Pivotal Cloud Foundry®

Version 1.7

© 2018 Pivotal Software, Inc.
# Table of Contents

- Pivotal Cloud Foundry Documentation Table of Contents
- Installing Pivotal Cloud Foundry
- Prerequisites to Deploying Operations Manager and Elastic Runtime
- Preparing Your Firewall for Deploying Pivotal Cloud Foundry
- Pivotal Cloud Foundry IaaS User Role Guidelines
- Installing Pivotal Cloud Foundry on AWS
- Deploying the CloudFormation Template for Pivotal Cloud Foundry on AWS
- Launching an Ops Manager Director Instance on AWS
- Configuring Ops Manager Director on AWS
- Deploying Elastic Runtime on AWS
- Deleting an AWS Installation from the Console
- Configuring Amazon EBS Encryption
- Creating a Proxy ELB for Diego SSH without CloudFormation
- Installing Pivotal Cloud Foundry on OpenStack
- Provisioning the OpenStack Infrastructure
- Configuring Ops Manager Director for OpenStack
- Installing Elastic Runtime on OpenStack
- Installing Pivotal Cloud Foundry on vSphere, vCloud, and vCloud Air
- Deploying Operations Manager to vSphere
- Configuring Ops Manager Director for VMware vSphere
- Deploying Operations Manager to vCloud Air and vCloud
- Configuring Ops Manager Director for vCloud Air and vCloud
- Configuring Elastic Runtime for vSphere, vCloud, and vCloud Air
- Provisioning a Virtual Disk in vSphere
- Using the Cisco Nexus 1000v Switch with Ops Manager
- Using Ops Manager Resurrection on VMware vSphere
- Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments
- Understanding Availability Zones in VMware Installations
- Upgrading Pivotal Cloud Foundry
- Upgrading Elastic Runtime and Other Pivotal Cloud Foundry Products
- PCF Dev Overview
- Using Operations Manager
- Understanding the Ops Manager Interface
- Adding and Deleting Products
- Understanding Floating Stemcells
- Creating UAA Clients for BOSH Director
- Understanding Pivotal Cloud Foundry User Types
- Starting and Stopping Pivotal Cloud Foundry Virtual Machines
- Creating and Managing Ops Manager User Accounts
- Logging in to Apps Manager
- Controlling Apps Manager User Activity with Environment Variables
- Configuring Your App Autoscaling Instance
- Managing Scheduled Scaling in the App Autoscaling Service
- Modifying Your Ops Manager Installation and Product Template Files
- Backing Up and Restoring Pivotal Cloud Foundry
- Backing Up Pivotal Cloud Foundry
- Restoring Pivotal Cloud Foundry from Backup

© Copyright Pivotal Software Inc, 2013-2018
1.7

© Copyright Pivotal Software Inc, 2013-2018
<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrating a Database in Cloud Foundry</td>
</tr>
<tr>
<td>Buildpacks</td>
</tr>
<tr>
<td>Java Buildpack</td>
</tr>
<tr>
<td>Getting Started Deploying Grails Apps</td>
</tr>
<tr>
<td>Getting Started Deploying Ratpack Apps</td>
</tr>
<tr>
<td>Getting Started Deploying Spring Apps</td>
</tr>
<tr>
<td>Tips for Java Developers</td>
</tr>
<tr>
<td>Configuring Service Connections for Grails</td>
</tr>
<tr>
<td>Configuring Service Connections for Play Framework</td>
</tr>
<tr>
<td>Configuring Service Connections for Spring</td>
</tr>
<tr>
<td>Cloud Foundry Eclipse Plugin</td>
</tr>
<tr>
<td>Cloud Foundry Java Client Library</td>
</tr>
<tr>
<td>Build Tool Integration</td>
</tr>
<tr>
<td>BOSH Custom Trusted Certificate Support</td>
</tr>
<tr>
<td>Ruby Buildpack</td>
</tr>
<tr>
<td>Getting Started Deploying Ruby Apps</td>
</tr>
<tr>
<td>Getting Started Deploying Ruby on Rails Apps</td>
</tr>
<tr>
<td>Deploy a Sample Ruby on Rails Application</td>
</tr>
<tr>
<td>Configure Rake Tasks for Deployed Apps</td>
</tr>
<tr>
<td>Tips for Ruby Developers</td>
</tr>
<tr>
<td>Environment Variables Defined by the Ruby Buildpack</td>
</tr>
<tr>
<td>Configure Service Connections for Ruby</td>
</tr>
<tr>
<td>Node.js Buildpack</td>
</tr>
<tr>
<td>Tips for Node.js Applications</td>
</tr>
<tr>
<td>Environment Variables Defined by the Node Buildpack</td>
</tr>
<tr>
<td>Configuring Service Connections for Node.js</td>
</tr>
<tr>
<td>Binary Buildpack</td>
</tr>
<tr>
<td>Go Buildpack</td>
</tr>
<tr>
<td>PHP Buildpack</td>
</tr>
<tr>
<td>Composer</td>
</tr>
<tr>
<td>Sessions</td>
</tr>
<tr>
<td>New Relic</td>
</tr>
<tr>
<td>PHP Buildpack Configuration</td>
</tr>
<tr>
<td>Deploying and Developing PHP Apps</td>
</tr>
<tr>
<td>Tips for PHP Developers</td>
</tr>
<tr>
<td>Python Buildpack</td>
</tr>
<tr>
<td>Staticfile Buildpack</td>
</tr>
<tr>
<td>Using Buildpacks</td>
</tr>
<tr>
<td>Buildpack Detection</td>
</tr>
<tr>
<td>Proxy Usage</td>
</tr>
<tr>
<td>Supported Binary Dependencies</td>
</tr>
<tr>
<td>Configuring a Production Server</td>
</tr>
<tr>
<td>Developing Buildpacks</td>
</tr>
<tr>
<td>Custom Buildpacks</td>
</tr>
<tr>
<td>Packaging Dependencies for Offline Buildpacks</td>
</tr>
<tr>
<td>Merging from Upstream Buildpacks</td>
</tr>
<tr>
<td>Upgrading Dependency Versions</td>
</tr>
<tr>
<td>Services</td>
</tr>
<tr>
<td>Overview</td>
</tr>
<tr>
<td>Service Broker API v2.8</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing Service Brokers</td>
<td>703</td>
</tr>
<tr>
<td>Access Control</td>
<td>706</td>
</tr>
<tr>
<td>Catalog Metadata</td>
<td>712</td>
</tr>
<tr>
<td>Dashboard Single Sign-On</td>
<td>716</td>
</tr>
<tr>
<td>Example Service Brokers</td>
<td>720</td>
</tr>
<tr>
<td>Binding Credentials</td>
<td>721</td>
</tr>
<tr>
<td>Application Log Streaming</td>
<td>723</td>
</tr>
<tr>
<td>Route Services</td>
<td>724</td>
</tr>
<tr>
<td>Manage Application Requests with Route Services</td>
<td>729</td>
</tr>
<tr>
<td>Supporting Multiple Cloud Foundry Instances</td>
<td>730</td>
</tr>
<tr>
<td>Logging and Metrics</td>
<td>731</td>
</tr>
<tr>
<td>Overview of the Loggregator System</td>
<td>732</td>
</tr>
<tr>
<td>Using Loggregator</td>
<td>732</td>
</tr>
<tr>
<td>Loggregator Components</td>
<td>732</td>
</tr>
<tr>
<td>Loggregator Guide for Cloud Foundry Operators</td>
<td>734</td>
</tr>
<tr>
<td>Application Logging in Cloud Foundry</td>
<td>735</td>
</tr>
<tr>
<td>Security Event Logging for Cloud Controller and UAA</td>
<td>738</td>
</tr>
<tr>
<td>Cloud Foundry Component Metrics</td>
<td>742</td>
</tr>
<tr>
<td>Deploying a Nozzle to the Loggregator Firehose</td>
<td>756</td>
</tr>
<tr>
<td>Cloud Foundry Data Sources</td>
<td>761</td>
</tr>
<tr>
<td>Installing the Loggregator Firehose Plugin for cf CLI</td>
<td>762</td>
</tr>
<tr>
<td>Pivotal Cloud Foundry Release Notes and Known Issues</td>
<td>763</td>
</tr>
<tr>
<td>Pivotal Elastic Runtime v1.7.0.0 Release Notes</td>
<td>764</td>
</tr>
<tr>
<td>Version 1.7.0.0 New Features</td>
<td>798</td>
</tr>
<tr>
<td>In memory...</td>
<td>800</td>
</tr>
<tr>
<td>Pivotal Cloud Foundry Ops Manager v1.7 Release Notes</td>
<td>801</td>
</tr>
<tr>
<td>Pivotal Elastic Runtime v1.7 Known Issues</td>
<td>811</td>
</tr>
<tr>
<td>Pivotal Cloud Foundry Ops Manager v1.7.0 Known Issues</td>
<td>813</td>
</tr>
</tbody>
</table>
Pivotal Cloud Foundry Documentation Table of Contents

- **Installing Pivotal Cloud Foundry**
  A quick guide to installing and getting started with Pivotal Cloud Foundry (PCF).

- **Upgrading Pivotal Cloud Foundry**
  A guide to upgrading Pivotal Cloud Foundry Operations Manager (Ops Manager), Pivotal Elastic Runtime, and product tiles.

- **PCF Dev**
  A guide to PCF Dev, a lightweight Pivotal Cloud Foundry (PCF) installation that runs on a single virtual machine (VM) on your workstation. PCF Dev is intended for application developers who want to develop and debug their applications locally on a PCF deployment.

- **Using Ops Manager**
  A guide to using the Pivotal Cloud Foundry Operations Manager interface to manage your PCF PaaS.

- **Elastic Runtime Concepts**
  An explanation of the components in Pivotal Cloud Foundry Elastic Runtime and how they work.

- **Operating Elastic Runtime**
  A guide to running the Elastic Runtime component of PCF.

- **Administrating Elastic Runtime**
  A guide to managing Elastic Runtime at the administrator level.

- **Using Apps Manager**
  A guide to using the web-based Apps Manager application for managing users, organizations, spaces, and applications.

- **Using the Cloud Foundry Command Line Interface (cf CLI)**
  A guide to the Cloud Foundry Command Line Interface (cf CLI) to deploy and manage your applications.

- **Deploying Applications**
  A guide for developers on deploying and troubleshooting applications running in Elastic Runtime (Cloud Foundry).

- **Buildpacks**
  A guide to using system buildpacks and extending your Elastic Runtime with custom buildpacks.

- **Custom Services**
  A guide to extending your Elastic Runtime with custom services.

- **Logging and Metrics**
  A guide to using Loggregator, the next generation system for aggregating and streaming logs and metrics from all of the user apps and system components in Elastic Runtime.

- **Release Notes and Known Issues**
  Release notes and known issues for Pivotal Operations Manager, Pivotal Elastic Runtime, and Pivotal Ops Metrics.
Installing Pivotal Cloud Foundry

Welcome to Pivotal Cloud Foundry!

The following IaaS-specific guides are intended to walk you through the process of getting your Pivotal Cloud Foundry (PCF) deployment up and running.

If you experience a problem while following the steps below, check the Known Issues, or refer to the PCF Troubleshooting Guide.

Once you have completed the steps in this guide, explore the documentation on docs.pivotal.io to learn more about Pivotal Cloud Foundry and the Pivotal product suite.

Prepare for Installation:

- Prerequisites to Deploying Operations Manager and Elastic Runtime
- Preparing Your Firewall for Deploying Pivotal Cloud Foundry
- Pivotal Cloud Foundry IaaS User Role Guidelines

Install Pivotal Cloud Foundry:

- Installing Pivotal Cloud Foundry on AWS
- Installing Pivotal Cloud Foundry on OpenStack
- Installing Pivotal Cloud Foundry on vSphere, vCloud, and vCloud Air

Note: Pivotal Cloud Foundry (PCF) for vCloud Air and vCloud Director is deprecated and availability is restricted to existing customers. Contact Support for more information.

Check out the 15-minute Getting Started with PCF tutorial for learning Pivotal Cloud Foundry app deployment concepts.

© Copyright Pivotal Software Inc, 2013-2018
This topic explains system requirements for deploying the Pivotal Operations Manager and Elastic Runtime applications.

General Requirements

The following are general requirements for deploying and managing the Pivotal Operations Manager and Elastic Runtime applications:

- User privileges that meet the minimum vCenter requirements according to the BOSH vSphere CPI.
- (Recommended) Ability to create a wildcard DNS record to point to your router or load balancer. Alternatively, you can use a service such as xip.io. (Example: 203.0.113.0.xip.io).

Elastic Runtime gives each application its own hostname in your app domain. With a wildcard DNS record, every hostname in your domain resolves to the IP address of your router or load balancer, and you do not need to configure an A record for each app hostname. For example, if you create a DNS record *.example.com pointing to your router, every application deployed to the example.com domain resolves to the IP address of your router.

- (Recommended) A network without DHCP available for deploying the Elastic Runtime VMs.

Sufficient IP Allocation:

- One IP address for each VM instance.
- An additional IP address for each instance that requires static IPs.
- An additional IP address for each errand.
- An additional IP address for each compilation worker. IPs needed = VM instances + static IPs + errands + compilation workers

The most recent version of the Cloud Foundry Command Line Interface (cf CLI).

One or more NTP servers.

vSphere/vCenter Requirements

- The most recent version of the Cloud Foundry Command Line Interface (cf CLI).
- One or more NTP servers.

The following are the minimum resource requirements for maintaining a Pivotal Cloud Foundry (PCF) deployment with Ops Manager and Elastic Runtime on vSphere:

- vSphere 6.0, 5.5, or 5.1.
- Disk space: 2TB recommended.
- Memory: 120GB
- Two public IP addresses: One for Elastic Runtime and one for Ops Manager
- vCPU cores: 80
- Overall CPU: 28 GHz
vSphere editions: standard and above.

Ops Manager must have HTTPS access to vCenter and ESX hosts on TCP port 443.

A configured vSphere cluster:

- If you enable vSphere DRS (Distributed Resource Scheduler) for the cluster, you must set the Automation level to Partially automated or Fully automated. If you set the Automation level to Manual, the BOSH automated installation will fail with a power_on_vm error when BOSH attempts to create virtual machines (VMs).
- Turn hardware virtualization off if your vSphere hosts do not support VT-x/EPT. If you are unsure whether the VM hosts support VT-x/EPT, then you can turn this setting off. If you leave this setting on and the VM hosts do not support VT-x/EPT, then each VM requires manual intervention in vCenter to continue powering on without the Intel virtualized VT-x/EPT. Refer to the vCenter help topic at Configuring Virtual Machines > Setting Virtual Processors and Memory > Set Advanced Processor Options for more information.

Ops Manager requires read/write permissions to the datacenter level of the vSphere Inventory Hierarchy to successfully install. Pivotal recommends using the default VMware Administrator System Role to achieve the appropriate permission level, or a custom role that has all privileges for all objects in the datacenter, including propagating privileges to children. For a complete list of required vSphere privileges, see the BOSH documentation.

**Note:** Be advised that Ops Manager might indicate that you do not have the appropriate rights to create/delete folders when this is untrue. If so, click Ignore errors and start the install to continue.

**Note:** For information on how IaaS user roles are configured, refer to the Pivotal Cloud Foundry IaaS User Role Guidelines topic.

### vCD/vCloud Air Requirements

The following are the minimum resource requirements for maintaining a Pivotal Cloud Foundry (PCF) deployment with Ops Manager and Elastic Runtime on vCloud Air:

- vCD 5.1, 5.2, or 5.6 (vCloud Air)
- Disk space: 1TB
- Memory: 120GB
- Two public IP addresses: One for Elastic Runtime and one for Ops Manager
- vCPU cores: 80
- Overall CPU: 28 GHz
- Virtual infrastructure administrator privileges to enable Elastic Runtime to automatically power VMs on and off

**Note:** For information on how IaaS user roles are configured, refer to the Pivotal Cloud Foundry IaaS User Role Guidelines topic.

**Note:** For more information about user privileges, refer to the "User Privileges by Role" section in the VMware vCloud Air User’s Guide.

### Amazon Web Services

The following are the minimum resource requirements for maintaining a Pivotal Cloud Foundry (PCF) deployment with Ops Manager and Elastic Runtime on Amazon Web Services infrastructure:

- 1 Elastic Load Balancer
- 1 Relational Database Service. We recommend at least a db.m3.xlarge instance with 100 GB of allocated storage.
- 5 S3 Buckets
- EC2 Instances:
  - 13 t2.micro
  - 15 t2.small
  - 2 m3.medium
  - 6 m3.xlarge
  - 3 m3.2xlarge

See Installing Pivotal Cloud Foundry on AWS using CloudFormation for more detailed installation requirements.
OpenStack

Pivotal has tested and certified Pivotal Cloud Foundry on Mirantis OpenStack versions 5.1 (IceHouse) and 6.1 (Juno). Other OpenStack releases and distributions based on Havana, Icehouse, and Juno may also function properly.

See Installing Pivotal Cloud Foundry on OpenStack for detailed requirements.
Preparing Your Firewall for Deploying Pivotal Cloud Foundry

This topic describes how to configure your firewall for Pivotal Cloud Foundry (PCF) and how to verify that PCF resolves DNS entries behind your firewall.

Configure Your Firewall for PCF

Ops Manager and Elastic Runtime require the following open TCP ports:

- **25555**: Routes from Ops Manager to the Ops Manager Director.
- **443**: Routes to HAPerxy or, if configured, your own load balancer
- **80**: Routes to HAPerxy or, if configured, your own load balancer
- **22 (Optional)**: Only necessary if you want to connect using SSH

UDP port **123** must be open if you want to use an external NTP server.

For more information about required ports for additional installed products, refer to the product documentation.

The following example procedure uses the Linux command `iptables` to configure a firewall.

```
1. Open `/etc/sysctl.conf`, a file that contains configurations for Linux kernel settings, with the command below:

   $ sudo vi /etc/sysctl.conf

2. Add the line `net.ipv4.ip_forward=1` to `/etc/sysctl.conf` and save the file.

3. If you are using Linux machines for your firewall and want to remove all existing filtering or Network Address Translation (NAT) rules, run the following commands:

   Note: This command destroys all iptables rules. You must back up or record your rules if you want to preserve them.

   $ iptables -flush
   $ iptables --flush -t nat

4. Add environment variables to use when creating the IP rules:

   ```
   $ export INTERNAL_NETWORK_RANGE=10.0.0.0/8
   $ export GATEWAY_INTERNAL_IP=10.0.0.1
   $ export PUBLIC_IP=203.0.113.242
   $ export GATEWAY_EXTERNAL_IP=PUBLIC_IP
   $ export PIVOTALCF_IP=10.0.0.2
   $ export HA_PROXY_IP=10.0.0.254
   ```

5. Run the following commands to configure IP rules for the specified chains:

   - **FORWARD**:
     ```
     $ iptables -A FORWARD -i eth1 -j ACCEPT
     $ iptables -A FORWARD -o eth1 -j ACCEPT
     ```

   - **POSTROUTING**:
     ```
     $ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
     $ iptables -t nat -A POSTROUTING -d SHA_PROXY_IP -s INTERNAL_NETWORK_RANGE -p tcp --dport 80 -j SNAT --to GATEWAY_EXTERNAL_IP
     $ iptables -t nat -A POSTROUTING -d SHA_PROXY_IP -s INTERNAL_NETWORK_RANGE -p tcp --dport 443 -j SNAT --to GATEWAY_EXTERNAL_IP
     ```

   - **PREROUTING**:

© Copyright Pivotal Software Inc, 2013-2018
### Verify PCF Resolves DNS Entries Behind a Firewall

When you install PCF in an environment that uses a strong firewall, the firewall might block DNS resolution. For example, if you use [xip.io](https://xip.io) to test your DNS configuration, the tests will fail without warning if the firewall prevents Elastic Runtime from accessing *.xip.io.

To verify that Elastic Runtime can correctly resolve DNS entries:

1. **SSH into the Pivotal Ops Manager VM.**
   - For more information, refer to the [SSH into Ops Manager](#) section of the Advanced Troubleshooting with the BOSH CLI topic.

2. Run any of the following network administration commands with the IP address of the VM:
   - `nslookup`
   - `dig`
   - `host`
   - The appropriate `traceroute` command for your OS

3. Review the output of the command and fix any blocked routes.
   - If the output displays an error message, review the firewall logs to determine which blocked route or routes you need to clear.

4. Repeat steps 1-3 with the Ops Manager Director VM and the HAProxy VM.
Pivotal Cloud Foundry IaaS User Role Guidelines

This topic describes practices recommended by Pivotal for creating secure IaaS user roles.

Pivotal Cloud Foundry (PCF) is an automated platform that connects to IaaS providers such as AWS and OpenStack. This connectivity typically requires accounts with appropriate permissions to act on behalf of the operator to access IaaS functionality such as creating virtual machines (VMs), managing networks and storage, and other related services.

Ops Manager and Elastic Runtime can be configured with IaaS users in different ways depending on your IaaS. Other product tiles and services might also use their own IaaS credentials. Refer to the documentation for those product tiles or services to configure them securely.

Least Privileged Users (LPUs)

Pivotal recommends following the principle of least privilege by scoping privileges to the most restrictive permissions possible for a given role. In the event that someone gets access to credentials by mistake or through malicious intent, LPUs limit the scope of the breach. Pivotal recommends following best practices for the particular IaaS you are deploying.

Configuring IaaS User Roles on AWS

Pivotal recommends using the CloudFormation templates for Pivotal Cloud Foundry to configure AWS deployments to create users with least privilege. Pivotal also recommends minimizing the use of master account credentials by creating an IAM role and instance profile with the minimum required EC2, VPC, and EBS credentials.

See the table below for more information on the two CloudFormation templates.

<table>
<thead>
<tr>
<th>Template Source</th>
<th>Location</th>
<th>User(s) Created</th>
<th>User Purpose</th>
<th>Uses IAM Role</th>
<th>Additional Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elastic Runtime</td>
<td>Pivotal Network Elastic Runtime Download [1]</td>
<td>ERT S3 user</td>
<td>Blob storage</td>
<td>No</td>
<td>Deploying Elastic Runtime on AWS</td>
</tr>
<tr>
<td>Ops Manager</td>
<td>Referenced in the ERT template</td>
<td>Ops Manager VM and Ops Manager Director</td>
<td>EC2, VPC, EBS, S3, ELB</td>
<td>Yes</td>
<td>Director User Config</td>
</tr>
</tbody>
</table>

For more Amazon-specific best practices, refer to the following Amazon documentation:


Configuring IaaS User Roles on vSphere

See the vCenter permissions recommendations in vSphere/vCenter Requirements.

Configuring IaaS User Roles on vCloud

See the installation instructions and follow the least privilege user configuration for accounts.

Configuring IaaS User Roles on OpenStack

See the installation instructions and follow the least privilege user configuration for tenants and identity.
Installing Pivotal Cloud Foundry on AWS

Page last updated:

This guide describes how to install Pivotal Cloud Foundry (PCF) on Amazon Web Services (AWS) using the PCF CloudFormation template.

The CloudFormation template for Pivotal Cloud Foundry describes the set of necessary AWS resources and properties. When you create an AWS stack using the PCF template, CloudFormation provisions all the infrastructure that you need to deploy PCF on AWS.

Pivotal strongly recommends using CloudFormation to install PCF on AWS. Contact Pivotal Support if you cannot use CloudFormation for your installation.

Prerequisites

You must have the following in order to follow the procedure described here:

- An AWS account that can accommodate the minimum resource requirements for a PCF installation.
- The appropriate region selected within your AWS account. See the Amazon documentation on regions and availability zones for help selecting the correct region for your deployment.
- The AWS CLI installed on your machine, and configured with user credentials that have admin access to your AWS account.
- Sufficiently high instance limits (or no instance limits) on your AWS account. Installing Pivotal Cloud Foundry requires more than the default 20 concurrent instances.
- A key pair to use with your Pivotal Cloud Foundry deployment. Create a key pair in AWS.
- A registered wildcard domain for your PCF installation. You will need this registered domain when configuring your SSL certificate and Cloud Controller. See the AWS docs on Creating a Server Certificate for more information.
- An SSL certificate for your PCF domain. This can be a self-signed certificate, but Pivotal only recommends using a self-signed certificate for testing and development. You should obtain a certificate from your Certificate Authority for use in production. See the AWS docs on SSL certificates for more information.

Install PCF on AWS

Complete the following procedures to install PCF using CloudFormation:

1. Deploying the CloudFormation Template for PCF on AWS
2. Launching an Ops Manager Director Instance on AWS
3. Configuring Ops Manager Director on AWS
4. (Optional) Installing the PCF IPsec Add-On
5. Deploying Elastic Runtime on AWS

Delete PCF on AWS

You can use the AWS console to remove an installation of all components, but retain the objects in your bucket for a future deployment:

- Deleting an AWS Installation from the Console
Additional AWS Configuration

See the following topics for additional AWS configuration information:

- Configuring Amazon EBS Encryption
- Creating a Proxy ELB for Diego SSH without CloudFormation
Deploying the CloudFormation Template for Pivotal Cloud Foundry on AWS

This topic describes how to deploy the CloudFormation template for Pivotal Cloud Foundry (PCF) on Amazon Web Services (AWS).

An AWS CloudFormation template describes a set of AWS resources and properties. Follow the instructions below to use a CloudFormation template to create the infrastructure that you need to deploy PCF to AWS.

The template is designed to output the resources necessary for two availability zones (AZ), with a private and public subnet designated for each AZ. The Elastic Load Balancer will be attached to the public subnet of both AZs to balance traffic across both environments. Three AZs is actually recommended as the desired number of AZs for a highly available deployment of PCF, however many AWS regions only have two AZs available.

Note: The CloudFormation template for Elastic Runtime includes a reference to another CloudFormation template for Ops Manager. For more information on how IaaS user roles are configured for each template, refer to the Pivotal Cloud Foundry IaaS User Role Guidelines topic.

Note: Before following the procedure below, confirm that you have selected the correct region within your AWS account. All of the AWS resources for your deployment must exist within a single region. See the Amazon documentation on regions and availability zones for help selecting the correct region for your deployment.

Step 1: Download the PCF CloudFormation Template

1. Sign in to Pivotal Network.

2. Select Elastic Runtime. From the Releases drop-down menu, select the release that you wish to install.

3. Download the PCF 1.7 CloudFormation script for AWS.

4. Save the file as pcf.json.

Step 2: Upload an SSL Certificate to AWS

You can add an SSL Certificate using two methods:

- The AWS CLI
- The AWS Certificate Manager

(Option) Create SSL Certificate using the AWS CLI

The AWS CLI must be installed on your machine and configured to a user account with admin access privileges on your AWS account.

1. Obtain or create an SSL server certificate. See the AWS docs on SSL certificates. When you create a certificate signing request (CSR) in the “Create a Server Certificate” instructions, you must use your system wildcard domain (example: *my-pcf-apps-domain.com) as the Common Name input.


   Note: If you are using a self-signed certificate or selected the “Generate Self-Signed RSA Certificate” option during the Deploying Elastic Runtime on AWS installation process, you can ignore the step above. However, make sure you upload the self-signed certificate to AWS and attach the certificate to the listeners on the AWS Elastic Load Balancer. Pivotal recommends only using a self-signed certificate for testing and development.

3. Upload your SSL certificate to AWS. For more information, see the AWS docs on uploading SSL certificate using the CLI.

   ```sh
d   aws iam upload-server-certificate
   --server-certificate-name YOUR-CERTIFICATE
   --certificate-body file://YOUR-PUBLIC-KEY-CERT-FILE.pem
   --private-key file://YOUR-PRIVATE-KEY-FILE.pem
   ```

Note: Before following the procedure below, confirm that you have selected the correct region within your AWS account. All of the AWS resources for your deployment must exist within a single region. See the Amazon documentation on regions and availability zones for help selecting the correct region for your deployment.

© Copyright Pivotal Software Inc, 2013-2018
For example:

```bash
$ aws iam upload-server-certificate
  --server-certificate-name myServerCertificate
  --certificate-body file://my-certificate.pem
  --private-key file://my-private-key.pem
```

4. After successfully uploading the certificate to your AWS account, you will see output metadata for your certificate. For example:

```json
{
  "ServerCertificateMetadata": {
    "ServerCertificateId": "ASCAI3HRFYUTD55KNAF64",
    "ServerCertificateName": "myServerCertificate",
    "Expiration": "2016-10-18T18:41:59Z",
    "Path": "/",
    "Arn": "arn:aws:iam::9240874958318:server-certificate/myServerCertificate",
    "UploadDate": "2015-10-19T19:10:57.404Z"
  }
}
```

5. Record the value of the `Arn` key to use when configuring your AWS resource stack. Alternatively, if you know the name of the certificate, you can run the following command to retrieve certificate metadata at a later point:

```bash
$ aws iam get-server-certificate --server-certificate-name YOUR-CERT-NAME
```

For example:

```bash
$ aws iam get-server-certificate --server-certificate-name myServerCertificate
```

**(Option)** Create SSL Certificate using the AWS Certificate Manager

Refer to the Amazon Certificate Manager documentation to use the AWS Certificate Manager to manage your SSL certificates and generate a certificate ARN. You will need the ARN in the next step.

### Step 3: Create a Resource Stack Using the CloudFormation Template

1. Log in to the AWS Console.

2. In the second column, under Management Tools, click CloudFormation.
3. Click Create New Stack.

4. Select Upload a template to Amazon S3.

5. Click Browse. Browse to and select the `pcf.json`, the Pivotal Cloud Foundry Cloudformation script for AWS file that you downloaded. Click Next.

6. On the next screen, name the stack `pcf-stack`.

7. In the Specify Parameters page, complete the following fields:
- **01NATKeyPair**: Use the drop-down menu to select the name of your pre-existing AWS key pair. If you do not have a pre-existing key pair, create one in [AWS](https://aws.amazon.com) and return to this step.
- **02NATInstanceType**: Do not change this value.
- **03OpsManagerIngress**: Do not change this value.
- **04RdsDBName**: Do not change this value.
- **05RdsUserName**: Enter a username for the RDS database.
- **06RdsPassword**: Enter a password for the RDS database.
- **07SSLCertificateARN**: Enter your SSL Certificate ARN.
- **08OpsManagerTemplate**: The default template link provided here works. Otherwise you can enter your own S3 bucket location of the Ops Manager CloudFormation script.
- **09ElbPrefix**: Prefix for the generated names of the ELBs. Any string you specify in this field will be prefixed to `-pcf-elb` to form the name of your ELBs. Leave empty to use the default prefix of `AWS::StackName`.
- **10AllowHttpOnElb**: Set this to **true** to listen for HTTP traffic on port 80. This is the default. Set it to **false** to only listen for traffic on ports 443 and 4443.

8. Click **Next**.

9. On the **Options** page, leave the fields blank and click **Next**.
10. On the Review page, select the I acknowledge that this template might cause AWS CloudFormation to create IAM resources checkbox and click Create.

AWS runs the CloudFormation script and creates the infrastructure that you need to deploy PCF to AWS. This may take a few moments. You can click on the Events tab to view the progress of the setup.

When the installation process successfully completes, AWS displays CREATE_COMPLETE as the status of the stack.
After completing this procedure, complete all of the steps in the following topics:

- Launching an Ops Manager Director Instance on AWS
- Configuring Ops Manager Director for AWS
- Deploying Elastic Runtime on AWS

Return to Installing Pivotal Cloud Foundry Using AWS CloudFormation.
Launching an Ops Manager Director Instance on AWS

This topic describes how to deploy Ops Manager Director after deploying the CloudFormation template for Pivotal Cloud Foundry (PCF) on Amazon Web Services (AWS).

Before beginning this procedure, ensure that you have successfully completed all of the steps in the Deploying the CloudFormation Template for PCF on AWS topic. After you complete this procedure, follow the instructions in the Configuring Ops Manager Director on AWS CloudFormation and Configuring Elastic Runtime on AWS CloudFormation topics.

Step 1: Open the Outputs Tab in AWS Stacks

1. In the dashboard of your AWS Console, click CloudFormation. The Stacks Dashboard displays.

2. Select the pcf-stack checkbox, then select the Outputs tab.

![Outputs Tab in AWS Stacks](image)

In the steps described below, use the information from the Value column of the Outputs tab to configure your PCF installation.

Step 2: Select a Pivotal Ops Manager AMI Instance
1. Log into the Pivotal Network and click on Ops Manager.

2. From the Releases dropdown, select the release you wish to install.

3. Select Pivotal Cloud Foundry Ops Manager for AWS to download the OpsManagers.x.x.xonAWSFulfillmentInstructions.pdf file. This document lists AMI IDs for Pivotal Ops Manager for specific regions.

4. Log in to the AWS Console. Navigate to the EC2 Dashboard.

5. In the left navigation panel, click AMIs.

6. Using the OpsManagers.x.x.xonAWSFulfillmentInstructions.pdf document, enter the AMI ID for your AWS region in the Public images search field. This search locates the appropriate Pivotal Ops Manager AMI for your region within public images.

7. Select this AMI and click Launch.

8. Choose m3.large for your instance type.

9. Click Next: Configure Instance Details.

Step 3: Configure Instance Details

1. Complete the Config Instance Details page with information from the Outputs tab in the AWS Stacks Dashboard.
Select the Network that matches the value of PcfVpc.
Select the Subnet that matches the value of PcfPublicSubnetId.

2. Set Auto-assign Public IP to Enable.

3. Click Next: Add Storage.

4. On the Add Storage page, adjust the Size (GiB) value. Pivotal recommends increasing this value to a minimum of 100 GB.

5. Click Next: Tag Instance.

6. On the Tag Instance page, add a Key Name with Value Ops Manager.


Step 4: Configure Security Group

1. Select the Select an existing security group option.

2. Select the Security Group ID that matches the value of PcfOpsManagerSecurityGroupId located in the Outputs tab of the Stacks dashboard.
3. Click Review and Launch.

Step 5: Deploy Ops Manager

1. Review the instance launch details. Click Launch.

2. Use the first drop-down menu to select Choose an existing key pair. Use the second drop-down menu to select the name of your pre-existing AWS key pair.

3. Select the acknowledgement checkbox.

4. Click Launch Instances. If successful, you will see the Launch Status Page.
5. Click **View Instances**. Or alternately, navigate to **Instances** from the left navigation panel of the EC2 Dashboard.

6. AWS deploys Ops Manager. This may take a few minutes. When complete, AWS displays an **Instance State** of **running** and a **Status Check** of **passed** when the Ops Manager deployment successfully completes.

### Step 6: Create a DNS Entry

> **Note**: Ops Manager 1.7 security features require you to create a fully qualified domain name in order to access Ops Manager during the initial configuration.

Create a DNS entry for the IP address that you used for Ops Manager. You must use this fully qualified domain name when you log into Ops Manager in the Configure Ops Manager Director for AWS step below.

### Step 7: Configure Ops Manager Director for AWS

After you complete this procedure, follow the instructions in the Configuring Ops Manager Director on AWS CloudFormation and Configuring Elastic Runtime on AWS CloudFormation topics.

Return to Installing Pivotal Cloud Foundry Using AWS CloudFormation.
Configuring Ops Manager Director on AWS

This topic describes how to configure the Ops Manager Director after deploying the CloudFormation template for Pivotal Cloud Foundry (PCF) on Amazon Web Services (AWS). Use this topic when Installing Pivotal Cloud Foundry Using AWS CloudFormation.

Before beginning this procedure, ensure that you have successfully completed all steps in the Deploying the CloudFormation Template for PCF on AWS and the Launching an Ops Manager Director Instance on AWS CloudFormation topics. After you complete this procedure, follow the instructions in the Deploying Elastic Runtime on AWS CloudFormation topic.

Step 1: Open the Outputs Tab in AWS Stacks

1. In the dashboard of your AWS Console, click CloudFormation. The Stacks Dashboard displays.

2. Select the pcf-stack checkbox, then select the Outputs tab.

Step 2: Access Ops Manager
1. In a web browser, navigate to the fully qualified domain you created in the Create a DNS Entry step of Launching an Ops Manager Director Instance on AWS.

2. When Ops Manager starts for the first time, you must choose one of the following:
   - **Use an Identity Provider**: If you use an Identity Provider, an external identity server maintains your user database.
   - **Internal Authentication**: If you use Internal Authentication, PCF maintains your user database.

**Use an Identity Provider (IdP)**

1. Log in to your IdP console and download the IdP metadata XML. Optionally, if your IdP supports metadata URL, you can copy the metadata URL instead of the XML.

2. Copy the IdP metadata XML or URL to the Ops Manager Use an Identity Provider log in page.

   ![IdP metadata](https://example.com/idp/metadata.xml)

   **Note**: The same IdP metadata URL or XML is applied for the BOSH Director. If you are using a separate IdP for BOSH, copy the metadata XML or URL from that IdP and enter it into the BOSH IdP Metadata text box in the Ops Manager log in page.

3. Enter your **Decryption passphrase**. Read the **End User License Agreement**, and select the checkbox to accept the terms.

4. Your Ops Manager log in page appears. Enter your username and password. Click Login.

5. Download your SAML Service Provider metadata (SAML Relying Party metadata) by navigating to the following urls:
   - 5a. Ops Manager SAML service provider metadata: https://OPS-MAN-FQDN:443/uaa/saml/metadata
   - 5b. BOSH Director SAML service provider metadata: https://BOSH-IP-ADDRESS:8443/uaa/saml/metadata

---

© Copyright Pivotal Software Inc, 2013-2018
6. Configure your IdP with your SAML Service Provider metadata. Import the Ops Manager SAML provider metadata from Step 5a above to your IdP. If your IdP does not support importing, provide the values below:
   - **Single sign on URL:** https://OPS-MAN-FQDN:443/uaa/saml/SSO/alias/OPS-MAN-FQDN
   - **Audience URI (SP Entity ID):** https://OPS-MAN-FQDN:443/uaa
   - **Name ID is Email Address**
   - SAML authentication requests are always signed

7. Import the BOSH Director SAML provider metadata from Step 5b to your IdP. If the IdP does not support an import, provide the values below.
   - **Single sign on URL:** https://BOSH-IP:8443/saml/SSO/alias/BOSH-IP
   - **Audience URI (SP Entity ID):** https://BOSH-IP:8443
   - **Name ID is Email Address**
   - SAML authentication requests are always signed

8. Return to the **Ops Manager Director** tile, and continue with the configuration steps below.

### Internal Authentication

1. When redirected to the **Internal Authentication** page, you must complete the following steps:
   - Enter a **Username, Password**, and **Password confirmation** to create an Admin user.
   - Enter a **Decryption passphrase** and the **Decryption passphrase confirmation**. This passphrase encrypts the Ops Manager datastore, and is not recoverable if lost.
   - If you are using an **Http proxy** or **Https proxy**, follow the these instructions.
   - Read the **End User License Agreement**, and select the checkbox to accept the terms.
   - Click **Setup Authentication**.

2. Log in to Ops Manager with the Admin username and password that you created in the previous step.
Step 3: AWS Config Page

1. Click the Ops Manager Director tile.

2. Select AWS Config to open the AWS Management Console Config page.
3. Select **Use AWS Keys** or **Use AWS Instance Profile**.
   - If you choose to use AWS keys, complete the fields with information from the Outputs tab for your stack in the AWS Console:
     - **Access Key ID**: Use the value of `PcfIamUserAccessKey`.
     - **AWS Secret Key**: Use the value of `PcfIamUserSecretAccessKey`.
   - If you choose to use an AWS instance profile, enter the name of your AWS Identity and Access Management (IAM) profile.

4. Complete the remainder of the AWS Management Console Config page with the following information.
   - **VPC ID**: Use the value of `PcfVpc` from your Outputs tab.
   - **Security Group ID**: Open the AWS EC2 Dashboard and click Security Groups. Select the security group with the Description `PCF VMs Security Group`. Copy the Group ID of this group into the Ops Manager Security Group ID field.
   - **Key Pair Name**: Use the name of your pre-existing AWS key pair. You selected this key pair name when you first deployed Ops Manager Director.
   - **SSH Private Key**: Open your AWS key pair `.pem` file in a text editor. Copy the contents of the `.pem` file and paste it into the SSH Private Key field.
   - **Region**: Select the region where you deployed Ops Manager.
   - **Encrypt EBS Volumes**: Select this checkbox to enable full encryption on persistent disks of all BOSH-deployed virtual machines (VMs), except for the Ops Manager VM and Director VM. See the Configuring Amazon EBS Encryption for PCF on AWS topic for details on using EBS encryption.

5. Click **Save**.
Step 4: Director Config Page

1. Select Director Config to open the Director Config page.

   ![Director Config](image)

   - NTP Servers (comma delimited):
     - 0.amazon.pool.ntp.org
     - 1.amazon.pool.ntp.org
     - 2.amazon.pool.ntp.org
     - 3.amazon.pool.ntp.org
   - Metrics IP Address:
   - Enable VM Resurrector Plugin
   - Recreate all VMs
     - This will force BOSH to recreate all VMs on the next deploy. Persistent disk will be preserved.
   - Keep Unreachable Director VMs

2. In the NTP Servers (comma delimited) field, enter at least two of the following NTP servers, separated by a comma:
   - 0.amazon.pool.ntp.org
   - 1.amazon.pool.ntp.org
   - 2.amazon.pool.ntp.org
   - 3.amazon.pool.ntp.org

3. (Optional) Enter your Metrics IP Address if you are Using JMX Bridge.

4. Select the Enable VM Resurrector Plugin checkbox to enable the Ops Manager Resurrector functionality and increase Elastic Runtime availability.

5. Select Recreate all VMs to force BOSH to recreate all VMs on the next deploy. This process does not destroy any persistent disk data.

6. Select Keep Unreachable Director VMs if you want to preserve Ops Manager Director VMs after a failed deployment for troubleshooting purposes.

   ![HM Pager Duty Plugin](image)

   - Service Key: Enter your API service key from PagerDuty.
   - HTTP Proxy: Enter an HTTP proxy for use with PagerDuty.
8. Select **HM Email Plugin** to enable Health Monitor integration with email.
   - **Host**: Enter your email hostname.
   - **Port**: Enter your email port number.
   - **Domain**: Enter your domain.
   - **From**: Enter the address for the sender.
   - **Recipients**: Enter comma-separated addresses of intended recipients.
   - **Username**: Enter the username for your email server.
   - **Password**: Enter the password for your email server.
   - **Enable TLS**: Select this checkbox to enable Transport Layer Security.

9. For **Blobstore Location**, select **S3 Compatible Blobstore** and complete the following steps:
In a browser, reference the Amazon Simple Storage Service (Amazon S3) table, and find the region for your AWS account.

Prepend https:// to the Endpoint for your region, and copy it into the Ops Manager S3 Endpoint field. For example, in the us-west-2 region, enter https://s3-us-west-2.amazonaws.com into the field.

Complete the following fields with information from the Outputs tab in the AWS Console:

- **Bucket Name**: Use the value of PcfOpsManagerS3Bucket.
- **Access Key ID**: Use the value of PcfIamUserAccessKey.
- **AWS Secret Key**: Use the value of PcfIamUserSecretAccessKey.
- Select **V2 Signature** or **V4 Signature**. If you select V4 Signature, enter your **Region**.
- **AWS recommends using Signature Version 4 when possible.**

**Note**: For more information about AWS S3 Signatures, see the Authenticating Requests documentation.

10. For **Database Location**, select **External MySQL Database**. Complete the following fields with information from the Outputs tab in the AWS Console.
Host: Use the value of PcfRdsAddress.
Port: Use the value of PcfRdsPort.
Username: Use the value of PcfRdsUsername.
Password: Use the value of PcfRdsPassword.

Database: Use the value of PcfRdsDBName.

Max Threads sets the maximum number of threads that the Ops Manager Director can run simultaneously. Pivotal recommends that you leave the field blank to use the default value, unless doing so results in rate limiting or errors on your IaaS.
12. (Optional) To add a custom URL for your Ops Manager Director, enter a valid hostname in Director Hostname. You can also use this field to configure a load balancer in front of your Ops Manager Director.

13. Click Save.

**Note:** For more information about AWS S3 Signatures, see the Authenticating Requests documentation.

---

### Step 5: Create Availability Zones Page

**Note:** Pivotal recommends at least three Availability Zones for a highly available installation of Elastic Runtime.

1. Select Create Availability Zones.

![Create Availability Zones](image)

2. Use the following steps to create one or more Availability Zones for your applications to use:
   - Click Add.
   - For Amazon Availability Zone, enter the value of `PcfPrivateSubnetAvailabilityZone` from the Outputs tab in the AWS Console.
   - (Optional) If you are using a second Amazon Availability Zone, click Add. Enter the value of `PcfPrivateSubnet2AvailabilityZone` from the Outputs tab in the AWS Console.
   - Click Save.

---

### Step 6: Create Networks Page

1. Select Create Networks.
2. Select Enable ICMP checks to enable ICMP on your networks. Ops Manager uses ICMP checks to confirm that components within your network are reachable.

3. Use the following steps to create one or more Ops Manager networks:
   - Click Add Network.
   - Enter a unique Name for the network.
   - Click Add Subnet to create one or more subnets for the network.
   - In the VPC Subnet ID field, use the value of PcfPrivateSubnetId from the Outputs tab in the AWS Console.
   - For CIDR, enter 10.0.16.0/20. Ops Manager deploy VMs to this CIDR block.
   - For Reserved IP Ranges, enter 10.0.16.1-10.0.16.9. Ops Manager avoids deploying VMs to any IP address in this range.
   - Enter 10.0.0.2 for DNS and 10.0.16.1 for Gateway.
   - Select which Availability Zones to use with the network.
   - (Optional) If you are using a second subnet, click Add Subnet. In the VPC Subnet ID field, use the value of PcfPrivateSubnet2Id. Enter the rest of the fields using the information provided above.
4. Click Save.

5. If the following ICMP error message appears, you can ignore the warning. Dismiss the warning, and move on to the next step.

---

**Step 7: Assign AZs and Networks Page**

1. Select Assign AZs and Networks.

2. Use the drop-down menu to select a **Singleton Availability Zone**. The Ops Manager Director installs in this Availability Zone.

3. Use the drop-down menu to select a **Network** for your Ops Manager Director.

4. Click **Save**.

---

**Step 8: Security Page**

1. Select **Security**.
2. In **Trusted Certificates**, enter a custom certificate authority (CA) certificate to insert into your organization’s certificate trust chain. This feature enables all BOSH-deployed components in your deployment to trust a custom root certificate. If you want to use Docker Registries for running app instances in Docker containers, use this field to enter your certificate for your private Docker Registry. See the Using Docker Registries topic for more information.

3. Choose **Generate passwords** or **Use default BOSH password**. Pivotal recommends that you use the **Generate passwords** option for greater security.

4. Click **Save**. To view your saved Director password, click the **Credentials** tab.

**Step 9: Resource Config Page**

1. Select **Resource Config**.

2. Adjust any values as necessary for your deployment. Under the **Instances**, **Persistent Disk Type**, and **VM Type** fields, choose **Automatic** from the
drop-down menu to allocate the recommended resources for the job. If the Persistent Disk Type field reads None, the job does not require persistent disk space.

Note: If you set a field to Automatic and the recommended resource allocation changes in a future version, Ops Manager automatically uses the updated recommended allocation.

Step 10: Complete the Ops Manager Director Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes. If the following ICMP error message appears, click Ignore errors and start the install.

3. Ops Manager Director installs. This may take a few moments. When the installation process successfully completes, the Changes Applied window appears.

4. After you complete this procedure, follow the instructions in the Deploying Elastic Runtime on AWS CloudFormation topic.
Deploying Elastic Runtime on AWS

Page last updated:

This topic describes how to install and configure Elastic Runtime after deploying the CloudFormation template for Pivotal Cloud Foundry (PCF) on Amazon Web Services (AWS). Use this topic when installing Pivotal Cloud Foundry on AWS.

Before beginning this procedure, ensure that you have successfully completed all steps in the Deploying the CloudFormation Template for PCF on AWS and Configuring Ops Manager Director after Deploying PCF on AWS using CloudFormation topics.

Note: If you plan to install the PCF IPsec add-on, you must do so before installing any other tiles. Pivotal recommends installing IPsec immediately after Ops Manager, and before installing the Elastic Runtime tile.

Step 1: Open the Outputs Tab in AWS

1. In the dashboard of your AWS Console, click CloudFormation. The Stacks Dashboard displays.

2. Select the pcf-stack checkbox, then select the Outputs tab.

In the steps described below, use the information from the Value column of the Outputs tab to configure your PCF installation.
Step 2: Add Elastic Runtime to Ops Manager

1. Navigate to the Pivotal Cloud Foundry Operations Manager Installation Dashboard.

2. If you have not downloaded Elastic Runtime, click the Pivotal Network link on the left to download the Elastic Runtime .pivotal file. Click **Import a Product** to add the tile to Ops Manager. For more information, refer to the *Adding and Deleting Products* topic.

3. Click the **Elastic Runtime** tile in the Installation Dashboard.

Step 3: Assign Availability Zones and Networks

1. Select **Assign AZ and Networks**. These are the Availability Zones that you create when configuring Ops Manager Director.

2. Select an Availability Zone under **Place singleton jobs**. Ops Manager runs any job with a single instance in this Availability Zone.

3. Select one or more Availability Zones under **Balance other jobs**. Ops Manager balances instances of jobs with more than one instance across the Availability Zones that you specify.

4. From the **Network** drop-down box, choose the network on which you want to run Elastic Runtime.
Step 4: Add CNAME Record for Your Custom Domain

5. Click Save.

Note: When you save this form, a verification error displays because the PCF security group blocks ICMP. You can ignore this error.
In the **Use the AWS CLI to upload your SSL Cert** step, you uploaded an SSL certificate for your PCF wildcard domain to AWS. In this step you redirect all wildcard queries for your domain to the DNS name of your ELB.

**Note:** Do not point your wildcard domain at the numeric IP address for your ELB because this changes frequently.

1. Find the DNS hostname of your ELB. The **Output** tab of the CloudFormation page in the AWS dashboard lists this as the value for the key `PcfElbDnsName`.

2. Log in to the DNS registrar that hosts your domain (for example, Network Solutions, GoDaddy, or Register.com).

3. Create a CNAME record with your DNS registrar that points `*.YOUR-DOMAIN.com` to the DNS hostname of your ELB.

4. Save changes within the web interface of your DNS registrar.

5. In the terminal, run the following `dig` command to confirm that you created your CNAME record successfully:

   ```
   dig xyz.MY-DOMAIN.COM
   ```

   You should see the CNAME record that you just created:

   ```
   ;; ANSWER SECTION:
   ```

**Note:** You must complete this step before proceeding to Cloud Controller configuration. A difficult-to-resolve problem can occur if the wildcard domain is improperly cached before the CNAME is registered.

---

### Step 5: Configure Domains

1. Select **Domains**.

   Elastic Runtime gives each deployed application its own hostname in the app domain, and each system component its own hostname on the system domain. Ensure that you have a wildcard DNS record configured for both domains.

   ```
   System Domain *
   pivotai.cf-app.com
   ```

   ```
   Apps Domain *
   pivotai.cf-app.com
   ```

   This is the default apps domain that pushed apps will use for their hostnames. Use the Cloud Foundry command line interface to add or delete domains. This also requires a wildcard DNS record.

2. Enter the system and application domains.
   - The **System Domain** defines your target when you push apps to Elastic Runtime.
   - The **Apps Domain** defines where Elastic Runtime should serve your apps.

   **Note:** Pivotal recommends that you use the same domain name but different subdomain names for your system and app domains. This prevents system and apps routes from overlapping. You will require two DNS entries: one for the system and the other for apps. For example, `system.EXAMPLE.COM` and `apps.EXAMPLE.COM`. Point both domains to your internal router IP address, which can be found under the status tab in the Elastic Runtime tile.
3. Click **Save**.

## Step 6: Configure Networking

1. Select **Networking**.

2. Leave the **Router IPs** and **HAProxy IPs** fields blank.

3. Under **Configure the point-of-entry to this environment**, choose one of the following:
   - **External Load Balancer with Encryption**: Select this option if your deployment uses an external load balancer that can forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing. Complete the fields for the **Router SSL Termination Certificate and Private Key** and **Router SSL Ciphers**.
   - **External Load Balancer without Encryption**: Select this option if your deployment will terminate SSL connection and forward unencrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing. You can also select this option if you are installing PCF on AWS.
   - **HAProxy**: Select this option to use HAProxy as your first point of entry. Complete the fields for **SSL Certificate and Private Key**, and **HAProxy SSL Ciphers**. Select **Disable HTTP traffic to HAProxy** if you want the HAProxy to only allow HTTPS traffic. You can also generate a self-signed certificate using your wildcard system domain.

   **Note:** For details about different SSL termination point options, which correspond to different points-of-entry for Elastic Runtime, see the [Providing a Certificate for your SSL Termination Point](#) topic.

4. If you are not using SSL encryption or if you are using self-signed certificates, select **Disable SSL certificate verification for this environment**. Selecting this checkbox also disables SSL verification for route services.
Configure security and routing services for your platform. It is usually preferable to use your own load balancer instead of an HAProxy instance as your point-of-entry to the platform.

In the **Choose whether or not to enable route services** section, choose either **Enable route services** or **Disable route services**. Route services are a class of **marketplace services** that perform filtering or content transformation on application requests and responses. See the **Route Services** topic for details.

For Loggregator Port, you must enter 4443. In AWS deployments, port 4443 forwards SSL traffic that supports WebSockets from the ELB. Do not use the default port of 443.

(Optional) Use the **Applications Subnet** field if you need to avoid address collision with a third-party service on the same subnet as your apps. Enter a CIDR subnet mask specifying the range of available IP addresses assigned to your app containers. The IP range must be different from the network used by the system VMs.

(Optional) You can change the value in the **Applications Network Maximum Transmission Unit (MTU)** field. Pivotal recommends setting the MTU value for your application network to 1454. Some configurations, such as networks that use GRE tunnels, may require a smaller MTU value.

(Optional) To accommodate larger uploads over connections with high latency, increase the timeout value in **Router Timeout to Backends**. This value is specified in seconds.

(Optional) Increase the value of **Load Balancer Unhealthy Threshold** to specify the amount of time, in seconds, that the router continues to accept connections before shutting down. During this period, healthchecks may report the router as unhealthy, which causes load balancers to failover to other routers. Set this value to an amount greater than or equal to the maximum time it takes your load balancer to consider a router instance
unhealthy, given contiguous failed healthchecks.

11. Click **Save**.

### Step 7: Configure Application Containers

1. **Select Application Containers.**

   **Enable microservice frameworks, private Docker registries, and other services that support your applications at a container level.**

   - [ ] Enable Custom Buildpacks
   - [ ] Allow SSH access to app containers
   - **Private Docker Insecure Registry Whitelist**
   
   Docker Images Disk Cleanup Scheduling on Cell VMs*
   - [ ] Never clean up Cell disk-space
   - [x] Routinely clean up Cell disk-space
   - [ ] Clean up disk-space once threshold is reached

   **Save**

2. The **Enable Custom Buildpacks** checkbox governs the ability to pass a custom buildpack URL to the `-b` option of the `cf push` command. By default, this ability is enabled, letting developers use custom buildpacks when deploying apps. Disable this option by disabling the checkbox. For more information about custom buildpacks, refer to the `buildpacks` section of the PCF documentation.

3. The **Allow SSH access to app containers** checkbox controls SSH access to application instances. Enable the checkbox to permit SSH access across your deployment, and disable it to prevent all SSH Access. See **Application SSH Overview** for information on SSH access permissions at the space and app scope.
4. You can configure Elastic Runtime to run app instances in Docker containers by supplying their IP address range(s) in the Private Docker Insecure Registry Whitelist textbox. See the Using Docker Registries topic for more information.

5. Select your preference for Docker Images Disk-Cleanup on Cell VMs. If you choose Clean up disk-space once threshold is reached, enter a Threshold of Disk-Used in megabytes.

6. Click Save.

**Step 8: Configure Application Developer Controls**

1. Select Application Developer Controls.

2. Enter your intended maximum file upload size.

3. Enter your default RAM memory allocation per app.

4. Enter your default total RAM memory (RAM) quota per Org. You can change this in the CLI.

5. Enter your maximum and default disk quotas per app.

6. Enter your default service instances quota per Org. You can change this in the CLI.

7. Click Save.

**Step 9: Review Application Security Groups**

Setting appropriate Application Security Groups is critical for a secure deployment. Type X in the box to acknowledge that once the Elastic Runtime deployment completes, you will review and set the appropriate application security groups. See Restricting App Access to Internal PCF Components for
Setting appropriate Application Security Groups that control application network policy is the responsibility of the Elastic Runtime administration team. Please refer to the Application Security Groups topic in the Pivotal Cloud Foundry documentation for more detail on completing this activity after the Elastic Runtime deployment completes.

Type X to acknowledge that you understand this message *

x

Save

Step 10: Configure Authentication and Enterprise SSO

1. Select Authentication and Enterprise SSO.
2. To authenticate user sign-ons, your deployment can use one of three types of user database: the UAA server’s internal user store, an external SAML identity provider, and an external LDAP server.

   a. To use the internal UAA, select the Internal option and follow the instructions in Configuring UAA Password Policy to configure your password policy.
   b. To connect to an external identity provider via SAML, scroll down to select the SAML Identity Provider option and follow the instructions in Configuring PCF for SAML.
   c. To connect to an external LDAP server, scroll down to select the LDAP Server option and follow the instructions in Configuring LDAP.

3. (Optional) In the Apps Manager Access Token Lifetime, Apps Manager Refresh Token Lifetime, Cloud Foundry CLI Access Token Lifetime, Cloud Foundry CLI Refresh Token Lifetime fields, you can change the lifetimes of tokens granted for Apps Manager and cf CLI login access and refresh. Most deployments use the defaults.

4. (Optional) The Proxy IPs Regular Expression field contains a pipe delimited set of regular expressions that UAA considers to be reverse proxy IP addresses. UAA respects the x-forwarded-for and x-forwarded-proto headers coming from IP addresses that match to these regular expressions. To configure UAA to respond properly to Router or HAProxy requests coming from public IP address(es), append a regular expression or regular expressions to match the public IP address(es).
Step 11: Create System Databases

You must create the databases required by Elastic Runtime on the RDS instance provisioned by the CloudFormation script.

1. Add the AWS-provided key pair to your SSH profile so that you can access the Ops Manager VM:
   ```
   ssh-add aws-keypair.pem
   ```

2. SSH into your Ops Manager using the Ops Manager FQDN and the username `ubuntu`:
   ```
   ssh ubuntu@OPS_MANAGER_FQDN
   ```

3. Run the following terminal command to log in to your RDS instance through the MySQL client, using values from your AWS dashboard Outputs tab to fill in the following output keys:
   ```
   mysql --host=PcfRdsAddress --user=PcfRdsUsername --password=PcfRdsPassword
   ```
   For example:
   ```
   mysql --host=pp19dd336auydlw.cpdgtp8njpud.us-west-2.rds.amazonaws.com --user=docs --password=jks563!fjlksd
   ```

4. Run the following MySQL commands to create databases for each of the five Elastic Runtime components that require a database:
   ```
   CREATE database uaa;
   CREATE database ccdb;
   CREATE database console;
   CREATE database notifications;
   CREATE database autoscale;
   CREATE database app_usage_service;
   ```

5. Type `exit` to quit the MySQL client and `exit` again to close your connection to the Ops Manager VM.

Step 12: Configure System Databases

Note: If you are performing an upgrade, do not modify your existing internal database configuration or you may lose data. You must migrate your existing data first before changing the configuration. See Upgrading Pivotal Cloud Foundry for additional upgrade information.

1. Select Databases.

2. Select the External Databases option.
3. For the Hostname and TCP Port fields, enter the corresponding values from the Outputs tab in the AWS Console, according to the following table:

<table>
<thead>
<tr>
<th>Elastic Runtime Field</th>
<th>Outputs Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>PcfRdsAddress</td>
</tr>
<tr>
<td>TCP Port</td>
<td>PcfRdsPort</td>
</tr>
</tbody>
</table>

4. For each database username and database password field, enter the corresponding values from the Outputs tab in the AWS Console, according to the following table:

<table>
<thead>
<tr>
<th>Elastic Runtime Field</th>
<th>Outputs Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE-NAME database username</td>
<td>PcfRdsUsername</td>
</tr>
<tr>
<td>DATABASE-NAME database password</td>
<td>PcfRdsPassword</td>
</tr>
</tbody>
</table>

5. Click Save.

Step 13: (Optional) Configure Internal MySQL

- Note: You only need to configure this section if you have selected Internal Databases - MySQL in the Databases section.

1. Select Internal MySQL

2. In the MySQL Proxy IPs field, enter one or more comma-delimited IP addresses that are not in the reserved CIDR range of your network. If a MySQL node fails, these proxies re-route connections to a healthy node. See the MySQL Proxy topic for more information.
3. For **MySQL Service Hostname**, enter an IP address or hostname for your load balancer. If a MySQL proxy fails, the load balancer re-routes connections to a healthy proxy. If you leave this field blank, components are configured with the IP address of the first proxy instance entered above.

⚠️ **Warning:** You must configure a load balancer to achieve complete high-availability.

4. In the **Replication canary time period** field, leave the default of 30 seconds or modify the value based on the needs of your deployment. Lower numbers cause the canary to run more frequently, which means that the canary reacts more quickly to replication failure but adds load to the database.

5. In the **Replication canary read delay** field, leave the default of 20 seconds or modify the value based on the needs of your deployment. This field configures how long the canary waits, in seconds, before verifying that data is replicating across each MySQL node. Clusters under heavy load can experience a small replication lag as write-sets are committed across the nodes.

6. **(Required):** In the **E-mail address** field, enter the email address where the MySQL service sends alerts when the cluster experiences a replication issue or when a node is not allowed to auto-rejoin the cluster.

7. Under **Automated Backups Configuration**, choose one of three options for MySQL backups:
   - Disable automatic backups of MySQL
   - Enable automated backups from MySQL to an S3 bucket or other S3-compatible file store saves your backups to an existing Amazon Web Services (AWS) or Ceph S3-compatible blobstore.
This option requires the following fields:

- For **S3 Bucket Name**, enter the name of your S3 bucket. Do not include an `s3://`, a trailing `/`, or underscores. If the bucket does not already exist, it will be created automatically.
- For **Bucket Path**, specify a folder within the bucket to hold your MySQL backups. Do not include a trailing `/`.
- For **AWS Access Key ID** and **AWS Secret Access Key**, enter your AWS or Ceph credentials.
- For **Cron Schedule**, enter a valid cron expression to schedule your automated backups. Cron uses your computer’s local time zone.
- **Enable automated backups from MySQL to a remote host via SCP** saves your backups to a remote host using secure copy protocol (SCP).
This option requires the following fields:

- For **Hostname**, enter the name of your SCP host.
- For **Port**, enter your SCP port. This should be the TCP port that your SCP host uses for SSH. The default port is **22**.
- For **Username**, enter your SSH username for the SCP host.
- For **Private key**, paste in your SSH private key.
- For **Destination directory**, enter the directory on the SCP host where you want to save backup files.
- For **Cron Schedule**, enter a valid cron expression to schedule your automated backups. Cron uses your computer's local time zone.
- Enable **Backup All Nodes** to make unique backups from each instance of the MySQL server rather than just the first MySQL server instance.

8. If you want to log audit events for internal MySQL, select **Enable server activity logging** under **Server Activity Logging**.

   a. For the **Event types** field, you can enter the events you want the MySQL service to log. By default, this field includes **connect** and **query**, which tracks who connects to the system and what queries are processed. For more information, see the Logging Events section of the MariaDB documentation.
Step 14: Configure File Storage

1. In the Elastic Runtime tile, select File Storage.

2. Select the External S3-Compatible Filestore option and complete the following fields:

   - For URL Endpoint:
     1. In a browser, open the Amazon Simple Storage Service (Amazon S3) table.
     2. Prepend `https://` to the Endpoint for your region and copy it into the Ops Manager URL Endpoint field.
For example, in the us-west-2 region, use [https://s3-us-west-2.amazonaws.com/](https://s3-us-west-2.amazonaws.com/).

- For S3 Signature Version and Region, use the V4 Signature values. AWS recommends using Signature Version 4.
- Select Server-side Encryption (available for AWS S3 only) to encrypt the contents of your S3 filestore. See the AWS S3 documentation for more information.
- Use the values in your AWS Outputs tab to complete the remaining fields as follows:

<table>
<thead>
<tr>
<th>Ops Manager Field</th>
<th>Outputs Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buildpacks Bucket Name</td>
<td>PcfElasticRuntimeS3BuildpacksBucket</td>
</tr>
<tr>
<td>Droplets Bucket Name</td>
<td>PcfElasticRuntimeS3DropletsBucket</td>
</tr>
<tr>
<td>Packages Bucket Name</td>
<td>PcfElasticRuntimeS3PackagesBucket</td>
</tr>
<tr>
<td>Resources Bucket Name</td>
<td>PcfElasticRuntimeS3ResourcesBucket</td>
</tr>
<tr>
<td>Access Key ID</td>
<td>PcfIamUserAccessKey</td>
</tr>
<tr>
<td>AWS Secret Key</td>
<td>PcfIamUserSecretAccessKey</td>
</tr>
</tbody>
</table>

3. Click Save.

### Step 15: (Optional) Configure System Logging

If you are forwarding logging messages to an external Reliable Event Logging Protocol (RELP) server, complete the following steps:

1. Select System Logging.

2. If you want to include security events in your log stream, select the Enable Cloud Controller security event logging checkbox. This logs all API requests, including the endpoint, user, source IP, and request result, in the Common Event Format (CEF).

3. Enter the IP address of your syslog server in **External Syslog Aggregator Hostname** and its port in **External Syslog Aggregator Port**. The default port for a syslog server is 514.

   - **Note**: The host must be reachable from the Elastic Runtime network, accept TCP connections, and use the RELP protocol. Ensure your syslog server listens on external interfaces.

4. Select an **External Syslog Network Protocol** to use when forwarding logs.

5. For the **Syslog Drain Buffer Size**, enter the number of messages the Doppler server can hold from Metron agents before the server starts to drop them. See the Loggregator Guide for Cloud Foundry Operators topic for more details.

6. Click Save.
Step 16: (Optional) Customize Apps Manager

The Custom Branding and Apps Manager sections customize the appearance and functionality of Apps Manager. Refer to the Custom Branding Apps Manager topic for more information.

1. Select Custom Branding.

Use this section to configure the text, colors, and images of the interface that developers see when they log in, create an account, reset their password, or use Apps Manager.

2. Select Apps Manager.
Use this section to control access and role options for Apps Manager, as well as specify the headers and sidebar links that it displays.

- Select **Enable Internal User Store** to use an internal user store in the PCF local UAA server. With the internal store enabled, PCF admins do not need to configure an external user store such as an LDAP / AD server.
- Select **Enable Non Admin Role Management** to allow Org managers and Space managers to assign roles to users in the Orgs and Spaces that they manage.

Both the **Enable Internal User Store** and the **Enable Non Admin Role Management** checkboxes must be selected if you want to invite new members to join an Org in Apps Manager. For more information about inviting users, see Inviting New Users. Ensure that you select the **Push Apps Manager** post-deploy errand on the **Errands** tab so that these edits are applied to your Apps Manager installation. Follow the steps in the “Configure Errands” below.

3. Click **Save** to save your settings for each section.

**Step 17: (Optional) Configure Email Notifications**

Elastic Runtime uses SMTP to send invitations and confirmations to Apps Manager users. You must complete the **Email Notifications** page if you want to enable end-user self-registration.

1. Select Email Notifications.
Configure Simple Mail Transfer Protocol for the Notifications application to send email notifications about your deployment. This application is deployed as an errand in Elastic Runtime. If you don’t need this service, you may leave this section blank and disable the Notifications and Notifications UI errands.

From Email
Address of SMTP Server
Port of SMTP Server
SMTP Server Credentials
Change
SMTP Enable Automatic STARTTLS
SMTP Authentication Mechanism
SMTP CRAMMD5 secret

2. Enter your reply-to and SMTP email information.

3. For SMTP Authentication Mechanism, select None.

4. Click Save.

Note: If you do not configure the SMTP settings using this form, the administrator must create orgs and users using the cf CLI tool. See Creating and Managing Users with the cf CLI for more information.

Step 18: (Optional) Add CCDB Restore Key

Perform this step if all of the following are true:

- You deployed Elastic Runtime previously
- You then stopped Elastic Runtime or it crashed
- You are re-deploying Elastic Runtime with a backup of your Cloud Controller database

1. Click Restore CCDB Encryption Key.

2. Enter your Cloud Controller DB Encryption Key.
Step 19: Configure Smoke Tests

The Smoke Tests errand runs basic functionality tests against your Elastic Runtime deployment after an installation or update. In this section, choose where to run smoke tests. In the **Errands** section, you can choose whether or not to run the Smoke Tests errand.

1. Select **Smoke Tests**.

2. If you have a shared apps domain, select **On-demand org and space**, which creates an ad-hoc org and space for running smoke tests and deletes them afterwards. Otherwise, select **Specified org and space** and complete the fields to specify where you want to run smoke tests.

   - If you eventually decide to delete your Elastic Runtime deployment's shared domain(s), you will need to specify a dedicated Cloud Foundry organization and space where the smoke tests can be run.

   - Choose whether to create a new org and space on demand for smoke tests:
     - **On-demand org and space** (this org and space are deleted after smoke tests finish running)
     - **Specified org and space** (the org and space must have a domain available for routing)

   - Complete the fields for the organization, space, and domain.

3. Click **Save**.

Step 20: (Optional) Enable Experimental Features

Use caution when enabling experimental features if you have other Pivotal Cloud Foundry service tiles installed in your Pivotal Cloud Foundry deployment. Not all of the services are guaranteed to work as expected with these features enabled.
Diego Cell Memory and Disk Overcommit

If your apps do not use the full allocation of disk space and memory set in the Resource Config tab, you might want to use this feature. These fields control the amount to overcommit disk and memory resources to each Diego Cell VM.

For example, you might want to use the overcommit if your apps use a small amount of disk and memory capacity compared with the Resource Config settings for Diego Cell.

Note: Due to the risk of app failure and the deployment-specific nature of disk and memory use, Pivotal has no recommendation about how much, if any, memory or disk space to overcommit.

To enable this feature, follow these steps:

1. Select Experimental Features.
2. Enter the total desired amount of Diego cell memory value in the Cell Memory Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell memory capacity settings that this field overrides.
3. Enter the total desired amount of Diego cell disk capacity value in the Cell Disk Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell disk capacity settings that this field overrides.

Note: Entries made to each of these two fields set the total amount of resources allocated, not the average.

4. Click Save.

CF CLI Connection Timeout

The CF CLI Connection Timeout field allows you to override the default 5 second timeout of the Cloud Foundry Command Line Interface (cf CLI) used within your PCF deployment. This timeout affects the cf command used to push Elastic Runtime errand apps such as Notifications, Autoscaler, Apps Manager and so on.

Set the value of this field to a higher value, in seconds, if you are experiencing domain name resolution timeouts when pushing errands in Elastic Runtime.

To modify your CF CLI connection timeout, perform the following steps:

1. Select Experimental Features.
2. Add a value, in seconds, to the CF CLI Connection Timeout field.
3. Click Save.

Step 21: Configure Errands Page

Errands are scripts that Ops Manager runs to automate tasks. By default, Ops Manager runs the post-install errands listed below when you deploy Elastic Runtime. However, you can prevent a specific post-install errand from running by deselecting its checkbox on the Errands page.

Note: Several errands deploy apps that provide services for your deployment, such as Autoscaling and Notifications. Once one of these apps is running, deselecting the checkbox for the corresponding errand on a subsequent deployment does not stop the app.
Run Smoke Tests verifies that your deployment can do the following:

- Push, scale, and delete apps
- Create and delete orgs and spaces

Push Apps Manager deploys the Apps Manager, a dashboard for managing apps, services, orgs, users, and spaces. Until you deploy Apps Manager, you must perform these functions through the cf CLI. After Apps Manager has been deployed, we recommend deselecting the checkbox for this errand on subsequent Elastic Runtime deployments. For more information about the Apps Manager, see Getting Started with the Apps Manager.

Notifications deploys an API for sending email notifications to your PCF platform users.

Notifications UI deploys a dashboard for users to manage notification subscriptions.

Deploy CF Autoscaling App enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the Bind a Service Instance section of the Managing Service Instances with the CLI topic.

Register Autoscaling Service Broker makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.

Step 22: Configure Router to Elastic Load Balancer

1. If you do not know it, find the name of your ELB by clicking Load Balancers in the AWS EC2 dashboard, for example, `pcf-stack-pcf-ssh-elb` and `pcf-stack-pcf-elb`.

Notifications app requires you configure SMTP with a username and password, even if SMTP Authentication Mechanism is set to none.

Notifications UI deploys a dashboard for users to manage notification subscriptions.

Deploy CF Autoscaling App enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the Bind a Service Instance section of the Managing Service Instances with the CLI topic.

Register Autoscaling Service Broker makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.

Notifications app requires that you configure SMTP with a username and password, even if SMTP Authentication Mechanism is set to none.

Notifications UI deploys a dashboard for users to manage notification subscriptions.

Deploy CF Autoscaling App enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the Bind a Service Instance section of the Managing Service Instances with the CLI topic.

Register Autoscaling Service Broker makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.

Notifications app requires that you configure SMTP with a username and password, even if SMTP Authentication Mechanism is set to none.

Notifications UI deploys a dashboard for users to manage notification subscriptions.

Deploy CF Autoscaling App enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the Bind a Service Instance section of the Managing Service Instances with the CLI topic.

Register Autoscaling Service Broker makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.
2. In the Elastic Runtime tile, click Resource Config.

3. In the ELB Name field of the Router row, enter the name of your load balancer. You may configure multiple load balancers by entering the names separated by commas.

4. In the ELB Name field of the Diego Brain row, enter the name of your SSH load balancer. You may configure multiple load balancers by entering the names separated by commas.

5. Click Save.
Step 23: (Optional) Disable Unused Resources

By default, Elastic Runtime uses an internal filestore and internal databases. If you configure Elastic Runtime to use external resources, you can disable the corresponding system-provided resources in Ops Manager to reduce costs and administrative overhead.
For more information regarding scaling instances, see the Zero Downtime Deployment and Scaling in CF and the Scaling Instances in Elastic Runtime topics.

Complete the following procedures to disable specific VMs in Ops Manager:

1. Click Resource Config.

2. If you configure Elastic Runtime to use an external S3-compatible filestore, edit the following fields:
   - NFS Server: Enter 0 in Instances.

3. If you configure Elastic Runtime to use an external Relational Database Service (RDS), edit the following fields:
   - MySQL Proxy: Enter 0 in Instances.
   - MySQL Server: Enter 0 in Instances.
   - Cloud Controller Database: Enter 0 in Instances.
   - UAA Database: Enter 0 in Instances.
   - Apps Manager Database: Enter 0 in Instances.

4. If you are using an External Load Balancer instead of HAProxy, enter 0 in the Instances field for HAProxy.

5. Click Save.

**Step 24: Download Stemcell**

This step is only required if your Ops Manager does not already have the stemcell version required by Elastic Runtime. There are two ways to determine if you need to download a stemcell: * Refer to the dependency list available on Pivnet for each Ops Manager and Elastic Runtime Tile to determine if you need to download the stemcell. * If the Stemcell section of a tile is colored orange, you will need to download a different stemcell.

1. Select Stemcell.

2. Log into the Pivotal Network and click on Stemcells.

3. Download the appropriate stemcell version targeted for your IaaS.

4. In Ops Manager, import the downloaded stemcell .tgz file.

**Step 25: Complete the Elastic Runtime Installation**

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes. If the following ICMP error message appears, click Ignore errors and start the install.
The install process generally requires a minimum of 90 minutes to complete. The image shows the Changes Applied window that displays when the installation process successfully completes.

Return to Installing Pivotal Cloud Foundry Using AWS CloudFormation.
When you deploy Pivotal Cloud Foundry (PCF) to Amazon Web Services (AWS), you provision a set of resources. This topic describes how to delete the AWS resources associated with a PCF deployment. You can use the AWS console to remove an installation of all components, but retain the objects in your bucket for a future deployment.

1. Log into your AWS Console.

2. Navigate to your EC2 dashboard. Select Instances from the menu on the left side.

3. Terminate all your instances.
4. Select **Load Balancers**. Delete all load balancers.
5. From the AWS Console, select RDS.

6. Select **Instances** from the menu on the left side. Delete the RDS instances.

7. Select **Create final Snapshot** from the drop-down menu. Click **Delete**.
8. From the AWS Console, select VPC.

9. Select Your VPCs from the menu on the left. Delete the VPCs.

10. Check the box to acknowledge that you want to delete your default VPC. Click Yes, Delete.
Configuring Amazon EBS Encryption

Pivotal Cloud Foundry (PCF) supports Amazon Elastic Block Store (EBS) Encryption for PCF deployments on AWS. Amazon EBS Encryption allows operators to use full disk encryption for all persistent disks on BOSH-deployed VMs. You can use this feature to meet data-at-rest encryption requirements or as a security best practice.

There is no performance penalty for using encrypted EBS volumes. Pivotal advises all users of PCF on AWS to check this box.

How to Enable EBS Encryption

1. Click the Ops Manager Director tile.

2. Select AWS Config to open the AWS Management Console Config page.
3. Select **Encrypt EBS Volumes**.

   Note: **Encrypt EBS Volumes** is a global setting. When selected, **Encrypt EBS Volumes** enables encryption on all VMs deployed by BOSH for all product tiles.

4. Click **Save**, and then return to the **Installation Dashboard**.

5. In Ops Manager, click **Apply Changes** and review any reported errors. The following error message lists jobs that cannot be encrypted due to unsupported instance types.
If you find a job that should be encrypted in the error list, modify the instance type for that job in the Resource Config page of the Elastic Runtime. Select an instance type that supports encryption. Pivotal recommends using t2.large.

6. After you make your changes in Elastic Runtime, return to Ops Manager and click Apply Changes.

**WARNING:** After you enable or disable Encrypt EBS Volumes and click Apply Changes, Ops Manager recreates all existing persistent VM disks.

**Limitations**

Using EBS Encryption is subject to the following limitations:

- Ops Manager and Director VMs are not encrypted.
- PCF does not support Amazon EBS Encryption for the following AWS instance types:
  - t2.micro
  - t2.small
  - t2.medium

  **Note:** PCF will remove this limitation in a future release.

- Ephemeral disks are not encrypted. The Encrypt EBS Volumes checkbox applies only to persistent disks.
- Compilation worker VMs are not encrypted because they do not have persistent disks.
Creating a Proxy ELB for Diego SSH without CloudFormation

Page last updated:

If you want to allow SSH connections to application containers, you may want to use an Elastic Load Balancer (ELB) as the SSH proxy.

Users who deploy a Pivotal Cloud Foundry (PCF) 1.6+ installation on Amazon Web Services (AWS) using the CloudFormation template will automatically have this ELB created for them. However, if you are not using the CloudFormation template, or you are upgrading from an earlier version of PCF, perform the following steps to create this ELB in AWS manually:

1. On the EC2 Dashboard, click Load Balancers.

2. Click Create Load Balancer, and configure a load balancer with the following information:

   - Enter a load balancer name.
   - Create LB Inside: Select the pcf-vpc VPC where your PCF installation lives.
   - Ensure that the Create an internal load balancer checkbox is not selected.

3. Under Load Balancer Protocol, ensure that this ELB is listening on TCP port 2222 and forwarding to TCP port 2222.

4. Under Select Subnets, select the public subnet.

5. On the Assign Security Groups page, create a new Security Group. This Security Group should allow inbound traffic on TCP port 2222.
6. The Configure Security Settings page displays a security warning because your load balancer is not using a secure listener. You can ignore this warning.

7. Click Next: Configure Health Check.

8. Select TCP in Ping Protocol on the Configure Health Check page. Ensure that the Ping Port value is 2222 and set the Health Check Interval to 30 seconds.

9. Click Next: Add EC2 Instances.

10. Accept the defaults on the Add EC2 Instances page and click Next: Add Tags.

11. Accept the defaults on the Add Tags page and click Review and Create.

12. Review and confirm the load balancer details, and click Create.

13. With your DNS service (for example, Amazon Route 53), create an ssh.system.YOUR-SYSTEM-DOMAIN DNS record that points to this ELB that you just created.
14. You can now use this ELB to the SSH Proxy of your Elastic Runtime installation.

15. In Elastic Runtime, select Resource Config, and enter the ELB that you just created in the Diego Brain row, under the ELB Names column.
Installing Pivotal Cloud Foundry on OpenStack

Page last updated:

This guide describes how to install Pivotal Cloud Foundry (PCF) on OpenStack Juno and Kilo distributions.

Install PCF on OpenStack

Complete the following procedures to install PCF on OpenStack:

1. Provisioning the OpenStack Infrastructure
2. Configuring Ops Manager Director for OpenStack
3. (Optional) Installing the PCF IPsec Add-On
4. Installing Elastic Runtime after Deploying PCF on OpenStack

Supported Versions

Pivotal’s automated testing environments have been built using OpenStack releases and distributions based on Havana, Icehouse, Juno, Kilo (Keystone v2, and v3), Liberty, and Mitaka from different vendors including Canonical, EMC, Mirantis, Red Hat, and SUSE. The nature of OpenStack as a collection of interoperable components requires OpenStack expertise to troubleshoot issues that may occur when installing Pivotal Cloud Foundry on particular releases and distributions.

Prerequisites

To deploy Pivotal Cloud Foundry on OpenStack, you must have a dedicated OpenStack project (formerly known as an OpenStack tenant) that meets the following requirements.

- You must have keystone credentials for the OpenStack project, including:
  - Auth URL
  - API key
  - Username
  - Project name
  - Region
  - SSL certificate for your wildcard domain (see below).

- Create any necessary OpenStack network objects.

- The following must be enabled for the project:
  - The ability to upload custom images to Glance.
  - The ability to create and modify VM flavors. See the VM flavor configuration table.
  - DHCP
  - The ability to allocate floating IPs.
  - The ability for VMs inside a project to send messages via the floating IP.
  - Permissions for VMs to boot directly from image.
  - One wildcard DNS domain. Pivotal recommends using two wildcard domains if system and apps need to be separated.

Note: For information on how IaaS user roles are configured, refer to the Pivotal Cloud Foundry IaaS User Role Guidelines topic.

Note: It is possible to avoid using wildcard DNS domains by using a service such as xip.io. However, this option requires granting external internet access from inside VMs.

- Your OpenStack project must have the following resources before you install Pivotal Cloud Foundry:
  - 118 GB of RAM
  - 22 available instances
16 small VMs (1 vCPU, 1024 MB of RAM, 10 GB of root disk)
3 large VMs (4 vCPU, 16384 MB of RAM, 10 GB of root disk)
3 extra-large VMs (8 vCPU, 16 GB of RAM, 160 GB of ephemeral disk)
56 vCPUs
1 TB of storage
Neutron networking with floating IP support

Note: If you are using IPSec, your resource usage will increase by approximately 36 bytes. View the Installing IPsec topic for information, including setting correct MTU values.

- Requirements for your Cinder back end:
  - PCF requires RAW root disk images. The Cinder back end for your OpenStack project must support RAW.
  - Pivotal recommends that you use a Cinder back end that supports snapshots. This is required for some BOSH functionalities.
  - Pivotal recommends enabling your Cinder back end to delete block storage asynchronously. If this is not possible, it must be able to delete multiple 200GB volumes within 300 seconds.

- Using an Overlay Network with VXLAN or GRE Protocols:
  - If an overlay network is being used with VXLAN or GRE protocols, the MTU of the created VMs must be adjusted to the best practices recommended by the plugin vendor (if any).
  - DHCP must be enabled in the internal network for the MTU to be assigned to the VMs automatically.
  - Review the Installing Elastic Runtime on OpenStack topic to adjust your MTU values.
  - Failure to configure your overlay network correctly could cause Apps Manager to fail since applications will not be able to connect to the UAA.

- Miscellaneous
  - Pivotal recommends granting complete access to the OpenStack logs to the operator managing the installation process.
  - Your OpenStack environment should be thoroughly tested and considered stable before deploying PCF.

Configure your OpenStack VM flavors as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Memory_MB</th>
<th>Disk</th>
<th>Ephemeral</th>
<th>VCPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>m1.small</td>
<td>2048</td>
<td>20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>m1.medium</td>
<td>4096</td>
<td>40</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>m1.large</td>
<td>8192</td>
<td>80</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>m1.xlarge</td>
<td>16384</td>
<td>160</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

Note: If you are using IPSec, your resource usage will increase by approximately 36 bytes. View the Installing IPsec topic for information, including setting correct MTU values.

Do not change the names of the VM flavors in the table below.
Provisioning the OpenStack Infrastructure

Page last updated:

This guide describes how to provision the OpenStack infrastructure that you need to install Pivotal Cloud Foundry (PCF) OpenStack. This document uses Mirantis OpenStack. Use this topic when Installing Pivotal Cloud Foundry on OpenStack.

After completing this procedure, complete all of the steps in the Configuring Ops Manager Director for OpenStack and Installing Elastic Runtime on OpenStack topics.

Step 1: Log In to the OpenStack Horizon Dashboard

1. Log in to the OpenStack Horizon Dashboard.

   ![Log In](image)

Step 2: Configure Security

This section describes adding key pairs for your PCF deployment.

Note: OpenStack 8 uses the Python Paramiko library, which uses the BER or DER format instead of PEM. If you are using OpenStack 8, import your own key using `ssh-keygen`, and skip to Step 3 below.

1. In the left navigation of your OpenStack dashboard, click Project > Compute > Access & Security.

2. Select the Key Pairs tab on the Access & Security page.

3. Click Create Key Pair.

4. Enter a Key Pair Name and click Create Key Pair.
5. In the left navigation, click **Access & Security** to refresh the page.

6. Select the **Security Groups** tab. Click **Create Security Group** and create a group with the following properties:
   - **Name**: opsmanager
   - **Description**: Ops Manager

7. Select the checkbox for the **opsmanager** Security Group and click **Manage Rules**.

8. Add the access rules for HTTP, HTTPS, and SSH as shown in the table below. The rules with **opsmanager** in the Remote column have restricted access to that particular Security Group.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Ether Type</th>
<th>IP Protocol</th>
<th>Port Range</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>Custom TCP</td>
<td>22 (SSH)</td>
<td>0.0.0.0/0 (CIDR)</td>
</tr>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>Custom TCP</td>
<td>80 (HTTP)</td>
<td>0.0.0.0/0 (CIDR)</td>
</tr>
<tr>
<td>Ingress</td>
<td>IPv4</td>
<td>Custom TCP</td>
<td>443 (HTTPS)</td>
<td>0.0.0.0/0 (CIDR)</td>
</tr>
</tbody>
</table>

   **Note**: Adjust the remote sources as necessary for your own security compliance. Pivotal recommends limiting remote access to Ops Manager to IP ranges within your organization.

   **Note**: If you intend to set up ICMP for your PCF deployment, you should add those rules here.

© Copyright Pivotal Software Inc, 2013-2018
Step 3: Create Ops Manager Image

Use one of the following tools to create the Ops Manager image in OpenStack:

- The OpenStack GUI
- The Glance CLI client

Note: If your Horizon Dashboard does not support file uploads, you must use the Glance client.

(Option) OpenStack GUI

1. Download the Pivotal Cloud Foundry Ops Manager for OpenStack image file from Pivotal Network.

2. In the left navigation of your OpenStack dashboard, click Project > Compute > Images.

3. Click Create Image. Complete the Create An Image page with the following information:
   - Name: Enter 'Ops Manager'.
   - Image Source: Select Image File.
   - Image File: Click Choose File. Browse to and select the image file that you downloaded from Pivotal Network.
   - Format: Select Raw.
   - Minimum Disk (GB): Enter 40.
   - Minimum RAM (MB): Enter 4096.
   - Ensure that the Public checkbox is not selected.
   - Select the Protected checkbox.

4. Click Create Image.
(Option) Glance CLI

1. Install the Glance CLI client.

2. Download the Pivotal Cloud Foundry Ops Manager for OpenStack image file from Pivotal Network.

3. Run `admin-openrc.sh` to download your `openstack.rc` file and target your OpenStack environment.

```
$ ./admin-openrc.sh
Please enter your OpenStack Password:
```

4. Run the following Glance command to upload the image file that you downloaded from Pivotal Network.

```
$ glance image-create --progress --disk-format raw --name "Ops Manager" --container-format bare --file PATH/DOWNLOADED-FILE
```

Step 4: Launch Ops Manager VM

1. In the left navigation of your OpenStack dashboard, click Project > Compute > Images.

2. Click Launch to initiate your project image.

3. Complete the Details, Access & Security, and Networking tabs of the Launch Instance form with the information below.

Details Tab

Select the Details tab and specify the following details:

- **Availability Zone:** Use the drop-down menu to select an availability zone. You use this availability zone when you Complete the Availability Zones Pages when Configuring Ops Manager Director.
- **Instance Name:** Enter `Ops Manager`.
- **Flavor:** Select `m1.large`.
- **Instance Count:** Do not change from the default.
- **InstanceBoot Source:** Select `Boot from image`.
- **Image Name:** Select the `Ops Manager` image.
Access & Security Tab

Select the Access & Security tab and specify the following details:

- **Key Pair**: Select the key pair that you created in Step 2: Configure Security. You need this key pair to log in to the Ops Manager instance from your workstation.

- **Security Groups**: Select the opsmanager checkbox. Deselect all other Security Groups.

Networking Tab

1. Select the Networking tab.
2. Under Available networks, select a private subnet. You add a Floating IP to this network in a later step.
3. Click Launch.
Step 5: Associate Floating IP Address

1. In the left navigation of your OpenStack dashboard, click Project > Compute > Instances.

2. Wait until the Power State of the Ops Manager instances shows as Running.

3. Record the private IP Address of the Ops Manager instance. You use this IP Address when you Complete the Create Networks Pages in Ops Manager.

4. Select the Ops Manager checkbox. Click the Actions drop-down menu and select Associate Floating IP.

5. Under IP Address, click +.

6. Under Pool, select an IP Pool and click Allocate IP.
7. Under **Port to be associated**, select your **Ops Manager** instance. Click **Associate**.

![Manage Floating IP Associations](image)

### Step 6: Add Blob Storage

1. In the left navigation of your OpenStack dashboard, click **Project > Object Store > Containers**.

2. Click **Create Container**. Create a container with the following properties:
   - **Container Name**: Enter `pcf`.
   - **Container Access**: Select `private`.

![Create Container](image)

### Step 7: Create a DNS Entry

**Note**: Ops Manager 1.7 security features require you to create a fully qualified domain name in order to access Ops Manager during the initial configuration.

Create a DNS entry for the IP address that you used for Ops Manager. You must use this fully qualified domain name when you log into Ops Manager in the Configure Ops Manager Director for OpenStack step below.

### Step 8: Configure Ops Manager Director for OpenStack

Now that you have completed this procedure, complete all of the steps in the Configuring Ops Manager Director for OpenStack and Installing Elastic Runtime on OpenStack topics.

Return to [Installing Pivotal Cloud Foundry on OpenStack](#).
Configuring Ops Manager Director for OpenStack

This topic describes how to configure the Ops Manager Director after deploying Pivotal Cloud Foundry (PCF) on OpenStack. Use this topic when Installing Pivotal Cloud Foundry on OpenStack.

Before beginning this procedure, ensure that you have successfully completed all steps in the Provisioning the OpenStack Infrastructure topic. After you complete this procedure, follow the instructions in the Installing Elastic Runtime on OpenStack topic.

Step 1: Access Ops Manager

1. In a web browser, navigate to the fully qualified domain you created in the Create a DNS Entry step of Provisioning the OpenStack Infrastructure.

2. When Ops Manager starts for the first time, you must choose one of the following:
   - Use an Identity Provider: If you use an Identity Provider, an external identity server maintains your user database.
   - Internal Authentication: If you use Internal Authentication, Pivotal Cloud Foundry (PCF) maintains your user database.

   ![Ops Manager Welcome Screen](image)

Use an Identity Provider

1. Log in to your IdP console and download the IdP metadata XML. Optionally, if your IdP supports metadata URL, you can copy the metadata URL instead of the XML.
2. Copy the IdP metadata XML or URL to the Ops Manager Use an Identity Provider log in page.

   Note: The same IdP metadata URL or XML is applied for the BOSH Director. If you are using a separate IdP for BOSH, copy the metadata XML or URL from that IdP and enter it into the BOSH IdP Metadata text box in the Ops Manager log in page.

3. Enter your Decryption passphrase. Read the End User License Agreement, and select the checkbox to accept the terms.

4. Your Ops Manager log in page appears. Enter your username and password. Click Login.

5. Download your SAML Service Provider metadata (SAML Relying Party metadata) by navigating to the following urls:
   - 5a. Ops Manager SAML service provider metadata: https://OPS-MAN-FQDN:443/uaa/saml/metadata
   - 5b. BOSH Director SAML service provider metadata: https://BOSH-IP-ADDRESS:8443/saml/metadata

   Note: To retrieve your BOSH-IP-ADDRESS, navigate to the Ops Manager Director tile > Status tab. Record the Ops Manager Director IP address.

6. Configure your IdP with your SAML Service Provider metadata. Import the Ops Manager SAML provider metadata from Step 5a above to your IdP. If your IdP does not support importing, provide the values below:
   - Single sign on URL: https://OPS-MAN-FQDN:443/uaa/saml/SSO/alias/OPS-MAN-FQDN
   - Name ID is Email Address
   - SAML authentication requests are always signed

7. Import the BOSH Director SAML provider metadata from Step 5b to your IdP. If the IdP does not support an import, provide the values below.
   - Single sign on URL: https://BOSH-IP:8443/saml/SSO/alias/BOSH-IP
   - Audience URI (SP Entity ID): https://BOSH-IP:8443
   - Name ID is Email Address
   - SAML authentication requests are always signed

8. Return to the Ops Manager Director tile, and continue with the configuration steps below.

Use Internal Authentication

1. When redirected to the Internal Authentication page, you must complete the following steps:
   - Enter a Username, Password, and Password confirmation to create an Admin user.
   - Enter a Decryption passphrase and the Decryption passphrase confirmation. This passphrase encrypts the Ops Manager datastore, and is not recoverable.
   - If you are using an Http proxy or Https proxy, follow the these instructions.
   - Read the End User License Agreement, and select the checkbox to accept the terms.
   - Click Setup Authentication.
2. Log in to Ops Manager with the Admin username and password you created in the previous step.

Step 2: Complete the OpenStack Config Page

1. In the left navigation of your OpenStack dashboard, click Project > Compute > Access & Security. Select the API Access tab.

2. Record the Service Endpoint for the Identity service. You use this Service Endpoint as the Authentication URL for Ops Manager in a later step.
3. In the PCF Ops Manager Installation Dashboard, click the **Ops Manager Director** tile.

4. Select **OpenStack Config**.
5. Complete the OpenStack Management Console Configuration page with the following information:

- **Authentication URL**: Enter the Service Endpoint for the Identity service that you recorded in a previous step.
- **Keystone Version**: Choose a Keystone version. If you choose v3, you must enter a **Domain** to authenticate against.
- **Username**: Enter your OpenStack Horizon username.
- **Password**: Enter your OpenStack Horizon password.
- **Tenant**: Enter your OpenStack tenant name.
- **Region**: Enter **RegionOne**, or another region if recommended by your OpenStack administrator.
- **Ignore Server Availability Zone**: Do not select the checkbox.
- **Security Group Name**: Enter `opsmanager`. You created this Security Group when provisioning the OpenStack Infrastructure.
- **Key Pair Name**: Enter the name of the key pair that you created in the Configure Security step of the Provisioning the OpenStack Infrastructure topic.
- **SSH Private Key**: In a text editor, open the key pair file that you downloaded in the Configure Security step of the Provisioning the OpenStack Infrastructure topic. Copy and paste the contents of the key pair file into the field.
- **Disable DHCP**: Do not select the checkbox unless your setup requires it.

(Optional) **API SSL Certificate**: If, in your OpenStack Dashboard, you have configured API SSL termination, enter your **API SSL Certificate**.
6. Click Save.

Step 3: (Optional) Complete the Advanced Config Page

![Advanced Infrastructure Configuration]

1. In Ops Manager, select Advanced Infrastructure Config.

2. If your OpenStack environment requires specific connection options, enter them in the Connection Options field in JSON format. For example:
   
   ```
   'connection_options' => [ 'read_timeout' => 200 ]
   ```

3. Click Save.

Step 4: Complete the Director Config Page

1. In the left navigation of your OpenStack dashboard, click Project > Compute > Access & Security. Select the API Access tab.

2. Click Download EC2 Credentials.

3. Unzip the downloaded credentials. In a text editor, open the `ec2rc.sh` file. Depending on your configuration, you may use the contents of this file to complete the Ops Manager Director Config page.

4. In Ops Manager, select Director Config.
5. Enter one or more NTP servers in the **NTP Servers (comma delimited)** field. For example, `us.pool.ntp.org`.

6. (Optional) Enter your **Metrics IP Address** if you are **Using JMX Bridge**.

7. Select the **Enable VM Resurrection Plugin** checkbox to enable the Ops Manager Resurrection functionality and increase Elastic Runtime availability.

8. Select **Recreate all VMs** to force BOSH to recreate all VMs on the next deploy. This process does not destroy any persistent disk data.

9. Select **Keep Unreachable Director VMs** if you want to preserve Ops Manager Director VMs after a failed deployment for troubleshooting purposes.

10. Select **HM Pager Duty Plugin** to enable Health Monitor integration with PagerDuty.
    - **Service Key**: Enter your API service key from PagerDuty.
    - **HTTP Proxy**: Enter an HTTP proxy for use with PagerDuty.
11. Select **HM Email Plugin** to enable Health Monitor integration with email.

- **Host**: Enter your email hostname.
- **Port**: Enter your email port number.
- **Domain**: Enter your domain.
- **From**: Enter the address for the sender.
- **Recipients**: Enter comma-separated addresses of intended recipients.
- **Username**: Enter the username for your email server.
- **Password**: Enter the password password for your email server.
- **Enable TLS**: Select this checkbox to enable Transport Layer Security.

12. For **Blobstore Location**, select **S3 Compatible Blobstore** and complete the following steps using information from the `ec2rc.sh` file:
13. Select a **Database Location**. By default, PCF deploys and manages a database for you. If you choose to use an **External MySQL Database**, complete the associated fields with information obtained from your external MySQL Database provider.

14. For **Max Threads**, enter the number of operations the Ops Manager Director can perform simultaneously.

15. (Optional) To add a custom URL for your Ops Manager Director, enter a valid hostname in **Director Hostname**. You can also use this field to
configure a load balancer in front of your Ops Manager Director.

16. Click Save.

Note: If you select to use an internal database, back up your data frequently to ensure you have saved the latest copy.

Step 5: Complete the Create Availability Zones Page

1. In Ops Manager, select Create Availability Zones.

2. Enter the name of the availability zone that you selected when Provisioning the OpenStack Infrastructure.

3. Click Save.

Step 6: Complete the Create Networks Page

1. In the left navigation of your OpenStack dashboard, click Project > Network > Networks.

2. Click the name of the network that contains the private subnet where you deployed the Ops Manager VM. The OpenStack Network Detail page displays your network settings.
3. In Ops Manager, select Create Networks.
4. Select **Enable ICMP checks** to enable ICMP on your networks. Ops Manager uses ICMP checks to confirm that components within your network are reachable. Review the **Configure Security** topic to ensure you have setup ICMP in your Security Group.

5. Use the following steps to create one or more Ops Manager networks using information from your OpenStack network:
   - Click **Add Network**.
   - Enter a unique **Name** for the network.
   - Click **Add Subnet** to create one or more subnets for the network.
   - For **Network ID**, use the ID of the network to which the subnet belongs.
   - For **CIDR**, use the **Network Address** from the OpenStack page.
   - For **Reserved IP Ranges**, use the first 10 IP addresses of the **Network Address** range, and the private IP address of the Ops Manager instance that you recorded in the **Associate Floating IP Address** step of the **Provisioning the OpenStack Infrastructure** topic.
   - For **DNS**, enter one or more Domain Name Servers.
   - For **Gateway**, use the **Gateway IP** from the OpenStack page.
   - For **Availability Zones**, select which Availability Zones to use with the network.
6. Click **Save**.

**Step 7: Complete the Assign AZs and Networks Page**

1. Select **Assign Availability Zones**.

2. From the **Singleton Availability Zone** drop-down menu, select the availability zone that you created in a previous step. The Ops Manager Director installs in this Availability Zone.

3. Use the drop-down menu to select the **Network** that you created in a previous step. Ops Manager Director installs in this network.

4. Click **Save**.

**Step 8: Complete the Security Page**

1. Select **Security**.
In **Trusted Certificates**, enter a custom certificate authority (CA) certificate to insert into your organization’s certificate trust chain. This feature enables all BOSH-deployed components in your deployment to trust a custom root certificate. If you want to use Docker Registries for running app instances in Docker containers, use this field to enter your certificate for your private Docker Registry. See the [Using Docker Registries](#) topic for more information.

3. Choose **Generate passwords** or **Use default BOSH password**. Pivotal recommends that you use the **Generate passwords** option for greater security.

4. Click **Save**. To view your saved Director password, click the **Credentials** tab.

**Step 9: Complete the Resource Config Page**

1. Select **Resource Config**.

2. Adjust any values as necessary for your deployment, such as increasing the persistent disk size. Select **Automatic** from the drop-down menu to
provision the amount of persistent disk predefined by the job. If the persistent disk field reads None, the job does not require persistent disk space.

Note: If you set a field to Automatic and the recommended resource allocation changes in a future version, Ops Manager automatically uses the updated recommended allocation.

3. Click Save.

Step 10: Complete Ops Manager Director Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes. If the following ICMP error message appears, click Ignore errors and start the install.

3. Ops Manager Director installs. The image shows the Changes Applied message that Ops Manager displays when the installation process successfully completes.

4. After you complete this procedure, follow the instructions in the Installing Elastic Runtime on OpenStack topic.

Return to Installing Pivotal Cloud Foundry on OpenStack.
Installing Elastic Runtime on OpenStack

This topic describes how to install and configure Elastic Runtime after deploying Pivotal Cloud Foundry (PCF) on OpenStack.

Use this topic when Installing Pivotal Cloud Foundry on OpenStack.

Before beginning this procedure, ensure that you have successfully completed all steps in the Provisioning the OpenStack Infrastructure topic and the Configuring Ops Manager Director for OpenStack topics.

Note: If you are performing an upgrade to PCF 1.7, please review Upgrading Pivotal Cloud Foundry for critical upgrade information.

Step 1: Add Elastic Runtime to Ops Manager

1. Navigate to the Pivotal Cloud Foundry Operations Manager Installation Dashboard.

2. Click the Pivotal Network link on the left to add Elastic Runtime to Ops Manager. For more information, refer to the Adding and Deleting Products topic.

Step 2: Assign Availability Zones and Networks

Note: Pivotal recommends at least three Availability Zones for a highly available installation of Elastic Runtime.

1. Select Assign AZ and Networks. These are the Availability Zones that you create when configuring Ops Manager Director.

2. Select an Availability Zone under Place singleton jobs. Ops Manager runs any job with a single instance in this Availability Zone.

3. Select one or more Availability Zones under Balance other jobs. Ops Manager balances instances of jobs with more than one instance across the Availability Zones that you specify.

4. From the Network drop-down box, choose the network on which you want to run Elastic Runtime.
5. Click Save.

**Note:** When you save this form, a verification error displays because the PCF security group blocks ICMP. You can ignore this error.

---

**Step 3: Configure Domains**
1. Select Domains.

Elastic Runtime gives each deployed application its own hostname in the app domain, and each system component its own hostname on the system domain. Ensure that you have a wildcard DNS record configured for both domains.

**System Domain**

```
pivot.cf-app.com
```

**Apps Domain**

```
pivot.cf-app.com
```

This is the default apps domain that pushed apps will use for their hostnames. Use the Cloud Foundry command line interface to add or delete domains. This also requires a wildcard DNS record.

2. Enter the system and application domains.

- The **System Domain** defines your target when you push apps to Elastic Runtime.
- The **Apps Domain** defines where Elastic Runtime should serve your apps.

**Note:** Pivotal recommends that you use the same domain name but different subdomain names for your system and app domains. Doing so allows you to use a single wildcard certificate for the domain while preventing apps from creating routes that overlap with system routes. For example, name your system domain `system.EXAMPLE.com` and your apps domain `apps.EXAMPLE.com`.

**Note:** You configured wildcard DNS records for these domains in an earlier step.

3. Click **Save**.

**Step 4: Configure Networking**

1. Select **Networking**.

2. *(Optional)* The values you enter in the **Router IPs** and **HAPRoxy IPs** fields depends on whether you are using your own load balancer or the HAPRoxy load balancer. Find your load balancer type in the table below to determine how to complete these fields.

   **Note:** If you choose to assign specific IP addresses in either the **Router IPs** or **HAPRoxy IPs** field, ensure that these IPs are in your subnet.

<table>
<thead>
<tr>
<th>LOAD BALANCER</th>
<th>ROUTER IP FIELD VALUE</th>
<th>HAPROXY IP FIELD VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your own load balancer</td>
<td>Enter the IP address or addresses for PCF that you registered with your load balancer. Refer to the Using Your Own Load Balancer topic for help using your own load balancer with PCF.</td>
<td>Leave this field blank.</td>
</tr>
<tr>
<td>HAPRoxy load balancer</td>
<td>Leave this field blank.</td>
<td>Enter at least one HAPRoxy IP address. Point your DNS to this address.</td>
</tr>
</tbody>
</table>

For help understanding the Elastic Runtime architecture, refer to the Architecture topic.

3. For **Router IPs**, enter one or more static IP addresses for your routers. These must be in the subnet that you configured in the Ops Manager **Create Networks** section. If you are using your own load balancer, configure it to point to these IPs. If you are using the Elastic Load Balancer (ELB), add the name of your ELB in the router column of the **Resource Config** section.
4. For HAProxy IPs, enter one or more IP addresses for HAProxy. You must point your DNS to this IP unless you are using your own load balancer, and HAProxy’s IP must be in your subnet.

5. Under Configure the point-of-entry to this environment, choose one of the following:
   - **External Load Balancer with Encryption**: Select this option if your deployment uses an external load balancer that can forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing. Complete the fields for the Router SSL Termination Certificate and Private Key and Router SSL Ciphers.
   - **External Load Balancer without Encryption**: Select this option if your deployment uses an external load balancer that cannot forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing.
   - **HAProxy**: Select this option to use HAProxy as your first point of entry. Complete the fields for SSL Certificate and Private Key, and HAProxy SSL Ciphers. Select Disable HTTP traffic to HAProxy if you want the HAProxy to only allow HTTPS traffic.

   ![Configure security and routing services for your platform. It is usually preferable to use your own load balancer instead of an HAProxy instance as your point-of-entry to the platform.](image)

   **Note**: For details about different SSL termination point options, which correspond to different points-of-entry for Elastic Runtime, see the [Providing a Certificate for your SSL Termination Point](#) topic.

6. If you are not using SSL encryption or if you are using self-signed certificates, select Disable SSL certificate verification for this environment. Selecting this checkbox also disables SSL verification for route services.

7. Select the **Disable insecure cookies on the Router** checkbox to turn on the secure flag for cookies generated by the router.
8. In the Choose whether or not to enable route services section, choose either Enable route services or Disable route services. Route services are a class of marketplace services that perform filtering or content transformation on application requests and responses. See the Route Services topic for details.

9. (Optional) Use the Applications Subnet field if you need to avoid address collision with a third-party service on the same subnet as your apps. Enter a CIDR subnet mask specifying the range of available IP addresses assigned to your app containers. The IP range must be different from the network used by the system VMs.

10. (Optional) The Loggregator Port defaults to 443 if left blank. Enter a new value to override the default.

11. (Optional) You can change the value in the Applications Network Maximum Transmission Unit (MTU) field. Pivotal recommends setting the MTU value for your application network to 1454. Some configurations, such as networks that use GRE tunnels, may require a smaller MTU value.

12. (Optional) To accommodate larger uploads over connections with high latency, increase the timeout value in Router Timeout to Backends. This value is specified in seconds.

13. (Optional) Increase the value of Load Balancer Unhealthy Threshold to specify the amount of time, in seconds, that the router continues to accept connections before shutting down. During this period, healthchecks may report the router as unhealthy, which causes load balancers to failover to other routers. Set this value to an amount greater than or equal to the maximum time it takes your load balancer to consider a router instance unhealthy, given contiguous failed healthchecks.

14. Click Save.

Step 5: Configure Application Containers

1. Select Application Containers.
2. The **Enable Custom Buildpacks** checkbox governs the ability to pass a custom buildpack URL to the `-b` option of the `cf push` command. By default, this ability is enabled, letting developers use custom buildpacks when deploying apps. Disable this option by disabling the checkbox. For more information about custom buildpacks, refer to the buildpacks section of the PCF documentation.

3. The **Allow SSH access to app containers** checkbox controls SSH access to application instances. Enable the checkbox to permit SSH access across your deployment, and disable it to prevent all SSH Access. See Application SSH Overview for information on SSH access permissions at the space and app scope.

4. You can configure Elastic Runtime to run app instances in Docker containers by supplying their IP address range(s) in the **Private Docker Insecure Registry Whitelist** textbox. See the Using Docker Registries topic for more information.

5. Select your preference for **Docker Images Disk-Cleanup on Cell VMs**. If you choose **Clean up disk-space once threshold is reached**, enter a **Threshold of Disk-Used** in megabytes.

6. Click **Save**.

---

**Step 6: Configure Application Developer Controls**

1. Select **Application Developer Controls**.
Configure restrictions and default settings for applications pushed to Cloud Foundry.

Maximum File Upload Size (MB) \( (\text{min: } 1024, \text{max: } 2048) \)

- 1024

Default App Memory (MB) \( (\text{min: } 64, \text{max: } 2048) \)

- 1024

Default App Memory Quota per Org (only applies the first time you install Elastic Runtime for this environment) \( (\text{min: } 10240, \text{max: } 102400) \)

- 10240

Maximum Disk Quota per App (MB) \( (\text{min: } 512, \text{max: } 10240) \)

- 2048

Default Disk Quota per App (MB) \( (\text{min: } 512, \text{max: } 10240) \)

- 1024

Default Service Instances Quota per Org (only applies the first time you install Elastic Runtime for this environment) \( (\text{min: } 0, \text{max: } 1000) \)

- 100

Save

2. Enter your intended maximum file upload size.

3. Enter your default RAM memory allocation per app.

4. Enter your default total RAM memory (RAM) quota per Org. You can change this in the CLI.

5. Enter your maximum and default disk quotas per app.

6. Enter your default service instances quota per Org. You can change this in the CLI.

7. Click Save.

---

Step 7: Review Application Security Groups

Setting appropriate Application Security Groups is critical for a secure deployment. Type `X` in the box to acknowledge that once the Elastic Runtime deployment completes, you will review and set the appropriate application security groups. See Restricting App Access to Internal PCF Components for instructions.
Step 8: Configure Authentication and Enterprise SSO

1. Select Authentication and Enterprise SSO.

Setting appropriate Application Security Groups that control application network policy is the responsibility of the Elastic Runtime administration team. Please refer to the Application Security Groups topic in the Pivotal Cloud Foundry documentation for more detail on completing this activity after the Elastic Runtime deployment completes.

Type X to acknowledge that you understand this message *

Save
2. To authenticate user sign-ons, your deployment can use one of three types of user database: the UAA server’s internal user store, an external SAML identity provider, and an external LDAP server.
   a. To use the internal UAA, select the Internal option and follow the instructions in Configuring UAA Password Policy to configure your password policy.
   b. To connect to an external identity provider via SAML, scroll down to select the SAML Identity Provider option and follow the instructions in Configuring PCF for SAML.
   c. To connect to an external LDAP server, scroll down to select the LDAP Server option and follow the instructions in Configuring LDAP.
3. (Optional) In the Apps Manager Access Token Lifetime, Apps Manager Refresh Token Lifetime, Cloud Foundry CLI Access Token Lifetime, Cloud Foundry CLI Refresh Token Lifetime fields, you can change the lifetimes of tokens granted for Apps Manager and cf CLI login access and refresh. Most deployments use the defaults.
4. (Optional) The Proxy IPs Regular Expression field contains a pipe delimited set of regular expressions that UAA considers to be reverse proxy IP addresses. UAA respects the x-forwarded-for and x-forwarded-host headers coming from IP addresses that match to these regular expressions. To configure UAA to respond properly to Router or HAProxy requests coming from public IP address(es), append a regular expression or regular expressions to match the public IP address(es).
Step 9: Configure System Databases

Note: If you are performing an upgrade, do not modify your existing internal database configuration or you may lose data. You must migrate your existing data first before changing the configuration. See Upgrading Pivotal Cloud Foundry for additional upgrade information.

1. Select Databases.

2. If you want to use internal databases for your deployment, select Internal Databases - MySQL and Postgres or Internal Databases - MySQL. If you want to use external databases such as Amazon Web Services (AWS) RDS, select External Databases and complete the following steps:
   - For Hostname, enter the hostname of your database.
   - For TCP Port, enter the port of your database.
   - For each database username and database password field, enter a unique username and password.

   ![Database Selection](image)

   Note: Pivotal recommends that you use internal databases unless you require the functionality of AWS RDS.

3. Click Save.

Step 10 (Optional) Configure Internal MySQL
1. Select **Internal MySQL**.

2. In the **MySQL Proxy IPs** field, enter one or more comma-delimited IP addresses that are not in the reserved CIDR range of your network. If a MySQL node fails, these proxies re-route connections to a healthy node. See the [MySQL Proxy](#) topic for more information.

3. For **MySQL Service Hostname**, enter an IP address or hostname for your load balancer. If a MySQL proxy fails, the load balancer re-routes connections to a healthy proxy. If you leave this field blank, components are configured with the IP address of the first proxy instance entered above.

   **Warning:** You must configure a load balancer to achieve complete high-availability.

4. In the **Replication canary time period** field, leave the default of 30 seconds or modify the value based on the needs of your deployment. Lower numbers cause the canary to run more frequently, which means that the canary reacts more quickly to replication failure but adds load to the database.

5. In the **Replication canary read delay** field, leave the default of 20 seconds or modify the value based on the needs of your deployment. This field configures how long the canary waits, in seconds, before verifying that data is replicating across each MySQL node. Clusters under heavy load can experience a small replication lag as write-sets are committed across the nodes.

6. [Required]: In the **E-mail address** field, enter the email address where the MySQL service sends alerts when the cluster experiences a replication issue or when a node is not allowed to auto-rejoin the cluster.

7. Under **Automated Backups Configuration**, choose one of three options for MySQL backups:
   - Disable automatic backups of MySQL
   - Enable automated backups from MySQL to an S3 bucket or other S3-compatible file store saves your backups to an existing Amazon Web Services (AWS) or Ceph [S3-compatible blobstore](#).
This option requires the following fields:

- **For S3 Bucket Name**, enter the name of your S3 bucket. Do not include an `s3://`, a trailing `/`, or underscores. If the bucket does not already exist, it will be created automatically.
- **For Bucket Path**, specify a folder within the bucket to hold your MySQL backups. Do not include a trailing `/`.
- **For AWS Access Key ID and AWS Secret Access Key**, enter your AWS or Ceph credentials.
- **For Cron Schedule**, enter a valid `cron` expression to schedule your automated backups. Cron uses your computer's local time zone.
- **Enable automated backups from MySQL to a remote host via SCP** saves your backups to a remote host using secure copy protocol (SCP).
This option requires the following fields:

- For **Hostname**, enter the name of your SCP host.
- For **Port**, enter your SCP port. This should be the TCP port that your SCP host uses for SSH. The default port is **22**.
- For **Username**, enter your SSH username for the SCP host.
- For **Private key**, paste in your SSH private key.
- For **Destination directory**, enter the directory on the SCP host where you want to save backup files.
- For **Cron Schedule**, enter a valid cron expression to schedule your automated backups. Cron uses your computer's local time zone.
- Enable **Backup All Nodes** to make unique backups from each instance of the MySQL server rather than just the first MySQL server instance.

**Note:** If you choose to enable automated MySQL backups, set the number of instances for the **Backup Prepare Node** under the **Resource Config** section of the Elastic Runtime tile to **1**.

8. If you want to log audit events for internal MySQL, select **Enable server activity logging** under **Server Activity Logging**.
   
a. For the **Event types** field, you can enter the events you want the MySQL service to log. By default, this field includes `connect` and `query`, which tracks who connects to the system and what queries are processed. For more information, see the [Logging Events](#) section of the MariaDB documentation.
9. Click Save.

Step 11: Configure File Storage

1. Select File Storage.

2. To use the PCF internal filestore, select the Internal option and click Save.

3. To use an external S3-compatible filestore for your Elastic Runtime file storage, select the External S3-Compatible Filestore option and complete the following procedure:
a. Enter the URL Endpoint for your filestore.
b. Enter your Access Key and Secret Key.
c. For S3 Signature Version and Region, use V4 Signature values.
d. Enter a Buildpacks Bucket Name.
e. Enter a Droplets Bucket Name.
f. Enter a Packages Bucket Name.
g. Enter a Resources Bucket Name.
h. Click Save.

Note: For more information about AWS S3 Signatures, see the Authenticating Requests documentation.

Step 12: (Optional) Configure System Logging

If you are forwarding logging messages to an external Reliable Event Logging Protocol (RELP) server, complete the following steps:

1. Select System Logging.

2. If you want to include security events in your log stream, select the Enable Cloud Controller security event logging checkbox. This logs all API requests, including the endpoint, user, source IP, and request result, in the Common Event Format (CEF).

3. Enter the IP address of your syslog server in External Syslog Aggregator Hostname and its port in External Syslog Aggregator Port. The default port for a syslog server is 514.

Note: The host must be reachable from the Elastic Runtime network, accept TCP connections, and use the RELP protocol. Ensure your syslog server listens on external interfaces.


5. For the Syslog Drain Buffer Size, enter the number of messages the Doppler server can hold from Metron agents before the server starts to drop them. See the Loggregator Guide for Cloud Foundry Operators topic for more details.

6. Click Save.

Step 13: (Optional) Customize Apps Manager

The Custom Branding and Apps Manager sections customize the appearance and functionality of Apps Manager. Refer to the Custom Branding Apps Manager topic for more information.
1. Select Custom Branding.

Customize colors, images, and text for Apps Manager and the Cloud Foundry login portal.

- Company Name
- Accent Color
- Main Logo (PNGs only)
- Square Logo/Favicon (PNGs only)
- Footer Text:
- Footer Links
  You may configure up to 3 links in the Apps Manager footer

Save

Use this section to configure the text, colors, and images of the interface that developers see when they log in, create an account, reset their password, or use Apps Manager.

2. Select Apps Manager.
Use this section to control access and role options for Apps Manager, as well as specify the headers and sidebar links that it displays.

- Select **Enable Internal User Store** to use an internal user store in the PCF local UAA server. With the internal store enabled, PCF admins do not need to configure an external user store such as an LDAP / AD server.
- Select **Enable Non Admin Role Management** to allow Org managers and Space managers to assign roles to users in the Orgs and Spaces that they manage.

Both the **Enable Internal User Store** and the **Enable Non Admin Role Management** checkboxes must be selected if you want to invite new members to join an Org in Apps Manager. For more information about inviting users, see [Inviting New Users](#). Ensure that you select the **Push Apps Manager** post-deploy errand on the **Errands** tab so that these edits are applied to your Apps Manager installation. Follow the steps in the "Configure Errands" below.

3. Click **Save** to save your settings for each section.

### Step 14: (Optional) Configure Email Notifications

Elastic Runtime uses SMTP to send invitations and confirmations to Apps Manager users. You must complete the **Email Notifications** page if you want to enable end-user self-registration.

1. Select **Email Notifications**.
Configure Simple Mail Transfer Protocol for the Notifications application to send email notifications about your deployment. This application is deployed as an errand in Elastic Runtime. If you don’t need this service, you may leave this section blank and disable the Notifications and Notifications UI errands.

2. Enter your reply-to and SMTP email information.

3. For SMTP Authentication Mechanism, select *none*.

4. Click Save.

Note: If you do not configure the SMTP settings using this form, the administrator must create orgs and users using the cf CLI tool. See Creating and Managing Users with the cf CLI for more information.

Step 15: (Optional) Add CCDB Restore Key

Perform this step if all of the following are true:

- You deployed Elastic Runtime previously
- You then stopped Elastic Runtime or it crashed
- You are re-deploying Elastic Runtime with a backup of your Cloud Controller database

1. Click Restore CCDB Encryption Key.

2. Enter your Cloud Controller DB Encryption Key.
Step 16: Configure Smoke Tests

The Smoke Tests errand runs basic functionality tests against your Elastic Runtime deployment after an installation or update. In this section, choose where to run smoke tests. In the Errands section, you can choose whether or not to run the Smoke Tests errand.

1. Select Smoke Tests.

2. If you have a shared apps domain, select On-demand org and space, which creates an ad-hoc org and space for running smoke tests and deletes them afterwards. Otherwise, select Specified org and space and complete the fields to specify where you want to run smoke tests.

3. Click Save.

Step 17: (Optional) Enable Experimental Features

Use caution when enabling experimental features if you have other Pivotal Cloud Foundry service tiles installed in your Pivotal Cloud Foundry deployment. Not all of the services are guaranteed to work as expected with these features enabled.
Diego Cell Memory and Disk Overcommit

If your apps do not use the full allocation of disk space and memory set in the Resource Config tab, you might want to use this feature. These fields control the amount to overcommit disk and memory resources to each Diego Cell VM.

For example, you might want to use the overcommit if your apps use a small amount of disk and memory capacity compared with the Resource Config settings for Diego Cell.

**Note:** Due to the risk of app failure and the deployment-specific nature of disk and memory use, Pivotal has no recommendation about how much, if any, memory or disk space to overcommit.

To enable this feature, follow these steps:

1. Select Experimental Features.
2. Enter the total desired amount of Diego cell memory value in the Cell Memory Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell memory capacity settings that this field overrides.
3. Enter the total desired amount of Diego cell disk capacity value in the Cell Disk Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell disk capacity settings that this field overrides.

**Note:** Entries made to each of these two fields set the total amount of resources allocated, not the average.

4. Click Save.

CF CLI Connection Timeout

The CF CLI Connection Timeout field allows you to override the default 5 second timeout of the Cloud Foundry Command Line Interface (cf CLI) used within your PCF deployment. This timeout affects the :cf: command used to push Elastic Runtime errand apps such as Notifications, Autoscaler, Apps Manager and so on.

Set the value of this field to a higher value, in seconds, if you are experiencing domain name resolution timeouts when pushing errands in Elastic Runtime.

To modify your CF CLI connection timeout, perform the following steps:

1. Select Experimental Features.
2. Add a value, in seconds, to the CF CLI Connection Timeout field.
3. Click Save.

Step 18: Configure Errands Page

Errands are scripts that Ops Manager runs to automate tasks. By default, Ops Manager runs the post-install errands listed below when you deploy Elastic Runtime. However, you can prevent a specific post-install errand from running by deselecting its checkbox on the Errands page.

**Note:** Several errands deploy apps that provide services for your deployment, such as Autoscaling and Notifications. Once one of these apps is running, deselecting the checkbox for the corresponding errand on a subsequent deployment does not stop the app.
### Errands

Errands are scripts that run at designated points during an installation.

#### Post-Deploy Errands

- **Run Smoke Tests** verifies that your deployment can do the following:
  - Push, scale, and delete apps
  - Create and delete orgs and spaces

- **Push Apps Manager** deploys the Apps Manager, a dashboard for managing apps, services, orgs, users, and spaces. Until you deploy Apps Manager, you must perform these functions through the `cf` CLI. After Apps Manager has been deployed, we recommend deselecting the checkbox for this errand on subsequent Elastic Runtime deployments. For more information about the Apps Manager, see [Getting Started with the Apps Manager](#).

- **Notifications** deploys an API for sending email notifications to your PCF platform users.

  Note: The Notifications app requires that you configure SMTP with a username and password, even if SMTP Authentication Mechanism is set to none.

- **Notifications-UI** deploys a dashboard for users to manage notification subscriptions.

- **Deploy CF Autoscaling App** enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the [Bind a Service Instance](#) section of the Managing Service Instances with the CLI topic.

  Note: The Autoscaling app requires the Notifications app to send scaling action alerts by email.

- **Register Autoscaling Service Broker** makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.

### Step 19: Enable Traffic to Private Subnet

Unless you are using your own load balancer, you must enable traffic flow to the OpenStack private subnet as follows. Give each HAProxy a way of routing traffic into the private subnet by providing public IPs as floating IPs.

1. Click Resource Config.
2. Enter one or more IP addresses in **Floating IPs** for each HAProxy.

3. Click **Save**.

### Step 20: (Optional) Disable Unused Resources

By default, Elastic Runtime uses an internal filestore and internal databases. If you configure Elastic Runtime to use external resources, you can disable the corresponding system-provided resources in Ops Manager to reduce costs and administrative overhead.

For more information regarding scaling instances, see the [Zero Downtime Deployment and Scaling in CF](#) and the [Scaling Instances in Elastic Runtime](#) topics.

Complete the following procedures to disable specific VMs in Ops Manager:

1. Click **Resource Config**.

2. If you configure Elastic Runtime to use an external S3-compatible filestore, edit the following fields:
   - **NFS Server**: Enter 0 in **Instances**.

3. If you configure Elastic Runtime to use an external Relational Database Service (RDS), edit the following fields:
   - **MySQL Proxy**: Enter 0 in **Instances**.
   - **MySQL Server**: Enter 0 in **Instances**.
Cloud Controller Database: Enter 0 in Instances.
UAA Database: Enter 0 in Instances.
Apps Manager Database: Enter 0 in Instances.

4. If you are using an External Load Balancer instead of HAProxy, enter 0 in the Instances field for HAProxy.

5. Click Save.

Step 21: Complete Elastic Runtime Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes. If the following ICMP error message appears, click Ignore errors and start the install.

3. Elastic Runtime installs. The image shows the Changes Applied message that Ops Manager displays when the installation process successfully completes.

Return to Installing Pivotal Cloud Foundry on OpenStack
Installing Pivotal Cloud Foundry on vSphere, vCloud, and vCloud Air

Page last updated:

Note: Pivotal Cloud Foundry (PCF) for vCloud Air and vCloud Director is deprecated and availability is restricted to existing customers. Contact Support for more information.

This guide describes how to install Pivotal Cloud Foundry (PCF) on vSphere, vCloud, and vCloud Air. If you experience a problem while following the steps below, refer to the Known Issues topics or to the Pivotal Cloud Foundry Troubleshooting Guide.

Note: If you are performing an upgrade to PCF 1.7, see Upgrading Pivotal Cloud Foundry for critical upgrade information.

Prerequisites

To install PCF on vSphere, vCloud, and vCloud Air, your IaaS configuration must have the minimum resource requirements and user privileges for a PCF installation. For a full list of requirements, see the Prerequisites to Deploying Operations Manager and Elastic Runtime and Pivotal Cloud Foundry IaaS User Role Guidelines topics.

If you are deploying PCF behind a firewall, see the Preparing Your Firewall for Deploying Pivotal Cloud Foundry topic.

Step 1: Install Ops Manager

Complete the following procedures to install Ops Manager on vSphere:

1. Deploying Operations Manager to vSphere
2. Configuring Ops Manager Director for VMware vSphere

Complete the following procedures to install Ops Manager on vCloud and vCloud Air:

1. Deploying Operations Manager to vCloud Air and vCloud
2. Configuring Ops Manager Director for vCloud Air and vCloud

Step 2: Install Elastic Runtime

To install Elastic Runtime on vSphere, vCloud, and vCloud Air, perform the procedures in the Configuring Elastic Runtime for vSphere, vCloud, and vCloud Air.

(Optional) Step 3: Install the IPsec Add-on

The PCF IPsec add-on secures network traffic within a PCF deployment and provides internal system protection if a malicious actor breaches your firewall. See the Securing Data in Transit with the IPsec Add-on topic for installation instructions.

Note: You must install the PCF IPsec add-on before installing any other tiles to enable the IPsec functionality. Pivotal recommends installing IPsec immediately after Ops Manager, and before installing the Elastic Runtime tile.

Step 4: Create New User Accounts

Once you have successfully deployed PCF, add users to your account. Refer to the Creating New Elastic Runtime User Accounts topic for more information.

© Copyright Pivotal Software Inc, 2013-2018
Step 5: Target Elastic Runtime

The next step is to use the cf CLI tool to target your Elastic Runtime installation. Make sure that you have installed the cf CLI tool. Refer to the PCF documentation for more information about using the cf command line tool.

Note: In Ops Manager, refer to Elastic Runtime > Credentials for the UAA admin name and password. You can also use the user that you created in Apps Manager, or create another user with the create-user command.

Additional Configuration

See the following topics for additional vSphere configuration information:

- Provisioning a Virtual Disk in vSphere
- Using the Cisco Nexus 1000v Switch with Ops Manager
- Using Ops Manager Resurrector on VMware vSphere
- Configuring SSL Termination for vSphere Deployments

For more information about using availability zones in vSphere, vCloud, or vCloud Air, see the Understanding Availability Zones in VMware Installations topic.
Deploying Operations Manager to vSphere

Page last updated:

Refer to this topic for help deploying Ops Manager to VMware vSphere. For help deploying Ops Manager to vCloud Air or vCloud, see the Deploying Operations Manager to vCloud Air and vCloud topic.

1. Refer to the Known Issues topic before getting started.

2. Download the Pivotal Cloud Foundry (PCF) Ops Manager .ova file at Pivotal Network. Click the Pivotal Cloud Foundry region to access the PCF product page. Use the dropdown menu to select an Ops Manager release.

3. Log into vCenter.

4. Select the VM and Templates view.

5. Right click on your datacenter and select New Folder.

6. Name the folder pivotal_cf and select it.

7. Select File > Deploy OVF Template.

8. Select the .ova file and click Next.
9. Review the product details and click Next.

10. Accept the license agreement and click Next.

11. Name the virtual machine and click Next.

12. Select a vSphere cluster and click Next.

13. If prompted, select a resource pool and click Next.

14. If prompted, select a host and click Next.

15. Select a storage destination and click Next.

16. Select a disk format and click Next. For information about disk formats, see Provisioning a Virtual Disk.

17. Select a network from the drop down list and click Next.
18. Enter network information and passwords for the Ops Manager VM admin user and click **Next**.

   **Note:** You must enter a default admin password, or else your Ops Manager VM will not boot up.

Keep this network information. The IP Address will be the location of the Ops Manager interface.

19. Check the **Power on after deployment** checkbox and click **Finish**. Once the VM boots, the interface is available at the IP address you specified.

   **Note:** It is normal to experience a brief delay before the interface is accessible while the web server and VM start up.

20. Create a DNS entry for the IP address that you used for Ops Manager. You must use this fully qualified domain name when you log into Ops Manager in the Configuring Ops Manager Director for VMware vSphere topic.

   **Note:** Ops Manager 1.7 security features require you to create a fully qualified domain name in order to access Ops Manager during the initial configuration.

Return to the Installing Pivotal Cloud Foundry Guide ➤
Configuring Ops Manager Director for VMware vSphere

This topic describes how to configure the Ops Manager Director for VMware vSphere.

Before you begin this procedure, ensure that you have successfully completed all steps in the Deploying Operations Manager to vSphere topic. After you complete this procedure, follow the instructions in the Configuring Elastic Runtime for vSphere, vCloud, and vCloud Air topic.

Step 1: Set Up Ops Manager

1. Navigate to the fully qualified domain of your Ops Manager in a web browser.

2. The first time you start Ops Manager, you must choose one of the following:
   - **Use an Identity Provider**: If you use an Identity Provider, an external identity server maintains your user database.
   - **Internal Authentication**: If you use Internal Authentication, PCF maintains your user database.

### Use an Identity Provider

1. Log in to your IdP console and download the IdP metadata XML. Optionally, if your IdP supports metadata URL, you can copy the metadata URL instead of the XML.

2. Copy the IdP metadata XML or URL to the Ops Manager Use an Identity Provider log in page.

   **Note**: The same IdP metadata URL or XML is applied for the BOSH Director. If you are using a separate IdP for BOSH, copy the metadata XML.
3. Enter your Decryption passphrase. Read the End User License Agreement, and select the checkbox to accept the terms.

4. Your Ops Manager log in page appears. Enter your username and password. Click Login.

5. Download your SAML Service Provider metadata (SAML Relying Party metadata) by navigating to the following urls:
   - 5a. Ops Manager SAML service provider metadata: https://OPS-MAN-FQDN:443/uaa/saml/metadata
   - 5b. BOSH Director SAML service provider metadata: https://BOSH-IP-ADDRESS:8443/saml/metadata

   **Note:** To retrieve your BOSH-IP-ADDRESS, navigate to the Ops Manager Director tile > Status tab. Record the Ops Manager Director IP address.

6. Configure your IdP with your SAML Service Provider metadata. Import the Ops Manager SAML provider metadata from Step 5a above to your IdP. If your IdP does not support importing, provide the values below:
   - Single sign on URL: https://OPS-MAN-FQDN:443/uaa/saml/SSO/alias/OPS-MAN-FQDN
   - Name ID is Email Address
   - SAML authentication requests are always signed

7. Import the BOSH Director SAML provider metadata from Step 5b to your IdP. If the IdP does not support an import, provide the values below.
   - Single sign on URL: https://BOSH-IP:8443/saml/SSO/alias/BOSH-IP
   - Audience URI (SP Entity ID): https://BOSH-IP:8443
   - Name ID is Email Address
   - SAML authentication requests are always signed

8. Return to the Ops Manager Director tile, and continue with the configuration steps below.

**Internal Authentication**

1. When redirected to the Internal Authentication page, you must complete the following steps:
   - Enter a Username, Password, and Password confirmation to create an Admin user.
   - Enter a Decryption passphrase and the Decryption passphrase confirmation. This passphrase encrypts the Ops Manager datastore, and is not recoverable.
   - If you are using an Http proxy or Https proxy, follow the PCF Director Proxy Settings instructions.
   - Read the End User License Agreement, and select the checkbox to accept the terms.

**Step 2: vCenter Config Page**

1. Log in to Ops Manager with the Admin username and password you created in the previous step.
2. Click the **Ops Manager Director** tile.

3. Select **vCenter Config**.
4. Enter the following information:
   - **vCenter Host**: The hostname of the vCenter that manages ESXi/vSphere.
   - **vCenter Username**: A vCenter username with create and delete privileges for virtual machines (VMs) and folders.
   - **vCenter Password**: The password for the vCenter used specified above.
   - **Datacenter Name**: The name of the datacenter as it appears in vCenter.
   - **Virtual Disk Type**: The Virtual Disk Type to provision for all VMs.
   - **Ephemeral Datastore Names (comma delimited)**: The names of the datastores that store ephemeral VM disks deployed by Ops Manager.
   - **Persistent Datastore Names (comma delimited)**: The names of the datastores that store persistent VM disks deployed by Ops Manager.
   - **VM Folder**: The vSphere datacenter folder (default: `pcf_vms`) where Ops Manager places VMs.
   - **Template Folder**: The vSphere datacenter folder (default: `pcf_templates`) where Ops Manager places VMs.
Disk path Folder: The vSphere datastore folder (default: `pcf_disk`) where Ops Manager creates attached disk images. You must not nest this folder.

5. Click Save.

**Note:** After your initial deployment, you will not be able to edit the VM Folder, Template Folder, and Disk path Folder names.

---

### Step 3: Director Config Page

1. Select Director Config.

2. In the NTP Servers (comma delimited) field, enter your NTP server addresses.

3. If you have installed and configured the JMX Bridge product, enter your Metrics IP Address.

4. Select the Enable VM Resurrector Plugin to enable Ops Manager Resurrector functionality and increase Elastic Runtime availability. For more information, see the Using Ops Manager Resurrector on VMware vSphere topic.

5. Select Recreate all VMs to force BOSH to recreate all VMs on the next deploy. This process does not destroy any persistent disk data.

6. Select Keep Unreachable Director VMs if you want to preserve Ops Manager Director VMs after a failed deployment for troubleshooting purposes.

7. Select HM Pager Duty Plugin to enable Health Monitor integration with PagerDuty:
   - **Service Key:** Enter your API service key from PagerDuty.
   - **HTTP Proxy:** Enter an HTTP proxy for use with PagerDuty.
8. Select **HM Email Plugin** to enable Health Monitor integration with email.
   - **Host**: Enter your email hostname.
   - **Port**: Enter your email port number.
   - **Domain**: Enter your domain.
   - **From**: Enter the address for the sender.
   - **Recipients**: Enter comma-separated addresses of intended recipients.
   - **Username**: Enter the username for your email server.
   - **Password**: Enter the password for your email server.
   - **Enable TLS**: Select this checkbox to enable Transport Layer Security.

9. For **Blobstore Location**, Pivotal recommends that you select **Internal**. However, if you select **S3 Compatible Blobstore**, complete the **S3 Endpoint**, **Bucket Name**, **Access Key**, **Secret Key**, **V2 Signature/V4 Signature**, and **Region** with information from your blobstore provider.
10. By default, Pivotal Cloud Foundry (PCF) deploys and manages an Internal database for you. If you choose to use an External MySQL Database, complete the associated fields with information obtained from your external MySQL Database provider: Host, Port, Username, Password, and Database.

11. Max Threads sets the maximum number of threads that the Ops Manager Director can run simultaneously. For vSphere, the default value is 32. Leave the field blank to use this default value. Pivotal recommends that you use the default value unless doing so results in rate limiting or errors on your IaaS.
To add a custom URL for your Ops Manager Director, enter a valid hostname in **Director Hostname**. You can also use this field to configure a load balancer in front of your Ops Manager Director.

Click **Save**.

**Note:** After your initial deployment, you will not be able to edit the Blobstore and Database locations.

### Step 4: Create Availability Zone Page

Ops Manager Availability Zones correspond to your vCenter clusters and resource pools. Multiple Availability Zones allow you to provide high-availability and load balancing to your applications. When you run more than one instance of an application, Ops Manager balances those instances across all of the Availability Zones assigned to the application. At least three availability zones are recommended for a highly available installation of Elastic Runtime.

1. **Select Create Availability Zones.**

![Create Availability Zones](image)

2. Use the following steps to create one or more Availability Zones for your applications to use:
   - Click **Add**.
   - Enter a unique **Name** for the Availability Zone.
   - Enter the name of an existing vCenter **Cluster** to use as an Availability Zone.
   - (Optional) Enter the name of a **Resource Pool** in the vCenter cluster that you specified above. The jobs running in this Availability Zone share the CPU and memory resources defined by the pool.

   **Note:** For more information about using availability zones in vSphere, see the Understanding Availability Zones in VMware Installations topic.

3. **Click Save**.

### Step 5: Create Networks Page

1. **Select Create Networks.**
2. Select **Enable ICMP checks** to enable ICMP on your networks. Ops Manager uses ICMP checks to confirm that components within your network are reachable.

3. Use the following steps to create one or more Ops Manager networks:
   - Click **Add Network**.
   - Enter a unique **Name** for the network.
   - Click **Add Subnet** to create one or more subnets for the network.
   - Enter the full path and **vSphere Network Name** as it displays in vCenter. For example, enter `YOUR-DIRECTORY-NAME/YOUR-NETWORK-NAME`. If your vSphere Network Name contains a forward slash character, replace the forward slash with the URL-encoded forward slash character `%2F`.
   - For **CIDR**, enter a valid CIDR block in which to deploy VMs. For example, enter `192.0.2.0/24`.
   - For **Reserved IP Ranges**, enter any IP addresses from the **CIDR** that you want to blacklist from the installation. Ops Manager will not deploy VMs to any address in this range.
   - Enter your **DNS** and **Gateway** IP addresses.
   - Select which **Availability Zones** to use with the network.

4. Click **Save**.
**Step 6: Assign AZs and Networks Page**

1. Select Assign AZs and Networks.

![Assign AZs and Networks Page](image)

2. Use the drop-down menu to select a **Singleton Availability Zone**. The Ops Manager Director installs in this Availability Zone.

3. Use the drop-down menu to select a **Network** for your Ops Manager Director.

4. Click **Save**.

**Step 7: Security Page**

2. In **Trusted Certificates**, enter a custom certificate authority (CA) certificate to insert into your organization’s certificate trust chain. This feature enables all BOSH-deployed components in your deployment to trust a custom root certificate. If you want to use Docker Registries for running app instances in Docker containers, use this field to enter your certificate for your private Docker Registry. See the **Using Docker Registries** topic for more information.

3. Choose **Generate passwords** or **Use default BOSH password**. Pivotal recommends that you use the **Generate passwords** option for greater security.

4. Click **Save**. To view your saved Director password, click the **Credentials** tab.

**Step 8: Resource Config Page**

1. Select **Resource Config**.

2. Adjust any values as necessary for your deployment. Under the **Instances**, **Persistent Disk Type**, and **VM Type** fields, choose **Automatic** from the
drop-down menu to allocate the recommended resources for the job. If the Persistent Disk Type field reads None, the job does not require persistent disk space.

Note: If you set a field to Automatic and the recommended resource allocation changes in a future version, Ops Manager automatically uses the updated recommended allocation.

3. Click Save.

Step 9: Complete the Ops Manager Director Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes on the right navigation.

3. After you complete this procedure, follow the instructions in the Configuring Elastic Runtime for vSphere and vCloud topic.
Deploying Operations Manager to vCloud Air and vCloud

This topic is a prerequisite to Configuring Ops Manager Director for vCloud Air and vCloud.

This topic describes how to configure the vCloud or vCloud Air Edge Gateways Configure Services screen and install Pivotal Cloud Foundry (PCF) Ops Manager for your Elastic Runtime environment.

Accessing the vShield Edge Gateway Services Interface

Follow these steps to access the vCloud or vCloud Air Edge Gateways Configure Services screen. For more information about edge gateway services, see the VMware vCloud Director documentation.

1. Log into vCloud or vCloud Air.
2. Click the Gateways tab and your virtual datacenter on the Gateways page. The Gateways > Gateways Details page appears.
3. Click Manage Advanced Gateway Settings on the right side of the Gateways > Gateways Details page. The vCloud Director > Administration > Edge Gateways page appears.
4. Select the gateway you want to configure, then click the gear icon and select Edge Gateway Services.

The Configure Services screen for your virtual datacenter displays with the following tabs:
- DHCP
- NAT
- Firewall
- Static Routing
- VPN
- Load Balancer
Configuring NAT Rules

The following section describe how to configure your vCloud or vCloud Air Edge Gateway to ensure that Elastic Runtime can access the web.

To do this, you configure the single source NAT rule (SNAT) and three destination NAT (DNAT) rules that Elastic Runtime requires:

- Elastic Runtime accesses the Internet using an SNAT rule.
- Elastic Runtime’s API endpoint, which is fronted by HAProxy, requires a DNAT rule to forward traffic from a public IP.
- Ops Manager also requires a DNAT rule to connect external sources on any port to its public IP, as illustrated:

vCloud or vCloud Air evaluates NAT rules in the order you list them in, from top to bottom, on the NAT tab of the Edge Gateways Configure Services screen. The image is an example of the configured SNAT rule and a DNAT rule.
Create SNAT and DNAT Rules

To allow outbound connections through Ops Manager public IP address, configure an SNAT rule. To enable inbound traffic over SSH to your Ops Manager VM, create a DNAT rule.

**Note:** Using the Elastic Runtime IP address for outbound connections can be problematic for DNS resolution.

1. In the Edge Gateways Configure Services screen, select the NAT tab.

2. Configure an SNAT rule:
   a. From the Applied on drop down menu, select the network where you want to apply the NAT rule.
   b. In the Original (Internal) source IP/range field, enter the IP range/subnet mask.
   c. In the Translated (External) source IP/range field, enter the Ops manager public IP.
   d. Ensure the checkbox Enabled is checked.

3. Create a destination NAT (DNAT) rule by following the same procedure, using the following configuration:
   - Applied on: Select your external network
   - Original (External) IP/range: Enter the public IP address for Ops Manager
   - Protocol: Select TCP & UDP
   - Original Port: Select 22
   - Translated (Internal) IP/range: Enter the private IP address of your Ops Manager
   - Translated port: 22

Create Firewall Rules for SNAT and DNAT

1. In the Edge Gateways Configure Services screen, select the Firewall tab.

2. Create a SNAT firewall rule allowing outbound traffic from all internal IP addresses to all IP external addresses.
3. Create a DNAT firewall rule allowing inbound traffic from the public IP to the private IP address of your Ops Manager.

Allow Inbound Web Traffic for Ops Manager
Repeat the steps above for ports 80 and 443 for the same public address.

Allow Inbound Web Traffic for Elastic Runtime
Repeat the steps above for ports 80 and 443 for the Elastic Runtime public IP address.

Setting up Static Routing
Select the Enable static routing checkbox.

Setting up Network Rules for Elastic Runtime DNS Resolution

1. In the Edge Gateways Configure Services screen, select the Load Balancer tab.

2. Click Pool Servers, then click Add.

The Add Load Balancer Member Pool wizard appears.

3. Name the pool Load Balancer to Elastic Runtime.

4. In the Configure Service step, enable the pool to support HTTP port 80 and HTTPS port 443. We recommend using the default balancing method, Round Robin.
5. In the Configure Health-Check step, enter Monitor Port 80 for HTTP and 443 for HTTPS. For both HTTP and HTTPS, change the Mode to TCP.

6. In the Manage Members step, click Add. Enter the IP address of the HAProxy VM. Specify 80 for the HTTP port values and 443 for the HTTPS port values.

7. Click Finish.

8. Click Virtual Servers.

9. Click Add.

10. Complete the new virtual server form with the following information:
    - Name: Load Balancer
    - Applied On: Select your external network
    - IP Address: Enter the public IP address of your Elastic Runtime instance
    - Pool: Select the Load Balancer to Elastic Runtime pool
    - Services: Enable HTTP on port 80 with a Persistence Method of None, and HTTPS on port 443 with a Persistence Method of Session Id
    - Enabled: Select this checkbox
11. Click OK to complete.

Deploying Ops Manager to vCloud or vCloud Air

The following procedures guide you through uploading and deploying Ops Manager as a vApp on vCloud or vCloud Air. Refer to the Known Issues topic before getting started.

Upload Ops Manager

You must either upload the Ops Manager vApp into your catalog or use a vApp that your cloud administrator uploaded to your organization’s catalog.

Note: vCloud and vCloud Air use the vCloud Director Web Console, which only supports 32-bit browsers like Firefox. It does not support Chrome. Refer to Article 2034554 in the VMware Knowledge Base for more information about browser versions that the vCloud Director supports.

1. Download Pivotal Cloud Foundry Operations Manager for vCloud Air and vCloud Director from Pivotal Network.

2. Log into vCloud Director.

3. Navigate to Catalogs > My Organization’s Catalogs and select a catalog or click Add to create a new catalog.

   If you are creating a new catalog:
   - Enter a name for the new catalog and click Next.
   - Select a storage type and click Next.
   - Specify sharing (if needed) and click Next.
   - Review your settings and click Finish.

4. Navigate to the vApp Templates tab for your catalog and click Upload.
5. Select Local file and browse to your .ovf file.

6. Enter a name for your Ops Manager vApp, enter a description, and click Upload.

vCloud Director transfers the OVF package to a staging environment, then uploads it to your catalog.

7. Navigate to the Home view and click Add vApp from Catalog.

8. Select your Ops Manager vApp and click Next.

9. Complete the Add vApp from Catalog wizard, changing the default settings as necessary for your environment. See Complete the vApp Wizard and Deploy Ops Manager for more information.

Complete the vApp Wizard and Deploy Ops Manager

After adding the Ops Manager vApp to your vCloud Director, you can finish the set up and deploy as follows:

1. Check the I agree checkbox to accept licenses and click Next.

2. Enter the name of your Ops Manager vApp, select the virtual data center where the vApp should run, and click Next.
3. Choose a storage policy and click Next.

4. Set the network mapping Destination to the network name, set IP allocation to Static—Manual, and click Next.

5. Enter the desired networking information, set an admin password for the Ops Manager vApp, and click Next.
The following list contains tips on entering specific networking information:

- **DNS**: If you are unsure of your Pivotal Ops Manager DNS, you can use the Google Public DNS value 8.8.8.8. For more information, refer to the [Using Google Public DNS](#) topic.
- **Default Gateway**: On the vCloud Air or vCloud Dashboard, click the Gateways tab and copy the GATEWAY IP value.
- **IP Address**: Navigate to the My Clouds > VMs page, locate the Pivotal Ops Manager VM, and copy the IP address from the IP Address column. If this column does not display, click the Customize Columns icon on the right side to set your column display preferences.

6. Review the hardware specifications of the virtual machine and click **Next**.

7. In the Ready to Complete dialog, check the **Power on vApp After This Wizard is Finished** checkbox and click **Finish**.

8. Navigate to the **Home** view to verify that your Ops Manager vApp is being created.

---

### Create a DNS Entry

- **Note**: Ops Manager 1.7 security features require you to create a fully qualified domain name in order to access Ops Manager during the initial configuration.

Create a DNS entry for the IP address that you used for Ops Manager. You must use this fully qualified domain name when you log into Ops Manager in the Installing Pivotal Cloud Foundry on vSphere and vCloud Air topic.

[Return to the Installing Pivotal Cloud Foundry Guide](#)
This topic describes how to configure the Ops Manager Director for VMWare vCloud Air and vCloud.

Before you begin this procedure, ensure that you have successfully completed all steps in the Deploying Operations Manager to vCloud Air and vCloud topic. After you complete this procedure, follow the instructions in the Configuring Elastic Runtime for vSphere, vCloud, and vCloud Air topic.

Step 1: Set Up Ops Manager

1. Navigate to the fully qualified domain of your Ops Manager in a web browser.

2. The first time you start Ops Manager, you must choose one of the following:
   - **Use an Identity Provider**: If you use an Identity Provider, an external identity server maintains your user database.
   - **Internal Authentication**: If you use Internal Authentication, PCF maintains your user database.

   ![Welcome to Ops Manager](image)

   Use an Identity Provider

   1. Log in to your IdP console and download the IdP metadata XML. Optionally, if your IdP supports metadata URL, you can copy the metadata URL instead of the XML.
2. Copy the IdP metadata XML or URL to the Ops Manager Use an Identity Provider log in page.

   ![Identity Provider Metadata](https://ops-preview.com/app/saml/metadata)

   **Note:** The same IdP metadata URL or XML is applied for the BOSH Director. If you are using a separate IdP for BOSH, copy the metadata XML or URL from that IdP and enter it into the BOSH IdP Metadata text box in the Ops Manager log in page.

3. Enter your Decryption passphrase. Read the End User License Agreement, and select the checkbox to accept the terms.

4. Your Ops Manager log in page appears. Enter your username and password. Click Login.

5. Download your SAML Service Provider metadata (SAML Relying Party metadata) by navigating to the following urls:
   - 5a. Ops Manager SAML service provider metadata: https://OPS-MAN-FQDN:443/uaa/saml/metadata
   - 5b. BOSH Director SAML service provider metadata: https://BOSH-IP-ADDRESS:8443/saml/metadata

   ![Download Metadata](https://downloadmetadata.com)

   **Note:** To retrieve your BOSH-IP-ADDRESS, navigate to the Ops Manager Director tile > Status tab. Record the Ops Manager Director IP address.

6. Configure your IdP with your SAML Service Provider metadata. Import the Ops Manager SAML provider metadata from Step 5a above to your IdP. If your IdP does not support importing, provide the values below:
   - **Single sign on URL:** https://OPS-MAN-FQDN:443/uaa/saml/SSO/alias/OPS-MAN-FQDN
   - **Audience URI (SP Entity ID):** https://OPS-MAN-FQDN:443/uaa
   - **Name ID is Email Address**
   - **SAML authentication requests are always signed**

7. Import the BOSH Director SAML provider metadata from Step 5b to your IdP. If the IdP does not support an import, provide the values below.
   - **Single sign on URL:** https://BOSH-IP:8443/saml/SSO/alias/BOSH-IP
   - **Audience URI (SP Entity ID):** https://BOSH-IP:8443
   - **Name ID is Email Address**
   - **SAML authentication requests are always signed**

8. Return to the Ops Manager Director tile, and continue with the configuration steps below.

### Internal Authentication

1. When redirected to the Internal Authentication page, you must complete the following steps:
   - Enter a Username, Password, and Password confirmation to create an Admin user.
   - Enter a Decryption passphrase and the Decryption passphrase confirmation. This passphrase encrypts the Ops Manager datastore, and is not recoverable.
   - If you are using an Http proxy or Https proxy, follow the PCF Director Proxy Settings instructions.
   - Read the End User License Agreement, and select the checkbox to accept the terms.
Step 2: vCloud Config Page

1. Log in to Ops Manager with the Admin username and password you created in the previous step.

2. Click the **Ops Manager Director** tile.

3. Select **vCloud Config**.
vCloud Director Config

vCloud API URL *
https://example.com

Organization name *
VDC-M903513742-9043

Username *
user@pivotal.com

Password *
********

Virtual Datacenter name *
VDC-M903513742-9043

Storage Profile name *
SSD-Accelerated

Catalog Name *
pcf_catalog

Save
4. Enter the following information.
   - **vCloud API URL**: The URL of the vCloud Director.
   - **Organization name**: The name of your organization.

   **Note**: vCloud Air and vCloud Director use case-sensitive organization names. The name that you supply in the Ops Manager **Organization name** field must match the vCD organization name exactly.

   - **Username**: The username for a user who has create and delete privileges for virtual machines (VMs) and folders.
   - **Password**: The password for the user provided in the previous step.
   - **Virtual Datacenter name**: The name of the virtual datacenter as it appears in vCloud Director. The name is an alphanumeric string with a `VDC` prefix. For example, `VDC-9903513742-9043`.
   - **Storage Profile name**: The name of the storage profile as it appears in vCloud Director.

5. Click **Save**.

### Step 3: Director Config Page

![Director Config](image-url)

- **NTP Servers (comma delimited)**
  - `time1.cf-app.com`

- **Metrics IP Address**
  - 

- **Enable VM Resurrecter Plugin**

- **Recreate all VMs**
  - This will force BOSH to recreate all VMs on the next deploy. Persistent disk will be preserved.

- **HIM Pager Duty Plugin**

  - **Service Key**
    - `pagerduty-service-key`

- **HTTP Proxy**
  - 

- **HIM Email Plugin**

  - **Host**
    - `mail.cf-app.com`

  - **Port**
    - `25`

  - **Domain**
    - `cf-app.com`
From*  
admin@cf-app.com

Recipients*  
user@email.com

Username

Password

Enable TLS

Blobstore Location
○ Internal
○ S3 Compatible Blobstore

S3 Endpoint*

Bucket Name*

Access Key*

Secret Key*

○ V2 Signature
○ V4 Signature

Region*

Database Location
○ Internal
○ External MySQL Database

Host*

Port*

Username*
1. Select Director Config.

2. In the NTP Servers (comma delimited) field, enter your NTP server addresses.

3. If you have installed and configured the JMX Bridge product, enter your Metrics IP Address.

4. Select the Enable VM Resurrector Plugin to enable Ops Manager Resurrector functionality and increase Elastic Runtime availability. For more information, see the Using Ops Manager Resurrector on VMware vSphere topic.

5. Select Recreate all VMs to force BOSH to recreate all VMs on the next deploy. This process does not destroy any persistent disk data.

   - Service Key: Enter your API service key from PagerDuty.
   - HTTP Proxy: Enter an HTTP proxy for use with PagerDuty.

7. Select HM Email Plugin to enable Health Monitor integration with email.
   - Host: Enter your email hostname.
   - Port: Enter your email port number.
   - Domain: Enter your domain.
   - From: Enter the address for the sender.
   - Recipients: Enter comma-separated addresses of intended recipients.
   - Username: Enter the username for your email server.
   - Password: Enter the password for your email server.
   - Enable TLS: Select this checkbox to enable Transport Layer Security.

8. For Blobstore Location, Pivotal recommends that you select Internal. However, if you select S3 Compatible Blobstore, complete the S3 Endpoint, Bucket Name, Access Key, Secret Key, V2 Signature/V4 Signature, and Region with information from your blobstore provider.

9. By default, Pivotal Cloud Foundry (PCF) deploys and manages an Internal database for you. If you choose to use an External MySQL Database, complete the associated fields with information obtained from your external MySQL Database provider: Host, Port, Username, Password, and Database.

10. Max Threads sets the maximum number of threads that the Ops Manager Director can run simultaneously. For vCloud, the default value is 4. Leave the field blank to use this default value. Pivotal recommends that you use the default value unless doing so results in rate limiting or errors on your IaaS.

11. Click Save.

Note: After your initial deployment, you will not be able to edit the Blobstore and Database locations.

Step 4: Create Networks Page

1. Select Create Networks.
2. Select **Enable ICMP checks** to enable ICMP on your networks. Ops Manager uses ICMP checks to confirm that components within your network are reachable.

3. Use the following steps to create one or more Ops Manager networks:
   - Click **Add Network**.
   - Enter a unique **Name** for the network.
   - Click **Add Subnet** to create one or more subnets for the network.
   - Enter the **vCloud Network Name** as it appears in vCloud Director.
   - For **CIDR**, enter a valid CIDR block in which to deploy VMs. For example, enter `192.0.2.0/24`.
   - For **Reserved IP Ranges**, enter any IP addresses from the **CIDR** that you want to blacklist from the installation. Ops Manager will not deploy VMs to any address in this range.
   - Enter your **DNS** and **Gateway** IP addresses.

4. Click **Save**.

### Step 5: Assign Networks Page

1. Select **Assign Networks**.
2. Use the drop-down menu to select the network that acts as the infrastructure and deployment network for Ops Manager. You can configure the Ops Manager Director to have an IP address on one network.

Step 6: Security Page


2. In Trusted Certificates, enter a custom certificate authority (CA) certificate to insert into your organization’s certificate trust chain. This feature
enables all BOSH-deployed components in your deployment to trust a custom root certificate. If you want to use Docker Trusted Registries for running app instances in Docker containers, use this field to enter your certificate for your private Docker Trusted Registry. See the Using Docker Trusted Registries topic for more information.

3. Choose Generate passwords or Use default BOSH password. Pivotal recommends that you use the Generate passwords option for greater security.

4. Click Save. To view your saved Director password, click the Credentials tab.

Step 7: Resource Config Page

1. Select Resource Config.

![Resource Config](image)

2. Adjust any values as necessary for your deployment, such as increasing the persistent disk size. Select Automatic from the drop-down menu to provision the amount of persistent disk predefined by the job. If the persistent disk field reads None, the job does not require persistent disk space.

   **Note:** If you set a field to Automatic and the recommended resource allocation changes in a future version, Ops Manager automatically uses the updated recommended allocation.

3. Click Save.

Step 8: Complete the Ops Manager Director Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes.

3. After you complete this procedure, follow the instructions in the Configuring Elastic Runtime for vSphere and vCloud topic.
Configuring Elastic Runtime for vSphere, vCloud, and vCloud Air

This topic describes how to configure the Pivotal Elastic Runtime components that you need to run Pivotal Cloud Foundry (PCF) for VMware vSphere, vCloud, or vCloud Air.

**Step 1: Add Elastic Runtime to Ops Manager**

1. Navigate to the Pivotal Network and click the Pivotal Cloud Foundry banner to access the PCF product page. Use the drop-down menu to select an Elastic Runtime release.

2. From the Available Products view, click Import a Product.


4. In the Available Products view, hover over Elastic Runtime and click Add.
5. Click the Elastic Runtime tile in the Installation Dashboard.

Step 2: Assign Availability Zones and Networks

1. Select **Assign AZs and Networks**. These are the Availability Zones that you create when configuring Ops Manager Director.

2. *(vSphere Only)* Select an Availability Zone under **Place singleton jobs**. Ops Manager runs any job with a single instance in this Availability Zone.

3. *(vSphere Only)* Select one or more Availability Zones under **Balance other jobs**. Ops Manager balances instances of jobs with more than one instance across the Availability Zones that you specify.

4. From the **Network** drop-down box, choose the network on which you want to run Elastic Runtime.
5. Click Save.

Note: When you save this form, a verification error displays because the PCF security group blocks ICMP. You can ignore this error.

Step 3: Configure Domains
1. Select **Domains**.

Elastic Runtime gives each deployed application its own hostname in the app domain, and each system component its own hostname on the system domain. Ensure that you have a wildcard DNS record configured for both domains.

System Domain *

pivotal.cf-app.com

Apps Domain *

pivotal.cf-app.com

This is the default apps domain that pushed apps will use for their hostnames. Use the Cloud Foundry command line interface to add or delete domains. This also requires a wildcard DNS record.

2. Enter the system and application domains.
   - The **System Domain** defines your target when you push apps to Elastic Runtime.
   - The **Apps Domain** defines where Elastic Runtime should serve your apps.

   **Note:** Pivotal recommends that you use the same domain name but different subdomain names for your system and app domains. Doing so allows you to use a single wildcard certificate for the domain while preventing apps from creating routes that overlap with system routes. For example, name your system domain `system.EXAMPLE.com` and your apps domain `apps.EXAMPLE.com`.

   **Note:** You configured wildcard DNS records for these domains in an earlier step.

3. Click **Save**.

**Step 4: Configure Networking**

1. Select **Networking**.

2. *(Optional)* The values you enter in the **Router IPs** and **HAPProxy IPs** fields depends on whether you are using your own load balancer or the HAProxy load balancer. Find your load balancer type in the table below to determine how to complete these fields.

   **Note:** If you choose to assign specific IP addresses in either the **Router IPs** or **HAPProxy IPs** field, ensure that these IPs are in your subnet.

<table>
<thead>
<tr>
<th>LOAD BALANCER</th>
<th>ROUTER IP FIELD VALUE</th>
<th>HAPPROXY IP FIELD VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your own load balancer</td>
<td>Enter the IP address or addresses for PCF that you registered with your load balancer. Refer to the <strong>Using Your Own Load Balancer</strong> topic for help using your own load balancer with PCF. Leave this field blank.</td>
<td>Enter at least one HAProxy IP address. Point your DNS to this address.</td>
</tr>
<tr>
<td>HAProxy load balancer</td>
<td>Leave this field blank.</td>
<td></td>
</tr>
</tbody>
</table>

For more information, refer to the **Configuring PCF SSL Termination** topic. For help understanding the Elastic Runtime architecture, refer to the Architecture topic.

3. For **Router IPs**, enter one or more static IP addresses for your routers. These must be in the subnet that you configured in the Ops Manager **Create Networks** section. If you are using your own load balancer, configure it to point to these IPs. If you are using the Elastic Load Balancer (ELB), add the name of your ELB in the router column of the **Resource Config** section.
4. For HAProxy IPs, enter one or more IP addresses for HAProxy. You must point your DNS to this IP unless you are using your own load balancer, and HAProxy's IP must be in your subnet.

5. Under Configure the point-of-entry to this environment, choose one of the following:
   - External Load Balancer with Encryption: Select this option if your deployment uses an external load balancer that can forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing. Complete the fields for the Router SSL Termination Certificate and Private Key and Router SSL Ciphers.
   - External Load Balancer without Encryption: Select this option if your deployment uses an external load balancer that cannot forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing.
   - HAProxy: Select this option to use HAProxy as your first point of entry. Complete the fields for SSL Certificate and Private Key, and HAProxy SSL Ciphers. Select Disable HTTP traffic to HAProxy if you want the HAProxy to only allow HTTPS traffic.

   **Note:** For details about different SSL termination point options, which correspond to different points-of-entry for Elastic Runtime, see the Providing a Certificate for your SSL Termination Point topic.

6. If you are not using SSL encryption or if you are using self-signed certificates, select Disable SSL certificate verification for this environment. Selecting this checkbox also disables SSL verification for route services.

7. Select the Disable insecure cookies on the Router checkbox to turn on the secure flag for cookies generated by the router.
8. In the Choose whether or not to enable route services section, choose either Enable route services or Disable route services. Route services are a class of marketplace services that perform filtering or content transformation on application requests and responses. See the Route Services topic for details.

9. (Optional) The Loggregator Port defaults to 443 if left blank. Enter a new value to override the default.

10. (Optional) Use the Applications Subnet field if you need to avoid address collision with a third-party service on the same subnet as your apps. Enter a CIDR subnet mask specifying the range of available IP addresses assigned to your app containers. The IP range must be different from the network used by the system VMs.

11. (Optional) You can change the value in the Applications Network Maximum Transmission Unit (MTU) field. Pivotal recommends setting the MTU value for your application network to 1454. Some configurations, such as networks that use GRE tunnels, may require a smaller MTU value.

12. (Optional) Increase the number of seconds in the Router Timeout to Backends field to accommodate larger uploads over connections with high latency.

13. (Optional) Increase the value of Load Balancer Unhealthy Threshold to specify the amount of time, in seconds, that the router continues to accept connections before shutting down. During this period, healthchecks may report the router as unhealthy, which causes load balancers to failover to other routers. Set this value to an amount greater than or equal to the maximum time it takes your load balancer to consider a router instance unhealthy, given contiguous failed healthchecks.

14. Click Save.

Step 5: Configure Application Containers

1. Select Application Containers.
2. The **Enable Custom Buildpacks** checkbox governs the ability to pass a custom buildpack URL to the `-b` option of the `cf push` command. By default, this ability is enabled, letting developers use custom buildpacks when deploying apps. Disable this option by disabling the checkbox. For more information about custom buildpacks, refer to the `buildpacks` section of the PCF documentation.

3. The **Allow SSH access to app containers** checkbox controls SSH access to application instances. Enable the checkbox to permit SSH access across your deployment, and disable it to prevent all SSH Access. See Application SSH Overview for information on SSH access permissions at the space and app scope.

4. You can configure Elastic Runtime to run app instances in Docker containers by supplying their IP address range(s) in the **Private Docker Insecure Registry Whitelist** textbox. See the Using Docker Registries topic for more information.

5. Select your preference for **Docker Images Disk-Cleanup on Cell VMs**. If you choose **Clean up disk-space once threshold is reached**, enter a **Threshold of Disk-Used** in megabytes.

6. Click **Save**.

---

**Step 6: Configure Application Developer Controls**

1. Select Application Developer Controls.
2. Enter your intended maximum file upload size.

3. Enter your default RAM memory allocation per app.

4. Enter your default total RAM memory (RAM) quota per Org. You can change this in the CLI.

5. Enter your maximum and default disk quotas per app.

6. Enter your default service instances quota per Org. You can change this in the CLI.

7. Click Save.

**Step 7: Review Application Security Group**

Setting appropriate Application Security Groups is critical for a secure deployment. Type X in the box to acknowledge that once the Elastic Runtime deployment completes, you will review and set the appropriate application security groups. See Restricting App Access to Internal PCF Components for instructions.
Step 8: Configure Authentication and Enterprise SSO

1. Select Authentication and Enterprise SSO.
2. To authenticate user sign-ons, your deployment can use one of three types of user database: the UAA server’s internal user store, an external SAML identity provider, and an external LDAP server.

   a. To use the internal UAA, select the Internal option and follow the instructions in Configuring UAA Password Policy to configure your password policy.

   b. To connect to an external identity provider via SAML, scroll down to select the SAML Identity Provider option and follow the instructions in Configuring PCF for SAML.

   c. To connect to an external LDAP server, scroll down to select the LDAP Server option and follow the instructions in Configuring LDAP.

3. (Optional) In the Apps Manager Access Token Lifetime, Apps Manager Refresh Token Lifetime, Cloud Foundry CLI Access Token Lifetime, Cloud Foundry CLI Refresh Token Lifetime fields, you can change the lifetimes of tokens granted for Apps Manager and cf CLI login access and refresh. Most deployments use the defaults.

4. (Optional) The Proxy IPs Regular Expression field contains a pipe delimited set of regular expressions that UAA considers to be reverse proxy IP addresses. UAA respects the \x-forwarded-for and \x-forwarded-proto headers coming from IP addresses that match to these regular expressions. To configure UAA to respond properly to Router or HAProxy requests coming from public IP address(es), append a regular expression or regular expressions to match the public IP address(es).
Step 9: Configure System Databases

Note: If you are performing an upgrade, do not modify your existing internal database configuration or you may lose data. You must migrate your existing data first before changing the configuration. See Upgrading Pivotal Cloud Foundry for additional upgrade information.

1. Select Databases.

Place the databases used by Elastic Runtime components like Cloud Controller and UAA.

Choose the location of your system databases:

- Internal Databases - MySQL and Postgres (the Postgres DBs are not highly-available, but this selection is required if you want to keep your system data from a pre-1.6.0 Elastic Runtime that you upgraded)
- Internal Databases - MySQL (preferred for complete high-availability)
- External Databases (preferred if, for example, you use AWS RDS)

Hostname

TCP Port

App Usage Service database username

App Usage Service database password

2. If you want to use internal databases for your deployment, select Internal Databases - MySQL and Postgres or Internal Databases - MySQL. If you want to use external databases such as Amazon Web Services (AWS) RDS, select External Databases and complete the following steps:
   - For Hostname, enter the hostname of your database.
   - For TCP Port, enter the port of your database.
   - For each database username and database password field, enter a unique username and password.

   Note: Pivotal recommends that you use internal databases unless you require the functionality of AWS RDS.

3. Click Save.

Step 10 (Optional) Configure Internal MySQL
1. Select Internal MySQL.

2. In the MySQL Proxy IPs field, enter one or more comma-delimited IP addresses that are not in the reserved CIDR range of your network. If a MySQL node fails, these proxies re-route connections to a healthy node. See the MySQL Proxy topic for more information.

3. For MySQL Service Hostname, enter an IP address or hostname for your load balancer. If a MySQL proxy fails, the load balancer re-routes connections to a healthy proxy. If you leave this field blank, components are configured with the IP address of the first proxy instance entered above.

   **Warning:** You must configure a load balancer to achieve complete high-availability.

4. In the Replication canary time period field, leave the default of 30 seconds or modify the value based on the needs of your deployment. Lower numbers cause the canary to run more frequently, which means that the canary reacts more quickly to replication failure but adds load to the database.

5. In the Replication canary read delay field, leave the default of 20 seconds or modify the value based on the needs of your deployment. This field configures how long the canary waits, in seconds, before verifying that data is replicating across each MySQL node. Clusters under heavy load can experience a small replication lag as write-sets are committed across the nodes.

6. (Required): In the E-mail address field, enter the email address where the MySQL service sends alerts when the cluster experiences a replication issue or when a node is not allowed to auto-rejoin the cluster.

7. Under Automated Backups Configuration, choose one of three options for MySQL backups:
   - Disable automatic backups of MySQL.
   - Enable automated backups from MySQL to an S3 bucket or other S3-compatible file store saves your backups to an existing Amazon Web Services (AWS) or Ceph S3-compatible blobstore.
This option requires the following fields:

- For **S3 Bucket Name**, enter the name of your S3 bucket. Do not include an `s3://` prefix, a trailing `/`, or underscores. If the bucket does not already exist, it will be created automatically.
- For **Bucket Path**, specify a folder within the bucket to hold your MySQL backups. Do not include a trailing `/`.
- For **AWS Access Key ID** and **AWS Secret Access Key**, enter your AWS or Ceph credentials.
- For **Cron Schedule**, enter a valid `cron` expression to schedule your automated backups. Cron uses your computer’s local time zone.
- **Enable automated backups from MySQL to a remote host via SCP**: saves your backups to a remote host using secure copy protocol (SCP).
This option requires the following fields:

- For **Hostname**, enter the name of your SCP host.
- For **Port**, enter your SCP port. This should be the TCP port that your SCP host uses for SSH. The default port is 22.
- For **Username**, enter your SSH username for the SCP host.
- For **Private key**, paste in your SSH private key.
- For **Destination directory**, enter the directory on the SCP host where you want to save backup files.
- For **Cron Schedule**, enter a valid cron expression to schedule your automated backups. Cron uses your computer's local time zone.
- Enable **Backup All Nodes** to make unique backups from each instance of the MySQL server rather than just the first MySQL server instance.

**Note:** If you choose to enable automated MySQL backups, set the number of instances for the **Backup Prepare Node** under the **Resource Config** section of the Elastic Runtime tile to 0.

8. If you want to log audit events for internal MySQL, select **Enable server activity logging** under **Server Activity Logging**.
   a. For the **Event types** field, you can enter the events you want the MySQL service to log. By default, this field includes `connect` and `query`, which tracks who connects to the system and what queries are processed. For more information, see the [Logging Events](https://mariadb.com/kb/en/gtid/) section of the MariaDB documentation.
9. Click Save.

Step 11: Configure File Storage

1. Select File Storage.

2. To use the PCF internal filestore, select the Internal option and click Save.

3. To use an external S3-compatible filestore for your Elastic Runtime file storage, select the External S3-Compatible Filestore option and complete the following procedure:
a. Enter the URL Endpoint for your filestore.

b. Enter your Access Key and Secret Key.

c. For S3 Signature Version and Region, use V4 Signature values.

d. Enter a Buildpacks Bucket Name.

e. Enter a Droplets Bucket Name.

f. Enter a Packages Bucket Name.

g. Enter a Resources Bucket Name.

h. Click Save.

Note: For more information about AWS S3 Signatures, see the Authenticating Requests documentation.

Step 12: (Optional) Configure System Logging

If you are forwarding logging messages to an external Reliable Event Logging Protocol (RELP) server, complete the following steps:

1. Select System Logging.

2. If you want to include security events in your log stream, select the Enable Cloud Controller security event logging checkbox. This logs all API requests, including the endpoint, user, source IP, and request result, in the Common Event Format (CEF).

3. Enter the IP address of your syslog server in External Syslog Aggregator Hostname and its port in External Syslog Aggregator Port. The default port for a syslog server is 514.

   Note: The host must be reachable from the Elastic Runtime network, accept TCP connections, and use the RELP protocol. Ensure your syslog server listens on external interfaces.


5. For the Syslog Drain Buffer Size, enter the number of messages the Doppler server can hold from Metron agents before the server starts to drop them. See the Loggregator Guide for Cloud Foundry Operators topic for more details.

6. Click Save.

Step 13: (Optional) Customize Apps Manager

The Custom Branding and Apps Manager sections customize the appearance and functionality of Apps Manager. Refer to the Custom Branding Apps Manager topic for more information.
1. Select **Custom Branding**.

![Customize colors, images, and text for Apps Manager and the Cloud Foundry login portal.](image)

Use this section to configure the text, colors, and images of the interface that developers see when they log in, create an account, reset their password, or use Apps Manager.

2. Select **Apps Manager**.
Use this section to control access and role options for Apps Manager, as well as specify the headers and sidebar links that it displays.

- Select **Enable Internal User Store** to use an internal user store in the PCF local UAA server. With the internal store enabled, PCF admins do not need to configure an external user store such as an LDAP / AD server.
- Select **Enable Non Admin Role Management** to allow Org managers and Space managers to assign roles to users in the Orgs and Spaces that they manage.

Both the **Enable Internal User Store** and the **Enable Non Admin Role Management** checkboxes must be selected if you want to invite new members to join an Org in Apps Manager. For more information about inviting users, see Inviting New Users. Ensure that you select the **Push Apps Manager** post-deploy errand on the Errands tab so that these edits are applied to your Apps Manager installation. Follow the steps in the “Configure Errands” below.

3. Click **Save** to save your settings for each section.

### Step 14: (Optional) Configure Email Notifications

Elastic Runtime uses SMTP to send invitations and confirmations to Apps Manager users. You must complete the **Email Notifications** page if you want to enable end-user self-registration.

1. Select **Email Notifications**.
2. Enter your reply-to and SMTP email information

3. Verify your authentication requirements with your email administrator and use the SMTP Authentication Mechanism drop-down menu to select None, Plain, or CRAM-MD5. If you have no SMTP authentication requirements, select None.

4. Click Save.

Note: If you do not configure the SMTP settings using this form, the administrator must create orgs and users using the cf CLI tool. See Creating and Managing Users with the cf CLI for more information.

Step 15: (Optional) Add CCDB Restore Key

Perform this step if all of the following are true:

- You deployed Elastic Runtime previously
- You then stopped Elastic Runtime or it crashed
- You are re-deploying Elastic Runtime with a backup of your Cloud Controller database

1. Click Restore CCDB Encryption Key.

2. Enter your Cloud Controller DB Encryption Key.
See Backing Up Pivotal Cloud Foundry for more information.

Step 16: Configure Smoke Tests

The Smoke Tests errand runs basic functionality tests against your Elastic Runtime deployment after an installation or update. In this section, choose where to run smoke tests. In the Errands section, you can choose whether or not to run the Smoke Tests errand.

1. Select Smoke Tests.

2. If you have a shared apps domain, select On-demand org and space, which creates an ad-hoc org and space for running smoke tests and deletes them afterwards. Otherwise, select Specified org and space and complete the fields to specify where you want to run smoke tests.

3. Click Save.

Step 17: (Optional) Enable Experimental Features

Use caution when enabling experimental features if you have other Pivotal Cloud Foundry service tiles installed in your Pivotal Cloud Foundry deployment. Not all of the services are guaranteed to work as expected with these features enabled.
Diego Cell Memory and Disk Overcommit

If your apps do not use the full allocation of disk space and memory set in the Resource Config tab, you might want to use this feature. These fields control the amount to overcommit disk and memory resources to each Diego Cell VM.

For example, you might want to use the overcommit if your apps use a small amount of disk and memory capacity compared with the Resource Config settings for Diego Cell.

**Note:** Due to the risk of app failure and the deployment-specific nature of disk and memory use, Pivotal has no recommendation about how much, if any, memory or disk space to overcommit.

To enable this feature, follow these steps:

1. Select Experimental Features.
2. Enter the total desired amount of Diego cell memory value in the Cell Memory Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell memory capacity settings that this field overrides.
3. Enter the total desired amount of Diego cell disk capacity value in the Cell Disk Capacity (MB) field. Refer to the Diego Cell row in the Resource Config tab for the current Cell disk capacity settings that this field overrides.

**Note:** Entries made to each of these two fields set the total amount of resources allocated, not the average.

4. Click Save.

CF CLI Connection Timeout

The CF CLI Connection Timeout field allows you to override the default 5 second timeout of the Cloud Foundry Command Line Interface (cf CLI) used within your PCF deployment. This timeout affects the `cf` command used to push Elastic Runtime errand apps such as Notifications, Autoscaler, Apps Manager and so on.

Set the value of this field to a higher value, in seconds, if you are experiencing domain name resolution timeouts when pushing errands in Elastic Runtime.

To modify your CF CLI connection timeout, perform the following steps:

1. Select Experimental Features.
2. Add a value, in seconds, to the CF CLI Connection Timeout field.
3. Click Save.

Step 18: Configure Errands

Errands are scripts that Ops Manager runs to automate tasks. By default, Ops Manager runs the post-install errands listed below when you deploy Elastic Runtime. However, you can prevent a specific post-install errand from running by deselecting its checkbox on the Errands page.

**Note:** Several errands deploy apps that provide services for your deployment, such as Autoscaling and Notifications. Once one of these apps is running, deselecting the checkbox for the corresponding errand on a subsequent deployment does not stop the app.
Run Smoke Tests verifies that your deployment can do the following:

- Push, scale, and delete apps
- Create and delete orgs and spaces

Push Apps Manager deploys the Apps Manager, a dashboard for managing apps, services, orgs, users, and spaces. Until you deploy Apps Manager, you must perform these functions through the cf CLI. After Apps Manager has been deployed, we recommend deselecting the checkbox for this errand on subsequent Elastic Runtime deployments. For more information about the Apps Manager, see Getting Started with the Apps Manager.

Notifications deploys an API for sending email notifications to your PCF platform users.

Note: The Notifications app requires that you configure SMTP with a username and password, even if SMTP Authentication Mechanism is set to none.

Notifications-UI deploys a dashboard for users to manage notification subscriptions.

Deploy CF Autoscaling App enables your deployment to automatically scale the number of instances of an app in response to changes in its usage load. To enable Autoscaling for an app, you must also bind the Autoscaling service to it. For more information, see the Bind a Service Instance section of the Managing Service Instances with the CLI topic.

Note: The Autoscaling app requires the Notifications app to send scaling action alerts by email.

Register Autoscaling Service Broker makes the Autoscaling service available to your applications. Without this errand, you cannot bind the Autoscaling app to your apps.

Step 19: (Optional) Configure Resources

Note: Ops Manager 1.7 defines specific instance types instead of custom sizes. Each instance adopts an instance type that is the closest match to its previous custom size for CPU, memory, and disk space. You can modify the automatically selected instance size by selecting a different type under Resource Config.

Scale the number of instances in order to reduce resources and configure your deployment.
By default, Elastic Runtime uses an internal filestore and internal databases. If you configure Elastic Runtime to use external resources, you can disable the corresponding system-provided resources in Ops Manager to reduce costs and administrative overhead.

For more information regarding scaling instances, see the [Zero Downtime Deployment and Scaling in CF](https://www.pivotal.io) and the [Scaling Instances in Elastic Runtime](https://www.pivotal.io).
Complete the following procedures to disable specific VMs in Ops Manager:

1. Click Resource Config.

2. If you configure Elastic Runtime to use an external S3-compatible filestore, edit the following fields:
   - NFS Server: Enter 0 in Instances.

3. If you configure Elastic Runtime to use an external Relational Database Service (RDS), edit the following fields:
   - MySQL Proxy: Enter 0 in Instances.
   - MySQL Server: Enter 0 in Instances.
   - Cloud Controller Database: Enter 0 in Instances.
   - UAA Database: Enter 0 in Instances.
   - Apps Manager Database: Enter 0 in Instances.

4. If you are using an External Load Balancer instead of HAProxy, enter 0 in the Instances field for HAProxy.

5. Click Save.

Step 20: Configure Stemcell

1. Select Stemcell. This page displays the stemcell version that shipped with Ops Manager.

You can also use this page to import a new stemcell version. You only need to import a new Stemcell if your Ops Manager does not already have the Stemcell version required by Elastic Runtime.

Step 21: Complete the Elastic Runtime Installation

1. Click the Installation Dashboard link to return to the Installation Dashboard.

2. Click Apply Changes. If the following ICMP error message appears, click Ignore errors and start the install.

The install process generally requires a minimum of 90 minutes to complete. The image shows the Changes Applied window that displays when the installation process successfully completes.
Changes Applied

Ops Manager Director was successfully installed.
We recommend that you export a backup of this installation from the actions menu.

Close  Return to Installation Dashboard
Provisioning a Virtual Disk in vSphere

When you create a virtual machine in VMware vSphere, vSphere creates a new virtual hard drive for that virtual machine. The virtual hard drive is contained in a virtual machine disk (VMDK). The disk format you choose for the new virtual hard drive can have a significant impact on performance.

You can choose one of three formats when creating a virtual hard drive:

- Thin Provisioned
- Thick Provisioned Lazy Zeroed
- Thick Provisioned Eager Zeroed

Thin Provisioned

Advantages:

- Fastest to provision
- Allows disk space to be overcommitted to VMs

Disadvantages:

- Slowest performance due to metadata allocation overhead and additional overhead during initial write operations
- Overcommitment of storage can lead to application disruption or downtime if resources are actually used
- Does not support clustering features

When vSphere creates a thin provisioned disk, it only writes a small amount of metadata to the datastore. It does not allocate or zero out any disk space. At write time, vSphere first updates the allocation metadata for the VMDK, then zeros out the block or blocks, then finally writes the data. Because of this overhead, thin provisioned VMDKs have the lowest performance of the three disk formats.

Thin provisioning allows you to overcommit disk spaces to VMs on a datastore. For example, you could put 10 VMs, each with a 50 GB VMDK attached to it, on a single 100 GB datastore, as long as the sum total of all data written by the VMs never exceeded 100 GB. Thin provisioning allows administrators to use space on datastores that would otherwise be unavailable if using thick provisioning, possibly reducing costs and administrative overhead.

Thick Provisioned Lazy Zeroed

Advantages:

- Faster to provision than Thick Provisioned Eager Zeroed
- Better performance than Thin Provisioned

Disadvantages:

- Slightly slower to provision than Thin Provisioned
- Slower performance than Thick Provisioned Eager Zero
- Does not support clustering features

When vSphere creates a thick provisioned lazy zeroed disk, it allocates the maximum size of the disk to the VMDK, but does nothing else. At the initial access to each block, vSphere first zeros out the block, then writes the data. Performance of a thick provisioned lazy zeroed disk is not as good a thick provisioned eager zero disk because of this added overhead.

Thick Provisioned Eager Zeroed

Advantages:

- Best performance
- Overwriting allocated disk space with zeros reduces possible security risks
- Supports clustering features such as Microsoft Cluster Server (MSCS) and VMware Fault Tolerance
Disadvantages:

- Longest time to provision

When vSphere creates a thick provisioned eager zeroed disk, it allocates the maximum size of the disk to the VMDK, then zeros out all of that space.

Example: If you create an 80 GB thick provisioned eager zeroed VMDK, vSphere allocates 80 GB and writes 80 GB of zeros.

By overwriting all data in the allocated space with zeros, thick provisioned eager zeroed eliminates the possibility of reading any residual data from the disk, thereby reducing possible security risks.

Thick provisioned eager zeroed VMDKs have the best performance. When a write operation occurs to a thick provisioned eager zeroed disk, vSphere writes to the disk, with none of the additional overhead required by thin provisioned or thick provisioned lazy zeroed formats.
Using the Cisco Nexus 1000v Switch with Ops Manager

Refer to the procedure in this topic to use Ops Manager with the Cisco Nexus 1000v Switch. First, configure Ops Manager through Step 4 in Configuring Ops Manager Director for VMware vSphere. Then configure your network according to the following steps.

1. From your Pivotal Cloud Foundry (PCF) Ops Manager Installation Dashboard, click the Ops Manager Director tile.

2. Select Create Networks.

3. Click the network name to configure the network settings. This is default if you have not changed the name.

4. Find the folder name and port group name for the switch, as you configured them in vCenter. For the example vSphere environment pictured below, a user might want to use the switch configured on the beer-apple port group, which is in the drinks-dc folder.
5. In the **vSphere Network Name** field, instead of entering your network name, enter the folder name and port group name for the switch, as you configured them in vCenter. For the example vSphere environment pictured above, you would enter **drinks-dc/beer-apple** to use the switch configured on the **beer-apple** port group.
6. Click **Save**.

7. Return to Configuring Ops Manager Director for VMware vSphere to complete the Ops Manager installation.
Using Ops Manager Resurrector on VMware vSphere

Page last updated:

The Ops Manager Resurrector increases Pivotal Cloud Foundry (PCF) Elastic Runtime availability in the following ways:

- Reacts to hardware failure and network disruptions by restarting virtual machines on active, stable hosts
- Detects operating system failures by continuously monitoring virtual machines and restarting them as required
- Continuously monitors the BOSH Agent running on each virtual machine and restarts the VMs as required

The Ops Manager Resurrector continuously monitors the status of all virtual machines in an Elastic Runtime deployment. The Resurrector also monitors the BOSH Agent on each VM. If either the VM or the BOSH Agent fail, the Resurrector restarts the virtual machine on another active host.

Limitations

The following limitations apply to using the Ops Manager Resurrector:

- The Resurrector does not monitor or protect the Ops Manager VM or the BOSH Director VM.
- The Resurrector might not be able to resolve issues caused by the loss of an entire host.
- The Resurrector does not monitor or protect data storage.

For increased reliability, in addition to using BOSH Resurrector, Pivotal recommends that you use vSphere High Availability to protect all of the VMs in your deployment, and that you use a highly-available storage solution.

Enabling vSphere High Availability

Follow the steps below to enable vSphere High Availability:

1. Launch the vSphere Management Console.
2. Right-click the cluster that contains the Pivotal Cloud Foundry (PCF) deployment and select Edit Settings.
3. Check the Turn on vSphere HA checkbox.
4. Click OK to enable vSphere High Availability on the cluster.
Enabling Ops Manager Resurrector

To enable the Ops Manager Resurrector:

1. Log into the Ops Manager web interface.

2. On the Product Dashboard, select **Ops Manager Director**.

3. In the left navigation menu, select **Director Config**.

4. Check **Enable VM Resurrector Plugin** and click **Save**.
Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments

To use SSL termination in Pivotal Cloud Foundry (PCF), you must configure the Pivotal-deployed HAProxy load balancer or your own load balancer. Pivotal recommends that you use HAProxy in lab and test environments only. Production environments should instead use a highly-available customer-provided load balancing solution.

Select an SSL termination method to determine the steps you must take to configure Elastic Runtime.

Using the Pivotal HAProxy Load Balancer

PCF deploys with a single instance of HAProxy for use in lab and test environments. You can use this HAProxy instance for SSL termination and load balancing to the PCF Routers. HAProxy can generate a self-signed certificate if you do not want to obtain a signed certificate from a well-known certificate authority.

Note: Certificates generated in Elastic Runtime are signed by the Operations Manager Certificate Authority. They are not technically self-signed, but they are referred to as ‘Self-Signed Certificates’ in the Ops Manager GUI and throughout this documentation.

To use the HAProxy load balancer, you must create a wildcard A record in your DNS and configure three fields in the Elastic Runtime product tile.

1. Create an A record in your DNS that points to the HAProxy IP address. The A record associates the System Domain and Apps Domain that you configure in the Domains section of the Elastic Runtime tile with the HAProxy IP address.

For example, with cf.example.com as the main subdomain for your CF install and an HAProxy IP address 203.0.113.1, you must create an A record in your DNS that serves example.com and points * .cf to 203.0.113.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.cf</td>
<td>A</td>
<td>203.0.113.1</td>
<td>example.com</td>
</tr>
</tbody>
</table>

2. Use the Linux `host` command to test your DNS entry. The `host` command should return your HAProxy IP address.

Example:

```
$ host cf.example.com
cf.example.com has address 203.0.113.1
$ host anything.example.com
anything.cf.example.com has address 203.0.113.1
```

3. From the PCF Ops Manager Dashboard, click on the Elastic Runtime tile.

4. Select Networking.

5. Leave the Router IPs field blank. HAProxy assigns the router IPs internally.

6. Enter the IP address for HAProxy in the HAProxy IPs field.

7. Provide your SSL certificate in the SSL Termination Certificate and Private Key field. See Providing a Certificate for your SSL Termination Point for details.

Return to the Getting Started Guide

Using Another Load Balancer

Production environments should use a highly-available customer-provided load balancing solution that does the following:

- Provides SSL termination with wildcard DNS location
- Provides load balancing to each of the PCF Router IPs
• Adds appropriate x-forwarded-for and x-forwarded-proto HTTP headers

You must register static IP addresses for PCF with your load balancer and configure three fields in the Elastic Runtime product tile.

1. Register one or more static IP address for PCF with your load balancer.

2. Create an A record in your DNS that points to your load balancer IP address. The A record associates the System Domain and Apps Domain that you configure in the Domains section of the Elastic Runtime tile with the IP address of your load balancer.

   For example, with cf.example.com as the main subdomain for your CF install and a load balancer IP address 198.51.100.1, you must create an A record in your DNS that serves example.com and points *.cf to 198.51.100.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.cf</td>
<td>A</td>
<td>198.51.100.1</td>
<td>example.com</td>
</tr>
</tbody>
</table>

3. From the PCF Ops Manager Dashboard, click on the Elastic Runtime tile.

4. Select Networking.

5. In the Router IPs field, enter the static IP address for PCF that you have registered with your load balancer.

6. Leave the HAProxy IPs field blank.

7. Provide your SSL certificate in the SSL Termination Certificate and Private Key field. See Providing a Certificate for your SSL Termination Point for details.

   **Note:** When adding or removing PCF routers, you must update your load balancing solution configuration with the appropriate IP addresses.

Return to the Installing Pivotal Cloud Foundry Guide
Understanding Availability Zones in VMware Installations

Note: Pivotal Cloud Foundry (PCF) for vCloud Air and vCloud Director is deprecated and availability is restricted to existing customers. Contact Support for more information.

Pivotal defines an Availability Zone (AZ) as an operator-assigned, functionally independent segment of network infrastructure. In cases of partial infrastructure failure, Pivotal Cloud Foundry (PCF) Elastic Runtime distributes and balances all instances of running applications across remaining AZs. Strategic use of Availability Zones contributes to the fault tolerance and high availability of a Elastic Runtime deployment.

It is recommended that customers use three Availability Zones to operate a highly available installation of Elastic Runtime.

AZ Differences Between VMware Products

Elastic Runtime on VMware vSphere supports distributing deployments across multiple AZs. See the section on AZs in Configuring Ops Manager Director for VMware vSphere. VMware vCloud and vCloud Air support only one AZ, which is the virtual data center you access with vShield Edge Gateway Services interface.

Balancing Across AZs During Failure: Example Scenario

An operator scales an application to four instances in an Elastic Runtime environment distributed across three availability zones: A1, A2, and A3. The environment allocates the instances according to the Diego Auction.

If A1 experiences a power outage or hardware failure, the two application instances running in A1 terminate while the application instances in zones A2 and A3 continue to run:

If A1 remains unavailable, Elastic Runtime balances new instances of the application across the remaining availability zones:
Upgrading Pivotal Cloud Foundry

Page last updated:

Note: Pivotal Cloud Foundry (PCF) for vCloud Air and vCloud Director is deprecated and availability is restricted to existing customers. Contact Support for more information.

This topic describes upgrading Pivotal Cloud Foundry to version 1.7. The upgrade procedures below describe upgrading Ops Manager, Elastic Runtime, and product tiles.

If you have already upgraded your Ops Manager to version 1.7 and want to upgrade individual products such as PCF Elastic Runtime, Pivotal MySQL, or RabbitMQ in your PCF deployment, see the Upgrading Products in a PCF Deployment topic.

Important: Before you upgrade to Pivotal Cloud Foundry 1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. All apps that do not already run on Diego will terminate during an upgrade from 1.6 to 1.7.

Important: The apps in your deployment continue to run during the upgrade. However, you cannot write to your deployment or make changes to apps during the upgrade.

Before You Upgrade

This section contains important guidelines that you must follow before beginning an upgrade to PCF 1.7. Failure to follow these instructions may jeopardize your existing deployment data or cause your upgrade to fail.

The tasks you need to complete before upgrade fall into three categories:

- **Choose Single-AZ or Multi-AZ** - choose a single-AZ or multi-AZ deployment.
- **Prepare Your Environment** — includes making sure you have sufficient disk space.
- **Prepare Databases and Apps** — includes backing up data and migrating any apps that currently use the DEA architecture to use the Diego architecture.
- **Check System Health** — perform a health check of your system, from virtual machines (VMs) to service tiles.

Choose Single-AZ or Multi-AZ

For AWS and OpenStack, Ops Manager 1.7 ships with a new multi-AZ functionality that results in multiple networks.

Note: The multi-AZ functionality ships with PCF 1.6 and later for vSphere.

When you upgrade to PCF 1.7 from a previous version instead of performing a new full install, you have a single-AZ deployment. If you want to use PCF 1.7 with multi-AZ functionality, choose one of the following procedures:

- **Upgrade and then apply multi-AZ:**
  1. Follow the upgrade instructions on this page to upgrade to PCF 1.7.
  2. Follow the instructions in this Knowledge Base article to enable advanced mode in Ops Manager.
  3. Follow the instructions in this Knowledge Base article to apply the multi-AZ feature to your deployment.

- Perform a new installation of PCF 1.7.

Prepare Your Environment for Upgrade

1. Install the releases from your currently deployed version to the target version in sequential order. For example, if your deployment uses Ops Manager release 1.5 and you are upgrading to 1.7, you must sequentially install 1.6 and 1.7.

2. Ensure that you are using Elastic Runtime (ERT) v. 1.6.9 or higher. For information about how to upgrade product tiles, see the Upgrading Elastic Runtime and Other Pivotal Cloud Foundry Products topic.
3. Ensure that every product tile on the Installation Dashboard is compatible with the new version of Ops Manager. For specific compatibility information, refer to the full Product Version Matrix. If a product does not meet this requirement, you must upgrade the product or remove the tile before upgrading Ops Manager.

4. For each product tile you have installed, review the upgrade documentation that is specific to the tile. For example, if you have RabbitMQ, you need to increase the number of HAproxy instances from one to two. See RabbitMQ for Pivotal Cloud Foundry Upgrades.

5. Confirm that you have adequate disk space for your upgrades. From your Installation Dashboard, select the Ops Manager Director tile. Select Status. If you need more space to handle your upgrades, select Settings > Resource Config. Increase your persistent disk space to 100 GB, or enough to handle the size of the resources.

6. If you have disabled lifecycle errands for any installed product in order to reduce deployment time, Pivotal recommends that you re-enable these errands before upgrading. See Adding and Deleting Products for more information.

7. Ensure that the VM resurrector is turned off:
   a. From your Installation Dashboard, select the Ops Manager Director tile.
   b. Click Director Config.
   c. Clear the Enable VM resurrector plugin checkbox.
   d. Click Save.
   e. Return to the Installation Dashboard, and click Apply Changes.

8. If you are upgrading a vSphere environment, ensure that you have the following information about your existing environment before starting the upgrade:
   • Note these IP addresses, which can be found in the vSphere web client, Manage > Settings > vApp Options. This is the same information you entered at the end of deploying. Ops Manager on vSphere.
     i. IP Address of the Ops Manager
     ii. Netmask
     iii. Default Gateway
     iv. DNS Servers
     v. NTP Servers
   • Note this VM hardware information so you can set up the new VM with similar settings. You can find this in the vSphere web client, Manage > Settings > VM Hardware.
     i. CPU
     ii. Memory
     iii. Hard Disk 1
     iv. Network Adapter 1 - When you set up the new VM, ensure your network adapters are configured properly and are on the same network.

### Prepare Databases and Apps for Upgrade

1. Back up all critical data prior to upgrading to PCF 1.7. For example, to backup a 1.6 environment, follow these instructions Backing Up Pivotal Cloud Foundry.

2. Before you upgrade to Pivotal Cloud Foundry 1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. Pivotal does not support DEA architecture in Pivotal Cloud Foundry 1.7. See the Migrating Apps to Diego topic for information.

### Check System Health Before Upgrade

1. Run `bosh cloudcheck` to confirm that the VMs are healthy. See the BOSH Cloudcheck topic.

2. Check the system health of installed products. In the Installation Dashboard, select the Status tab for each service tile. Confirm that all jobs are healthy.

3. (Optional) Check the logs for errors before proceeding with the upgrade. See the Viewing Logs in the Command Line Interface topic.

4. There should be no outstanding changes in Ops Manager or any other tile. All tiles should be green. Click Apply Changes if necessary.

5. After applying changes, click Recent Install Logs to confirm that the changes completed cleanly. `cleanup complete "type": "step\_finished", "id": "clean\_up\_bosh\_cleaning\_up" exited with 0.`
Follow the steps below to upgrade Ops Manager and keep all installed products.

1. From the Product Dashboard, select **Actions > Export installation settings**.

   This exports the current PCF installation with all of its assets. When you export an installation, the export contains the base VM images and necessary packages, and references to the installation IP addresses. As a result, an exported file can be very large, as much as 5 GB or more.
   - The export time depends on the size of the exported file.
   - Some browsers do not provide feedback on the status of the export process, and may appear to hang.

2. Download the latest Ops Manager VM Template from the [Pivotal Network](https://www.pivotalnetwork.com) site.
3. Record the IP address of the existing Ops Manager VM.
4. To avoid IP conflicts, power off the existing Ops Manager VM.
5. Deploy the new Ops Manager VM:
   - [Launching an Ops Manager Director Instance on AWS](https://www.pivotalnetwork.com)
   - [Provisioning the OpenStack Infrastructure](https://www.pivotalnetwork.com)
   - [Deploying Operations Manager to vSphere](https://www.pivotalnetwork.com)
   - [Deploying Operations Manager to vCloud Air and vCloud](https://www.pivotalnetwork.com)
6. When redirected to the **Welcome to Ops Manager** page, select **Import Existing Installation**.

---

**Note:** Ops Manager 1.7 uses the User Account and Authentication (UAA), instead of only local user account authentication. When you import your pre-1.7 file to Ops Manager, your user name changes to `admin` and your password remains the same. It also prompts you to create a shared passphrase, which is distinct from your password. Your passphrase on a new import will be the same as your password. However, you must change the passphrase for security purposes.

**Note:** Some operating systems may automatically unzip the exported installation. If this occurs, create a zip file of the unzipped export. Do not start compressing at the “installation” folder level. Instead, start compressing at the level containing the `config.yml` file.
7. When prompted, enter the following:
   - **Decryption Passphrase**, which is the same as your password.
   - Click **Choose File** and browse to the installation zip file exported in Step 3 above.

8. Click **Import**.

   **Note:** Some browsers do not provide feedback on the status of the import process, and may appear to hang.

9. Before you see the new PCF 1.7 Installation Dashboard, a Security Features alert appears. Take note of your new **username**. Ensure you change your decryption passphrase before sharing it with other users. Click **Continue**.

10. A “Successfully imported installation” message appears upon completion.
11. Click **Apply Changes**. This immediately imports and applies upgrades to all tiles in a single transaction.

12. Click each service tile, select the **Status** tab, and confirm that all VMs appear and are in good health.

13. Remove the original Ops Manager VM from your IaaS if the new installation functions correctly.

**After the Upgrade**

- Advise your application developers about the Diego architecture. See the [Migrating Apps to Diego](#) and [Diego Architecture](#) topics for more information.

- Ops Manager 1.7 defines specific instance types instead of custom sizes for vSphere or vCloud. Each instance adopts an instance type that is the closest match to its previous custom size for CPU, memory, and disk space. After the upgrade, you can modify the automatically selected instance size by selecting a different **type** under **Resource Config** for each of your installed tiles.

- In Ops Manager 1.7, all product tiles use floating stemcells by default. This increases the security of your deployment by enabling tiles to automatically use the latest patched version of a stemcell, but it may significantly increase the amount of time required by a tile upgrade. Review the [Understanding Floating Stemcells](#) topic for more information.

- (Optional) You can review your product tiles and configure values accordingly. For example, for Elastic Runtime, review the installation topic for new features:
  - **Launching an Ops Manager Director Instance on AWS**
  - **Provisioning the OpenStack Infrastructure**
  - **Deploying Operations Manager to vSphere**
  - **Deploying Operations Manager to vCloud Air and vCloud**
Upgrading Elastic Runtime and Other Pivotal Cloud Foundry Products

Page last updated:

Note: Pivotal Cloud Foundry (PCF) for vCloud Air and vCloud Director is deprecated and availability is restricted to existing customers. Contact Support for more information.

This topic describes how to upgrade Elastic Runtime (ERT) and other product tiles alone, without upgrading Ops Manager. For example, you might need to do this if a security update for ERT is released or if new features are introduced in a point release of a product tile.

The topic Upgrading Pivotal Cloud Foundry describes how to perform a complete upgrade of PCF, including Ops Manager, ERT and other installed products.

WARNING: Before you upgrade to Elastic Runtime 1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. All apps that do not already run on Diego will terminate during an upgrade from 1.6 to 1.7.

Note: The apps in your deployment continue to run during the upgrade. However, you cannot write to your deployment or make changes to apps during the upgrade.

Refer to the Product Compatibility Matrix before upgrading Elastic Runtime for Pivotal Cloud Foundry.

Elastic Runtime Snapshot

Current Pivotal Cloud Foundry Elastic Runtime Details

- Version: 1.7.0
- Release Date: 29 April 2016
- Software component version: Cloud Foundry 235
- Compatible Ops Manager Version(s): 1.7.x
  - vSphere support? Yes
  - AWS support? Yes
  - OpenStack support? Yes

Upgrading Elastic Runtime

Refer to the Product Version Matrix before upgrading Elastic Runtime for Pivotal Cloud Foundry.

Note: Before you upgrade to Ops Manager 1.7.x, you must first upgrade PCF Elastic Runtime to version 1.6.9 or higher.

Install using the Pivotal Operations Manager

To install Elastic Runtime for PCF without upgrading Ops Manager, follow the procedure for installing PCF products:

1. Download the product file from Pivotal Network.

2. Upload the product file to your Ops Manager installation.

3. Click Add next to the uploaded product description in the Available Products view to add this product to your staging area.

4. Click the newly added tile to review any configurable options.

5. Click Apply Changes to install the service.

Before You Upgrade

Before you upgrade to Elastic Runtime 1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. Pivotal
does not support DEA architecture in Pivotal Cloud Foundry 1.7. See the Migrating Apps to Diego topic for information.

Upgrading PCF Products

Important: Read the Known Issues section of the Pivotal Cloud Foundry Release Notes before getting started.

This section describes how to upgrade individual products like Pivotal Cloud Foundry Elastic Runtime, Pivotal MySQL, or RabbitMQ in your Pivotal Cloud Foundry (PCF) deployment.

Note: In Ops Manager 1.7, all product tiles use floating stemcells by default. This increases the security of your deployment by enabling tiles to automatically use the latest patched version of a stemcell, but it may significantly increase the amount of time required by a tile upgrade. Review the Understanding Floating Stencells topic for more information.

Note: To upgrade your PCF product to a target release, you must install all releases from your currently deployed version to the target version in sequential order. For example, if your deployment uses Elastic Runtime release 1.5 and you are upgrading to 1.7, you must sequentially install 1.6 and 1.7.

Note: The Single Sign-On service tile operates in lockstep with Pivotal Elastic Runtime.
- The SSO v1.0.x tiles are compatible with PCF v1.6.x
- The SSO v1.1.x tiles are compatible with PCF v1.7.x

If you are upgrading from PCF 1.6 to PCF 1.7 and you are using SSO v1.0.x, you must update SSO v1.1.0 service tile at the same time as you update the ERT upgrade.

1. Browse to Pivotal Network and sign in.

2. Download the latest PCF release for the product or products you want to upgrade. Every product is tied to exactly one stemcell. Download the stemcell that matches your product and version.

3. Confirm that you have adequate disk space for your upgrades. From your Installation Dashboard, select the Ops Manager Director tile. Select Status. If you need more space to handle your upgrades, select Settings > Resource Config. Increase your persistent disk space to 100 GB, or enough to handle the size of the resources.

4. Browse to the Pivotal Cloud Foundry Operations Manager web interface and click Import a Product.

Note: As of release 1.4.0.0, Pivotal Ops Manager no longer supports older versions of PCF products. You must update all products to at least version 1.2 (except for RabbitMQ, which must be updated to at least version 1.3.4.0) before importing them into Ops Manager 1.4.0.0.

5. Upload the new version of a product you want to upgrade.

6. Under Available Products, click Upgrade for the uploaded product.
7. Repeat the import, upload, and upgrade steps for each product you downloaded.

8. If you are upgrading a product that uses a self-signed certificate from version 1.1 to 1.2, you must configure the product to trust the self-signed certificate.
   To do this:
   - Click the product tile.
   - In the left-hand column, select the setting page containing the SSL certificate configuration. For example, for Elastic Runtime, select the HAProxy page.
   - Check the Trust Self-Signed Certificates box.
   - Click Save.

9. Click Apply changes.

After You Upgrade

- Advise your application developers about the Diego architecture. See the Migrating Apps to Diego and the Diego Architecture topics for more information.

- Ops Manager 1.7 defines specific instance types instead of custom sizes for vSphere or vCloud. Each instance adopts an instance type that is the closest match to its previous custom size for CPU, memory, and disk space. You can modify the automatically selected instance size by selecting a different type under Resource Config for each of your installed tiles.
PCF Dev Overview

This guide describes how to install and use PCF Dev, a lightweight Pivotal Cloud Foundry (PCF) installation that runs on a single virtual machine (VM) on your workstation. PCF Dev is intended for application developers who want to develop and debug their applications locally on a PCF deployment.

PCF Dev includes Pivotal Application Service (PAS), Redis, RabbitMQ, and MySQL. It also supports all Cloud Foundry Command Line Interface (cf CLI) functionality. See the Comparing PCF Dev to Pivotal Cloud Foundry table below for more product details.

Prerequisites

- **VirtualBox: 5.0+**: PCF Dev uses VirtualBox as its virtualizer.
- The latest version of the [cf CLI](https://github.com/pivotal-cf/cf): Use the cf CLI to push and scale apps.
- You must have an Internet connection for DNS. See [Using PCF Dev Offline](#) if you do not have an Internet connection.
- At least 3 GB of available memory on your host machine. Pivotal recommends running on a host system with at least 8 GB of total RAM.

Installing PCF Dev

- [Installing PCF Dev on Mac OS X](#)
- [Installing PCF Dev on Linux](#)
- [Installing PCF Dev on Microsoft Windows](#)

Configuring and Using PCF Dev

- [Configuring PCF Dev](#)
- [Using PCF Dev](#)
- [Using Services in PCF Dev](#)
- [Using Spring Cloud Services in PCF Dev](#)
- [Using PCF Dev Behind a Proxy](#)
- [Using PCF Dev Offline](#)
- [PCF Dev on AWS](#)
- [Frequently Asked Questions](#)

Comparing PCF Dev to Pivotal Cloud Foundry

PCF Dev mirrors [PCF](https://www.pivotal.io) in its key product offerings. If an application runs on PCF Dev, it runs on PCF with no modification in almost all cases. Review the table below for key product details.

<table>
<thead>
<tr>
<th></th>
<th>PCF Dev</th>
<th>PCF</th>
<th>CF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space required</strong></td>
<td>20 GB</td>
<td>100GB+</td>
<td>50GB+</td>
</tr>
<tr>
<td><strong>Memory required</strong></td>
<td>3 GB</td>
<td>50GB+</td>
<td>variable</td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td><code>cf dev start</code></td>
<td>Ops Manager</td>
<td><code>bosh create-env</code></td>
</tr>
<tr>
<td><strong>Estimated time-to-deploy</strong></td>
<td>10 Minutes</td>
<td>Hour+</td>
<td>Hour+</td>
</tr>
<tr>
<td><strong>Out-of-the-Box Services</strong></td>
<td>Redis MySQL RabbitMQ</td>
<td>Redis MySQL RabbitMQ GemFire</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>PAS</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Logging/Metrics</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Routing</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature</td>
<td>PCF Dev</td>
<td>PCF</td>
<td>CF</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Compatible with CF CLI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Deploy apps with any supported buildpack</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Supports Multi-Tenancy</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Diego Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Docker Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User-Provided Services</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>High Availability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Integration with 3rd party Authorization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BOSH Director (i.e., can perform additional BOSH deployments)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Day Two Lifecycle Operations (e.g., rolling upgrades, security patches)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ops Manager</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apps Manager</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tile Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Developers have root-level access across cluster</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pre-provisioned</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Does not depend on BOSH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Using Operations Manager

Operations Manager is a web application that you use to deploy and manage a Pivotal Cloud Foundry (PCF) PaaS. This is a guide to deploying and using Ops Manager.

Browser Support

Ops Manager is compatible with current and recent versions of all major browsers. Pivotal recommends using the current version of Chrome, Firefox, or Safari for the best Ops Manager experience.

Operations Manager API

Use the Ops Manager API to automate any Ops Manager task. To view the Ops Manager API documentation, browse to

https://YOUR-OPS-MANAGER-FQDN/docs

Using Operations Manager and Installed Products

- Understanding the Ops Manager Interface
- Adding and Deleting Products
- Understanding Floating Stemcells
- Configuring Ops Manager Director for AWS
- Configuring Amazon EBS Encryption
- Configuring Ops Manager Director for VMware vSphere
- Configuring Ops Manager Director for vCloud Air and vCloud
- Creating UAA Clients for BOSH Director
- Configuring Ops Manager Director for OpenStack
- Deploying Elastic Runtime on AWS
- Configuring Elastic Runtime for vSphere and vCloud
- Installing Elastic Runtime on OpenStack
- Understanding Pivotal Cloud Foundry User Types
- Starting and Stopping Pivotal Cloud Foundry Virtual Machines
- Creating and Managing Ops Manager User Accounts
- Logging into the Apps Manager
- Controlling Apps Manager User Activity with Environment Variables
- Configuring Your App Autoscaling Instance
- Managing Scheduled Scaling in the App Autoscaling Service
- Deleting an AWS Installation from the Console
- Modifying Your Ops Manager Installation and Product Template Files

Back Up

- Backing Up and Restoring Pivotal Cloud Foundry
- Creating a Proxy ELB for Diego SSH without CloudFormation

© Copyright Pivotal Software Inc, 2013-2018 210
Monitoring, Logging, and Troubleshooting

- Monitoring Virtual Machines in Pivotal Cloud Foundry
- Pivotal Cloud Foundry Troubleshooting Guide
- Troubleshooting Ops Manager for VMware vSphere
- Recovering MySQL from Elastic Runtime Downtime
- Advanced Troubleshooting with the BOSH CLI
- Pivotal Cloud Foundry Security Overview and Policy
Understanding the Ops Manager Interface

This topic describes key features of the Pivotal Cloud Foundry (PCF) Operations Manager interface.

PCF Ops Manager

- **A — Available Products**: Displays a list of products you have imported that are ready for installation. If an upgrade is available, an active Upgrade button appears when you hover over the name of the product. Click the Import a Product link to add a new product to Ops Manager. Click Delete unused products to delete any unused products.

- **B — Installation Dashboard**: Displays a product tile for each installed product.

- **C — Actions menu**: Includes the following options:
  - Download activity data: Downloads a directory containing the config file for the installation, the deployment history, and version information.
  - View diagnostic report: Displays various types of information about the configuration of your deployment.
  - Export settings: Exports the current installation with all of its assets. When you export an installation, the exported file contains references to the installation IP addresses. It also contains the base VM images and necessary packages. As a result, an export can be very large (as much as 5 GB or more).
  - Delete this installation: Deletes the current installation with all of its assets.

- **D — User account menu**: Use this menu to change your password or log out.

- **E — Pending Changes view**: Displays queued installations and updates that will install during the next deploy. Click Apply Changes to apply any pending changes to your deployment.

⚠️ Note: When an update depends on prerequisites, the prerequisites automatically install first.

Settings Menu

Navigate to the Settings menu by clicking on your user name located at the upper right corner of the screen.

In the settings menu, you can change the following account information:
Update Decryption Passphrase

Switch Identity Providers by entering your Decryption passphrase, Saml idp metadata, and optionally, your Bosh idp metadata. For more information about setting up your Identity Provider, view the following instructions for your configuration:

- Amazon Web Services
- OpenStack
- vSphere and vCloud

Account Settings

To change your email and password, navigate to Actions menu.
Adding and Deleting Products

Refer to this topic for help adding and deleting additional products from your Pivotal Cloud Foundry (PCF) installation, such as Pivotal RabbitMQ® for PCF.

Note: In Ops Manager 1.7, all product tiles use floating stemcells by default. This increases the security of your deployment by enabling tiles to automatically use the latest patched version of a stemcell, but it may significantly increase the amount of time required by a tile upgrade. Review the Understanding Floating Stemcells topic for more information.

Adding and Importing Products

1. Download PCF-compatible products at Pivotal Network.

2. From the Available Products view, click Import a Product.

3. To import a product, select the .pivotal file that you downloaded from Pivotal Network or received from your software distributor, then click Open. After the import completes, the product appears in the Available Products view.

4. Hover over the product name in the AvailableProducts view to expose the Add button, then click Add.

5. The product tile appears in the Installation Dashboard.

Note: If the product requires configuration, the tile appears orange.

6. If necessary, configure the product.

7. (Optional) In the product configuration view, select the Errands pane to configure post-install errands or review the default settings. Post-install errands are scripts that automatically run after a product installs, before Ops Manager makes the product available for use. For more information about post-install errands, see Understanding Lifecycle Errands.

© Copyright Pivotal Software Inc, 2013-2018
The **Broker Registrar** checkbox is an example of an errand available for a product. When you select this checkbox, this errand registers service brokers with the Cloud Controller and also updates any broker URL and credential values that have changed since the previous registration.

8. In the Pending Changes view, click **Apply Changes** to start installation and run post-install lifecycle errands for the product.

### Deleting a Product

1. From the Installation Dashboard, click the trash icon on a product tile to remove that product. In the **Delete Product** dialog box that appears, click **Confirm**.

   > **Note:** You cannot delete the Ops Manager Director product.

2. In the Pending Changes view, click **Apply Changes**.

   After you delete a product, the product tile is removed from the installation and the Installation Dashboard. However, the product appears in the Available Products view.
Understanding Floating Stemcells

This topic describes how floating stemcells work in Pivotal Cloud Foundry (PCF) v1.7 and later, and the consequences for upgrading product tiles in Ops Manager.

To increase the security of your deployment, all product tiles use floating stemcells by default. This enables tiles to automatically use the latest patched version of a stemcell.

Floating stemcells allow upgrade to the minor versions of stemcells but not the major versions. For example, a stemcell can float from 1234.56 to 1234.99 but not from 1234.991 to 1235.0.

When an operator upgrades a product tile, Ops Manager checks to see whether there is a new minor, or patch, version of the stemcell. If an minor stemcell is available from the current major line, Ops Manager installs the upgraded tile and all compatible product tiles in the deployment on the new stemcell. This ensures that PCF can quickly propagate a patched stemcell to all VMs in the deployment when a vulnerability is discovered.

Operators can now perform certain deployment-wide updates, such as CVEs, by uploading a new stemcell instead of uploading .pivotal files for each tile, which reduces the time spent waiting for files to upload. Operators can upload new stemcells using the Ops Manager API or through a product tile in the Ops Manager Installation Dashboard.

However, operators who want to upgrade a single product tile may face significantly longer wait times, depending on the number of tiles in the deployment and the availability of a new stemcell.
Creating UAA Clients for BOSH Director

This topic describes the process of creating a UAA client for the BOSH Director. You must create an automation client to run BOSH from a script or set up a continuous integration pipeline.

Local Authentication

To perform this procedure, the UAAC client must be installed on the Ops Manager virtual machine (VM).

1. Open a terminal and SSH into the Ops Manager VM. Provide your SSH key, or when prompted, enter the password you configured for SSH access during Ops Manager deployment.

   $ ssh ubuntu@OPS-MANAGER-FQDN
   Password: ************

2. Navigate to the Ops Manager Installation Dashboard and select the Ops Manager Director tile. In Ops Manager Director, click the Status tab, and copy the Ops Manager Director IP address.

3. Using the `uaac target` command, target Ops Manager Director UAA on port `8443` using the IP address you copied, and specify the location of the root certificate. The default location is `/var/tempest/workspaces/default/root_ca_certificate`.


   Target: https://10.85.16.4:8443

   **Note:** You can also curl or point your browser to the following endpoint to obtain the root certificate: https://OPS-MANAGER-FQDN/api/v0/security/root_ca_certificate

4. Log in to the Ops Manager Director UAA and retrieve the owner token. Perform the following step to obtain the values for `UAA-LOGIN-CLIENT-PASSWORD` and `UAA-ADMIN-CLIENT-PASSWORD`:

   - Select the Ops Manager Director tile from the Ops Manager Installation Dashboard.
   - Click the Credentials tab, and locate the entries for Uaa Login Client Credentials and Uaa Admin User Credentials.
   - For each entry, click Link to Credential to obtain the password.

   $ uaac token owner get login -s UAA-LOGIN-CLIENT-PASSWORD
   User name: admin
   Password: UAA-ADMIN-CLIENT-PASSWORD

   Successfully fetched token via owner password grant.
   Target: https://10.85.16.4:8443
   Context: admin, from client login

   **Note:** To obtain the password for the UAA login and admin clients, you can also curl or point your browser to the following endpoints:
   https://OPS-MANAGER-FQDN/api/v0/deployed/director/credentials/uaa_login_client_credentials
   and
   https://OPS-MANAGER-FQDN/api/v0/deployed/director/credentials/uaa_admin_user_credentials

5. Create a new UAA Client with `bosh.admin` privileges.

   $ uaac client add ci --authorized_grant_types client_credentials
   --authorities bosh.admin --secret CI-SECRET

   scope: uaa.none
   client_id: ci
   resource_ids: none
   authorized_grant_types: client_credentials
   autoapprove: none
   authorities: bosh.admin
   name: ci
   lastmodified: 1469727130702
   id: ci

© Copyright Pivotal Software Inc, 2013-2018
6. Set the client and secret as environment variables on the VM.

```
$ ubuntu@ip-10-0-0-12:~$ export BOSH_CLIENT=ci
$ ubuntu@ip-10-0-0-12:~$ export BOSH_CLIENT_SECRET=CI-SECRET
```

7. Target BOSH using the client. Replace `OPS-MANAGER-DIRECTOR-IP` with the IP address of your Ops Manager Director VM.

```
$ bosh --ca-cert /var/tempest/workspaces/default/root_ca_certificate \
  target OPS-MANAGER-DIRECTOR-IP
```

You can now use the UAA client you created to run BOSH in automated or scripted environments, such as continuous integration pipelines.

---

**SAML Authentication to the BOSH Director**

Typically, there is no browser access to a BOSH Director in order to authenticate using SAML. Ops Manager provides an option to create UAA clients during SAML configuration so that BOSH can be automated via scripts and tooling.

1. Select **Provision an admin client in the Bosh UAA** when configuring Ops Manager for SAML.

2. After deploying Ops Manager Director (BOSH), click the Credentials tab in the Ops Manager Director tile.

3. Click the link for the **Uaa Bosh Client Credentials** to get the client name and secret.

4. Open a terminal and SSH into the Ops Manager VM. Provide your SSH key, or, when prompted, enter the password you configured for SSH access during Ops Manager deployment.

```
$ ssh ubuntu@OPS-MANAGER-FQDN
Password: ***********
```

5. Set the client and secret as environment variables on the VM.

```
$ ubuntu@ip-10-0-0-12:~$ export BOSH_CLIENT=bosh_admin_client
$ ubuntu@ip-10-0-0-12:~$ export BOSH_CLIENT_SECRET=CLIENT_SECRET
```

6. Target BOSH using the client. Replace `OPS-MANAGER-DIRECTOR-IP` with the IP address of your Ops Manager Director VM.

```
$ bosh --ca-cert /var/tempest/workspaces/default/root_ca_certificate \
  target OPS-MANAGER-DIRECTOR-IP
```
Understanding Pivotal Cloud Foundry User Types

This topic describes the types of users in a Pivotal Cloud Foundry (PCF) deployment, their roles and permissions, and who creates and manages their user accounts.

The users who run a PCF deployment and have admin privileges are operators. With Elastic Runtime installed to host apps, you add two more user types: Elastic Runtime users who develop the apps and manage the development environment, and end users who just run the apps.

PCF distinguishes between these three user types and multiple user roles that exist within a single user type. Roles are assigned categories that more specifically define functions that a user can perform. A user may serve in more than one role at the same time.

Operators

Operators have the highest, admin-level permissions. We also refer to operators as Ops Manager admins and Elastic Runtime admins because they perform an admin role within these contexts.

Tools and Tasks

Operators fulfill system administrator roles covering the entire PCF deployment. They work primarily with their IaaS and Ops Manager, to configure and maintain Elastic Runtime component VMs. The component VMs, in turn, support the VMs that host applications. Typical operator tasks include:

- Deploying and configuring Ops Manager, Elastic Runtime, and other product and service tiles.
- Maintaining and upgrading PCF deployments.
- Creating user accounts for Elastic Runtime users and the orgs that Elastic Runtime users work within.
- Creating service plans that define the access granted to end users.

User Accounts

When Ops Manager starts up for the first time, the operator specifies one of the following authentication systems for operator user accounts:

- Internal authentication, using a new UAA database that Ops Manager creates.
- External authentication, through an existing identity provider accessed via SAML protocol.

The operator can then use the UAAC to create more operator accounts.

Elastic Runtime Users

Elastic Runtime users are app developers, managers, and auditors who work within orgs and spaces, the virtual compartments within a deployment where Elastic Runtime users can run apps and locally manage their roles and permissions.

A Role-Based Access Control (RBAC) system defines and maintains the different Elastic Runtime user roles:

- Org Manager, Org Auditor, Org Billing Manager
- Space Manager, Space Developer, Space Auditor

The Orgs, Roles, Spaces, Permissions topic describes the Elastic Runtime user roles, and what actions they can take within the orgs and spaces they belong to. Some of these permissions depend on the values of environment variables.

Tools

Space Developer users work with their software development tools and the apps deployed on host VMs.

All Elastic Runtime users use system tools such as the cf CLI, PCF Metrics, and Apps Manager, a dashboard for managing Elastic Runtime users, orgs, spaces, and apps.
**User Accounts**

When an operator configures Elastic Runtime for the first time, they specify one of the following authentication systems for Elastic Runtime user accounts:

1. Internal authentication, using a new UAA database created for Elastic Runtime. This system-wide UAA differs from the Ops Manager internal UAA, which only stores Ops Manager Admin accounts.

2. External authentication, through an existing identity provider accessed via SAML or LDAP protocol.

In either case, Elastic Runtime user role settings are saved internally in the Cloud Controller Database, separate from the internal or external user store.

Org and Space Managers then use Apps Manager to invite and manage additional Elastic Runtime users within their orgs and spaces. Elastic Runtime users with proper permissions can also use the cf CLI to assign user roles.

Operators can log into Apps Manager by using the **UAA Administrator User** credentials under the **Credentials** tab of the Elastic Runtime tile. These UAA Admin credentials grant them the role of Org Manager within all orgs in the deployment. The UAA Admin can also use the UAAC to create new user accounts and the cf CLI to assign user roles.

**End Users**

End users are the people who log into and use the apps hosted on Elastic Runtime. They do not interact directly with Elastic Runtime components or interfaces. Any interactions or roles they perform within the apps are defined by the apps themselves, not Elastic Runtime.

**User Accounts and SSO**

App developers can configure apps any way they want to grant end user access individually. In a deployment with **Single Sign-On Service for Pivotal Cloud Foundry** installed, they can also offer end users a single login that accesses multiple apps.

The Single Sign-On (SSO) service can save user account information in an external database accessed via SAML or LDAP, or in the internal Elastic Runtime user store, along with Elastic Runtime User accounts.

To make the SSO service available to developers, an operator creates service plans that give login access to specific groups of end users. A Space Manager then creates a local instance of the service plan, and registers apps with it. Apps registered to the plan instance then become available via SSO to all end users covered by the plan.

**User Types Summary**

The following table summarizes PCF user types, their roles, the tools they use, the System of Record (SOR) that stores their accounts, and what accounts they can provision.

<table>
<thead>
<tr>
<th>User Type</th>
<th>Available Roles</th>
<th>Tools They Use</th>
<th>Account SOR</th>
<th>Accounts They Can Provision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operators</strong></td>
<td>Admin (UAA Admin, SSO Plan Admin, other system admins)</td>
<td>• IaaS UI</td>
<td>Ops Manager user store via UAA or External store via SAML</td>
<td>Operators and Elastic Runtime Users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PivNet</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ops Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• cf CLI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UAA CL (UAAC)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SSO Dashboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Marketplace</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Elastic Runtime Users</strong></td>
<td>• UAA Administrator</td>
<td>• cf CLI</td>
<td>Elastic Runtime user store via UAA or External store via SAML</td>
<td>Elastic Runtime Users within permitted orgs and spaces, and End Users</td>
</tr>
<tr>
<td></td>
<td>• Org Manager</td>
<td>• CAPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Org Auditor</td>
<td>• Apps Manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Org Billing Manager</td>
<td>• PCF Metrics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Space Manager</td>
<td>• Marketplace</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Space Developer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Space Auditor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End Users</td>
<td>Defined by apps they use</td>
<td>Hosted apps</td>
<td>Individual apps or Elastic Runtime user store via SSO</td>
<td>N/A</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------</td>
<td>-----</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018
221
Starting and Stopping Pivotal Cloud Foundry Virtual Machines

Page last updated:

This topic describes starting and stopping the component virtual machines (VMs) that make up a Pivotal Cloud Foundry (PCF) deployment. This procedure uses the BOSH Command Line Interface (CLI). See Prepare to Use the BOSH CLI for help setting up this tool.

Dependencies between the components in your PCF deployment require that you start and stop the VMs for those components in a specific order. These orders are specified below in the start order and stop order tables.

Note: When you deploy Elastic Runtime, the process automatically starts VMs. These steps are not required to install any part of your deployment.

Finding the Names for Your PCF Virtual Machines

You need the full names for the VMs to start or stop them using the BOSH CLI. To find full names for the VMs running each component, run `bosh vms`:

```
$ bosh vms
Acting as user 'director' on 'p-bosh-399383d452762dc:3522'
Deployment 'cf-1ef2da789c0ed8f3567f'

Director task 26
Task 26 done
```

You can see the full name of each VM in the VM column of the terminal output. Each full name includes:

- A prefix indicating the component function of the VM. The table below associates each component VM function with a prefix.
- The word `partition`
- An identifier string specific to your deployment
- An `INDEX` suffix. For component processes that run on a single VM instance, INDEX is always 0. For processes running on multiple VMs, INDEX is a sequentially numbered value that uniquely identifies each VM.
- An `ID` in parentheses that is the UUID (Universally Unique Identifier) for that instance and can generally be used interchangeably with the index value from the CLI.

For any component, you can look for its prefix in the `bosh vms` output to find the full name of the VM or VMs that run it. In the example shown here, the full name of one of the two Diego Cell VMs is `dieuo_cell-partition-bb35e96d6d3184a2d672/2`.

Starting PCF Virtual Machines
In the order specified in the **Start Order table** below, run `bosh start VM-NAME` for each component in your PCF deployment. Use the full name of the component VM as listed in your `bosh vms` terminal output with the `INDEX` at the end. In the example here, the first component you would start is the NATS VM, by running `bosh start nats-partition-458f9d7042365f1b10e9:0`.

Processing deployment manifest

You are about to start nats-partition-458f9d7042365f1b10e9/0

Detecting deployment changes

Start nats-partition-458f9d7042365f1b10e9/0? (type 'yes' to continue): yes

Performing `start nats-partition-458f9d7042365f1b10e9/0`...

... Started updating job nats-partition-458f9d7042365f1b10e9 > nats-partition-458f9d7042365f1b10e9/0 (canary). Done (00:00:43)

Task 42 done

nats-partition-458f9d7042365f1b10e9/0 has been started

**Note:** To start a specific instance of a VM, include the `INDEX` space delimited, at the end of its full name. In the example here, you could start only the first Diego Cell instance by running: `bosh start diego_cell-partition-458f9d7042365f1b10e9:0`

<table>
<thead>
<tr>
<th>Start Order</th>
<th>Component</th>
<th>Job/index name prefix (in <code>bosh vms</code> output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NATS</td>
<td>nats-</td>
</tr>
<tr>
<td>2</td>
<td>consul</td>
<td>consul_server-</td>
</tr>
<tr>
<td>3</td>
<td>etcd</td>
<td>etcd_server-</td>
</tr>
<tr>
<td>4</td>
<td>Diego Database</td>
<td>diego_database-</td>
</tr>
<tr>
<td>5</td>
<td>NFS Server</td>
<td>nfs_server-</td>
</tr>
<tr>
<td>6</td>
<td>Router</td>
<td>router-</td>
</tr>
<tr>
<td>7</td>
<td>MySQL Proxy</td>
<td>mysql_proxy-</td>
</tr>
<tr>
<td>8</td>
<td>MySQL Server</td>
<td>mysql-</td>
</tr>
<tr>
<td>9</td>
<td>Cloud Controller Database</td>
<td>ccdb-</td>
</tr>
<tr>
<td>10</td>
<td>UAA Database</td>
<td>uaadb-</td>
</tr>
<tr>
<td>11</td>
<td>Apps Manager Database</td>
<td>consoledb-</td>
</tr>
<tr>
<td>12</td>
<td>Cloud Controller</td>
<td>cloud_controller-</td>
</tr>
<tr>
<td>13</td>
<td>HAProxy</td>
<td>ha_proxy-</td>
</tr>
<tr>
<td>14</td>
<td>Health Manager</td>
<td>health_manager-</td>
</tr>
<tr>
<td>15</td>
<td>Clock Global</td>
<td>clock_global-</td>
</tr>
<tr>
<td>16</td>
<td>Cloud Controller Worker</td>
<td>cloud_controller_worker-</td>
</tr>
<tr>
<td>17</td>
<td>Collector</td>
<td>collector-</td>
</tr>
<tr>
<td>18</td>
<td>UAA</td>
<td>uaa-</td>
</tr>
<tr>
<td>19</td>
<td>Diego Brain</td>
<td>diego_brain-</td>
</tr>
<tr>
<td>20</td>
<td>Diego Cell</td>
<td>diego_cell-</td>
</tr>
<tr>
<td>21</td>
<td>Doppler Server</td>
<td>doppler-</td>
</tr>
<tr>
<td>22</td>
<td>Loggregator Traffic Controller</td>
<td>loggregator_trafficcontroller-</td>
</tr>
</tbody>
</table>

**Stopping PCF Virtual Machines**

In the order specified in the **Stop Order table** below, run `bosh stop VM-NAME INDEX-OR-ID --hard` for each component in your PCF deployment. Use the full name of the component VM as listed in your `bosh vms` terminal output with the `INDEX` at the end. In the example here, you could stop only the first Diego Cell instance by running: `bosh stop diego_cell-partition-458f9d7042365f1b10e9:0`.

Processing deployment manifest

You are about to stop diego_cell-partition-458f9d7042365f1b10e9/0

Detecting deployment changes

Stop diego_cell-partition-458f9d7042365f1b10e9/0? (type 'yes' to continue): yes

Performing `stop diego_cell-partition-458f9d7042365f1b10e9/0`...

... Started updating job diego_cell-partition-458f9d7042365f1b10e9 > diego_cell-partition-458f9d7042365f1b10e9/0 (canary). Done (00:00:43)

Task 42 done

diego_cell-partition-458f9d7042365f1b10e9/0 has been stopped
name of the component VM as listed in your `bosh vms` terminal output, with the INDEX, space delimited, at the end. In the example here, the first component you would stop is the Loggregator Traffic Controller VM, by running:

```
bosh stop loggregator_trafficcontroller-partition-458f9d7042365f6810e9 0 --hard
```

<table>
<thead>
<tr>
<th>Stop Order</th>
<th>Component</th>
<th>Job/index name prefix (in <code>bosh vms</code> output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loggregator Traffic Controller</td>
<td>loggregator_trafficcontroller-</td>
</tr>
<tr>
<td>2</td>
<td>Doppler Server</td>
<td>doppler-</td>
</tr>
<tr>
<td>4</td>
<td>Diego Cell</td>
<td>diego_cell-</td>
</tr>
<tr>
<td>5</td>
<td>Diego Brain</td>
<td>diego_brain-</td>
</tr>
<tr>
<td>6</td>
<td>UAA</td>
<td>uaa-</td>
</tr>
<tr>
<td>7</td>
<td>Collector</td>
<td>collector-</td>
</tr>
<tr>
<td>8</td>
<td>Cloud Controller Worker</td>
<td>cloud_controller_worker-</td>
</tr>
<tr>
<td>9</td>
<td>Clock Global</td>
<td>clock_global-</td>
</tr>
<tr>
<td>10</td>
<td>Health Manager</td>
<td>health_manager-</td>
</tr>
<tr>
<td>11</td>
<td>HAPProxy</td>
<td>ha_proxy-</td>
</tr>
<tr>
<td>12</td>
<td>Cloud Controller</td>
<td>cloud_controller-</td>
</tr>
<tr>
<td>13</td>
<td>Apps Manager Database</td>
<td>consoledb-</td>
</tr>
<tr>
<td>14</td>
<td>UAA Database</td>
<td>uaadb-</td>
</tr>
<tr>
<td>15</td>
<td>Cloud Controller Database</td>
<td>ccdb-</td>
</tr>
<tr>
<td>16</td>
<td>MySQL Server</td>
<td>mysql-</td>
</tr>
<tr>
<td>17</td>
<td>MySQL Proxy</td>
<td>mysql_proxy-</td>
</tr>
<tr>
<td>18</td>
<td>Router</td>
<td>router-</td>
</tr>
<tr>
<td>19</td>
<td>NFS Server</td>
<td>nfs_server-</td>
</tr>
<tr>
<td>20</td>
<td>Diego Database</td>
<td>diego_database-</td>
</tr>
<tr>
<td>21</td>
<td>etcd</td>
<td>etcd_server-</td>
</tr>
<tr>
<td>22</td>
<td>consul</td>
<td>consul_server-</td>
</tr>
<tr>
<td>23</td>
<td>NATS</td>
<td>nats-</td>
</tr>
</tbody>
</table>

**Note:** To stop a specific instance of a VM, include the INDEX, space delimited, at the end of its full name. In the example here, you could stop only the second Diego Cell instance by running:

```
bosh stop diego_cell-partition-bb35e966d311842a2d672 1 --hard
```
Creating and Managing Ops Manager User Accounts

Page last updated:

Pivotal Cloud Foundry supports multiple user accounts in Ops Manager. A User Account and Authentication (UAA) module co-located on the Ops Manager VM manages access permissions to Ops Manager.

When Ops Manager boots for the first time, you create an admin user. However, you do not create additional users through the Ops Manager web interface. If you want to create additional users who can log into Ops Manager, you must use the UAA API, either through curl or the UAA Command Line Client (UAAC).

Follow these steps to add or remove users via the UAAC. If you do not already have the UAAC installed, run `gem install cf-uaac` from a terminal window.

### Adding Users to Ops Manager

1. Target your Ops Manager UAA:

   ```
   $ uaac target https://YOUR-OPSMAN-FQDN/uaa/
   ```

2. Get your token:

   ```
   $ uaac token owner get
   Client ID: opsman
   Client Secret: [Press Enter]
   Username: Admin
   Password: *******
   
   Successfully fetched token via client credentials grant.
   Target https://YOUR-OPSMAN-FQDN/uaa/
   ```

3. Add a user:

   ```
   $ uaac user add YOUR-USER-NAME -p YOUR-USER-PASSWORD -e YOUR-USER-EMAIL@EXAMPLE.COM
   ```

### Removing Users from Ops Manager

1. Target your Ops Manager UAA:

   ```
   $ uaac target https://YOUR-OPSMAN-FQDN/uaa/
   ```

2. Get your token:

   ```
   $ uaac token owner get
   Client ID: opsman
   Client Secret: [Press Enter]
   Username: Admin
   Password: *******
   
   Successfully fetched token via client credentials grant.
   Target https://YOUR-OPSMAN-FQDN/uaa/
   ```

3. Delete a user:

   ```
   $ uaac user delete YOUR-USER-NAME
   ```

Note: You can only manage users on the Ops Manager UAA module if you chose to use Internal Authentication instead of an external Identity Provider when configuring Ops Manager.
Logging in to Apps Manager

Log in as Admin User

Complete the following steps to log in to Apps Manager as the Admin user:

1. If you do not know the system domain for the deployment, then select Pivotal Elastic Runtime > Settings > Domains to locate the configured system domain.

2. Open a browser and navigate to apps.YOUR-SYSTEM-DOMAIN. For example, if the system domain is system.example.com, then point your browser to apps.system.example.com.

3. Log in using UAA credentials for the Admin user. To obtain these credentials, refer to Pivotal Elastic Runtime > Credentials > UAA > Admin Credentials.

4. After you log in, Apps Manager appears.
Controlling Apps Manager User Activity with Environment Variables

This topic describes three environment variables that you can use to manage user interactions with Pivotal Cloud Foundry Apps Manager, and how to set these variables using the cf CLI.

Understanding Apps Manager Environment Variables

The following environment variables control which users can create Orgs and perform user management actions on Apps Manager.

**ENABLE_INTERNAL_USER_STORE**

This variable defaults to `false`, which disables the internal user store. This means that an admin must designate an external LDAP / AD user store as the source of user accounts.

Set this variable to `true` to let users register for new accounts, manage their account and password, and invite new members into their Orgs by clicking an "Invite New Members" link. This setting enables an internal user store within a Pivotal Cloud Foundry (PCF) installation's own local User Account and Authentication (UAA) Server. With the internal store enabled, Cloud Foundry admins do not need to configure an external user store, such as an LDAP / AD server, to create user accounts.

**ENABLE_NON_ADMIN_ORG_CREATION**

This variable defaults to `false`, which prevents non-admin users from being able to create Orgs.

Set this variable to `true` to allow any user to create an Org. This setting activates two links on the user's Org Dashboard:

- The **Create a New Org** link in the drop-down menu in the left navigation panel
- The **Create Org** link in the **MY ORGS** section of the **My Account** page, which you access from the user name drop-down menu

**ENABLE_NON_ADMIN_ROLE_MANAGEMENT**

This variable defaults to `false`, which prevents users with the Space Manager role from managing existing users and roles within their Spaces. This allows admins to centrally manage users and roles.

Set this variable to `true` to allow users with the Space Manager role to manage existing users and roles within their Spaces.

**Note:** The previous `ENABLE_NON_ADMIN_USER_MANAGEMENT` environment variable that controlled many of the above features was deprecated in Pivotal Elastic Runtime 1.6.

Changing an Environment Variable Value

**Note:** To run the commands discussed in this section, you must log in to the cf CLI with your UAA Administrator user credentials. In Pivotal Operations Manager, refer to **Elastic Runtime > Credentials** for the UAA admin name and password.

To change an environment variable value:

1. In a terminal window, set your target API to your Apps Manager URL by running `cf api API-URL`, where API-URL is the URL of the Cloud Controller in your Elastic Runtime instance.
2. Run `cf login` to log in as an admin. When prompted, provide your UAA Administrator user credentials. When prompted further, choose `system` as your org and `system` as your space.

   **Note:** If you are already logged in with UAA Administrator user credentials, switch to the correct org and space by running `cf target -o system -s system`. You do not need to log in again.

3. Set an environment variable for either `apps-manager-blue` or `apps-manager-green`, depending on which is currently live. To learn more about how Apps Manager uses blue-green deployment to reduce downtime, review [Using Blue-Green Deployment to Reduce Downtime and Risk](#). For example, to let users self-manage their org and space roles as described above, run the following command to set an environment variable for your live Apps Manager.

   ```bash
cf set-env apps-manager-blue ENABLE_NON_ADMIN_ROLE_MANAGEMENT true
   ```

4. Reinitialize your live Apps Manager with the new environment variable value.

   ```bash
cf restart apps-manager-blue
   ```
Configuring Your App Autoscaling Instance

Page last updated:

The App Autoscaling service scales bound applications in response to load.

An instance of the App Autoscaling service examines the CPU usage of an application bound to it every few minutes. In response to load changes, the service scales your app up and down according to the thresholds, minimums, and maximums that you provide.

Follow the steps below to configure your App Autoscaling service instance.

1. Log in to the Apps Manager: Logging into the Apps Manager

2. Select a space containing an App Autoscaling service instance from the org dashboard or from the left navigation bar.

3. In the Services section of the space dashboard, under your App Autoscaling service instance name, click Manage.

4. By default, new autoscaling instances are paused. If paused, click Turn On to enable monitoring of your application and start the autoscaling process.

5. Click the wrench icon on your Autoscaling dashboard.

Note: You must specifically have the role of Space Developer to access the Manage link for the app autoscaling service. Space Managers, Space Auditors, and all Org roles do not have the permission to make changes to App Autoscaling. See Managing User Accounts and Permissions Using the Apps Manager for help managing user roles.
6. Change the configuration settings and click **Save**. See the **Configuration Options** section of this topic for information about the configuration settings.

7. Examine the App Autoscaling service instance dashboard to confirm your changes.
Configuration Options

You can set the absolute maximum and minimum number of instances for your app, as well as the CPU thresholds for an app that trigger the autoscaling service.

Instance Counts

The `Instances` values specify the absolute minimum and maximum number of instances autoscaling can set for an application.

- **Min**: Default value: 2. The minimum number of instances to which autoscaling can scale your app. Autoscaling never scales your application below this number of instances.
- **Max**: Default value: 5. The maximum number of instances to which autoscaling can scale your app. Autoscaling never scales your application above this number of instances.

Note: **Min** and **Max** cannot be set to less than 1 or greater than 20. **Min** must be less than or equal to **Max**.

CPU Thresholds

The `CPU thresholds` values specify the upper and lower limits of CPU utilization that trigger the autoscaling service.

The autoscaling service calculates CPU utilization as a moving average across the CPUs of all currently running instances of an application.

- **Low**: Default value: 20. When the autoscaling service instance detects CPU utilization below this threshold, it reduces the number of instances of the app by one.
- **High**: Default value: 80. When the autoscaling service instance detects CPU utilization above below this threshold, it increases the number of instances of the app by one.

Manual Scaling

If you manually scale an application bound to an autoscaling service instance, the autoscaling service stops monitoring and autoscaling your application.

To re-enable monitoring and scaling, click **Turn On** on the App Autoscaling service instance dashboard.
Autoscaling for my-app is currently paused

TURN ON
Managing Scheduled Scaling in the App Autoscaling Service

Follow the steps below to manage your App Autoscaling service instance.

1. Log in to the Apps Manager: Logging into the Apps Manager

2. Select a space containing an App Autoscaling service instance from the org dashboard or from the left navigation bar.

3. In the Services section of the space dashboard, under your App Autoscaling service instance name, click Manage.

   ![Manage link](image)

   Note: To access the Manage link for the app autoscaling service, you must have the role of Space Developer. See Managing User Accounts and Permissions Using the Apps Manager for help managing user roles.

4. By default, new autoscaling instances are paused. If paused, click Turn On to enable monitoring of your application and start the autoscaling process.

   ![Turn On button](image)

5. Click the clock icon on your Autoscaling dashboard.

   ![Clock icon](image)

6. In the Scheduling interface, create a new rule by editing the date and time fields and choosing values for the number of minimum and maximum instances. When finished, click Save. See the Rule Types section of this topic for more information.
7. After saving, the left side of the Scheduling interfaces shows your rule. Click your rule to edit it.

8. Edit your existing rule and click **Save** to save your changes.

9. In the left pane of the Scheduling interfaces, click the X for a rule to delete it.
10. Close the Scheduling interfaces to return to your Autoscaling dashboard. The Scheduling section of the Autoscaling dashboard displays the next occurring rule and summary information about your rules.

Rule Types

Scheduled scaling rules affect the minimum and maximum instance count values for your application. When the autoscaling service runs a scheduled scaling rule, it changes the Min and Max values of the instance count of your application to the values specified in the rule.

One-time Rules

The autoscaling service runs a one-time scheduled scaling rule once only. After running a one-time scheduled scaling rule, the service removes the rule from the list of existing rules.

Note: You must schedule one-time rules to occur at a time in the future.

Recurring Rules
The autoscaling service runs a recurring scheduled scaling rule on a regular basis. You select one or more days of the week for a rule, and the autoscaling service runs the rule on those days every week.

Click **pause** for a particular rule to stop the autoscaling service from running that rule. Click **play** to resume running that rule.

**Note:** The autoscaling service does not run a recurring rule for the first time until the date specified in the rule.

**Scheduling Example**

- The rule shown in the image below recurs every Monday and Friday at 4AM, starting on Friday, November 28, 2014. This rule changes the minimum number of instances of the app to 10 and the maximum to 20.

![Scheduling Example Image](image1.png)

- The rule shown in the image below recurs every Wednesday at 4AM, starting on Friday, November 28, 2014. This rule changes the minimum number of instances of the app to 1 and the maximum to 3.

![Scheduling Example Image](image2.png)

Based on the two rules above, starting on Friday, November 28, 2014, the autoscaling service scales the minimum and maximum instance counts for the application as follows:

- Every Monday, the autoscaling service scales the minimum up to 10 and the maximum to 20.
- Every Wednesday, the autoscaling service scales the minimum down to 1 and the maximum to 3.
- Every Friday, the autoscaling service scales the minimum back up to 10 and the maximum to 20.
Modifying Your Ops Manager Installation and Product Template Files

Page last updated:

This topic describes how to modify your Ops Manager installation by decrypting and editing the YAML files that Ops Manager uses to store configuration data. Operators can use these procedures to view and change values that they cannot access through the Ops Manager web interface. They can also modify the product templates that Ops Manager uses to create forms and obtain user input.

Operators may want to modify the Ops Manager installation and product template files for a number of reasons, including the following:

- To change the User Account and Authentication (UAA) admin password of their deployment
- To retrieve key values
- To migrate content across different Pivotal Cloud Foundry (PCF) releases

**WARNING:** Be careful when making changes to your Ops Manager installation and product template files. Use spaces instead of tabs, and remember that YAML files use whitespace as a delimiter. Finally, Pivotal does not officially support these procedures, so use them at your own risk.

Understand Installation and Product Template Files

During the installation process, Ops Manager combines information from the installation and product template files to generate the manifests that define your deployment.

- **Installation file:** PCF stores user-entered data and automatically generated values for Ops Manager in an installation YAML file on the Ops Manager virtual machine (VM). PCF encrypts and stores this file in the directory
  /var/tempest/workspaces/default. You must decrypt this file to view the contents, edit them as necessary, then re-encrypt them.

- **Product templates:** Ops Manager uses product templates to create forms and obtain user input. The job_types and property_blueprint key-value pairs in a product template determine how the jobs and properties sections display in the installation file. Ops Manager stores product templates as YAML files in the directory
  /var/tempest/workspaces/default/metadata on the Ops Manager VM. These files are not encrypted, so you can edit them without decrypting. User input does not alter these files.

**Note:** Upgrading Ops Manager may eliminate your changes to the installation and product template files.

Modify the Installation File

Perform the following steps to locate, decrypt, and edit your Ops Manager installation file:

1. SSH into the Ops Manager VM by following the steps in the SSH into Ops Manager section of the Advanced Troubleshooting with the BOSH CLI topic.

2. cd into the scripts directory:

   ```
   $ cd /home/tempest-web/tempest/web/scripts/
   ```

3. Run the following command to decrypt the installation YAML file and make a temporary copy of the decrypted file. When prompted for a passphrase, enter the decryption passphrase you created when you launched Ops Manager for the first time:

   ```
   $ sudo -u tempest-web /decrypt /var/tempest/workspaces/default/installation.yml /tmp/installation.yml
   ```

4. Open /tmp/installation.yml to view or edit values.

5. If you plan to make changes, make a backup of the original installation YAML file:

   ```
   $ cp /var/tempest/workspaces/default/installation.yml ~/installation-orig.yml
   ```

6. If you have made changes to your copy of the installation YAML file, you must encrypt it and overwrite the original with it:

   ```
   ```

© Copyright Pivotal Software Inc, 2013-2018
7. Delete the temporary copy of the decrypted file:

   ```
   rm /tmp/installation.yml
   ```

8. Restart the Ops Manager web interface:

   ```
   sudo service tempest-web stop && sudo service tempest-web start
   ```

9. Navigate to Ops Manager in a browser and enter your decryption passphrase.

10. Log in to Ops Manager and click **Apply Changes**.

11. If Ops Manager cannot load your changes, see the [Revert To Your Backup](#) section of this topic to restore your previous settings.

---

## Modify Product Template Files

Perform the following steps to locate and edit your Ops Manager product template files:

1. SSH into the Ops Manager VM by following the steps in the [SSH into Ops Manager](#) section of the Advanced Troubleshooting with the BOSH CLI topic.

2. On the Ops Manager VM, navigate to the `/var/tempest/workspaces/default/metadata` directory.

   ```
   cd /var/tempest/workspaces/default/metadata
   ```

3. The `/var/tempest/workspaces/default/metadata` directory contains the product templates as YAML files. If you plan to make changes, make a backup of the original product template YAML file:

   ```
   cp /var/tempest/workspace/default/metadata/YOUR-PRODUCT-TEMPLATE.yml ~/YOUR-PRODUCT-TEMPLATE-orig.yml
   ```

4. Open and edit the product template YAML file as necessary. For more information about product templates, see the [Product Template Reference](#) topic.

5. Navigate to Ops Manager to see your changes.

6. If Ops Manager cannot load your changes, see the [Revert To Your Backup](#) section of this topic to restore your previous settings.

---

## Revert to Your Backup

Perform the following steps to revert to your backup of an installation or product template file:

1. SSH into the Ops Manager VM by following the steps in the [SSH into Ops Manager](#) section of the Advanced Troubleshooting with the BOSH CLI topic.

2. Overwrite the modified file with the backup:
   - For the installation file, run the following command:
     ```
     cp ~/installation-orig.yml /var/tempest/workspaces/default/installation.yml
     ```
   - For a product template file, run the following command:
     ```
     cp ~/YOUR-PRODUCT-TEMPLATE-orig.yml /var/tempest/workspaces/default/metadata/YOUR-PRODUCT-TEMPLATE.yml
     ```

3. Restart the Ops Manager web interface:

   ```
   sudo service tempest-web stop && sudo service tempest-web start
   ```

4. Navigate to Ops Manager in a browser and enter your decryption passphrase.

5. Log in to Ops Manager and click **Apply Changes**.
Back up and Restore Pivotal Cloud Foundry

This section explains how to back up and restore critical backend components. The exact procedures you follow depend on where your deployment stores its backend data. Consider the following points:

- If your deployment uses external databases (for example, AWS RDS) then you must back up your data according to the instructions provided by your database provider.
- If your PCF deployment originated from 1.5.x or earlier, follow the backup/restore instructions for PostgreSQL databases and for the MySQL server.
- If your PCF deployment originated from 1.6.0 or later, follow the backup/restore instructions for the MySQL server (but not those for PostgreSQL databases).
- If you do not know the original version of your PCF deployment, perform the following steps to determine what databases your deployment uses:
  1. From the Ops Manager Installation Dashboard, click **Pivotal Elastic Runtime**.
  2. Click **Databases** to determine whether your deployment uses internal databases with MySQL and PostgreSQL, internal databases with MySQL only, or external databases.

To create a backup, see the **Backing Up Pivotal Cloud Foundry** topic.

To restore a backup, see the **Restoring Pivotal Cloud Foundry from Backup** topic.
Backing Up Pivotal Cloud Foundry

Page last updated:

This topic describes the procedure for backing up each critical backend Elastic Runtime component. Pivotal recommends frequently backing up your installation settings before making any changes to your PCF deployment, such as configuration of any tiles in Ops Manager.

To back up a deployment, export installation settings, download the BOSH Deployment Manifest, temporarily stop the Cloud Controller, create and export backup files for each critical backend component, and restart the Cloud Controller. It is also important to record your Cloud Controller Database encryption credentials which you will need if you contact Pivotal Support for help restoring your installation.

To restore your backup, see the Restoring Pivotal Cloud Foundry from Backup topic.

Record the Cloud Controller Database Encryption Credentials

From the Installation Dashboard, select Pivotal Elastic Runtime > Credentials and locate the Cloud Controller section. Record the Cloud Controller DB Encryption Credentials. You must provide these credentials if you contact Pivotal Support for help restoring your installation.

Export Installation Settings

Pivotal recommends that you back up your installation settings by exporting frequently. Always export an installation before following the steps in the Import Installation Settings section of the Restoring Pivotal Cloud Foundry from Backup topic.

Note: Exporting your installation only backs up your installation settings. It does not back up your virtual machines (VMs) or any external MySQL databases.

From the Installation Dashboard in the Ops Manager interface, click the gear icon and select Export installation settings. This option is only available after you have deployed at least one time.

Export installation settings exports the current PCF installation settings and assets. When you export an installation, the exported file contains the base VM images, all necessary packages, and references to the installation IP addresses. As a result, an exported installation file can exceed 5 GB in size.
Target the BOSH Director

1. Install Ruby and the BOSH CLI Ruby gem on a machine outside of your PCF deployment.

2. From the Installation Dashboard in Ops Manager, select Ops Manager Director > Status and record the IP address listed for the Director. You access the BOSH Director using this IP address.

3. Click Credentials and record the Director credentials.

4. From the command line, run `bosh target` to log into the BOSH Director using the IP address and credentials that you recorded:

   ```
   $ bosh target 192.0.2.3
   Target set to 'microbosh-1234abcd1234abcd1234'
   Your username: director
   Enter password: ****************
   Logged in as 'director'
   ```

   **Note:** If `bosh target` does not prompt you for your username and password, run `bosh login`.

Download BOSH Manifest

1. Run `bosh deployments` to identify the name of your current BOSH deployment:
2. Run `bosh download manifest DEPLOYMENT-NAME LOCAL-SAVE-NAME` to download and save each BOSH deployment manifest. You need this manifest to locate information about your databases. For each manifest, you will need to repeat these instructions. Replace `DEPLOYMENT-NAME` with the name of the current BOSH deployment. For this procedure, use `cf.yml` as the `LOCAL-SAVE-NAME`.

```
$ bosh download manifest cf-example cf.yml
Deployment manifest saved to: `cf.yml`
```

## Back Up Critical Backend Components

Your Elastic Runtime deployment contains several critical data stores that must be present for a complete restore. This section describes the procedure for backing up the databases and the servers associated with your PCF installation. For more information about which data stores you might be using, see [Back up and restoring PCF](#).

If you are running PostgreSQL and are on the default internal databases, follow the instructions below for backing up the Cloud Controller database, the UAA database, the Apps Manager database, MySQL server, and the NFS server.

If you are running your databases on the internal MySQL database only, follow the instructions below for backing up the MySQL server and the NFS server.

If you are running your databases or filestores externally, disregard instructions for backing up the Cloud Controller, UAA, and Apps Manager Databases and ensure that you back up your external databases and filestores, as well as the NFS server.

**Note:** To follow the backup instructions below, your network must be configured to allow access to the BOSH Director VM from your local machine. If you do not have local administrator access, use the `scp` command to copy the TAR file to the BOSH Director VM. For example:

```
scp vcap@192.0.2.10:nfs.tar.gz

and

scp vcap@192.0.2.3:/nfs.tar.gz
```

### Stop Cloud Controller

1. From a command line, run `bosh deployment DEPLOYMENT-MANIFEST` to select your PCF deployment. The manifest is located in `/var/tempest/workspaces/default/deployments/` on the Ops Manager VM. For example:

```
$ bosh deployment /var/tempest/workspaces/default/deployments/cf-bd784.yml
Deployment set to: `/var/tempest/workspaces/default/deployments/cf-bd784.yml`
```

2. Run `bosh vms CF-DEPLOYMENT-NAME` to view a list of VMs in your PCF deployment. `CF-DEPLOYMENT-NAME` corresponds to the name of your PCF release deployment, which is also the filename of your manifest file without the `.yml` ending. For example:

```
$ bosh vms cf-bd784

+-----------------+----------------+-----------------+-------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+
|-----------------+----------------+----------------+-------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------|
|-----------------+----------------+----------------+-------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------+----------------+-------------|
```

3. Perform the following steps for each Cloud Controller VM, excluding the Cloud Controller Database VM:

   a. SSH onto the VM:

```
$ bosh ssh JOB-NAME
```
b. From the VM, list the running processes:

```bash
$ monit summary
```

c. Stop all processes that start with `cloud_controller_`:

```bash
$ monit stop PROCESS-NAME
```

---

**Back Up the Cloud Controller Database**

**Note:** Follow these instructions only if you are using a PostgreSQL database.

1. In the BOSH deployment manifest, locate the Cloud Controller database (CCDB) component under the `ccdb` key and record the IP address:

```yaml
ccdb:
  address: 192.0.2.96
  port: 2544
  db_scheme: postgres
```

2. From the Installation Dashboard in Ops Manager, select Elastic Runtime and click Credentials > Link to Credential. Record the Cloud Controller database VM credentials.

<table>
<thead>
<tr>
<th>Cloud Controller Database (Postgres)</th>
<th>VM Credentials</th>
<th>Link to Credential</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. SSH into the Cloud Controller database VM as the admin using the IP address and password recorded in the previous steps.

```bash
$ ssh vcap@192.0.2.96
Password:***********
```

4. Run `find /var/vcap | grep ‘bin/pg_dump’` to find the locally installed psql client on the CCDB VM. For example:

```bash
$ sudo vcap sudo find /var/vcap | grep 'bin/pg_dump'
/var/vcap/data/packages/postgres/5.1/bin/pg_dump
```

5. Run `pg_dump` from the locally installed psql client to export the database:

```bash
$ /var/vcap/data/packages/postgres/5.1/bin/pg_dump -h 192.0.2.96 -U admin -p 2544 ccdb > ccdb.sql
```

6. Exit from the Cloud Controller database VM.

7. Run `scp` to copy the exported database to your local machine.

```bash
$ scp vcap@192.0.2.96:~/ccdb.sql .
```

---

**Back Up the UAA Database**

**Note:** Follow these instructions only if you are using a PostgreSQL database.

1. In the BOSH deployment manifest, locate the `uaadb` component and record the IP address:

```yaml
uaadb:
  address: 192.0.2.101
  port: 2544
  db_scheme: postgresql
```

2. From the Installation Dashboard in Ops Manager, select Elastic Runtime and click Credentials > Link to Credential. Record the UAA database VM credentials.
3. SSH into the UAA database VM as the admin using the IP address and password recorded in the previous steps.

4. Run `find /var/vcap | grep 'bin/pg_dump'` to find the locally installed psql client on the UAA database VM.

```
$ find /var/vcap | grep 'bin/pg_dump'
/var/vcap/data/packages/postgres/5.1/bin/pg_dump
```

5. Run `pg_dump` from the locally installed psql client to export the database:

```
$ /var/vcap/data/packages/postgres/5.1/bin/pg_dump -h 192.0.2.101 -U root -p 2544 uaa > uaa.sql
```

6. Exit from the UAA database VM.

7. Run `scp` to copy the exported database to your local machine.

```
$ scp vcap@192.0.2.101:~/uaa.sql
```

---

**Back Up the Apps Manager Database**

**Note:** Follow these instructions only if you are using a PostgreSQL database.

1. In the BOSH deployment manifest, `cf.yml`, locate the `databases` component and record the IP address and password:

   databases:
   address: 192.0.2.104
   port: 2544
   db_scheme: postgresql
   roles:
   - tag: admin
     name: root
     password: ****************
   databases:
   - tag: console
     name: console

2. From the Installation Dashboard in Ops Manager, select Elastic Runtime and click Credentials > Link to Credential. Record the Apps Manager database VM credentials.

<table>
<thead>
<tr>
<th>Apps Manager Database (Postgres)</th>
<th>VM Credentials</th>
<th>Link to Credential</th>
</tr>
</thead>
</table>

3. SSH into the Apps Manager database VM as the admin using the IP address and password recorded in the previous steps.

4. Run `find /var/vcap | grep 'bin/pg_dump'` to find the locally installed psql client on the Apps Manager database VM.

```
$ find /var/vcap | grep 'bin/pg_dump'
/var/vcap/data/packages/postgres/5.1/bin/pg_dump
```

5. Run `pg_dump` from the locally installed psql client to export the database:

```
$ /var/vcap/data/packages/postgres/5.1/bin/pg_dump -h 192.0.2.104 -U root -p 2544 console > console.sql
```

6. Exit from the Apps Manager database VM.

7. Run `scp` to copy the exported database to your local machine.

```
$ scp vcap@192.0.2.104:~/console.sql
```
Back Up NFS Server

1. In the BOSH deployment manifest, locate the `nfs_server` component and record the address:

```yaml
nfs_server:
  address: 192.0.2.10
  network: 192.0.2.0/24
  syslog_aggregator:
    address: port:
```

2. From the Installation Dashboard in Ops Manager, select Elastic Runtime and click Credentials > Link to Credential. Record the NFS Server VM credentials.

3. SSH into the NFS server VM and create a TAR file:

   ```bash
   $ ssh vcap@192.0.2.10 'cd /var/vcap/store && tar cz shared' > nfs.tar.gz
   ```

   **Note:** The TAR file that you create to back up NFS server might be large. To estimate the size of the TAR file before you create it, run the following command:
   ```bash
   ssh vcap@192.0.2.10 'tar -cf - /dir/to/archive/ | wc -c'
   ```

---

Back Up Pivotal MySQL Server

**Note:** The Elastic Runtime deploy contains an embedded MySQL Server that serves as the data store for the Application Usage Events, Notifications, and Autoscaler services. If you are not using PostgreSQL, and in Elastic Runtime tile>Database you selected to use an Internal Database - MySQL, then the MySQL server also hosts your databases for Cloud Controller, UAA, and Apps Manager.

There are two ways to backup the MySQL Server:

- **Manual backup:** If you have not set up automatic backups, you need to do a manual backup of your MySQL server.
- **Automatic backup:** If you set automatic backup in your ERT configuration, you do not need to manually backup your MySQL Server. Automatic backup requires S3-compatible blobstores. For more information, see:
  - AWS: Configure Internal MySQL
  - OpenStack: Configure Internal MySQL
  - vSphere Configure Internal MySQL

**Backing up MySQL Server Manually**

**Note:** The procedure to manually backup MySQL below does not cover steps that are specific to a particular IaaS. It is important to tailor these instructions based on your infrastructure and configuration requirements.

**Prerequisites**
1. Spin up a virtual machine (VM) to use for your MySQL backup and restore. Ensure the VM has sufficient disk space to receive the data dump. Configure network access to your p-mysql deployment.

2. Install the `mysqldump` client of your choice.

3. In a terminal window, execute the following command. To backup all databases in the MySQL deployment, use `--all-databases`:

   ```
   $ mysqldump -u root -p -h $MYSQL_NODE_IP --all-databases > user_databases.sql
   ```

   To backup a single database, specify the database name:

   ```
   $ mysqldump -u root -p -h $MYSQL_NODE_IP $DB_NAME > user_databases.sql
   ```

   **Note:** The backup data you obtain from running the command above is not encrypted or compressed. Pivotal recommends encrypting the backup artifact, before storing it off-site.

4. Store the backup in a different location than the MySQL primary storage, preferably off-site.

**Start Cloud Controller**

1. Run `bosh vms` to view a list of VMs in your selected deployment. The names of the Cloud Controller VMs begin with `cloud_controller`. 
2. Perform the following steps for each Cloud Controller VM, excluding the Cloud Controller Database VM:

   a. SSH onto the VM:

   ```bash
   $ bosh ssh JOB-NAME
   ```

   b. From the VM, list the running processes:

   ```bash
   $ monit summary
   ```

   c. Start all processes that start with `cloud_controller_`:

   ```bash
   $ monit start PROCESS-NAME
   ```

Follow the steps in the [Restoring Pivotal Cloud Foundry from Backup](#) topic to restore a backup, import an installation to restore your settings, or to share your settings with another user.
Restoring Pivotal Cloud Foundry from Backup

This topic describes the procedure for restoring Elastic Runtime from a backup. To create a backup, see the Backing Up Pivotal Cloud Foundry topic.

To restore a deployment, you must import installation settings, temporarily stop the Cloud Controller, restore the state of each critical backend component from its backup file, and restart the Cloud Controller. Using the BOSH manifest to locate your critical backend components is necessary to perform these steps. Manifests are automatically downloaded to the Ops Manager virtual machine. However, if you are using a separate jumpbox, you must manually download the BOSH deployment manifest.

Note: The procedure described in this topic restores a running Elastic Runtime deployment to the state captured by backup files. This procedure does not deploy Elastic Runtime. See the Installing PCF Guide for information about deploying Elastic Runtime.

Import Installation Settings

Note: Pivotal recommends that you export your installation settings before importing from a backup. See the Export Installation Settings section of the Backing Up Pivotal Cloud Foundry topic for more information.

Import installation settings imports the settings and assets of an existing PCF installation. Importing an installation overwrites any existing installation. You must provision a new Ops Manager in order to import settings.

1. Deploy the new Ops Manager VM:
   - Launching an Ops Manager Director Instance on AWS
   - Provisioning the OpenStack Infrastructure
   - Deploying Operations Manager to vSphere
   - Deploying Operations Manager to vCloud Air and vCloud

2. When redirected to the Welcome to Ops Manager page, select Import Existing Installation.

   ![Welcome to Ops Manager](image)

   Welcome to Ops Manager
   Select an Authentication System
   - Use an Identity Provider
   - Internal Authentication
   Upgrading Ops Manager?
   - Import Existing Installation

3. When prompted, enter the following:
   - Decryption Passphrase, which is the same as your password.
   - Click Choose File and browse to the installation zip file that you exported in the Export Installation Settings section.
4. Click **Import**.

   **Note:** Some browsers do not provide feedback on the status of the import process, and may appear to hang.

5. Before you see the new PCF 1.7 **Installation Dashboard**, a Security Features alert appears. Take note of your new **username**. Ensure you change your decryption passphrase before sharing it with other users. Click **Continue**.

6. A “Successfully imported installation” message appears upon completion.

7. Click **Apply Changes**. This immediately imports and applies upgrades to all tiles in a single transaction.

**Restoring BOSH Using Ops Manager**

1. From the **Product Installation Dashboard**, click the **Ops Manager Director** tile.

2. Make a change to your configuration in order to trigger a new deployment. For example, you can adjust the number of NTP servers in your deployment. Choose a change in configuration which suits your specific deployment.

3. Follow the instructions in **SSH into Ops Manager**. This example assumes an Amazon Web Services deployment:

   ```
   $ ssh -i ops_mgr.pem ubuntu@OPS-MGR-IP
   ```

4. Rename, move, or delete the **bosh-state.json** file. Removing **bosh-state.json** causes Ops Manager to treat the deploy as a new deployment, recreating missing Virtual Machines (VMs) including BOSH. The new deployment ignores existing VMs such as your Pivotal Cloud Foundry deployment:

   ```
   $ cd /var/tempest/workspaces/default/deployments/  
   $ sudo mv bosh-state.json bosh-state.json.old
   ```
5. Return to the Product Installation Dashboard, and click Apply Changes.

Target the BOSH Director

1. Install Ruby and the BOSH CLI Ruby gem on a machine outside of your PCF deployment.

2. From the Installation Dashboard in Ops Manager, select Ops Manager Director > Status and record the IP address listed for the Director. You access the BOSH Director using this IP address.

![Ops Manager Director Dashboard](image)

3. Click Credentials and record the Director credentials.

![Credentials Page](image)

4. From the command line, run `bosh target` to log into the BOSH Director using the IP address and credentials that you recorded:

```
$ bosh target 192.0.2.3
Target set to `microbosh-1234abcd1234abcd1234'
Your username: director
Enter password: ****************
Logged in as 'director'
```

**Note:** If `bosh target` does not prompt you for your username and password, run `bosh login`.

Download BOSH Manifest

1. Run `bosh deployments` to identify the name of your current BOSH deployment:

```
$ bosh deployments
+--------------------------+--------------------------+-----------------------------+--------------------------+
| Name                     | Release(s)               | Stemcell(s)                 |
+--------------------------+--------------------------+-----------------------------+--------------------------+
| cf-example               | cf-mysql/10              | bosh-vsphere-esxi-ubuntu-trusty-go_agent/2690.3 |
|                          |                          |                            |
+--------------------------+--------------------------+-----------------------------+--------------------------+
```

© Copyright Pivotal Software Inc, 2013-2018
2. Run `bosh download manifest DEPLOYMENT-NAME LOCAL-SAVE-NAME` to download and save each BOSH deployment manifest. You need this manifest to locate information about your databases. For each manifest, you will need to repeat these instructions. Replace `DEPLOYMENT-NAME` with the name of the current BOSH deployment. For this procedure, use `cf.yml` as the `LOCAL-SAVE-NAME`.

```
$ bosh download manifest cf-example cf.yml
Deployment manifest saved to `cf.yml`
```

---

Restoring Critical Backend Components

Your Elastic Runtime deployment contains several critical data stores that must be present for a complete restore. This section describes the procedure for restoring the databases and servers associated with your PCF installation. For more information about which data stores you may be using, see Backing Up and Restoring PCF.

You must restore each of the following:

- **Cloud Controller database** (Only if you are using PostgreSQL databases)
- **Apps Manager database** (Only if you are using PostgreSQL databases)
- **UAA database** (Only if you are using PostgreSQL databases)
- **NFS server** (In all cases)
- **MySQL server** (In all cases except if you are using external databases)

### Stop Cloud Controller

1. From a command line, run `bosh deployment DEPLOYMENT-MANIFEST` to select your PCF deployment. The manifest is located in `/var/tempest/workspaces/default/deployments/` on the Ops Manager VM. For example:

```
$ bosh deployment /var/tempest/workspaces/default/deployments/cf-bd784.yml
Deployment set to `/var/tempest/workspaces/default/deployments/cf-bd784.yml`
```

2. Run `bosh vms CF-DEPLOYMENT-NAME` to view a list of VMs in your PCF deployment. `CF-DEPLOYMENT-NAME` corresponds to the name of your PCF release deployment, which is also the filename of your manifest file without the `.yml` ending. For example:

```
$ bosh vms cf-bd784
+-------------------------------------------+---------+----------------------------------+--------------+
| Job/index                                      | State   | Resource Pool                        | IPs            |
| ccdb-partition-bd784/0                       | running | ccdb-partition-bd784                  | 10.85.xx.xx     |
| cloud_controller-partition-bd784/0           | running | cloud_controller-partition-bd784      | 10.85.xx.xx     |
| cloud_controller_worker-partition-bd784/0    | running | cloud_controller-partition-bd784      | 10.85.xx.xx     |
| clock_global-partition-bd784/0               | running | clock_global-partition-bd784          | 10.85.xx.xx     |
| nats-partition-bd784/0                       | running | nats-partition-bd784                  | 10.85.xx.xx     |
| router-partition-bd784/0                     | running | router-partition-bd784                | 10.85.xx.xx     |
| uaa-partition-bd784/0                        | running | uaa-partition-bd784                   | 10.85.xx.xx     |
+-------------------------------------------+---------+----------------------------------+--------------+
```

3. Perform the following steps for each Cloud Controller VM, excluding the Cloud Controller Database VM:

   a. SSH onto the VM:

      ```
      $ bosh ssh JOB-NAME
      ```

   b. From the VM, list the running processes:

      ```
      $ monit summary
      ```

   c. Stop all processes that start with `cloud_controller_`:

      ```
      $ monit stop PROCESS-NAME
      ```

### Restore the Cloud Controller Database
Use the Cloud Controller Database (CCDB) password and IP address to restore the Cloud Controller Database by following the steps detailed below. Find the IP address in your BOSH deployment manifest. To find your password in the Ops Manager Installation Dashboard, select Elastic Runtime and click Credentials>Link to Credential.

1. Use `scp` to send the Cloud Controller Database backup file to the Cloud Controller Database VM.

```bash
$ scp ccdb.sql vcap@YOUR-CCDB-VM-IP-ADDRESS:~/.
```

2. SSH into the Cloud Controller Database VM.

```bash
$ ssh vcap@YOUR-CCDB-VM-IP
```

3. Log in to the psql client.

```bash
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 ccdb
```

4. Drop the database schema and create a new one to replace it.

```sql
ccdb=# drop schema public cascade;
ccdb=# create schema public;
```

5. Restore the database from the backup file.

```bash
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 ccdb < ~/ccdb.sql
```

### Restore the Apps Manager Database from its backup state

1. Use the Apps Manager Database password and IP address to restore the Cloud Controller Database by running the following command. Find the IP address in your BOSH deployment manifest. To find your password in the Ops Manager Installation Dashboard, select Elastic Runtime and click Credentials>Link to Credential.

2. Use `scp` to copy the database backup file to the UAA Database VM.

```bash
$ scp console.sql vcap@YOUR-CONSOLE-DB-VM-ADDRESS:~/.
```

3. SSH into the UAA Database VM.

```bash
$ ssh vcap@YOUR-CONSOLE-DB-VM-ADDRESS
```

4. Log into the psql client.

```bash
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 console
```

5. Drop the existing database schema and create a new one to replace it.

```sql
console=# drop schema public cascade;
console=# create schema public;
```

6. Restore the database from the backup file.

```bash
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 console < ~/console.sql
```

### Restore UAA Database
Drop the UAA Database tables

1. Find your UAA Database VM ID. To view all VM IDs, run `bosh vms` from a command line:

   $ bosh vms

2. SSH into the UAA Database VM using the vcap user and password. If you do not have this information recorded, find it in the Ops Manager Installation Dashboard. Click the Elastic Runtime tile and select Credentials>Link to Credential.

   $ ssh vcap@YOUR-UAADB-VM-IP-ADDRESS

3. Run `find /var/vcap | grep 'bin/psql'` to find the locally installed psql client on the UAA Database VM.

   $ [YOUR-UAADB-VM-IP-ADDRESS]# find /var/vcap | grep 'bin/psql'

4. Log in to the psql client:

   $ [YOUR-UAADB-VM-IP-ADDRESS]# /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 uaa

5. Run the following commands to drop the tables:

   uaa=# drop schema public cascade;
   uaa=# create schema public;
   uaa=#

6. Exit the UAA Database VM.

   $ exit

Restore the UAA Database from its backup state

1. Use the UAA Database password and IP address to restore the UAA Database by running the following commands. You can find the IP address in your BOSH deployment manifest. To find your password in the Ops Manager Installation Dashboard, select Elastic Runtime and click Credentials>Link to Credential.

   2. Use `scp` to copy the database backup file to the UAA Database VM.

      $ scp uaa.sql vcap@YOUR-UAADB-VM-IP-ADDRESS:

   3. SSH into the UAA Database VM.

      $ ssh vcap@YOUR-UAADB-VM-IP-ADDRESS

   4. Restore the database from the backup file.

      $ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 uaa < ~/uaa.sql

Restore NFS

Use the NFS password and IP address to restore the NFS by following the steps detailed below. Find the IP address in your BOSH deployment manifest. To find your password in the Ops Manager Installation Dashboard, select Elastic Runtime and click Credentials>Link to Credential.
1. Run `ssh YOUR-NFS-VM-IP-ADDRESS` to enter the NFS VM.

   $ ssh vcap@192.0.2.10

2. Log in as root user. When prompted for a password, enter the vcap password you used to `ssh` into the VM:

   $ sudo su

3. Temporarily change the permissions on `/var/vcap/store` to add write permissions for all.

   $ chmod a+w /var/vcap/store

4. Use `scp` to send the NFS backup tarball to the NFS VM from your local machine.

   $ scp nfs.tar.gz vcap@YOUR-NFS-VM-IP-ADDRESS:/var/vcap/store

5. `cd` into the `store` folder on the NFS VM.

   $ cd /var/vcap/store

6. Decompress and extract the contents of the backup archive.

   $ tar xzf nfs.tar.gz

7. Change the permissions on `/var/vcap/store` to their prior setting.

   $ chmod a-w /var/vcap/store

8. Exit the NFS VM.

   $ exit

### Restore MySQL Database

Restoring a database will delete all data that existed in the database prior to the restore. Executing the SQL dump will drop, recreate, and refill the specified databases and tables.

![Warning:](image)

**Warning:** Restoring a database will delete all data that existed in the database prior to the restore. Restoring a database using a full backup artifact, produced by `mysqldump --all-databases` for example, will replace all data and user permissions.

1. From the same VM you used to perform the manual backup, or a similar VM, restore from the data dump:

   $ mysql -u root -p $MYSQL_NODE_IP < user_databases.sql

2. Re-apply user privileges:

   $ mysql -u root -p $MYSQL_NODE_IP -e "FLUSH PRIVILEGES"

   This command tells the cluster to re-load user permissions using the data that has just been restored.

### Start Cloud Controller

1. Run `bosh vms` to view a list of VMs in your selected deployment. The names of the Cloud Controller VMs begin with `cloud_controller`. 
2. Perform the following steps for each Cloud Controller VM, excluding the Cloud Controller Database VM:

a. SSH onto the VM:

```bash
$ bosh ssh JOB-NAME
```

b. From the VM, list the running processes:

```bash
$ monit summary
```

c. Start all processes that start with `cloud_controller_`:

```bash
$ monit start PROCESS-NAME
```
This topic covers strategies for monitoring virtual machine (VM) status and performance in Pivotal Cloud Foundry (PCF).

**Monitoring VMs Using the Ops Manager Interface**

Click any product tile and select the Status tab to view monitoring information.

The columns display the following information:

<table>
<thead>
<tr>
<th>VM Data Point</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Each job represents a component running on one or more VMs that Ops Manager deployed.</td>
</tr>
<tr>
<td>Index</td>
<td>For jobs that run across multiple VMs, the index value indicates the order in which the job VMs were deployed. For jobs that run on only one VM, the VM has an index value of 0.</td>
</tr>
<tr>
<td>IPs</td>
<td>IP address of the job VM.</td>
</tr>
<tr>
<td>CID</td>
<td>Uniquely identifies the VM.</td>
</tr>
<tr>
<td>Load Avg15</td>
<td>CPU load average over 15 minutes.</td>
</tr>
<tr>
<td>CPU</td>
<td>Current CPU usage.</td>
</tr>
<tr>
<td>Memory</td>
<td>Current memory usage.</td>
</tr>
<tr>
<td>Swap</td>
<td>Swap file percentage.</td>
</tr>
<tr>
<td>System Disk</td>
<td>System disk space usage.</td>
</tr>
<tr>
<td>Ephem. Disk</td>
<td>Ephemeral disk space usage.</td>
</tr>
<tr>
<td>Pers. Disk</td>
<td>Persistent disk space usage.</td>
</tr>
<tr>
<td>Logs</td>
<td>Download link for the most recent log files.</td>
</tr>
</tbody>
</table>

**Operations Manager VM Disk Space**
The Ops Manager stores its logs on the Ops Manager VM in the /tmp directory.

Note: The logs collect over time and do not self-delete. To prevent the VM from running out of disk space, restart the VM to clear the log entries from /tmp.

Monitoring in vSphere
To monitor VMs using the vSphere client:

1. Connect to a vCenter Server instance using the vSphere client.
2. Navigate to the Hosts And Clusters or VMs And Templates inventory view.
3. In the inventory tree, select a virtual machine.
4. Select the Performance tab from the content pane on the right.

VMware vSphere Server provides alarms that monitor VMs, as well as clusters, hosts, datacenters, datastores, networks, and licensing. To view preconfigured alarms, including disk usage alarms, related to a particular VM:

1. In the vSphere client, select the VM you want to monitor.
2. At the bottom left of the client window, click Alarms.
3. If a VM starts to run out of disk space, an alarm appears in the bottom panel.

Monitoring in vCloud Air
vCenter Operations Manager collects performance data from the virtual machines and disk drives in a deployment.

vCenter Hyperic specifically monitors operating systems, middleware, and applications.

Use vCenter Operations Manager and vCenter Hyperic to monitor the following services on the vCloud Director cells in your PCF deployment:

- **vmware-vcd-watchdog**: Watchdog service for the cell.
- **vmware-guestosd**: VMware Tools service. Provides heartbeat, shutdown, restart, and custom script execution functionality.
- **vmware-vcd-log-collection-agent**: Log collection service for the cell.
- **vmware-vcd-cell**: vCloud services for the cell.
This guide provides help with diagnosing and resolving issues encountered during a Pivotal Cloud Foundry (PCF) installation. For help troubleshooting issues that are specific to PCF deployments on VMware vSphere, refer to the topic on Troubleshooting Ops Manager for VMware vSphere.

An install or update can fail for many reasons. Fortunately, the system tends to heal or work around hardware or network faults. By the time you click the Install or Apply Changes button again, the problem may be resolved.

Some failures produce only generic errors like Exited with 1. In cases like this, where a failure is not accompanied by useful information, retry clicking Install or Apply Changes.

When the system does provide informative evidence, review the Common Problems section at the end of this guide to see if your problem is covered there.

Besides whether products install successfully or not, an important area to consider when troubleshooting is communication between VMs deployed by Pivotal Cloud Foundry. Depending on what products you install, communication takes the form of messaging, routing, or both. If they go wrong, an installation can fail. For example, in an Elastic Runtime installation the PCF VM tries to push a test application to the cloud during post-installation testing. The installation fails if the resulting traffic cannot be routed to the HA Proxy load balancer.

Viewing the Debug Endpoint

The debug endpoint is a web page that provides information useful in troubleshooting. If you have superuser privileges and can view the Ops Manager Installation Dashboard, you can access the debug endpoint.

- In a browser, open the URL: https://OPS-MANAGER-FQDN/debug

The debug endpoint offers three links:

- Files allows you to view the YAML files that Ops Manager uses to configure products that you install. The most important YAML file, `installation.yml`, provides networking settings and describes `microboah`. In this case, `microboah` is the VM whose BOSH Director component is used by Ops Manager to perform installations and updates of Elastic Runtime and other products.

- Components describes the components in detail.

- Rails log shows errors thrown by the VM where the Ops Manager web application (a Rails application) is running, as recorded in the `production.log` file. See the next section to learn how to explore other logs.

Logging Tips

Identifying Where to Start

This section contains general tips for locating where a particular problem is called out in the log files. Refer to the later sections for tips regarding specific logs (such as those for Elastic Runtime Components).

- Start with the largest and most recently updated files in the job log
- Identify logs that contain ‘err’ in the name
- Scan the file contents for a “failed” or “error” string

Viewing Logs for Elastic Runtime Components

To troubleshoot specific Elastic Runtime components by viewing their log files, browse to the Ops Manager interface and follow the procedure below.
1. In Ops Manager, browse to the **Pivotal Elastic Runtime > Status** tab. In the **Job** column, locate the component of interest.

2. In the **Logs** column for the component, click the download icon.

3. Browse to the **Pivotal Elastic Runtime > Logs** tab.

4. Once the zip file corresponding to the component of interest moves to the **Downloaded** list, click the linked file path to download the zip file.

5. Once the download completes, unzip the file.

The contents of the log directory vary depending on which component you view. For example, the Diego cell log directory contains subdirectories for the **metron_agent**, **rep**, **monit**, and **garden** processes. To view the standard error stream for **garden**, download the Diego cell logs and open

diego.0.job > garden >
garden.stderr.log

**Viewing Web Application and BOSH Failure Logs in a Terminal Window**

You can obtain diagnostic information from the Operations Manager by logging in to the VM where it is running. To log in to the Operations Manager VM, you need the following information:

- The IP address of the PCF VM shown in the **Settings** tab of the Ops Manager Director tile.
- Your **import credentials**. Import credentials are the username and password used to import the PCF .ova or .ovf file into your virtualization system.

Complete the following steps to log in to the Operations Manager VM:
1. Open a terminal window.

2. Run `ssh IMPORT-USERNAME@PCF-VM-IP-ADDRESS` to connect to the PCF installation VM.

3. Enter your import password when prompted.

4. Change directories to the home directory of the web application:
   ```bash
cd /home/tempest-web/tempest/web/
```

5. You are now in a position to explore whether things are as they should be within the web application.

   You can also verify that the `microbosh` component is successfully installed. A successful MicroBOSH installation is required to install Elastic Runtime and any products like databases and messaging services.

6. Change directories to the BOSH installation log home:
   ```bash
cd /var/tempest/workspaces/default/deployments/micro
```

7. You may want to begin by running a `tail` command on the current log:
   ```bash
cd /var/tempest/workspaces/default/deployments/micro
```

   If you are unable to resolve an issue by viewing configurations, exploring logs, or reviewing common problems, you can troubleshoot further by running BOSH diagnostic commands with the BOSH Command Line Interface (CLI).

   **Note:** Do not manually modify the deployment manifest. Operations Manager will overwrite manual changes to this manifest. In addition, manually changing the manifest may cause future deployments to fail.

### Viewing the VMs in Your Deployment

To view the VMs in your PCF deployment, perform the following steps specific to your IaaS.

#### Amazon Web Services (AWS)

1. Log in to the [AWS Console](https://aws.amazon.com).

2. Navigate to the EC2 Dashboard.

3. Click **Running Instances**.

4. Click the gear icon in the upper right.

5. Select the following: [job], [deployment], [director], [index].

6. Click **Close**.

#### OpenStack

1. Install the [novaclient](https://github.com/openstackClients/python-novaclient).

2. Point novaclient to your OpenStack installation and tenant by exporting the following environment variables:

   ```bash
   $ export OS_AUTH_URL=YOUR_KEYSTONE_AUTH_ENDPOINT
   $ export OS_TENANT_NAME=TENANT_NAME
   $ export OS_USERNAME=USERNAME
   $ export OS_PASSWORD=PASSWORD
   ``

3. List your VMs by running the following command:

   ```bash
   $ nova list --fields metadata
   ```

#### vSphere

© Copyright Pivotal Software Inc, 2013-2018
1. Log into vCenter.

2. Select Hosts and Clusters.

3. Select the top level object that contains your PCF deployment. For example, select Cluster, datastore or Resource Pool.

4. In the top tab, click Related Objects.

5. Select Virtual Machines.

6. Right click on the Table heading and select Show/Hide Columns.

7. Select the following boxes: job, deployment, director, index.

Viewing Apps Manager Logs in a Terminal Window

The Apps Manager provides a graphical user interface to help manage organizations, users, applications, and spaces.

When troubleshooting Apps Manager performance, you might want to view the Apps Manager application logs. To view the Apps Manager application logs, follow these steps:

1. Run `cf login -a api.MY-SYSTEM-DOMAIN -u admin` from a command line to log in to PCF using the UAA Administrator credentials. In Pivotal Ops Manager, refer to Pivotal Elastic Runtime > Credentials for these credentials.

   ```
   $ cf login -a api.example.com -u admin
   API endpoint: api.example.com
   Password:******
   Authenticating... OK
   ```

2. Run `cf target -o system -s apps-manager` to target the system org and the apps-manager space.

   ```
   $ cf target -o system -s apps-manager
   ```

3. Run `cf logs apps-manager` to tail the Apps Manager logs.

   ```
   $ cf logs apps-manager
   Connected, tailing logs for app apps-manager in org system / space apps-manager as admin...
   ```

Changing Logging Levels for the Apps Manager

The Apps Manager recognizes the `LOG_LEVEL` environment variable. The `LOG_LEVEL` environment variable allows you to filter the messages reported in the Apps Manager log files by severity level. The Apps Manager defines severity levels using the Ruby standard library `Logger` class.

By default, the Apps Manager `LOG_LEVEL` is set to `info`. The logs show more verbose messaging when you set the `LOG_LEVEL` to `debug`.

To change the Apps Manager `LOG_LEVEL`, run `cf set-env apps-manager LOG_LEVEL` with the desired severity level.

```
$ cf set-env apps-manager LOG_LEVEL debug
```

You can set `LOG_LEVEL` to one of the six severity levels defined by the Ruby Logger class:

- **Level 5:** `unknown` – An unknown message that should always be logged
- **Level 4:** `fatal` – An unhandleable error that results in a program crash
- **Level 3:** `error` – A handleable error condition
- **Level 2:** `warn` – A warning
- **Level 1:** `info` – General information about system operation
- **Level 0:** `debug` – Low-level information for developers

© Copyright Pivotal Software Inc, 2013-2018
Once set, the Apps Manager log files only include messages at the set severity level and above. For example, if you set `LOG_LEVEL` to `fatal`, the log includes `fatal` and `unknown` level messages only.

Common Issues

Compare evidence that you have gathered to the descriptions below. If your issue is covered, try the recommended remediation procedures.

BOSH Does Not Reinstall

You might want to reinstall BOSH for troubleshooting purposes. However, if PCF does not detect any changes, BOSH does not reinstall. To force a reinstall of BOSH, select Ops Manager Director > Resource Sizes and change a resource value. For example, you could increase the amount of RAM by 4 MB.

Creating Bound Missing VMs Times Out

This task happens immediately following package compilation, but before job assignment to agents. For example:

```
```

This is most likely a NATS issue with the VM in question. To identify a NATS issue, inspect the agent log for the VM. Since the BOSH director is unable to reach the BOSH agent, you must access the VM using another method. You will likely also be unable to access the VM using TCP. In this case, access the VM using your virtualization console.

To diagnose:

1. Access the VM using your virtualization console and log in.
2. Navigate to the Credentials tab of the Elastic Runtime tile and locate the VM in question to find the VM credentials.
4. Run `cd /var/vcap/bosh/log`
5. Open the file `current`
6. First, determine whether the BOSH agent and director have successfully completed a handshake, represented in the logs as a “ping-pong”:

```
```

This handshake must complete for the agent to receive instructions from the director.
7. If you do not see the handshake, look for another line near the beginning of the file, prefixed `INFO: loaded new infrastructure settings`. For example:

```
```

This is a JSON blob of key/value pairs representing the expected infrastructure for the BOSH agent. For this issue, the following section is the most important:
"mbus"=>"nats://nats:nats@192.0.2.17:4222"

This key/value pair represents where the agent expects the NATS server to be. One diagnostic tactic is to try pinging this NATS IP address from the VM to
determine whether you are experiencing routing issues.

Install Exits With a Creates/Updates/Deletes App Failure or With a 403 Error
Scenario 1: Your PCF install exits with the following 403 error when you attempt to log in to the Apps Manager:
{"type": "step_finished", "id": "apps-manager.deploy"}
`fetch': 403 => Net::HTTPForbidden for https://login.api.example.net/oauth/authorizeresponse_type=code&client_id=portal&redirect_uri=https%3...
-- unhandled response (Mechanize::ResponseCodeError)

Scenario 2: Your PCF install exits with a creates/updates/deletes an app (FAILED -

error message with the following stack trace:

1)

1) App CRUD creates/updates/deletes an app
Failure/Error: Unable to find matching line from backtrace
CFoundry::TargetRefused:
Connection refused - connect(2)

In either of the above scenarios, ensure that you have correctly entered your domains in wildcard format:
1. Browse to the Operations Manager fully qualified domain name (FQDN).
2. Click the Elastic Runtime tile.
3. Select HAProxy and click Generate Self-Signed RSA Certificate.
4. Enter your system and app domains in wildcard format, as well as optionally any custom domains, and click Save. Refer to Elastic Runtime > Cloud
Controller for explanations of these domain values.

Install Fails When Gateway Instances Exceed Zero
If you configure the number of Gateway instances to be greater than zero for a given product, you create a dependency on Elastic Runtime for that
product installation. If you attempt to install a product tile with an Elastic Runtime dependency before installing Elastic Runtime, the install fails.
To change the number of Gateway instances, click the product tile, then select Settings > Resource sizes > INSTANCES and change the value next to the
product Gateway job.
To remove the Elastic Runtime dependency, change the value of this field to 0 .

Out of Disk Space Error

© Copyright Pivotal Software Inc, 2013-2018

265

1.7


PCF displays an Out of Disk Space error if log files expand to fill all available disk space. If this happens, rebooting the PCF installation VM clears the tmp directory of these log files and resolves the error.

Installing Ops Manager Director Fails

If the DNS information for the PCF VM is incorrectly specified when deploying the PCF .ova file, installing Ops Manager Director fails at the “Installing Micro BOSH” step.

To resolve this issue, correct the DNS settings in the PCF Virtual Machine properties.

Deleting Ops Manager Fails

Ops Manager displays an error message when it cannot delete your installation. This scenario might happen if the Ops Manager Director cannot access the VMs or is experiencing other issues. To manually delete your installation and all VMs, you must do the following:

1. Use your IaaS dashboard to manually delete the VMs for all installed products, with the exception of the Ops Manager VM.

2. SSH into your Ops Manager VM and remove the installation.yml file from /var/tempest/workspaces/default/.

Note: Deleting the installation.yml file does not prevent you from reinstalling Ops Manager. For future deploys, Ops Manager regenerates this file when you click Save on any page in the Ops Manager Director.

Your installation is now deleted.

Installing Elastic Runtime Fails

If the DNS information for the PCF VM becomes incorrect after Ops Manager Director has been installed, installing Elastic Runtime with Pivotal Operations Manager fails at the “Verifying app push” step.

To resolve this issue, correct the DNS settings in the PCF Virtual Machine properties.

Cannot Attach Disk During MicroBOSH Deploy to vCloud

When attempting to attach a disk to a MicroBOSH VM, you might receive the following error:

The requested operation cannot be performed because disk XXXXXXXXX was not created properly.

Possible causes and recommendations:

- If the account used during deployment lacks permission to access the default storage profile, attaching the disk might fail.
- vCloud Director can incorrectly report a successful disk creation even if the operation fails, resulting in subsequent error messages. To resolve this issue, redeploy MicroBOSH.

Ops Manager Hangs During MicroBOSH Install or HAProxy States “IP Address Already Taken”

During an Ops Manager installation, you might receive the following errors:

- The Ops Manager GUI shows that the installation stops at the “Setting MicroBOSH deployment manifest” task.
- When you set the IP address for the HAproxy, the “IP Address Already Taken” message appears.

When you install Ops Manager, you assign it an IP address. Ops Manager then takes the next two consecutive IP addresses, assigns the first to MicroBOSH, and reserves the second. For example:

203.0.113.1 - Ops Manager (User assigned)
203.0.113.2 - MicroBOSH (Ops Manager assigned)
203.0.113.3 - Reserved (Ops Manager reserved)

To resolve this issue, ensure that the next two subsequent IP addresses from the manually assigned address are unassigned.
Poor PCF Performance

If you notice poor network performance by your PCF deployment and your deployment uses a Network Address Translation (NAT) gateway, your NAT gateway may be under-resourced.

Troubleshoot

To troubleshoot the issue, set a custom firewall rule in your IaaS console to route traffic originating from your private network directly to an S3-compatible object store. If you see decreased average latency and improved network performance, perform the solution below to scale up your NAT gateway.

Scale Up Your NAT Gateway

Perform the following steps to scale up your NAT gateway:

1. Navigate to your IaaS console.
2. Spin up a new NAT gateway of a larger VM size than your previous NAT gateway.
3. Change the routes to direct traffic through the new NAT gateway.
4. Spin down the old NAT gateway.

The specific procedures will vary depending on your IaaS. Consult your IaaS documentation for more information.

Common Issues Caused by Firewalls

This section describes various issues you might encounter when installing Elastic Runtime in an environment that uses a strong firewall.

DNS Resolution Fails

When you install PCF in an environment that uses a strong firewall, the firewall might block DNS resolution. To resolve this issue, refer to the Troubleshooting DNS Resolution Issues section of the Preparing Your Firewall for Deploying PCF topic.
Troubleshooting Ops Manager for VMware vSphere

This guide provides help with diagnosing and resolving issues that are specific to Pivotal Cloud Foundry (PCF) deployments on VMware vSphere.

For infrastructure-agnostic troubleshooting help, refer to the Pivotal Cloud Foundry Troubleshooting Guide.

Common Issues

The following sections list common issues you might encounter and possible resolutions.

PCF Installation Fails

If you modify the vCenter Statistics Interval Duration setting from its default setting of 5 minutes, the PCF installation might fail at the MicroBOSH deployment stage, and the logs might contain the following error message:

```
The specified parameter is not correct, interval
```

This failure happens because Ops Manager expects a default value of 5 minutes, and the call to this method fails when the retrieved value does not match the expected default value.

To resolve this issue, launch vCenter, navigate to Administration > vCenter Server Settings > Statistics, and reset the vCenter Statistics Interval Duration setting to 5 minutes.

BOSH Automated Installation Fails

Before starting an Elastic Runtime deployment, you must set up and configure a vSphere cluster.

If you enable vSphere DRS (Distributed Resource Scheduler) for the cluster, you must set the Automation level to Partially automated or Fully automated.

If you set the Automation level to Manual, the BOSH automated installation will fail with a `power_on_vm` error when BOSH attempts to create virtual VMs.

Ops Manager Loses Its IP Address After HA or Reboot
Ops Manager can lose its IP address and use DHCP due to an issue in the open source version of VMware Tools. Review the support topic for this issue in order to troubleshoot this problem.

Cannot Connect to the OVF Via a Browser

If you deployed the OVF file but cannot connect to it via a browser, check that the network settings you entered in the wizard are correct.

1. Access the PCF installation VM using the vSphere Console. If your network settings are misconfigured, you will not be able to SSH into the installation VM.

2. Log in using the credentials you provided when you imported the PCF .ova in vCenter.

3. Confirm that the network settings are correct by checking that the ADDRESS, NETMASK, GATEWAY, and DNS-NAMESERVERS entries are correct in /etc/network/interfaces.

4. If any of the settings are wrong, run `sudo vi /etc/network/interfaces` and correct the wrong entries.

5. In vSphere, navigate to the Summary tab for the VM and confirm that the network name is correct.

   ![Virtual Machine Screenshot]

   6. If the network name is wrong, right click on the VM, select Edit Settings > Network adapter 1, and select the correct network.

   7. Reboot the installation VM.

Installation Fails with Failed Network Connection

If you experience a communication error while installing Ops Manager or MicroBOSH Director, check the following settings.

- Ensure that the routes are not blocked. vSphere environments use NSX for firewall, NAT/SNAT translation and load balancing. All communication between PCF VMs and vCenter or ESXi hosts route through the NSX firewall and are blocked by default.

- Open port 443. Ops Manager and MicroBOSH Director VMs require access to vCenter and all ESXi through port 443.

- Allocate more IP addresses. BOSH requires that you allocate a sufficient number of additional dynamic IP addresses when configuring a reserved IP range during installation. BOSH uses these IPs during installation to compile and deploy VMs, install Elastic Runtime, and connect to services. We recommend that you allocate at least 36 dynamic IP addresses when deploying Ops Manager and Elastic Runtime.
Recovering MySQL from Elastic Runtime Downtime

This topic describes the procedure for recovering a terminated Elastic Runtime cluster using a process known as bootstrapping.

When to Bootstrap

You must bootstrap a cluster that loses quorum. A cluster loses quorum when less than half of the nodes can communicate with each other for longer than the configured grace period. If a cluster does not lose quorum, individual unhealthy nodes automatically rejoin the cluster after resolving the error, restarting the node, or restoring connectivity.

You can detect lost quorum through the following symptoms:

- All nodes appear "Unhealthy" on the proxy dashboard, viewable at `proxy-BOSH-JOB-INDEX.p-mysql.YOUR-SYSTEM-DOMAIN`:

  ![Switchboard](image)

  - All responsive nodes report the value of `wsrep_cluster_status` as "non-Primary":

    ```
    mysql> SHOW STATUS LIKE 'wsrep_cluster_status';
    +----------------+---------------+
    | Variable_name  | Value         |
    +----------------+---------------+
    | wsrep_cluster_status | non-Primary |
    +----------------+---------------+
    ```

  - All responsive nodes respond with `ERROR 1047` when queried with most statement types:

    ```
    mysql> select * from mysql.user;
    ERROR 1047 (08S01) at line 1: WSREP has not yet prepared node for application use
    ```

  See the [Cluster Scaling, Node Failure, and Quorum](#) topic for more details about determining cluster state.

  Follow the steps below to recover a cluster that has lost quorum.

Step 1: Choose the Correct Manifest

1. Log into the BOSH director by running `bosh target DIRECTOR-URL` followed by `bosh login USERNAME PASSWORD`.
2. Run `bosh deployments`.

© Copyright Pivotal Software Inc, 2013-2018 270
Scenario 1: Virtual Machines Running, Cluster Disrupted

In this scenario, nodes are up and running, but the cluster has been disrupted. You can run the bootstrap errand without recreating the VMs.

1. Run `bosh run errand bootstrap`. The errand command prints the following message when finished running:

```
+--------------------------------------------------+--------------------+------------------------------------------------+------------+
| Instance                                      | State                  | Resource Pool       | IPs         |
| mySQL-partition-a813339fde9330e9b905/0         | failing                | mySQL-partition-a813339fde9330e9b905/0            | 203.0.113.55 |
| mySQL-partition-a813339fde9330e9b905/1         | failing                | mySQL-partition-a813339fde9330e9b905/1            | 203.0.113.56 |
| mySQL-partition-a813339fde9330e9b905/2         | failing                | mySQL-partition-a813339fde9330e9b905/2            | 203.0.113.57 |
+--------------------------------------------------+--------------------+------------------------------------------------+------------+
```

2. If the output of `bosh instances` shows the state of the jobs as `unknown/unknown`, proceed to Scenario 2.
Bootstrap errand completed

[siderr]
+ echo 'Started bootstrap errand ...'
+ JOB_DIRS=var/vcap/jobs/bootstrap
+ CONFIG_PATH=var/vcap/jobs/bootstrap/config/config.yml
+ var/vcap/packages/bootstrap/bin/cf-mysql-bootstrap --configPath=var/vcap/jobs/bootstrap/config/config.yml
+ echo 'Bootstrap errand completed'
+ exit 0

Errand 'bootstrap' completed successfully (exit code 0)

Note: Sometimes the bootstrap errand fails on the first try. If this happens, run the command again in a few minutes.

2. If the errand fails, try performing the steps automated by the errand manually by following the Manual Bootstrapping procedure.

Scenario 2: Virtual Machines Terminated or Lost

In this scenario, severe circumstances such as power failure have terminated all of your VMs. You need to recreate the VMs before you can recover the cluster.

1. To recreate terminated or lost VMs, perform the following steps:

   a. If you use the VM Resurrector, disable it.
   b. Run the BOSH Cloud Check interactive command. When prompted, select Recreate VM. If this option fails, select Delete VM reference.

   c. Re-enable the VM Resurrector if you want to continue to use it.

Note: Do not proceed to the next step until all VMs are in the starting or failing state.
2. Complete the following steps to prepare your deployment for the bootstrap errand:

   a. Run `bosh edit deployment` to launch a `vi` editor and modify the deployment.
   b. Search for the jobs section: `jobs`
   c. Search for the `mysql-partition`: `mysql-partition`
   d. Search for the update section: `update`
   e. Change `max_in_flight` to 3.
   f. Below the `max_in_flight` line, add a new line: `canaries: 0`
   g. Set `update.serial` to false.
   h. Run `bosh deploy`.

3. Run `bosh run errand bootstrap`.

4. Run `bosh instances` and examine the output to confirm that the errand completes successfully. Some instances may still appear as `failing`.

5. Complete the following steps to restore the BOSH configuration:

   a. Run `bosh edit deployment`.
   b. Re-set `canaries` to 1, `max_in_flight` to 1, and `serial` to true in the same manner as above.
   c. Run `bosh deploy`.
   d. Validate that all mysql instances are in `running` state.

   **Note:** You must reset the values in the BOSH manifest to ensure successful future deployments and accurate reporting of the status of your jobs.

6. If this procedure fails, try performing the steps automated by the errand manually by following the Manual Bootstrapping procedure.

**Manual Bootstrapping**

**Note:** The following steps are prone to user error and can result in lost data if followed incorrectly. Please follow the Run the Bootstrap Errand instructions above first, and only resort to the manual process if the errand fails to repair the cluster.

If the bootstrap errand cannot recover the cluster, you need to perform the steps automated by the errand manually.

- If the output of `bosh instances` shows the state of the jobs as `failing` (Scenario 1), proceed directly to the manual steps below.
- If the output of `bosh instances` shows the state of the jobs as `unknown/unknown`, perform Steps 1-2 of Scenario 2, substitute the manual steps below for Step 3, and then perform Steps 4-5 of Scenario 2.

1. SSH to each node in the cluster and, as root, shut down the `mariadb` process.

   ```
   $ monit stop mariadb_ctrl
   ```

   Re-bootstrapping the cluster will not be successful unless all other nodes have been shut down.

2. Choose a node to bootstrap by locating the node with the highest transaction sequence number (`seqno`). You can obtain the `seqno` of a stopped node in one of two ways:
   - If a node shut down gracefully, the `seqno` is in the Galera state file of the node.
   - If the node crashed or was killed, the `seqno` in the Galera state file of the node is `-1`. In this case, the `seqno` may be recoverable from the database.

   ```
   $ cat /var/vcap/store/mysql/grastate.dat | grep 'seqno:
   ```

1. Run the following command to start up the database, log the recovered sequence number, and exit.

   ```
   $ /var/vcap/packages/mariadb/bin/mysqld --wsrep-recover
   ```

2. Scan the error log for the recovered sequence number. The last number after the group id `{uuid}` is the recovered `seqno`:

   ```
   $ grep "Recovered position" /var/vcap/syslog/mysql/mysql.err.log | tail -1
   150225 18:09:42 mysqld_safe WSREP: Recovered position e3935c73b797-11ed-9a6f-0b73eb46-15
   ```
If the node never connected to the cluster before crashing, it may not have a group id (uuid in grastate.dat). In this case, you cannot recover the seqno. Unless all nodes crashed this way, do not choose this node for bootstrapping.

3. Choose the node with the highest seqno value as the bootstrap node. If all nodes have the same seqno, you can choose any node as the bootstrap node.

**Note:** Only perform these bootstrap commands on the node with the highest seqno. Otherwise, the node with the highest seqno will be unable to join the new cluster unless its data is abandoned. Its mariadb process will exit with an error. See the [Cluster Scaling, Node Failure, and Quorum](#) topic for more details on intentionally abandoning data.

4. On the bootstrap node, update the state file and restart the mariadb process.

   ```bash
   $ echo -n "NEEDS_BOOTSTRAP" > /var/vcap/store/mysql/state.txt
   $ monit start mariadb_ctrl
   ```

5. Check that the mariadb process has started successfully.

   ```bash
   $ watch monit summary
   ```

   It can take up to ten minutes for monit to start the mariadb process.

6. Once the bootstrapped node is running, start the mariadb process on the remaining nodes using monit.

   ```bash
   $ monit start mariadb_ctrl
   ```

7. Verify that the new nodes have successfully joined the cluster. The following command displays the total number of nodes in the cluster:

   ```sql
   mysql> SHOW STATUS LIKE 'wsrep_cluster_size';
   ```

8. Complete the following steps to restore the BOSH configuration:

   a. Run `bosh edit deployment`.
   b. Re-set `canaries` to 1, `max_in_flight` to 1, and `serial` to true in the same manner as above.
   c. Run `bosh deploy`.
   d. Validate that all mysql instances are in `running` state.

   **Note:** You must reset the values in the BOSH manifest to ensure successful future deployments and accurate reporting of the status of your jobs.
Advanced Troubleshooting with the BOSH CLI

To perform advanced troubleshooting, you must log in to the BOSH Director. From there, you can run specific commands using the BOSH Command Line Interface (CLI). BOSH Director diagnostic commands have access to information about your entire Pivotal Cloud Foundry (PCF) installation.

The BOSH Director runs on the virtual machine (VM) that Ops Manager deploys on the first install of the Ops Manager Director tile. BOSH Director diagnostic commands have access to information about your entire Pivotal Cloud Foundry (PCF) installation.

Note: For more troubleshooting information, refer to the Troubleshooting Guide.

Note: Verify that no BOSH Director tasks are running on the Ops Manager VM before running any commands. You should not proceed with troubleshooting until all BOSH Director tasks have completed or you have ended them. See the Bosh CLI Commands for more information.

Prepare to Use the BOSH CLI

This section guides you through preparing to use the BOSH CLI.

Gather Information

Before you begin troubleshooting with the BOSH CLI, collect the information you need from the Ops Manager interface.

1. Open the Ops Manager interface by navigating to the Ops Manager fully qualified domain name (FQDN). Ensure that there are no installations or updates in progress.

2. Click the Ops Manager Director tile and select the Status tab.

3. Record the IP address for the Director job. This is the IP address of the VM where the BOSH Director runs.

4. Select the Credentials tab.

5. Click Link to Credential to view and record the Director Credentials.
6. Return to the **Installation Dashboard**.

7. *(Optional)* To prepare to troubleshoot the job VM for any other product, click the product tile and repeat the procedure above to record the IP address and VM credentials for that job VM.

8. Log out of Ops Manager.

   ![Ops Manager Dashboard](image)

   **Note:** You must log out of the Ops Manager interface to use the BOSH CLI.

---

**SSH into Ops Manager**

Use SSH to connect to the Ops Manager web application VM.

To SSH into the Ops Manager VM:

**vSphere:**

You need the credentials used to import the PCF .ova or .ovf file into your virtualization system.

1. From a command line, run `ssh ubuntu@OPS-MANAGER-FQDN`.

2. When prompted, enter the password that you set during the .ova deployment into vCenter:

   ```
   $ ssh ubuntu@OPS-MANAGER-FQDN
   Password: ***********
   ```

**AWS and OpenStack:**

1. Locate the Ops Manager FQDN on the AWS EC2 instances page or the OpenStack Access & Security page.

2. Change the permissions on the `.pem` file to be more restrictive:

   ```
   $ chmod 600 ops_mgr.pem
   ```

3. Run the `ssh` command:

   ```
   ssh -i ops_mgr.pem ubuntu@OPS-MANAGER-FQDN
   ```

---

**Log in to the BOSH Director**

Log in to the BOSH Director using one of the following options below:

- **Internal User Store Login via UAA** - target and log in to the Director using BOSH.
External User Store Login via SAML - use an external user store to log in to the BOSH Director.

Internal User Store Login via UAA

1. Target the BOSH UAA on Ops Manager with the UAAC command:

   ```
   $ uaac target --ca-cert /var/tempest/workspaces/default/root_ca_certificate https://DIRECTOR_IP:8443
   ```

2. Run `bosh target DIRECTOR-IP-ADDRESS` to target your Ops Manager VM using the BOSH CLI.

3. Retrieve the Director password from the Ops Manager Director > Credentials tab. Alternatively, launch a browser and visit the following URL to obtain the password:

   ```
   https://{OPSMANAGER-FQDN}/api/v0/deployed/director/credentials/director_credentials
   ```

4. Log in using the BOSH Director credentials:

   ```
   $ bosh --ca-cert /var/tempest/workspaces/default/root_ca_certificate target DIRECTOR-IP-ADDRESS
   Target set to 'DIRECTOR_UUID'
   Your username: director
   Enter password: (DIRECTOR_CREDENTIAL)
   Logged in as 'director'
   ```

External User Store Login via SAML

1. Log in to your identity provider and use the following information to configure SAML Service Provider Properties:

   - **Service Provider Entity ID**: bosh-uaa
   - **ACS URL**: https://DIRECTOR-IP-ADDRESS:8443/saml/SSO/alias/bosh-uaa
   - **Binding**: HTTP Post
   - **SLO URL**: https://DIRECTOR-IP-ADDRESS:8443/saml/SSO/alias/bosh-uaa
   - **Binding**: HTTP Redirect
   - **Name ID**: Email Address

2. Log in to BOSH using your SAML credentials:

   ```
   $ bosh login
   Email: admin
   Password: (One Time Code (Get one at https://192.0.2.16:8888/passcode):
   ```

   If you do not have browser access to the BOSH Director, run `sshuttle` on a local Linux workstation to browse the BOSH Director IP as if it were a local address. Retrieve a UAA passcode using the browser:

   ```
   $ git clone https://github.com/apenwarr/sshuttle.git
   $ cd sshuttle
   $ ./sshuttle -r username@opsmanagerIP 0.0.0.0/0 -vv
   ```

3. Click Log in with organization credentials [SAML].

   ![SAML Login](image)

4. Copy the **Temporary Authentication Code** that appears in your browser.
5. You see a login confirmation. For example:

```
Logged in as admin@example.org
```

Select a Product Deployment to Troubleshoot

When you import and install a product using Ops Manager, you deploy an instance of the product described by a YAML file. Examples of available products include Elastic Runtime, MySQL, or any other service that you imported and installed.

Perform the following steps to select a product deployment to troubleshoot:

1. Identify the YAML file that describes the deployment you want to troubleshoot.
   You identify the YAML file that describes a deployment by its filename. For example, to identify Elastic Runtime deployments, run the following command:
   ```bash
   find /var/tempest/workspaces/default/deployments -name cf-*.yml
   ```
   The table below shows the naming conventions for deployment files.

<table>
<thead>
<tr>
<th>Product</th>
<th>Deployment Filename Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elastic Runtime</td>
<td>cf-&lt;20-character_random_string&gt;.yml</td>
</tr>
<tr>
<td>MySQL Dev</td>
<td>cf_services-&lt;20-character_random_string&gt;.yml</td>
</tr>
<tr>
<td>Other</td>
<td>&lt;20-character_random_string&gt;.yml</td>
</tr>
</tbody>
</table>

   **Note:** Where there is more than one installation of the same product, record the release number shown on the product tile in Operations Manager. Then, from the YAML files for that product, find the deployment that specifies the same release version as the product tile.

2. Run `bosh status` and record the UUID value.

3. Open the `DEPLOYMENT-FILENAME.yml` file in a text editor and compare the `director_uuids` value in this file with the UUID value that you recorded. If the values do not match, perform the following steps:
   a. Replace the `director_uuids` value with the UUID value.
   b. Run `bosh deployment DEPLOYMENT-FILENAME.yml` to reset the file for your deployment.

4. Run `bosh deployment DEPLOYMENT-FILENAME.yml` to instruct the BOSH Director to apply BOSH CLI commands against the deployment described by the YAML file that you identified:

```
$ bosh deployment /var/tempest/workspaces/default/deployments/cf-cca1234abcd.yml
```

Use the BOSH CLI for Troubleshooting

This section describes three BOSH CLI commands commonly used during troubleshooting.

- **VMS:** Lists all VMs in a deployment
- **Cloudcheck:** Runs a cloud consistency check and interactive repair
- **SSH:** Starts an interactive session or executes commands with a VM

**BOSH VMS**

`bosh vms` provides an overview of the virtual machines that BOSH manages as part of the current deployment.
When troubleshooting an issue with your deployment, `bosh vms` may show a VM in an unknown state. Run `bosh cloudcheck` on a VM in an unknown state to instruct BOSH to diagnose problems with the VM.

You can also run `bosh vms` to identify VMs in your deployment, then use the `bosh sch` command to SSH into an identified VM for further troubleshooting.

`bosh vms` supports the following arguments:

- `-details`: Report also includes Cloud ID, Agent ID, and whether or not the BOSH Resurrection has been enabled for each VM
- `-vitals`: Report also includes load, CPU, memory usage, swap usage, system disk usage, ephemeral disk usage, and persistent disk usage for each VM
- `-dns`: Report also includes the DNS A record for each VM

**Note:** The **Status** tab of the Elastic Runtime product tile displays information similar to the `bosh vms` output.

### BOSH Cloudcheck

Run the `bosh cloudcheck` command to instruct BOSH to detect differences between the VM state database maintained by the BOSH Director and the actual state of the VMs. For each difference detected, `bosh clouddcheck` can offer the following repair options:

- **Reboot VM**: Instructs BOSH to reboot a VM. Rebooting can resolve many transient errors.
- **Ignore problem**: Instructs BOSH to do nothing. You may want to ignore a problem in order to run `bosh ssh` and attempt troubleshooting directly on the machine.
- **Reassociate VM with corresponding instance**: Updates the BOSH Director state database. Use this option if you believe that the BOSH Director state database is in error and that a VM is correctly associated with a job.
- **Recreate VM using last known apply spec**: Instructs BOSH to destroy the server and recreate it from the deployment manifest that the installer provides. Use this option if a VM is corrupted.
- **Delete VM reference**: Instructs BOSH to delete a VM reference in the Director state database. If a VM reference exists in the state database, BOSH...
expects to find an agent running on the VM. Select this option only if you know that this reference is in error. Once you delete the VM reference, BOSH can no longer control the VM.

Example Scenarios

Unresponsive Agent

```bash
$ bosh cloudcheck
ccdb/0 (vm-5e37133f-bc33-450e-98b1-f86d5b63502a) is not responding:

- Ignore problem
- Reboot VM
- Recreate VM using last known apply spec
- Delete VM reference (DANGEROUS!)
```

Missing VM

```bash
$ bosh cloudcheck
VM with cloud ID `vm-5e37133f-bc33-450e-98b1-f86d5b63502a` missing:

- Ignore problem
- Recreate VM using last known apply spec
- Delete VM reference (DANGEROUS!)
```

Unbound Instance VM

```bash
$ bosh cloudcheck
VM `vm-5e37133f-bc33-450e-98b1-f86d5b63502a` reports itself as `ccdb/0` but does not have a bound instance:

- Ignore problem
- Delete VM (unless it has persistent disk)
- Reassociate VM with corresponding instance
```

Out of Sync VM

```bash
$ bosh cloudcheck
VM `vm-5e37133f-bc33-450e-98b1-f86d5b63502a` is out of sync:
expected `cf-d7293430724a2c421061; ccdb/0` got `cf-d7293430724a2c421061; nats/0`:

- Ignore problem
- Delete VM (unless it has persistent disk)
```

BOSH SSH

Use `bosh ssh` to SSH into the VMs in your deployment.

Follows the steps below to use `bosh ssh`:

1. Run `ssh-keygen -t rsa` to provide BOSH with the correct public key.
2. Accept the defaults.
3. Run `bosh ssh`.
4. Select a VM to access.
5. Create a password for the temporary user that the `bosh ssh` command creates. Use this password if you need sudo access in this session.

Example:
$ bosh ssh
1. hx_proxy/0
2. nats/0
3. etcd_and_metrics/0
4. etcd_and_metrics/1
5. etcd_and_metrics/2
6. health_manager/0
7. nfs_server/0
8. ccdb/0
9. cloud_controller/0
10. clock_global/0
11. cloud_controller_worker/0
12. router/0
13. uaa/0
14. login/0
15. consoledb/0
16. dea/0
17. loggregator/0
18. loggregator_traffic_controller/0
19. loggregator_traffic_controller/1
20. push-apps-manager/0
21. smoke-tests/0

Choose an instance: 17
Enter password (use it to sudo on remote host): *******
Target deployment `cf_services-2c3c918a135ab5f91ee1`

Setting up ssh artifacts

### Standard SSH

In most cases, operators should use the `bosh ssh` command in the BOSH CLI to SSH into the BOSH Director and other VMs in their deployment. However, operators can also use standard `ssh` by performing the procedures below.

1. Locate the IP address of your BOSH Director and your BOSH Director credentials by following the steps above.

2. SSH into the BOSH Director with the public key you used with `bosh-init` to deploy the BOSH Director:

```bash
$ ssh BOSH-DIRECTION-IP -i PATH-TO-PUBLIC-KEY
```

3. Enter your BOSH Director credentials to log in.

From the BOSH Director, you can SSH into the other VMs in your deployment by performing the following steps:

1. Identify the private IP address of the component VM you want to SSH into by doing one of the following:
   - Perform the steps above to use the BOSH CLI to log in to your BOSH Director and use `bosh vms` to list the IP addresses of your component VMs.
   - Navigate to your IaaS console and locate the IP address of the VM. For example, Amazon Web Services users can locate the IP addresses of component VMs in the VPC Dashboard of the AWS Console.

2. SSH into the component VM:

```bash
$ ssh COMPONENT-VM-PRIVATE-IP
```
Pivotal Cloud Foundry Security Overview and Policy

Page last updated:

This document outlines our security policy and is addressed to operators deploying Pivotal Cloud Foundry (PCF) using Pivotal Cloud Foundry Operations Manager.

For a comprehensive overview of the security architecture of each PCF component, refer to the Cloud Foundry Security topic.

How Pivotal Monitors for Security Vulnerabilities

Pivotal receives private reports on vulnerabilities from customers and from field personnel via our secure disclosure process. We also monitor public repositories of software security vulnerabilities to identify newly discovered vulnerabilities that might affect one or more of our products.

How to Report a Vulnerability

Pivotal encourages users who become aware of a security vulnerability in our products to contact Pivotal with details of the vulnerability. Please send descriptions of any vulnerabilities found to security@pivotal.io. Please include details on the software and hardware configuration of your system so that we can reproduce the issue.

Note: We encourage use of encrypted email. Our public PGP key is located at http://www.pivotal.io/security.

Notification Policy

PCF has many customer stakeholders who need to know about security updates. When there is a possible security vulnerability identified for a PCF component, we do the following:

1. Assess the impact to PCF.
2. If the vulnerability would affect a PCF component, we schedule an update for the impacted component(s).
3. Update the affected component(s) and perform system tests.
4. Announce the fix publicly via the following channels:
   a. Automated notification to end users who have downloaded or subscribed to a PCF product on Pivotal Network when a new, fixed version is available.

Classes of Vulnerabilities

Attackers can exploit vulnerabilities to compromise user data and processing resources. This can affect data confidentiality, integrity, and availability to different degrees. For vulnerabilities related to Ubuntu provided packages, Pivotal follows Canonical’s priority levels. For other vulnerabilities, Pivotal follows Common Vulnerability Scoring System v3.0 standards when assessing severity.

Pivotal reports the severity of vulnerabilities using the following severity classes:

High

High severity vulnerabilities are those that can be exploited by an unauthenticated or authenticated attacker, from the Internet or those that break the guest/host Operating System isolation. The exploitation could result in the complete compromise of confidentiality, integrity, and availability of user data and/or processing resources without user interaction. Exploitation could be leveraged to propagate an Internet worm or execute arbitrary code between Virtual Machines and/or the Host Operating System. This rating also applies to those vulnerabilities that could lead to the complete compromise of availability when the exploitation is by a remote unauthenticated attacker from the Internet or through a breach of virtual machine isolation.

© Copyright Pivotal Software Inc, 2013-2018 282
Moderate

Moderate vulnerabilities are those in which the ability to exploit is mitigated to a significant degree by configuration or difficulty of exploitation, but in certain deployment scenarios could still lead to the compromise of confidentiality, integrity, or availability of user data and/or processing resources.

Low

Low vulnerabilities are all other issues that have a security impact. These include vulnerabilities for which exploitation is believed to be extremely difficult, or for which successful exploitation would have minimal impact.

Release Policy

PCF schedules regular releases of software in the PCF Suite to address Low / Medium severity vulnerability exploits. These patch releases take place during the first week each month. When High severity vulnerability exploits are identified, PCF releases fixes to software in the PCF Suite on-demand, with as fast a turnaround as possible.

Alerts/Actions Archive

http://www.pivotal.io/security
Cloud Foundry Concepts

Cloud Foundry is an open platform as a service, providing a choice of clouds, developer frameworks, and application services. Cloud Foundry makes it faster and easier to build, test, deploy and scale applications. It is an open source project and is available through a variety of private cloud distributions and public cloud instances.

This guide presents an overview of how Cloud Foundry works and a discussion of key concepts. Refer to this guide to learn more about Cloud Foundry fundamentals.

General Concepts

- Cloud Foundry Overview
- How Applications are Staged
- High Availability in Cloud Foundry
- Orgs, Spaces, Roles, and Permissions
- Understanding Cloud Foundry Security
- Using Docker in Cloud Foundry
- Understanding Application Security Groups

Architecture

- Cloud Foundry Components
- Cloud Controller
- Droplet Execution Agent
- Messaging (NATS)
- (Go)Router
- User Account and Authentication (UAA) Server
- Warden
- HTTP Routing

Diego

- Diego Architecture
- Differences Between DEA and Diego Architectures
- Understanding Application SSH
- How the Diego Auction Allocates Jobs
Cloud Foundry Overview

The Industry-Standard Cloud Platform

Cloud platforms let anyone deploy network apps or services and make them available to the world in a few minutes. When an app becomes popular, the cloud easily scales it to handle more traffic, replacing with a few keystrokes the build-out and migration efforts that once took months. Cloud platforms represent the next step in the evolution of IT, enabling you to focus exclusively on your applications and data without worrying about underlying infrastructure.

Not all cloud platforms are created equal. Some have limited language and framework support, lack key app services, or restrict deployment to a single cloud. Cloud Foundry (CF) has become the industry standard. It is an open source platform that you can deploy to run your apps on your own computing infrastructure, or deploy on an IaaS like AWS, vSphere, or OpenStack. You can also use a PaaS deployed by a commercial CF cloud provider. A broad community contributes to and supports Cloud Foundry. The platform's openness and extensibility prevent its users from being locked into a single framework, set of app services, or cloud.

Cloud Foundry is ideal for anyone interested in removing the cost and complexity of configuring infrastructure for their apps. Developers can deploy their apps to Cloud Foundry using their existing tools and with zero modification to their code.

How Cloud Foundry Works

To flexibly serve and scale apps online, Cloud Foundry has subsystems that perform specialized functions. Here's how some of these main subsystems work.

How the Cloud Balances Its Load

Clouds balance their processing loads over multiple machines, optimizing for efficiency and resilience against point failure. A Cloud Foundry installation accomplishes this at three levels:

1. **BOSH** creates and deploys virtual machines (VMs) on top of a physical computing infrastructure, and deploys and runs Cloud Foundry on top of this cloud. To configure the deployment, BOSH follows a manifest document.

2. The **CF Cloud Controller** runs the apps and other processes on the cloud's VMs, balancing demand and managing app lifecycles.

3. The **router** routes incoming traffic from the world to the VMs that are running the apps that the traffic demands, usually working with a customer-provided load balancer.

How Apps Run Anywhere

Cloud Foundry designates two types of VMs: the component VMs that constitute the platform's infrastructure, and the host VMs that host apps for the
outside world. Within CF, the Diego system distributes the hosted app load over all of the host VMs, and keeps it running and balanced through demand surges, outages, or other changes. Diego accomplishes this through an auction algorithm.

To meet demand, multiple host VMs run duplicate instances of the same app. This means that apps must be portable. Cloud Foundry distributes app source code to VMs with everything the VMs need to compile and run the apps locally. This includes the OS stack that the app runs on, and a buildpack containing all languages, libraries, and services that the app uses. Before sending an app to a VM, the Cloud Controller stages it for delivery by combining stack, buildpack, and source code into a droplet that the VM can unpack, compile, and run. For simple, standalone apps with no dynamic pointers, the droplet can contain a pre-compiled executable instead of source code, language, and libraries.

How CF Organizes Users and Workspaces

To organize user access to the cloud and to control resource use, a cloud operator defines Orgs and Spaces within an installation and assigns Roles such as admin, developer, or auditor to each user. The User Authentication and Authorization (UAA) server supports access control as an OAuth2 service, and can store user information internally or connect to external user stores through LDAP or SAML.

Where CF Stores Resources

Cloud Foundry uses the git system on GitHub to version-control source code, buildpacks, documentation, and other resources. Developers on the platform also use GitHub for their own apps, custom configurations, and other resources. To store large binary files, such as droplets, CF maintains an internal or external blobstore. To store and share temporary information, such as internal component states, CF uses the distributed value-store systems Consul and etcd.

How CF Components Communicate

Cloud Foundry components communicate with each other by posting messages internally using http and https protocols, and by sending NATS messages to each other directly.

How to Monitor and Analyze a CF Deployment

As the cloud operates, the Cloud Controller VM, router VM, and all VMs running apps continuously generate logs and metrics. The Loggregator system aggregates this information in a structured, usable form, the Firehose. You can use all of the output of the Firehose, or direct the output to specific uses, such as monitoring system internals or analyzing user behavior, by applying nozzles.

Using Services with CF

Typical apps depend on free or metered services such as databases or third-party APIs. To incorporate these into an app, a developer writes a Service Broker, an API that publishes to the Cloud Controller the ability to list service offerings, provision the service, and enable apps to make calls out to it.

How Pivotal Cloud Foundry Differs from Open Source Cloud Foundry

Open source software provides the basis for the Pivotal Cloud Foundry platform. Elastic Runtime is the Pivotal distribution of Cloud Foundry software for hosting apps. Pivotal offers additional commercial features, enterprise services, support, docs, certs, etc.
How Applications Are Staged

Cloud Foundry has used two architectures for managing application containers: Diego and Droplet Execution Agents (DEAs). For information about how DEA applications are staged, see the Staging Apps with DEAs section of the Droplet Execution Agent topic.

This topic describes how the Diego architecture stages buildpack applications and Docker images.

How Diego Stages Buildpack Applications

1. At the command line, the developer enters the directory containing her application and uses the Cloud Foundry Command Line Interface (cf CLI) to issue a push command.

2. The cf CLI tells the Cloud Controller to create a record for the application.

3. The Cloud Controller stores the application metadata. Application metadata can include the app name, number of instances the user specified, and the buildpack, and other information about the application.

4. Before uploading all the application files, the cf CLI issues a resource match request to the Cloud Controller to determine if any of the application files already exist in the resource cache. When the application files are uploaded, the cf CLI omits files that exist in the resource cache by supplying the result of the resource match request. The uploaded application files are combined with the files from the resource cache to create the application package.

5. The Cloud Controller stores the application package in the blobstore.

6. The cf CLI issues an app start command.

7. The Cloud Controller issues a staging request to Diego, which then schedules a Cell to run the staging Task. The Task downloads buildpacks and if present, the app’s buildpack cache. It then uses the buildpack that is detected automatically or specified with the -b flag to build the droplet. The Task uses the instructions in the buildpack to stage the application.

8. The Diego Cell streams the output of the staging process so the developer can troubleshoot application staging problems.

9. The Task packages the resulting staged application into a tarball called a “droplet” and the Diego Cell stores it in the blobstore. The Task also uploads the buildpack cache to the blobstore for use the next time the application is staged.
10. The **Diego Bulletin Board System** reports to the Cloud Controller that staging is complete. Staging must complete within 15 minutes or the staging is considered failed. Apps are given a minimum of 1GB memory to stage, even if the requested running memory is smaller.

11. Diego schedules the application as a [Long Running Process](#) on one or more Diego Cells.

12. The Diego Cells report the status of the application to the Cloud Controller.

See the [Diego Architecture](#) topic for more information.

### How Diego Stages Docker Images

1. At the command line, the developer enters the name of a Docker image in an accessible Docker Registry and uses the cf CLI to issue a push command.

2. The cf CLI tells the Cloud Controller to create a record for the Docker image.

3. The Cloud Controller issues a staging request to Diego, which then schedules a Cell to run the staging Task.

4. The Diego Cell streams the output of the staging process so the developer can troubleshoot staging problems.

5. The Task fetches the metadata associated with the Docker image and returns a portion of it to the Cloud Controller, which stores it in the Cloud Controller database (CCDB).

6. The Cloud Controller uses the Docker image metadata to construct a Long Running Process that runs the start command specified in the Dockerfile. The Cloud Controller also takes into account any user-specified overrides specified in the Dockerfile, such as custom environment variables.

7. The Cloud Controller submits the Long Running Process to Diego. Diego schedules the Long Running Process on one or more Diego Cells.

8. The Cloud Controller instructs Diego and the Gorouter to route traffic to the Docker image.
High Availability in Cloud Foundry

This topic explains how to configure Cloud Foundry for high availability (HA) and how Cloud Foundry is designed to ensure HA at multiple layers.

Configuring High Availability

This section describes how to configure system components to ensure high availability. You accomplish this by scaling component VMs and locating them in multiple Availability Zones (AZs), so that their redundancy and distribution minimizes downtime during ongoing operation, product updates, and platform upgrades.

Scaling component VMs means changing the number of VM instances dedicated to running a functional component of the system. Scaling usually means increasing this number, while scaling down or scaling back means decreasing it.

Deploying or scaling applications to at least two instances per app also helps maintain high availability. For information about scaling applications and maintaining app uptime, see Scaling an Application Using cf scale and Using Blue-Green Deployment to Reduce Downtime and Risk.

Availability Zones

During product updates and platform upgrades, the VMs in a deployment restart in succession, rendering them temporarily unavailable. During outages, VMs go down in a less orderly way. Spreading components across Availability Zones and scaling them to a sufficient level of redundancy maintains high availability during both upgrades and outages and can ensure zero downtime.

Deploying Cloud Foundry across three or more AZs and assigning multiple component instances to different AZ locations lets a deployment operate uninterrupted when entire AZs become unavailable. Cloud Foundry maintains its availability as long as a majority of the AZs remain accessible. For example, a three-AZ deployment stays up when one entire AZ goes down, and a five-AZ deployment can withstand an outage of up to two AZs with no impact on uptime.

Vertical and Horizontal Scaling

You can scale platform capacity vertically by adding memory and disk, or horizontally by adding more VMs running instances of Cloud Foundry components.

To scale vertically, ensure that you allocate and maintain enough of the following:

- Free space on host VMs, whether they are Diego cells or DEAs, so that apps expected to deploy can successfully be staged and run.
- Disk space and memory in your deployment such that if one host VM is down, all instances of apps can be placed on the remaining Host VMs.
- Free space to handle one AZ going down if deploying in multiple AZs.

Scaling up the following components horizontally also increases your capacity to host applications. The nature of the applications you host on Cloud Foundry should determine how you should scale vertically vs. horizontally.
Scalable Components

You can horizontally scale most Cloud Foundry components to multiple instances to achieve the redundancy required for high availability. You should also distribute the instances of multiply-scaled components across different availability zones (AZs). If you use more than three AZs, ensure that you use an odd number of AZs. For more information regarding zero downtime deployment, see the Scaling Instances in Elastic Runtime topic.

The following table provides recommended instance counts for a high-availability deployment:

<table>
<thead>
<tr>
<th>Job</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diego Cell</td>
<td>≥ 3 The optimal balance between CPU/memory sizing and instance count depends on the performance characteristics of the apps that run on Diego cells. Scaling vertically with larger Diego cells makes for larger points of failure, and more apps go down when a cell fails. On the other hand, scaling horizontally decreases the speed at which the system rebalances apps. Rebalancing 100 cells takes longer and demands more processing overhead than rebalancing 20 cells.</td>
</tr>
<tr>
<td>Diego Brain</td>
<td>≥ 2 One per AZ, or two if only one AZ.</td>
</tr>
<tr>
<td>Diego BBS</td>
<td>≥ 3 Set this to an odd number equal to or one greater than the number of AZs you have, in order to maintain quorum. Distribute the instances evenly across the AZs, at least one instance per AZ.</td>
</tr>
<tr>
<td>Consul</td>
<td>≥ 3 Set this to an odd number equal to or one greater than the number of AZs you have, in order to maintain quorum. Distribute the instances evenly across the AZs, at least one instance per AZ.</td>
</tr>
<tr>
<td>MySQL Server</td>
<td>3 You might run a single NATS instance if you lack the resources to deploy two stable NATS servers. Components using NATS are resilient to message failures and the BOSH resurrector recovers the NATS VM quickly if it becomes non-responsive.</td>
</tr>
<tr>
<td>MySQL Proxy</td>
<td>2</td>
</tr>
<tr>
<td>NATS Server</td>
<td>≥ 2</td>
</tr>
<tr>
<td>Cloud Controller</td>
<td>≥ 2 Scale the Cloud Controller to accommodate the number of requests to the API and the number of apps in the system.</td>
</tr>
<tr>
<td>Router</td>
<td>≥ 2 Scale the router to accommodate the number of incoming requests. Additional instances increase available bandwidth. In general, this load is much less than the load on Diego cells.</td>
</tr>
<tr>
<td>HAProxy</td>
<td>0 or 1</td>
</tr>
<tr>
<td>UAA</td>
<td>≥ 2</td>
</tr>
<tr>
<td>Doppler Server</td>
<td>≥ 2 Deploying additional Doppler servers splits traffic across them. Pivotal recommends to have at least two per Availability Zone.</td>
</tr>
<tr>
<td>Loggregator TC</td>
<td>≥ 2 Deploying additional Loggregator Traffic Controllers allows you to direct traffic to them in a round-robin manner. Pivotal recommends to have at least two per Availability Zone.</td>
</tr>
<tr>
<td>etcd</td>
<td>≥ 3 Set this to an odd number equal to or one greater than the number of AZs you have, in order to maintain quorum. Distribute the instances evenly across the AZs, at least one instance per AZ.</td>
</tr>
</tbody>
</table>

Blob Storage

For storing blobs, large binary files, the best approach for high availability is to use external storage such as Amazon S3 or an S3-compatible service.

If you store blobs internally using either NFS or WebDAV, these components run as single instances and you cannot scale them. For these deployments, use the high availability features of your IaaS to immediately recover your WebDAV or NFS Server VM if it fails. Contact Pivotal Support if you need assistance.

The singleton Collector and Compilation components do not affect platform availability.

Supporting Component Scaling

Ops Manager Resurrector

Enable the Ops Manager Resurrector.
Resource Pools

Configure your resource pools according to the requirements of your deployment.

Each IaaS has different ways of limiting resource consumption for scaling VMs. Consult with your IaaS administrator to ensure additional VMs and related resources, like IPs and storage, will be available when scaling.

For Amazon Web Services, review the documentation regarding scaling instances. If you are using OpenStack, see the topic regarding managing projects and users. For vSphere, review the Configuring Ops Manager Director for VMware vSphere topic.

Databases

For database services deployed outside Cloud Foundry, plan to leverage your infrastructure’s high availability features and to configure backup and restore where possible. For more information on scaling internal database components, see the Scaling Instances in Elastic Runtime topic.

Note: Data services may have single points of failure depending on their configuration.

Contact Pivotal Support if you need assistance.

How CF Maintains High Availability

This section explains how Pivotal Cloud Foundry (PCF) deployments include several layers of HA to keep applications running in the face of system failure. These layers include availability zones (AZs), application health management, process monitoring, and VM resurrection.

Availability Zones

PCF supports deploying applications instances across multiple AZs. This level of high availability requires that you define AZs in your IaaS. PCF balances the applications you deploy across the AZs you defined. If an AZ goes down, you still have application instances running in another.

You can configure your deployment so that Diego cells are created across these AZs. Follow the configuration for your specific IaaS (vSphere, OpenStack, or AWS).

Health Management for App Instances

If you lose application instances for any reason, such as a bug in the app or an AZ going down, PCF restarts new instances to maintain capacity. Under Diego architecture, the nsync, BBS, and Cell Rep components track the number of instances of each application that are running across all of the Diego cells. When these components detect a discrepancy between the actual state of the app instances in the cloud and the desired state as known by the Cloud Controller, they advise the Cloud Controller of the difference and the Cloud Controller initiates the deployment of new application instances.

Process Monitoring

PCF uses a BOSH agent, monit, to monitor the processes on the component VMs that work together to keep your applications running, such as nsync, BBS, and Cell Rep. If monit detects a failure, it restarts the process and notifies the BOSH agent on the VM. The BOSH agent notifies the BOSH Health Monitor, which triggers responders through plugins such as email notifications or paging.

Resurrection for VMs

BOSH detects if a VM is present by listening for heartbeat messages that are sent from the BOSH agent every 60 seconds. The BOSH Health Monitor listens for those heartbeats. When the Health Monitor finds that a VM is not responding, it passes an alert to the Resurrector component. If the Resurrector is enabled, it sends the IaaS a request to create a new VM instance to replace the one that failed.

To enable the Resurrector, see the following pages for your particular IaaS: AWS, OpenStack, or vSphere.
Orgs, Spaces, Roles, and Permissions

Page last updated:

PCF uses a role-based access control (RBAC) system to grant Elastic Runtime users permissions appropriate to their role within an org or a space. This topic describes how orgs and spaces work within a PCF deployment, and how different Elastic Runtime User roles operate within those contexts.

Admins, Org Managers, and Space Managers can assign user roles using the `cf CLI` or Apps Manager.

Note: Before you assign a space role to a user, you must assign an org role to the user.

Orgs

An org is a development account that an individual or multiple collaborators can own and use. All collaborators access an org with user accounts. Collaborators in an org share a resource quota plan, applications, services availability, and custom domains.

Before you assign a space role to a user, you must assign an org role to the user. The error message occurs when you try to set a space role before setting an org role for the user.

User Accounts

A user account represents an individual person within the context of a Cloud Foundry installation. A user can have different roles in different spaces within an org, governing what level and type of access they have within that space.

Spaces

Every application and service is scoped to a space. Each org contains at least one space. A space provides users with access to a shared location for application development, deployment, and maintenance. Each space role applies only to a particular space.

Roles and Permissions

A user can have one or more roles. The combination of these roles defines the user’s overall permissions in the org and within specific spaces in that org.

• Org Managers are managers or other users who need to administer the account.

Note: An Org Manager needs explicit administrator permissions to perform certain actions. Refer to the Creating and Managing Users with the UAA CLI (UAAC) topic to learn how to create a user with admin rights.

• Org Auditors view but cannot edit user information and org quota usage information.

• Space Managers are managers or other users who administer a space within an org.

• Space Developers are application developers or other users who manage applications and services in a space.

• Space Auditors view but cannot edit the space.

<table>
<thead>
<tr>
<th>User Role</th>
<th>Org Manager</th>
<th>Org Auditor</th>
<th>Space Manager</th>
<th>Space Developer</th>
<th>Space Auditor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope of operation</td>
<td>Org</td>
<td>Org</td>
<td>Space</td>
<td>Space</td>
<td>Space</td>
</tr>
<tr>
<td>Add and edit users and roles</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
</tr>
<tr>
<td>View users and roles</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Create and assign Org and Space quota plans</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View Org quota plans</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Create Orgs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>View Orgs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feature</td>
<td>Default</td>
<td>Notes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edit, rename, and delete Orgs</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Spaces</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View Spaces</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edit Spaces</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete Spaces</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename Spaces</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View the status, number of instances, service bindings, and resource use of applications</td>
<td>✔️ ✔️ ✔️ ✔️</td>
<td>Defaults to no. Yes if Elastic Runtime tile &gt; Settings tab &gt; Apps Manager section &gt; Enable Non Admin Role Management checkbox is selected.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add private domains†</td>
<td>✔️</td>
<td>No by default, unless feature flag <code>user_org_creation</code> is set to <code>true</code>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deploy, run, and manage applications</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiate and bind services to applications</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Associate routes†, instance counts, memory allocation, and disk limit of applications</td>
<td>✔️ ✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename applications</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create and manage Application Security Groups</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Defaults to no. Yes if Elastic Runtime tile > Settings tab > Apps Manager section > Enable Non Admin Role Management checkbox is selected.

† No by default, unless feature flag `user_org_creation` is set to `true`.  

© Copyright Pivotal Software Inc, 2013-2018
Understanding Cloud Foundry Security

Pivotal protects customers from security threats by minimizing network surface area, applying security controls, isolating customer applications and data in containers, and encrypting connections.

Cloud Foundry:

- Implements role-based access controls, applying and enforcing roles and permissions to ensure that users can only view and affect the spaces for which they have been granted access.
- Ensures security of application bits in a multi-tenant environment.
- Prevents possible denial of service attacks through resource starvation.

Before you read this document, you might want to review the general system architecture.

System Boundaries and Access

As the image below shows, in a typical deployment of Cloud Foundry, the components run on virtual machines (VMs) that exist within a VLAN. In this configuration, the only access points visible on a public network are a load balancer that maps to one or more Cloud Foundry routers and, optionally, a NAT VM and a jumpbox. Because of the limited number of contact points with the public internet, the surface area for possible security vulnerabilities is minimized.

Note: Pivotal recommends that you also install a NAT VM for outbound requests and a jumpbox to access the BOSH Director, though these access points are optional depending on your network configuration.
Protocols

All traffic from the public internet to the Cloud Controller and UAA happens over HTTPS. Inside the boundary of the system, components communicate over a publish-subscribe (pub-sub) message bus, NATS, and also HTTP.

BOSH

Operators deploy Cloud Foundry with BOSH. The BOSH Director is the core orchestrating component in BOSH: it controls VM creation and deployment, as well as other software and service lifecycle events. You use HTTPS to ensure secure communication to the BOSH Director.

Note: Pivotal recommends that you deploy the BOSH Director on a subnet that is not publicly accessible, and access the BOSH Director from a jumpbox on the subnet or through VPN.

BOSH includes the following functionality for security:

- Communicates with the VMs it launches over NATS. Because NATS cannot be accessed from outside Cloud Foundry, this ensures that published messages can only originate from a component within your deployment.
- Provides an audit trail through the `bosh tasks` command. This command shows all actions that an operator has taken with BOSH.
- Allows you to set up individual login accounts for each operator. BOSH operators have root access.

Note: BOSH does not encrypt data stored on BOSH VMs. Your IaaS might encrypt this data.
Authentication and Authorization

User Account and Authentication (UAA) is the central identity management service for the Elastic Runtime platform and its various components.

UAA acts as an OAuth 2.0 Authorization Server and issues access tokens for applications that request platform resources. The tokens are based on the JSON Web Token and are digitally signed by UAA.

Operators can configure the identity store in UAA. If users register an account with the Cloud Foundry platform, UAA acts as the user store and stores user passwords in the UAA database using bcrypt. UAA also supports connecting to external user stores through LDAP and SAML. Once an operator has configured the external user store, such as a corporate Microsoft Active Directory, users can use their LDAP credentials to gain access to the Cloud Foundry platform instead of registering a separate account. Alternatively, operators can use SAML to connect to an external user store and enable single sign-on for users into the Cloud Foundry platform.

Managing User Access with Role-Based Access Control

Applications that users deploy to Cloud Foundry exist within a space. Spaces exist within orgs. To view and access an org or a space, a user must be a member of it. Cloud Foundry uses role-based access control (RBAC), with each role granted permissions to either an org or a specified space. For more information about roles and permissions, refer to the Orgs, Spaces, Roles, and Permissions topic.

For more information, see Getting Started with the Apps Manager and Managing User Accounts and Permissions Using the Apps Manager.

Application Isolation with Containers

Each application deployed to Cloud Foundry runs within its own self-contained environment, a Garden container that isolates processes, memory, and the file system. Cloud Foundry operators can configure whether contained applications can directly interact with other applications or other Cloud Foundry system components.

Applications are typically allowed to invoke other applications in Cloud Foundry only by leaving the system and re-entering through the load balancer positioned in front of the Cloud Foundry routers. To isolate applications and control outgoing traffic, each Garden container uses a dedicated virtual network interface (VNI) that consists of a pair of ethernet addresses, one visible to the application instance running in the container, and the other visible to the host VM's root namespace. The pair is configured to use IPs in a small and static subnet.

This restrictive operating environment is designed for security and stability. If a spoofing attack bypasses the physical firewall for your deployment, Cloud Foundry network traffic rules help prevent the attack from accessing application containers.

Application Traffic

When an app instance starts, the cell (or DEA) on the host VM allocates an IP address and also assigns an arbitrary port to the application container. The application uses the PORT environment variable provided in the container environment to determine which port to listen on. Because the host assigns a random value to the PORT environment variable, the value is generally unique for each application instance.

A host VM has a single IP address. If you configure the deployment with the cluster on a VLAN, as recommended, then all traffic goes through the following levels of network address translation, as shown in the diagram below:

- **Inbound** requests flow from the load balancer through the router to the host cell (or DEA), then into the app container. The router determines which application instance receives each request.
- **Outbound** traffic flows from the app container to the cell (or DEA), then to the gateway on the cell's virtual network interface. This gateway might be a NAT to external networks depending on your IaaS.

© Copyright Pivotal Software Inc, 2013-2018
Network Traffic Rules

Administrators can configure rules to prevent system access from external networks and between internal components, and to restrict applications from establishing connections over the virtual network interface. Admins can configure these rules at two levels. Application Security Groups apply network traffic rules at the container level, and Network Properties of cells (or DEAs) apply them at the host VM level.

Application Security Groups

To target applications with specific network traffic rules, Application Security Groups (ASGs) define traffic rules for individual containers. The rules specify the protocols, addresses, and ports that are allowed for outgoing traffic. Because they are “allow” rules, their order of evaluation is unimportant when multiple application security groups apply to the same space, org, or deployment. The cell uses these rules to filter and log outbound network traffic.

When applications are first staging, they need traffic rules loose enough to let them pull resources in from across the network. Once they are running, the traffic rules can be more restrictive and secure. To distinguish between these two security needs, admins can define one ASG for when an app stages, and a different one for when it runs.

To provide granular control in securing a deployment, an administrator can assign ASGs to apply across a Cloud Foundry deployment, or to specific spaces or orgs within a deployment. You define ASGs on the cf CLI using `cf create-security-group` and `cf bind-security-group`.

Host-level Network Properties

Operators can configure the `allow_networks` and `deny_networks` parameters for DEAs to restrict communication between system components and applications. Any Cloud Foundry ASG configurations will overwrite these configurations. For more information, see DEA Network Properties.

Note: Pivotal recommends that you use Cloud Foundry ASGs to specify egress access rules for your applications. This functionality enables you to more securely restrict application outbound traffic to predefined routes.

Container Mechanics

Container isolation is achieved by namespacing kernel resources that would otherwise be shared. The intended level of isolation is set such that multiple containers present on the same host cannot detect each other. Every container includes a private root filesystem – each container has its own Process ID (PID), namespace, network namespace, and mount namespace.

This container filesystem is created by stacking a read-only base filesystem and a container-specific read-write filesystem, commonly known as an overlay filesystem. The read-only filesystem contains the minimal set of operating system packages and Garden-specific modifications common to all containers. Containers can share the same read-only base filesystem because all writes are applied to the read-write filesystem. The read-write filesystem is unique to each container and is created by formatting a large sparse file of a fixed size. This fixed size prevents the read-write filesystem from overflowing into unallocated space.

Resource control is managed using Linux control groups (`cgroups`) or Windows job objects. Associating each container with its own cgroup or job object limits the amount of memory that the container may use. Linux cgroups also require the container to use a fair share of CPU compared to the relative CPU share of other containers.

Note: BOSH does not support a RedHat Enterprise Linux OS stemcell. This is due to an inherent security issue with the way RedHat handles user namespaces and container isolation.

Security for Service Broker Integration

The Cloud Controller authenticates every request with the Service Broker API using HTTP or HTTPS, depending on which protocol that you specify during broker registration. The Cloud Controller rejects any broker registration that does not contain a username and password.

Service instances bound to an app contain credential data. Users specify the binding credentials for user-provided service instances, while third-party brokers specify the binding credentials for managed service instances. The VCAP_SERVICES environment variable contains credential information for any service bound to an app. Cloud Foundry constructs this value from encrypted data that it stores in the Cloud Controller Database (CCDB).
A third-party broker might offer a dashboard client in its catalog. Dashboard clients require a text string defined as `client_secret`. Cloud Foundry does not store this secret in the CCDB. Instead, Cloud Foundry passes the secret to the UAA component for verification using HTTP or HTTPS.

Software Vulnerability Management

Cloud Foundry manages software vulnerability using releases and BOSH stemcells. New Cloud Foundry releases are created with updates to address code issues, while new stemcells are created with patches for the latest security fixes to address any underlying operating system issues.

Ensuring Security for Application Artifacts

Cloud Foundry secures both the code and the configuration of an application using the following functionality:

- Application developers push their code using the Cloud Foundry API. Cloud Foundry secures each call to the CF API using the UAA and SSL.
- The Cloud Controller uses RBAC to ensure that only authorized users can access a particular application.
- The Cloud Controller stores the configuration for an application in an encrypted database table. This configuration data includes user-specified environment variables and service credentials for any services bound to the app.
- Cloud Foundry runs the app inside a secure container. For more information, see the Application Isolation with Containers section.
- Cloud Foundry operators can configure network traffic rules to control inbound communication to and outbound communication from an app. For more information, see the Network Traffic Rules section.

Security Event Logging and Auditing

For operators, Cloud Foundry provides an audit trail through the `bosh tasks` command. This command shows all actions that an operator has taken with the platform. Additionally, operators can redirect Cloud Foundry component logs to a standard syslog server using the `syslog_daemon_config` property in the `metron_agent` job of `cf-release`.

For users, Cloud Foundry records an audit trail of all relevant API invocations of an app. The CLI command `cf events` returns this information.

Recommendations for Running a Secure Deployment

To help run a secure deployment, Pivotal recommends the following:

- Configure UAA clients and users using a BOSH manifest. Limit and manage these clients and users as you would any other kind of privileged account.
- Deploy within a VLAN that limits network traffic to individual VMs. This reduces the possibility of unauthorized access to the VMs within your BOSH-managed cloud.
- Enable HTTPS for applications and SSL database connections to protect sensitive data transmitted to and from applications.
- Ensure that the jumpbox is secure, along with the load balancer and NAT VM.
- Encrypt stored files and data within databases to meet your data security requirements. Deploy using industry standard encryption and the best practices for your language or framework.
- Prohibit promiscuous network interfaces on the trusted network.
- Review and monitor data sharing and security practices with third-party services that you use to provide additional functionality to your application.
- Store SSH keys securely to prevent disclosure, and promptly replace lost or compromised keys.
- Use Cloud Foundry’s RBAC model to restrict your users’ access to only what is necessary to complete their tasks.
- Use a strong passphrase for both your Cloud Foundry user account and SSH keys.
- Use the IPsec add-on to encrypt IP data traffic within your deployment.

Note: The selected third-party broker controls how securely to communicate managed service credentials.
Using Docker in Cloud Foundry

This topic describes Docker image support in Cloud Foundry with Diego and outlines how Cloud Foundry uses Diego to run Docker images. For more information about Diego, see the Diego Architecture topic.

A Docker image consists of a collection of layers. Each layer consists of one or both of the following:

- Raw bits to download and mount. These bits form the file system.
- Metadata that describes commands, users, and environment for the layer. This metadata includes the ENTRYPOINT and CMD directives, and is specified in the Dockerfile.

Understanding How Garden-Linux Creates Containers

Diego uses Garden-Linux to construct Linux containers. Garden-Linux builds Linux containers with the same kernel resource isolation features used by all Linux containers: namespaces and cgroups.

Linux container creation requires that a file system is mounted as the root file system of the container. Garden-Linux supports mounting Docker images as root file systems for the containers it constructs. Garden-Linux fetches and caches the individual layers associated with a Docker image, then combines and mounts them as the root file system, using the same libraries that power Docker.

This process yields a container with contents that exactly match the contents of the associated Docker image.

Understanding Diego Process Determination and Monitoring

Once Garden-Linux creates a container, Diego runs and monitors the processes inside of it.

To determine which processes to run, the Cloud Controller fetches the metadata associated with the Docker image and returns it to the Cloud Controller. The Cloud Controller uses this metadata to perform the following actions:

- Runs the start command as the user specified in the Docker image
- Instructs Diego and the Gorouter to route traffic to the lowest-numbered port exposed in the Docker image

Note: When launching an application on Diego, the Cloud Controller honors any user-specified overrides such as a custom start command or custom environment variables.

Docker Security Concerns in a Multi-Tenant Environment

The attack surface area for a Docker-based container running on Diego remains somewhat higher than that of a buildpack application because Docker allows users to fully specify the contents of their root file systems. A buildpack application runs on a trusted root filesystem.

The Garden-Linux team has implemented a host of features to allow the platform to run Docker images more securely in a multi-tenant context. In particular, Cloud Foundry uses the user-namespacing feature found on modern Linux kernels to ensure that users cannot gain escalated privileges on the host even if they escalate privileges within a container.

The Cloud Controller always runs Docker containers on Diego with user namespaces enabled. This security restriction prevents certain features, such as the ability to mount FuseFS devices, from working in Docker containers.

To mitigate security concerns, Cloud Foundry recommends that you run only trusted Docker containers on the platform. By default, the Cloud Controller does not allow Docker-based applications to run on the platform.

To allow Docker-based applications to run, a Cloud Controller administrator can enable the diego_docker feature flag with the following command:

```bash
$ cf enable-feature-flag diego_docker
```

To disallow Docker-based applications, a Cloud Controller administrator can run the following command:

```bash
$ cf disable-feature-flag diego_docker
```
Pushing a Docker Image with the Cloud Foundry Command Line Interface (cf CLI)

Follow these instructions to deploy updated or new Docker images using Cloud Foundry Command Line Interface (cf CLI).

Ensure that you are running cf CLI 6.13.0 or a later version. See Installing the cf CLI for installation instructions.

Note: See Docker Support in CF + Diego for information about pushing a docker image with an earlier version of the cf CLI.

Pushing a Docker image using Docker Hub

Run
```
cf push lattice-app -o cloudfoundry/MY-DOCKER-IMAGE
```
to deploy a Docker image. Replace `MY-DOCKER-IMAGE` with the name of an image from an accessible Docker Registry.

For example, the following command pushes the `lattice-app` on Docker Hub to Cloud Foundry:
```
$ cf push lattice-app -o cloudfoundry/lattice-app
```

Pushing a Docker image using Docker Trusted Registries

To deploy a Docker image using Docker Trusted Registries, run:
```
$ cf push my-app -o MY-PRIVATE-REGISTRY.DOMAIN:5000/image/name:v2
```

Replace `MY-PRIVATE-REGISTRY.DOMAIN` with your domain name, which resolves to the private registry. In this example, 5000 is the port on which your private registry serves traffic. `image/name` is the name of the Docker image repository. `v2` is a tag in that repository that refers to a specific image.

Docker Caveats

This section contains known issues and limitations with running Docker images in Cloud Foundry.

- Diego supports running Docker images only from Docker registries that present the v2 registry API.
- Diego does not currently support fetching images from private repositories.
- Diego currently requires that the source registry for a Docker image be available when creating new application instances. If the registry is unavailable, Diego cannot start or restart applications.

Note: Disabling the `diego_docker` feature flag stops all Docker-based apps in your deployment within a few convergence cycles, on the order of a minute.
Understanding Application Security Groups

Page last updated:

This page assumes you are using cf CLI v6.4 or later.

Introduction

This topic provides an overview of Application Security Groups (ASGs), and describes how to manage and administer them. Many of the steps below require the Cloud Foundry Command Line Interface (cf CLI) tool.

Note: If you are creating ASGs for the first time, see Restricting App Access to Internal PCF Components.

Application Security Groups

Application Security Groups (ASGs) are a collections of egress rules that specify the protocols, ports, and IP address ranges where app or task instances send traffic. Because ASGs define allow rules, their order of evaluation is unimportant when multiple ASGs apply to the same space or deployment. The platform sets up rules to filter and log outbound network traffic from app and task instances. ASGs apply to both buildpack-based and Docker-based apps and tasks.

When apps or tasks begin staging, they need traffic rules permissive enough to allow them to pull resources from the network. After an app or task is running, the traffic rules can be more restrictive and secure. To distinguish between these two security requirements, administrators can define one ASG for app and task staging, and another for app and task runtime.

To provide granular control when securing a deployment, an administrator can assign ASGs to apply to all app and task instances for the entire deployment, or assign ASGs to spaces to apply only to apps and tasks in a particular space.

ASGs can be complicated to configure correctly, especially when the specific IP addresses listed in a group change. To simplify securing a deployment while still permitting apps reach external services, operators can deploy the services into a subnet that is separate from their Cloud Foundry deployment. Then the operators can create ASGs for the apps that whitelist those service subnets, while denying access to any virtual machine (VM) hosting other apps.

For examples of typical ASGs, see the Typical Application Security Groups section of this topic.

Default ASGs

Elastic Runtime defines one default ASG, default_security_group. This group allows all outbound traffic from application containers on public and private networks except for the link-local range, 169.254.0.0/16, which is blocked.

WARNING: For security, Elastic Runtime administrators must modify the default ASGs so that outbound network traffic cannot access internal components.

The ASG is defined in the Cloud Controller configuration as follows:

```json
security_group_definitions:
- name: default_security_group
  rules:
  - protocol: all
    destination: 0.0.0.0-169.253.255.255
  - protocol: all
    destination: 169.255.0.0-255.255.255.255
```

ASG Sets

ASGs are applied by configuring ASG sets differentiated by scope, platform-wide or space specific, and lifecycle, staging or running.

Currently, four ASG sets exist in Cloud Foundry:

- Platform-wide staging ASG set, also called “default-staging”
The following table indicates the differences between the four sets.

<table>
<thead>
<tr>
<th>When an ASG is bound to the...</th>
<th>the ASG rules are applied to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform-wide staging ASG set</td>
<td>the staging lifecycle for all apps and tasks.</td>
</tr>
<tr>
<td>Platform-wide running ASG set</td>
<td>the running lifecycle for all app and task instances.</td>
</tr>
<tr>
<td>Space-scoped staging ASG set</td>
<td>the staging lifecycle for apps and tasks in a particular space.</td>
</tr>
<tr>
<td>Space-scoped running ASG set</td>
<td>the running lifecycle for app and task instances in a particular space.</td>
</tr>
</tbody>
</table>

Typically, ASGs applied during the staging lifecycle are more permissive than the ASGs applied during the running lifecycle. This is because staging often requires access to different resources, such as dependencies.

You use different commands to apply an ASG to each of the four sets. For more information, see the Procedures section of this topic.

Note: To apply a staging ASG to apps within a space, you must use the CC API. The cf CLI command supports space-scoped running ASGs, but not space-scoped staging ASGs.

The Structure and Attributes of ASGs

ASG rules are specified as a JSON array of ASG objects. An ASG object has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>tcp, udp, icmp, or all</td>
<td>Required</td>
</tr>
<tr>
<td>destination</td>
<td>A single IP address, an IP address range like 192.0.2.0-192.0.2.50, or a CIDR block to allow network access to</td>
<td>Required when protocol is tcp or udp</td>
</tr>
<tr>
<td>ports</td>
<td>A single port, multiple comma-separated ports, or a single range of ports that can receive traffic. Examples: 443, 80,8080,8081, 8080-8081</td>
<td>Required when protocol is tcp or udp</td>
</tr>
<tr>
<td>code</td>
<td>ICMP code</td>
<td>Required when protocol is icmp</td>
</tr>
<tr>
<td>type</td>
<td>ICMP type</td>
<td>Required when protocol is icmp</td>
</tr>
<tr>
<td>log</td>
<td>Set to true to enable logging. For more information about how to configure system logs to be sent to a syslog drain, see the Using Log Management Services topic.</td>
<td>Logging is only supported when the protocol type is tcp.</td>
</tr>
<tr>
<td>description</td>
<td>An optional text field for operators managing security group rules</td>
<td></td>
</tr>
</tbody>
</table>

Process for Administering ASGs

The following table outlines the flow of tasks that the administrator carries out over the lifecycle of ASGs. Procedures for each of these tasks are given in Managing ASGs with the cf CLI below.

Note: If you are creating ASGs for the first time, see Restricting App Access to Internal PCF Components.

<table>
<thead>
<tr>
<th>Task</th>
<th>For more information, see</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Review the existing ASGs. If this is a new deployment, probably these are the Default ASGs alone.</td>
<td>View ASGs</td>
</tr>
<tr>
<td>2. Create new ASGs.</td>
<td>Create ASGs</td>
</tr>
<tr>
<td>3. Update the existing ASGs.</td>
<td>Update ASGs</td>
</tr>
<tr>
<td>4. Bind ASGs to an ASG set.</td>
<td>Bind ASGs</td>
</tr>
<tr>
<td>5. If you need to delete ASGs, unbind and delete them.</td>
<td>Unbind ASGs Delete ASGs</td>
</tr>
</tbody>
</table>
Managing ASGs with the cf CLI

This section provides the commands you need to create and manage ASGs.

View ASGs

Run the following cf CLI commands to view information about existing ASGs:

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>cf security-groups</td>
<td>All ASGs</td>
</tr>
<tr>
<td>cf staging-security-groups</td>
<td>All ASGs applied to the platform-wide staging ASG set</td>
</tr>
<tr>
<td>cf running-security-groups</td>
<td>All ASGs applied to the platform-wide running ASG set</td>
</tr>
<tr>
<td>cf security-group SECURITY-GROUP</td>
<td>All rules in the ASG named SECURITY-GROUP, for example, cf security-group dns</td>
</tr>
</tbody>
</table>

Create ASGs

To create an ASG, perform the following steps:

1. Create a rules file: a JSON-formatted single array containing objects that describe the rules. See the following example, which allows ICMP traffic of code 1 and type 0 to all destinations, and TCP traffic to 10.0.11.0/24 on ports 80 and 443. Also see The Structure and Attributes of ASGs.

   ```
   [  
   {  
   "protocol": "icmp",  
   "destination": "0.0.0.0/0",  
   "type": 0,  
   "code": 1  
   },  
   {  
   "protocol": "tcp",  
   "destination": "10.0.11.0/24",  
   "ports": "80,443",  
   "log": true,  
   "description": "Allow http and https traffic from ZoneA"  
   }  
   ]
   ```

2. Run `cf create-security-group SECURITY-GROUP PATH-TO-RULES-FILE`. Replace `SECURITY-GROUP` with the name of your security group, and `PATH-TO-RULES-FILE` with the absolute or relative path to a rules file.

   ```
   cf create-security-group my-asg ~/workspace/my-asg.json
   ```

After the ASG is created, you must bind it to an ASG set before it takes effect. See Bind ASGs below.

Bind ASGs

To apply an ASG, you must first bind it to an ASG set.

To bind an ASG to the platform-wide staging ASG set, run `cf bind-staging-security-group SECURITY-GROUP`. Replace `SECURITY-GROUP` with the name of your security group.

Example:

```
cf bind-staging-security-group my-asg
```
To bind an ASG to the platform-wide running ASG set, run `cf bind-running-security-group SECURITY-GROUP` command. Replace `SECURITY-GROUP` with the name of your security group.

Example:

```
$ cf bind-running-security-group my-asg
```

To bind an ASG to a space-scoped running ASG set, run `cf bind-security-group SECURITY-GROUP ORG SPACE`. Replace `SECURITY-GROUP` with the name of your security group. Replace `ORG` and `SPACE` with the org and space where you want to bind the ASG set.

Example:

```
$ cf bind-security-group my-asg my-org my-space
```

To bind an ASG to a space-scoped staging ASG set, run the following Cloud Controller (CC) API commands:

```
GET /v2/security_groups/:guid/staging_spaces
PUT /v2/spaces/:guid/staging_security_groups/:security_group_guid data
PUT /v2/security_groups/:guid/staging_spaces/:space_guid
DELETE /v2/spaces/:guid/staging_security_groups/:security_group_guid data
DELETE /v2/security_groups/:guid/staging_spaces/:space_guid
```

These API calls require administrator access. Additionally, the payload returned from API `GET` calls to `/v2/spaces/` and `/v2/spaces/:guid` includes a link to the `staging_security_groups_url`.

For more information about using these CC API commands, see the Cloud Foundry API documentation.

Update ASGs

To update an existing ASG, perform the following steps.

1. Edit the ASG rules in the JSON file.

2. Run `cf update-security-group SECURITY-GROUP PATH-TO-RULES-FILE`. Replace `SECURITY-GROUP` with the name of the existing ASG you want to change, and `PATH-TO-RULES-FILE` with the absolute or relative path to a rules file.

In the following example, `my-asg` is the name of a security group, and `~/workspace/my-asg-v2.json` is the path to a rules file.

```
$ cf update-security-group my-asg ~/workspace/my-asg-v2.json
```

**Note:** Updating an ASG does not affect started apps until you restart them. To restart all of the apps in an org or a space, use the app-restarter cf CLI plugin.

Unbind ASGs

**Note:** Unbinding an ASG does not affect started apps until you restart them. To restart all of the apps in an org or a space, use the app-restarter cf CLI plugin.

To unbind an ASG from the platform-wide staging ASG set, run `cf unbind-staging-security-group SECURITY-GROUP`. Replace `SECURITY-GROUP` with the name of your security group.

Example:

```
$ cf unbind-staging-security-group my-asg
```

To unbind an ASG from the platform-wide running ASG set, run `cf unbind-running-security-group SECURITY-GROUP`. Replace `SECURITY-GROUP` with the name of your security group.

Example:

```
$ cf unbind-running-security-group my-asg
```
of your security group.

Example:

```
$ cf unbind-running-security-group my-asg
```

To unbind an ASG from a specific space, run

```
$ cf unbind-security-group SECURITY-GROUP ORG SPACE
```

Replace `SECURITY-GROUP` with the name of your security group. Replace `ORG` and `SPACE` with the org and space where you want to unbind the ASG set.

Example:

```
$ cf unbind-security-group my-asg my-org my-space
```

Delete ASGs

Note: You can only delete unbound ASGs. To unbind ASGs, see Unbind ASGs above.

To delete an ASG, run

```
$ cf delete-security-group SECURITY-GROUP
```

Replace `SECURITY-GROUP` with the name of your security group.

Example:

```
$ cf delete-security-group my-asg
```

Typical ASGs

Below are examples of typical ASGs. Configure your ASGs in accordance with your organization’s network access policy for untrusted apps.

<table>
<thead>
<tr>
<th>ASG</th>
<th>For access to</th>
</tr>
</thead>
<tbody>
<tr>
<td>dns</td>
<td>DNS, either public or private</td>
</tr>
<tr>
<td>public-networks</td>
<td>Public networks, excluding IaaS metadata endpoints</td>
</tr>
<tr>
<td>private-networks</td>
<td>Private networks in accordance with RFC-1918</td>
</tr>
<tr>
<td>load-balancers</td>
<td>The internal Elastic Runtime load balancer and others</td>
</tr>
<tr>
<td>internal-proxy</td>
<td>Internal proxies</td>
</tr>
<tr>
<td>internal-databases</td>
<td>Internal databases</td>
</tr>
</tbody>
</table>

DNS

To resolve hostnames to IP addresses, apps require DNS server connectivity, which typically use port 53. Administrators should create or update a DNS ASG with appropriate rules. Administrators may further restrict the DNS servers to specific IP addresses or ranges of IP addresses.

Example `dns` ASG:

```
{
  "protocol": "tcp",
  "destination": "0.0.0.0/0",
  "ports": ["53"]
},
{
  "protocol": "udp",
  "destination": "0.0.0.0/0",
  "ports": ["53"]
}
```
Public Networks

Apps often require public network connectivity to retrieve app dependencies, or to integrate with services available on public networks. Example app dependencies include public Maven repositories, NPM, RubyGems, and Docker registries.

**Note:** You should exclude IaaS metadata endpoints, such as `169.254.169.254`, because the metadata endpoint can expose sensitive environment information to untrusted apps. The example below accounts for this recommendation.

```json
```

Private Networks

Network connections that are commonly allowable in private networks include endpoints such as proxy servers, Docker registries, load balancers, databases, messaging servers, directory servers, and file servers. Configure appropriate private network ASGs as appropriate. You may find it helpful to use a naming convention with `private_networks` as part of the ASG name, such as `private_networks_databases`.

**Note:** You should exclude any private networks and IP addresses that app and task instances should not have access to.

```json
```

Marketplace Services

Each installed Marketplace Service requires its own set of ASG rules to function properly. See the installation instructions for each installed Marketplace Service to determine which ASG rules it requires. For more information about how to provision and integrate services, see the Services Overview topics.

© Copyright Pivotal Software Inc, 2013-2018
About the ASG Creator Tool

The ASG Creator is a command line tool that you can use to create JSON rules files. The ASG Creator lets you specify IP addresses, CIDRs, and IP address ranges that you want to disallow traffic to, as well as the addresses that you want to allow traffic to. Based on these disallow/allow (exclude/include) lists that you provide as input, the ASG Creator formulates a JSON file of allow rules.

In turn, the JSON file is the input for the `cf bind-security-group` command that creates an ASG.

You can download the latest release of the ASG Creator from the Cloud Foundry incubator repository on Github: [https://github.com/cloudfoundry-incubator/asg-creator/releases/latest](https://github.com/cloudfoundry-incubator/asg-creator/releases/latest)

ASG Logging

The KB article [How to use Application Security Group (ASG) logging](https://kb.pivotal.io/articles/learning-your-pivotal-products/how-to-use-application-security-group-ASG-logging) describes how you can use ASGs to correlate emitted logs back to an app.
Cloud Foundry Components

Page last updated:

Cloud Foundry components include a self-service application execution engine, an automation engine for application deployment and lifecycle management, and a scriptable command line interface (CLI), as well as integration with development tools to ease deployment processes. Cloud Foundry has an open architecture that includes a buildpack mechanism for adding frameworks, an application services interface, and a cloud provider interface.

Refer to the descriptions below for more information about Cloud Foundry components. Some descriptions include links to more detailed documentation.

![Cloud Foundry Components Diagram]

Routing

Router

The router routes incoming traffic to the appropriate component, either a Cloud Controller component or a hosted application running on a Diego Cell.

The router periodically queries the Diego Bulletin Board System (BBS) for which cells and containers each application is currently running on. Then it recomputes new routing tables based on the IP addresses of each cell virtual machine (VM) and the host-side port numbers for the cell’s containers.

Authentication

OAuth2 Server (UAA) and Login Server

The OAuth2 server (the UAA) and Login Server work together to provide identity management.

App Lifecycle

Cloud Controller and Diego Brain

The Cloud Controller (CC) directs the deployment of applications. When a developer pushes an application to Cloud Foundry, she is targeting the Cloud Controller. The Cloud Controller then directs the Diego Brain through the CC-Bridge to coordinate individual Diego cells to stage and run applications.

In pre-Diego architecture, the Cloud Controller’s Droplet Execution Agent (DEA) performed these app lifecycle tasks.

The Cloud Controller also maintain records of orgs, spaces, user roles, services, and more.
nsync, BBS, and Cell Reps

To keep applications available, cloud deployments must constantly monitor their states and reconcile them with their expected states, starting and stopping processes as required. In pre-Diego architecture, the Health Manager (HM9000) performed this function. The nsync, BBS, and Cell Reps use a more distributed approach.

The nsync, BBS, and Cell Rep components work together along a chain to keep apps running. At one end is the user. At the other end are the instances of applications running on widely-distributed VMs, which may crash or become unavailable.

Here is how the components work together:

- **nsync** receives a message from the Cloud Controller when the user scales an app. It writes the number of instances into a DesiredLRP structure in the Diego BBS database.
- **BBS** uses its convergence process to monitor the DesiredLRP and ActualLRP values. It launches or kills application instances as appropriate to ensure the ActualLRP count matches the DesiredLRP count.
- **Cell Rep** monitors the containers and provides the ActualLRP value.

App Storage and Execution

**Blobstore**

The blobstore is a repository for large binary files, which Github cannot easily manage because Github is designed for code. Blobstore binaries include:

- Application code packages
- Buildpacks
- Droplets

You can configure the blobstore as either an internal server or an external S3 or S3-compatible endpoint. See this Knowledge Base article for more information about the blobstore.

**Diego Cell**

Application instances, application tasks, and staging tasks all run as Garden containers on the Diego Cell VMs. The Diego cell rep component manages the lifecycle of those containers and the processes running in them, reports their status to the Diego BBS, and emits their logs and metrics to Loggregator.

In pre-Diego CF architecture, the DEA node performed the task of managing the applications and containers on a VM.

**Services**

**Service Brokers**
Applications typically depend on services such as databases or third-party SaaS providers. When a developer provisions and binds a service to an application, the service broker for that service is responsible for providing the service instance.

**Messaging**

**Consul and BBS**

Cloud Foundry component VMs communicate with each other internally through HTTP and HTTPS protocols, sharing temporary messages and data stored in two locations:

- A Consul server stores longer-lived control data, such as component IP addresses and distributed locks that prevent components from duplicating actions.
- Diego’s Bulletin Board System (BBS) stores more frequently updated and disposable data such as cell and application status, unallocated work, and heartbeat messages. The BBS is currently implemented in etcd.

The route-emitter component uses the NATS protocol to broadcast the latest routing tables to the routers. In pre-Diego CF architecture, the NATS Message Bus carried all internal component communications.

**Metrics and Logging**

**Metrics Collector and Loggregator**

The metrics collector gathers metrics and statistics from the components. Operators can use this information to monitor a Cloud Foundry deployment.

The Loggregator (log aggregator) system streams application logs to developers.
HTTP Routing

Page last updated:

This topic describes features of HTTP routing handled by the Cloud Foundry (CF) router.

Sticky Sessions

The CF router supports session affinity or “sticky sessions” for incoming HTTP requests to compatible apps.

Sticky sessions means that when multiple instances of an app are running on CF, requests from a particular client always reach the same app instance. This allows apps to store session data specific to a user session.

- To support sticky sessions, configure your app to return a JSESSIONID cookie in responses. The app generates a JSESSIONID as a long hash in the following format:

```
1A530637289A03B779A44E8D551427
```

- If an app returns a JSESSIONID cookie to a client request, the CF routing tier generates a unique VCAP_ID for the app instance based on its GUID in the following format:

```
3235211e-fa53-4161-8d11-61539327913
```

- On subsequent requests, the client must provide both the JSESSIONID and VCAP_ID cookies.

The CF routing tier uses the VCAP_ID cookie to forward client requests to the same app instance every time. The JSESSIONID cookie is forwarded to the app instance to enable session continuity. If the app instance identified by the VCAP_ID crashes, the router attempts to route the request to a different instance of the app. If the router finds a healthy instance of the app, it initiates a new sticky session.

Note: CF does not persist or replicate HTTP session data across app instances. If an app instance crashes or is stopped, session data for that instance is lost. If you require session data to persist across crashed or stopped instances, or to be shared by all instances of an app, store session data in a CF marketplace service that offers data persistence.

HTTP Headers

HTTP traffic passed from the CF router to an app includes the following HTTP headers:

- X-Forwarded-Proto: gives the scheme of the HTTP request from the client. The scheme is HTTP if the client made an insecure request or HTTPS if the client made a secure request. Developers can configure their apps to reject insecure requests by inspecting the HTTP headers of incoming traffic and rejecting traffic that includes X-Forwarded-Proto with the scheme of HTTP.

- X-Forwarded-For: gives the IP address of the client originating the request.

If your load balancer terminates TLS upstream from the CF router, it must append these headers to requests forwarded to the CF router. For more information, see the Securing Traffic into Cloud Foundry topic.

SSL/TLS Termination

Depending on your needs, you can configure your deployment to terminate SSL/TLS at the CF router, the CF router and the load balancer, or the load balancer only. For more information, see the Securing Traffic into Cloud Foundry topic.

Transparent Retries

If the CF router cannot establish a TCP connection with a selected application instance, the router considers the instance ineligible for requests for 30 seconds, and the router transparently attempts to connect to another application instance. Once the router has established a TCP connection with an application instance, the router forwards the HTTP request.

See the Round-Robin Load Balancing topic below for more information about how the router forwards requests to application instances.
Round-Robin Load Balancing

The CF router uses the round-robin algorithm for load balancing incoming requests to application instances. The router maintains a dynamically updated list of application instances for each route, and forwards each request for a given route to the next application instance in the list.

WebSockets

WebSockets is a protocol providing bi-directional communication over a single, long-lived TCP connection, commonly implemented by web clients and servers. WebSockets are initiated via HTTP as an upgrade request. The CF Router supports this upgrade handshake, and will hold the TCP connection open with the selected application instance.

To support WebSockets, operators should configure their load balancer to pass WebSockets requests through as opaque TCP connections. If you are also terminating TLS at your load balancer, you may find that your load balancer does not support operating in TCP mode for some requests, and terminating TLS for others. Operators have the following options:

- Configure your load balancer to listen on a non-standard port (the built-in CF load balancer listens on 8443 by default for this purpose), and forward requests to this port in TCP mode. Application clients must make WebSockets upgrade requests to this port.
- Add a second load balancer listening in TCP mode on standard port 80. Configure DNS with a new hostname to be used for WebSockets. This hostname should resolve to the load balancer serving port 80 in TCP mode.
This topic provides an overview of the structure and components of Diego, the container management system for Pivotal Cloud Foundry versions 1.6 and newer.

Diego Architecture

Cloud Foundry has used two architectures for managing application containers: Droplet Execution Agents (DEA) and Diego. With the DEA architecture, the Cloud Controller schedules and manages applications on the DEA nodes. In the newer Diego architecture, Diego components replace the DEAs and the Health Manager (HM9000), and assume application scheduling and management responsibility from the Cloud Controller.

Refer to the following diagram and descriptions for information about the way Diego handles application requests.

1. The Cloud Controller passes requests to stage and run applications to the Cloud Controller Bridge (CC-Bridge).
2. The CC-Bridge translates staging and running requests into Tasks and Long Running Processes (LRPs), then submits these to the Bulletin Board System (BBS) through an API over HTTP.
3. The BBS submits the Tasks and LRPs to the Auctioneer, part of the Diego Brain.
4. The Auctioneer distributes these Tasks and LRPs to Cells through an Auction.

5. Once the Auctioneer assigns a Task or LRP to a Cell, an in-process Executor creates a Garden container in the Cell. The Task or LRP runs in the container.

6. The BBS tracks desired LRPs, running LRP instances, and in-flight Tasks. It also periodically analyzes this information and corrects discrepancies to ensure consistency between ActualLRP and DesiredLRP counts.

7. The Metron Agent, part of the Cell, forwards application logs, errors, and metrics to the Cloud Foundry Loggregator. For more information, see the Application Logging in Cloud Foundry topic.

Diego Core Components

Components in the Diego core run and monitor Tasks and LRPs. The core consists of the following major areas:

- **Brain**
- **Cells**
- **Database VMs**
- **Access VMs**
- **Consul**

Diego Brain

Diego Brain components distribute Tasks and LRPs to Diego Cells, and correct discrepancies between ActualLRP and DesiredLRP counts to ensure fault-tolerance and long-term consistency. The Diego Brain consists of the Auctioneer.

Auctioneer

- Uses the auction package to run Diego Auctions for Tasks and LRPs
- Communicates with Cell Reps over HTTP
- Maintains a lock in the BBS that restricts auctions to one Auctioneer at a time

Refer to the Auctioneer repo on GitHub for more information.

Diego Cell Components

Diego Cell components manage and maintain Tasks and LRPs.

Rep

- Represents a Cell in Diego Auctions for Tasks and LRPs
- Mediates all communication between the Cell and the BBS
- Ensures synchronization between the set of Tasks and LRPs in the BBS with the containers present on the Cell
- Maintains the presence of the Cell in the BBS
- Runs Tasks and LRPs by asking the in-process Executor to create a container and RunAction recipes

Refer to the Rep repo on GitHub for more information.

Executor

- Runs as a logical process inside the Rep
- Implements the generic Executor actions detailed in the API documentation
Streams `STDOUT` and `STDERR` to the Metron agent running on the Cell

Refer to the [Executor repo](https://github.com) on GitHub for more information.

**Garden**

- Provides a platform-independent server and clients to manage Garden containers
- Defines the `garden-linux` interface for container implementation

Refer to the [Garden repo](https://github.com) on GitHub for more information.

**Metron Agent**

Forwards application logs, errors, and application and Diego metrics to the Loggregator `Doppler` component

Refer to the [Metron repo](https://github.com) on GitHub for more information.

**Database VMs**

**Diego Bulletin Board System**

- Maintains a real-time representation of the state of the Diego cluster, including all desired LRPs, running LRP instances, and in-flight Tasks
- Provides an RPC-style API over HTTP to Diego Core components and external clients, including the SSH Proxy, CC-Bridge, and Route Emitter.
- Ensure consistency and fault tolerance for Tasks and LRPs by comparing desired state, stored in the database, with actual state, from running instances
- Acts to keep `DesiredLRP` count and `ActualLRP` count synchronized in the following ways:
  - If the `DesiredLRP` count exceeds the `ActualLRP` count, requests a start auction from the Auctioneer
  - If the `ActualLRP` count exceeds the `DesiredLRP` count, sends a stop message to the Rep on the Cell hosting an instance
- Limits the number of containers that can be concurrently in the starting state to 200. If your deployment requires more than 200 containers, containers after the first 200 will not be started until some in the first 200 leave the starting state.
- Monitors for potentially missed messages, resending them if necessary

For more information, see the [Bulletin Board System repo](https://github.com) on GitHub.

**etcd**

- Provides a consistent key-value data store to Diego

**Access VMs**

**File Server**

- This “blobstore” serves static assets that can include general-purpose App Lifecycle binaries and application-specific droplets and build artifacts.

Refer to the [File Server repo](https://github.com) on GitHub for more information.

**SSH Proxy**

- Brokers connections between SSH clients and SSH servers running inside instance containers

Refer to [Understanding Application SSH, Application SSH Overview](https://github.com), or the [Diego SSH Github repo](https://github.com) for more information.
Consul

- Provides dynamic service registration and load balancing through DNS resolution
- Provides a consistent key-value store for maintenance of distributed locks and component presence

Refer to the Consul repo on GitHub for more information.

Consuladapter

Consuladapter provides a driver for interfacing with etcd.

Refer to the Consuladapter repo on GitHub for more information.

Cloud Controller Bridge Components

The Cloud Controller Bridge (CC-Bridge) components translate app-specific requests from the Cloud Controller to the BBS. These components include the following:

Stager

- Translates staging requests from the Cloud Controller into generic Tasks and LRP
- Sends a response to the Cloud Controller when a Task completes

Refer to the Stager repo on GitHub for more information.

CC-Uploader

- Mediates uploads from the Executor to the Cloud Controller
- Translates simple HTTP POST requests from the Executor into complex multipart-form uploads for the Cloud Controller

Refer to the CC-Uploader repo on GitHub for more information.

Nsync

- Listens for app requests to update the DesiredLRPs count and updates DesiredLRPs through the BBS
- Periodically polls the Cloud Controller for each app to ensure that Diego maintains accurate DesiredLRPs counts

Refer to the Nsync repo on GitHub for more information.

TPS

- Provides the Cloud Controller with information about currently running LRP to respond to cf apps and cf app APP_NAME requests
- Monitors ActualLRP activity for crashes and reports them the Cloud Controller

Refer to the TPS repo on GitHub for more information.

Platform-specific Components

Garden Backends

Garden contains a set of interfaces that each platform-specific backend must implement. These interfaces contain methods to perform the following actions:
• Create and delete containers
• Apply resource limits to containers
• Open and attach network ports to containers
• Copy files into and out of containers
• Run processes within containers
• Stream STDOUT and STDERR data out of containers
• Annotate containers with arbitrary metadata
• Snapshot containers for redeploy without downtime

Refer to the Garden repo on GitHub for more information.

Current Implementations
• Garden-Linux provides a Linux-specific implementation of a Garden interface.

App Lifecycle Binaries

The following three platform-specific binaries deploy applications and govern their lifecycle:

• The Builder, which stages a CF application. The CC-Bridge runs the Builder as a Task on every staging request. The Builder performs static analysis on the application code and does any necessary pre-processing before the application is first run.
• The Launcher, which runs a CF application. The CC-Bridge sets the Launcher as the Action on the DesiredLRP for the application. The Launcher executes the start command with the correct system context, including working directory and environment variables.
• The Healthcheck, which performs a status check on running CF application from inside the container. The CC-Bridge sets the Healthcheck as the Monitor action on the DesiredLRP for the application.

Current Implementations
• Buildpack App Lifecycle implements the Cloud Foundry buildpack-based deployment strategy.
• Docker App Lifecycle implements a Docker deployment strategy.

Other Components

Route-Emitter

• Monitors DesiredLRP and ActualLRP states, emitting route registration and unregistration messages to the Cloud Foundry router when it detects changes
• Periodically emits the entire routing table to the Cloud Foundry router

Refer to the Route-Emitter repo on GitHub for more information.
Differences Between DEA and Diego Architectures

This topic describes components and functions that changed significantly when Cloud Foundry migrated to Diego architecture for version 1.5. This information will inform those who are familiar with Cloud Foundry’s DEA-based architecture and want to learn what has changed under Diego and how its new or changed components work.

Key Differences

The DEA architecture system is largely written in Ruby and the Diego architecture system is written in Go. When Cloud Foundry contributors decided to migrate the system’s core code from Ruby to Go, the rewrite offered the opportunity to make improvements to Cloud Foundry’s overall design.

In a pre-Diego Cloud Foundry deployment, the Cloud Controller’s Droplet Execution Agent (DEA) scheduled and managed applications on DEA nodes while the Health Manager (HM9000) kept them running. The Diego system assumes application scheduling and management responsibility from the Cloud Controller, replacing the DEA and Health Manager.

DEA architecture made no distinction between machine jobs that run once and jobs that run continuously. Diego recognizes the difference and uses it to allocate jobs to virtual machines (VMs) more efficiently, replacing the DEA Placement Algorithm with the Diego Auction.

In addition to these broad changes, the Cloud Foundry migration to Diego architecture includes smaller changes and renamings. The following sections describe pre-Diego components and their newer analogs, and the table provides a summary.

Changed Components and Functions

DEA Node → Diego Cell

The pre-Diego Droplet Execution Agent (DEA) node component managed application instances, tracked started instances, and broadcast state messages on each application VM. These functions are now performed by the Diego cell.

Warden → Garden

Pre-Diego application instances lived inside Warden containers, which are analogous to Garden containers in Diego architecture. Containerization ensures that application instances run in isolation, get their fair share of resources, and are protected from “noisy neighbors,” or other applications running on the same machine.

Warden could only manage containers on VMs running Linux, but the Garden subsystem supports VMs running diverse operating systems. The Garden front end presents the same container management operations that Warden used, with code that is abstracted away from any platform specifics. A platform-specific Garden Backend running on each VM translates the commands into machine code tailored to the native operating system.

The Diego SSH package enables developers to log into containers and access running application instances, a functionality that did not exist pre-Diego.

Warden Container-Level Traffic Rules

For network security, pre-Diego releases of Cloud Foundry supported allow and deny rules that governed outbound traffic from all Warden containers running on the same DEA node. Newer releases use container-specific Application Security Groups (ASGs) to restrict traffic at a more granular level. Cloud Foundry recommends using ASGs exclusively, but when a pre-Diego deployment defined both Warden rules and ASGs, they were evaluated in a strict priority order.

Pre-Diego Cloud Foundry returned an allow, deny, or reject result for the first rule that matched the outbound traffic request parameters, and did not evaluate any lower-priority rules. Cloud Foundry evaluated the network traffic rules for an application in the following order:

1. Security Groups: The rules described by the Default Staging set, the Default Running set, and all security groups bound to the space.

2. Warden allow rules: Any Warden Server configuration allow rules. Set Warden Server configuration rules in the Droplet Execution Agent (DEA) configuration section of your deployment manifest.

4. **Hard-coded reject rule**: Cloud Foundry returns a reject result for all outbound traffic from a container if not allowed by a higher-priority rule.

---

**Health Manager (HM9000) → nsync, BBS, and Cell Rep**

The function of the Health Manager (HM9000) component in pre-Diego releases of Cloud Foundry was replaced by the coordinated actions of the nsync, BBS, and Cell Reps. In pre-Diego architecture, the Health Manager (HM9000) had four core responsibilities:

- Monitor applications to determine their state (e.g. running, stopped, crashed, etc.), version, and number of instances. HM9000 updates the actual state of an application based on heartbeats and `droplet.exited` messages issued by the DEA node running the application.
- Determine applications’ expected state, version, and number of instances. HM9000 obtains the desired state of an application from a dump of the Cloud Controller database.
- Reconcile the actual state of applications with their expected state. For instance, if fewer than expected instances are running, HM9000 will instruct the Cloud Controller to start the appropriate number of instances.
- Direct Cloud Controller to take action to correct any discrepancies in the state of applications.

HM9000 was essential to ensuring that apps running on Cloud Foundry remained available. HM9000 restarted applications whenever the DEA node running an app shut down for any reason, when Warden killed the app because it violated a quota, or when the application process exited with a non-zero exit code.

Refer to the HM9000 readme for more information about the HM9000 architecture.

---

**DEA Placement Algorithm → Diego Auction**

In pre-Diego architecture, the Cloud Controller used the DEA Placement Algorithm to select the host DEA nodes for application instances that needed hosting.

Diego architecture moves this allocation process out of the Cloud Controller and into the Diego Brain, which uses the Diego Auction algorithm. The Diego Auction prioritizes one-time tasks like staging apps without affecting the uptime of ongoing, running applications like web servers.

---

**Message Bus (NATS)**

Pre-Diego Cloud Foundry used NATS, a lightweight publish-subscribe and distributed queueing messaging system, for internal communication between components. Diego retains NATS for some communications, but adds messaging via HTTP and HTTPS protocols, through which components share information in the Consul and Diego BBS servers.

---

**DEA / Diego Differences Summary**

<table>
<thead>
<tr>
<th>DEA architecture</th>
<th>Diego architecture</th>
<th>Function</th>
<th>Δ notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 and below</td>
<td>1.6 and above</td>
<td>CF version numbers</td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td>Go</td>
<td>Source code language</td>
<td></td>
</tr>
<tr>
<td>DEA</td>
<td>Diego Brain</td>
<td>High-level coordinator that allocates processes to containers in application VMs and keeps them running</td>
<td>DEA is part of the Cloud Controller. Diego is outside the Cloud Controller.</td>
</tr>
<tr>
<td>DEA Node</td>
<td>Diego Cell</td>
<td>Mid-level manager on each VM that runs apps as directed and communicates “heartbeat”, application status and container location, and other messages</td>
<td>Runs on each VM that hosts apps, as opposed to special-purpose component VMs.</td>
</tr>
<tr>
<td>Warden</td>
<td>Garden</td>
<td>Low-level manager and API protocol on each VM for creating, configuring, destroying, monitoring, and addressing application containers</td>
<td>Warden is Linux-only. Garden uses platform-specific Garden-backends to run on multiple OS.</td>
</tr>
<tr>
<td>DEA Placement Algorithm</td>
<td>Diego Auction</td>
<td>Algorithm used to allocate processes to VMs</td>
<td>Diego Auction distinguishes between Task and Long-Running Process (LRP) job types</td>
</tr>
<tr>
<td>Health Manager</td>
<td>nsync, BBS, and Cell Reps</td>
<td>System that monitors application instances and keeps instance counts in sync with the number that should be</td>
<td>nSync syncs between Cloud Controller and Diego, BBS syncs within Diego, and Cell Reps</td>
</tr>
<tr>
<td>NATS Message Bus</td>
<td>Bulletin Board System (BBS) and Consul via http/s, and NATS</td>
<td>running</td>
<td>sync between cells and the Diego BBS. BBS stores most runtime data; Consul stores control data.</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
Understanding Application SSH

This document describes details about the Elastic Runtime SSH components for access to deployed application instances. Elastic Runtime supports native SSH access to applications and load balancing of SSH sessions with the load balancer for your Elastic Runtime deployment.

The SSH Overview document describes procedural and configuration information on application SSH access.

SSH Components

The Elastic Runtime SSH includes the following central components, which are described in more detail below:

- An implementation of an SSH proxy server.
- A lightweight SSH daemon.

If these components are deployed and configured correctly, they provide a simple and scalable way to access containers apps and other long running processes (LRPs).

SSH Daemon

The SSH daemon is a lightweight implementation that is built around the Go SSH library. It supports command execution, interactive shells, local port forwarding, and secure copy. The daemon is self-contained and has no dependencies on the container root file system.

The daemon is focused on delivering basic access to application instances in Elastic Runtime. It is intended to run as an unprivileged process, and interactive shells and commands will run as the daemon user. The daemon only supports one authorized key, and it is not intended to support multiple users.

The daemon can be made available on a file server and Diego LRPs that want to use it can include a download action to acquire the binary and a run action to start it. Elastic Runtime applications will download the daemon as part of the lifecycle bundle.

SSH Proxy Authentication

The SSH proxy hosts the user-accessible SSH endpoint and is responsible for authentication, policy enforcement, and access controls in the context of Elastic Runtime. After a user has successfully authenticated with the proxy, the proxy will attempt to locate the target container and create an SSH session to a daemon running inside the container. After both sessions have been established, the proxy will manage the communication between the user’s SSH client and the container’s SSH Daemon.
How the Diego Auction Allocates Jobs

Page last updated:

The Diego Auction balances application processes, also called jobs, over the virtual machines (VMs) in a Cloud Foundry installation. When new processes need to be allocated to VMs, the Diego Auction determines which ones should run on which machines. The auction algorithm balances the load on VMs and optimizes application availability and resilience. This topic explains how the Diego Auction works at a conceptual level.

The Diego Auction replaces the Cloud Controller DEA placement algorithm, which performed the function of allocating processes to VMs in the pre-Diego Cloud Foundry architecture.

Refer to the Auction repo on GitHub for source code and more information.

Tasks and Long-Running Processes

The Diego Auction distinguishes between two types of jobs: Tasks and Long-Running Processes (LRPs).

- **Tasks** run once, for a finite amount of time. A common example is a staging task that compiles an app’s dependencies, to form a self-contained droplet that makes the app portable and runnable on multiple VMs. Other examples of tasks include making a database schema change, bulk importing data to initialize a database, and setting up a connected service.

- **Long-Running Processes** run continuously, for an indefinite amount of time. LRPs terminate only if stopped or killed, or if they crash. Examples include web servers, asynchronous background workers, and other applications and services that continuously accept and process input. To make high-demand LRPs more available, Diego may allocate multiple instances of the same application to run simultaneously on different VMs, often spread across Availability Zones that serve users in different geographic regions.

The Diego Auction process repeats whenever new jobs need to be allocated to VMs. Each auction distributes a current batch of work, Tasks and LRPs, that can include newly-created jobs, jobs left unallocated in the previous auction, and jobs left orphaned by failed VMs. Diego does not redistribute jobs that are already running on VMs. Only one auction can take place at a time, which prevents placement collisions.

Ordering the Auction Batch

The Diego Auction algorithm allocates jobs to VMs to fulfill the following outcomes, in decreasing priority order:

1. Keep at least one instance of each LRP running.
2. Run all of the Tasks in the current batch.
3. Distribute as much of the total desired LRP load as possible over the remaining available VMs, by spreading multiple LRP instances broadly across VMs and their Availability Zones.

To achieve these outcomes, each auction begins with the Diego Auctioneer component arranging the batch’s jobs into a priority order. Some of these jobs may be duplicate instances of the same process that Diego needs to allocate for high-traffic LRPs, to meet demand. So the Auctioneer creates a list of multiple LRP instances based on the desired instance count configured for each process.

For example, if the process LRP-A has a desired instance count of 3 and a memory load of 2, and process LRP-B has 2 desired instances and a load of 5, the Auctioneer creates a list of jobs for each process as follows:

<table>
<thead>
<tr>
<th>Process</th>
<th>Desired Instances</th>
<th>Load</th>
<th>Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRP-B</td>
<td>2</td>
<td>5</td>
<td>![LRP-B-1, LRP-B-2]</td>
</tr>
</tbody>
</table>

The Auctioneer then builds an ordered sequence of LRP instances by cycling through the list of LRPs in decreasing order of load. With each cycle, it adds another instance of each LRP to the sequence, until all desired instances of the LRP have been added. With the example above, the Auctioneer would order the LRPs like this:
The Auctioneer then builds an ordered sequence for all jobs, both LRPs and Tasks. Reflecting the auction batch priority order, the first instances of LRPs are first priority. Tasks are next, in decreasing order of load. Duplicate LRP jobs come last.

Adding one-time Task-C (load = 4) and Task-D (load = 3) to the above example, the priority order becomes:

```
Auction Sequence

Priority Group 1
LRP-B.1
LRP-A.1

Priority Group 2
Task-C
Task-D

Priority Group 3
LRP-B.2
LRP-A.2
LRP-A.3
```

**Auctioning the Batch to the Cells**

With all jobs sorted in priority order, the Auctioneer allocates each in turn to one of the VMs. The process resembles an auction, where VMs “bid” with their suitability to run each job. Facilitating this process, each app VM has a resident Cell that monitors and allocates the machine’s operation. The Cell participates in the auction on behalf of the virtual machine that it runs on.

Starting with the highest-priority job in the ordered sequence, the Auctioneer polls all the Cells on their fitness to run the currently-auctioned job. Cells “bid” to host each job according to the following priorities, in decreasing order:

1. Allocate all jobs only to Cells that have the correct software stack to host them, and sufficient resources given their allocation so far during this auction.
2. Allocate LRP instances into Availability Zones that are not already hosting other instances of the same LRP.
3. Within each Availability Zone, allocate LRP instances to run on Cells that are not already hosting other instances of the same LRP.
4. Allocate any job to the Cell that has lightest load, from both the current auction and jobs it has been running already. In other words, distribute the total load evenly across all Cells.

Our example auction sequence has seven jobs: five LRP instances and two Tasks. The following diagram shows how the Auctioneer might distribute this work across four Cells running in two Availability Zones:
If the Auctioneer reaches the end of its sequence of jobs, having distributed all jobs to the Cells, it submits requests to the Cells to execute their allotted work. If the Cells ran out of capacity to handle all jobs in the sequence, the Auctioneer carries the unallocated jobs over and merges them into the next auction batch, to be allocated in the next auction.

**Triggering Another Auction**

The Diego Auction process repeats to adapt a Cloud Foundry deployment to its changing workload. For example, the Cloud Controller of initiates a new auction when it detects that the actual number of running instances of LRPs does not match the number desired. The Cloud Controller’s BBS component monitors the number of instances of each LRP that are currently running. The BBS component periodically compares this number with the desired number of LRP instances, as configured by the user. If the actual number falls short of what is desired, the BBS triggers a new auction. In the case of a surplus of application instances, the BBS kills the extra instances and initiates another auction.

The Cloud Controller also triggers an auction whenever a Cell fails. After any auction, if a Cell responds to its work request with a message that it cannot perform the work after all, the Auctioneer carries the unallocated work over into the next batch. But if the Cell fails to respond entirely, for example if its connection times out, the unresponsive Cell may still be running its work. In this case, the Auctioneer does not automatically carry the Cell’s work over to the next batch. Instead, the Auctioneer defers to the BBS to continue monitoring the states of the Cells, and to re-assign unassigned work later if needed.
Operator's Guide
This guide covers networking and user management for Pivotal Cloud Foundry (PCF) operators.

Table of Contents

- Understanding the Elastic Runtime Network Architecture
- Identifying the API Endpoint for your Elastic Runtime Instance
- Creating New Elastic Runtime User Accounts
- Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments
- Using Your Own Load Balancer
- Configuring Proxy Settings for All Applications
- Restricting App Access to Internal PCF Components
- Configuring Application Security Groups for Email Notifications
- Configuring SSN Access for PCF
- Identifying Elastic Runtime Jobs Using vCenter
- Understanding the Effects of Single Components on a Pivotal Cloud Foundry Upgrade
- Configuring System Logging in Elastic Runtime
- Configuring UAA Password Policy
- Configuring Authentication and Enterprise SSO for Elastic Runtime
- Adding Existing SAML or LDAP Users to a Pivotal Cloud Foundry Deployment
- Switching Application Domains
- Scaling Elastic Runtime
- Scaling Down Your MySQL Cluster
- Using Docker Registries
- Custom Branding Apps Manager
- Monitoring App and Service Instance Usage
- Deploying Diego for Windows
- Operating Diego for Windows
- The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment
- Providing a Certificate for your SSL Termination Point
Understanding the Elastic Runtime Network Architecture

The diagram below shows the key Pivotal Cloud Foundry (PCF) Elastic Runtime network components.

### Load Balancer

Elastic Runtime includes an HAProxy load balancer for terminating SSL. If you do not want to serve SSL certificates for Elastic Runtime on your own load balancer use the HAProxy. If you do choose to manage SSL yourself, omit the HAProxy by setting the number of instances to zero in Ops Manager.

### Router

The routers in Elastic Runtime are responsible for routing HTTP requests from web clients to application instances in a load balanced fashion. The routers are dynamically configured based on users mapping of applications to location URLs called routes, and updated by the runtime service as application instances are dynamically distributed.

For high availability, the routers are designed to be horizontally scalable. Configure your load balancer to distribute incoming traffic across all router instances.

Refer to the Cloud Foundry Architecture topic for more information about Cloud Foundry components.
Identifying the API Endpoint for your Elastic Runtime Instance

The API endpoint for your Elastic Runtime deployment, its target URL, is the API endpoint of the deployment's Cloud Controller. Find your Cloud Controller API endpoint by consulting your cloud operator, from the Apps Manager, or from the command line.

From the Apps Manager

Log in to the Apps Manager for your Elastic Runtime instance, then click Tools in the left navigation panel. The Getting Started section of the Tools page shows your API endpoint.

```
<table>
<thead>
<tr>
<th>GETTING STARTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ cf help</td>
</tr>
<tr>
<td>$ cf login -a https://api.your_endpoint.com</td>
</tr>
<tr>
<td>API endpoint: https://api.your_endpoint.com</td>
</tr>
<tr>
<td>Username&gt; your_username</td>
</tr>
<tr>
<td>Password&gt; your_password</td>
</tr>
<tr>
<td>Org&gt; your_org</td>
</tr>
<tr>
<td>Space&gt; your_space</td>
</tr>
<tr>
<td>$ cf push your_app</td>
</tr>
</tbody>
</table>
```

From the Command Line

From a command line, use the `cf api` command to view your API endpoint.

Example:

```
$ cf api
API endpoint: https://api.example.com (API version: 2.2.0)
```
Creating New Elastic Runtime User Accounts

Page last updated:

When you first deploy your Elastic Runtime PaaS, there is only one user: an administrator. At this point, you can add accounts for new users who can then push applications using the Cloud Foundry Command Line Interface (cf CLI).

How to add users depends on whether or not you have SMTP enabled, as described in the options below.

Option 1: Adding New Users when SMTP is Enabled

If you have enabled SMTP, your users can sign up for accounts and create their own orgs. They do this using the Pivotal Cloud Foundry (PCF) Apps Manager, a self-service tool for managing organizations, users, applications, and application spaces.

Instruct users to complete the following steps to log in and get started using the Apps Manager.

1. Browse to apps.YOUR-SYSTEM-DOMAIN. Refer to Elastic Runtime > Domains to locate your system domain.

2. Select Create an Account.

3. Enter your email address and click Create an Account. You will receive an email from the Apps Manager when your account is ready.

4. When you receive the new account email, follow the link in the email to complete your registration.

5. You will be asked to choose your organization name.

You now have access to the Apps Manager. Refer to the Apps Manager documentation at docs.pivotal.io for more information about using the Apps Manager.

Option 2: Adding New Users when SMTP is Not Enabled

If you have not enabled SMTP, only an administrator can create new users, and there is no self-service facility for users to sign up for accounts or create orgs.

The administrator creates users with the cf CLI. See Creating and Managing Users with the cf CLI.

Return to the Installing Pivotal Cloud Foundry Guide
Using Your Own Load Balancer

This guide describes how to use your own load balancer and forward traffic to your Elastic Runtime router IP address.

Pivotal Cloud Foundry (PCF) deploys with a single instance of HAProxy for use in lab and test environments. Production environments should use a highly-available customer-provided load balancing solution that does the following:

- Provides load balancing to each of the PCF Router IPs
- Supports SSL termination with wildcard DNS location
- Adds appropriate `x-forwarded-for` and `x-forwarded-proto` HTTP headers to incoming requests
- (Optional) Supports WebSockets

**Prerequisites**

To integrate your own load balancer with PCF, you must ensure the following:

- WebSocket connections are not blocked for Loggregator functionality.
- The load balancer must be able to reach the Gorouter IPs.

Follow the instructions below to use your own load balancer.

**Step 1: Deploy PCF Installation VM**

Deploy a PCF Installation virtual machine. The procedure you follow depends on the IaaS you use:

- Deploying Operations Manager to vSphere
- Deploying Operations Manager to vCloud Air and vCloud

**Step 2: Register PCF IP Address**

In your load balancer, register the IP addresses that you assigned to PCF.

**Step 3: Configure Pivotal Ops Manager and Ops Manager Director**

Configure your Pivotal Operations Manager and Ops Manager Director as described in installing Pivotal Cloud Foundry, then add Elastic Runtime.

Do not click Install after adding Elastic Runtime.

**Step 4: Configure Networking**

1. In Pivotal Operations Manager, click the Elastic Runtime tile.
2. Select Networking.

This section allows you to employ various security and routing services for your platform. It is usually preferable to use your own load balancer instead of an HAProxy instance as your point-of-entry to the platform.
Configure the point-of-entry to this environment

- Select this option if you have an external load balancer, and it can forward encrypted traffic (SSL, HTTPS, etc.) to the Elastic Runtime Router(s). You may also select this option for a non-production environment where load balancing is not required, by configuring a single Router with TLS enabled as your point-of-entry to the Elastic Runtime platform.

**Router SSL Termination Certificate and Private Key**

```
-----BEGIN CERTIFICATE-----
MIIFxgCCACsCAQExDTEQMBAAUDATAQofEFAxGzbzElMAxGA1UEBhMC
WMsCzAUBgNVBAgTAlMNRwYQYDVQQHEw1TWMgY2RjbmNpbwIiMSwg
MREwHQYDVQQK
ExzQaZvZv0FsDepartSmYXKfLCJDnXhNMMwWggTgfDQYJKiYXZfN2kL:
chAkJWMTawJQkZiZiwhNAQgBFfKbYQ2XkYXK3NUwZ2aZWXyW6KqQb
qmM0YwwaYwHhckMCTUwNjA4MjEyMzA5MjYwLzA3MjEyMzA5MjYw
-----END CERTIFICATE-----
```

Change

**Router SSL Ciphers**

- Select this option if you have an external load balancer, but it will NOT forward any encrypted traffic to the Elastic Runtime Router(s). You may also select this option for a non-production environment where load balancing is not required, by configuring a single Router with TLS disabled as your point-of-entry to the Elastic Runtime platform.
- Select this option if you want to use HAProxy as your first point-of-entry instead of your own load balancer. Note that HAProxy does not provide IP failover, so it is not a robust load balancer.

- **Disable SSL certificate verification for this environment**
- **Disable insecure cookies on the Router**

Choose whether to enable route services. Route services enable you to proxy requests to your app over TLS to arbitrary URLs before hitting your app.

- **Enable route services**
- **Ignore SSL certificate verification on route services**
- **Disable route services**
3. In the Router IPs field, enter the IP address or addresses for PCF that you registered with your load balancer in Step 2.

4. In the HAProxy IPs field, delete any existing IP addresses. This field should be blank.

5. Under Configure the point-of-entry to this environment, choose one of the following:
   - External Load Balancer with Encryption: Select this option if your deployment uses an external load balancer that can forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing. Complete the fields for the Router SSL Termination Certificate and Private Key and Router SSL Ciphers.
   - External Load Balancer without Encryption: Select this option if your deployment uses an external load balancer that cannot forward encrypted traffic to the Elastic Runtime Router, or for a development environment that does not require load balancing.

6. If you are not using SSL encryption or if you are using self-signed certificates, select Disable SSL certificate verification for this environment.

7. Select the Disable insecure cookies on the Router checkbox to turn on the secure flag for cookies generated by the router.

8. In the Choose whether or not to enable route services section, choose either Enable route services or Disable route services. Route services are a class of marketplace services that perform filtering or content transformation on application requests and responses. See the Route Services topic for details.
   - If you enable route services, check Ignore SSL certificate verification on route services for the routing tier to reject requests that are not signed by a trusted CA.

9. Optionally, use the Applications Subnet field if you need to avoid address collision with a third-party service on the same subnet as your apps. Enter a CIDR subnet mask specifying the range of available IP addresses assigned to your app containers. The IP range must be different from the network used by the system VMs.

10. Optionally, you can change the value in the Applications Network Maximum Transmission Unit (MTU) field. Pivotal recommends setting the MTU value for your application network to 1454. Some configurations, such as networks that use GRE tunnels, may require a smaller MTU value.

11. Optionally, increase the number of seconds in the Router Timeout to Backends field to accommodate larger uploads over connections with high latency.

12. Click Save.

Step 5: Finalize Changes

1. Return to the Ops Manager Installation Dashboard
2. Click Install.
Configuring Proxy Settings for All Applications

This topic describes how to globally configure proxy settings for all applications in your Pivotal Cloud Foundry (PCF) deployment. Some environments restrict access to the Internet by requiring traffic to pass through an HTTP or HTTPS proxy. PCF operators can use the Cloud Foundry Command Line Interface (cf CLI) to provide the proxy settings to all applications, including system applications and service brokers.

**Note:** Incorrectly configuring proxy settings can prevent applications from connecting to the Internet or accessing required resources. They can also cause errands to fail and break system applications and service brokers. Although errands, system applications, and service brokers do not need to connect to the Internet, they often need to access other resources on PCF. Incorrect proxy settings can break these connections.

Set Environment Variables

To globally configure proxy settings for PCF applications, perform the following steps to set three environment variables for both the staging environment variable group and the running environment variable group.

For more information about variable groups, see the Environment Variable Groups section in the Cloud Foundry Environment Variables topic.

This procedure explains how to set proxy information for both staging and running applications. However, you can set proxy settings for only staging or only running applications.

1. Target your Cloud Controller with the cf CLI if you have not installed the cf CLI, see the Installing the cf CLI topic.

   ```bash
   $ cf api YOUR-SYSTEM-DOMAIN
   Setting api endpoint to api.YOUR-SYSTEM-DOMAIN...
   OK
   API endpoint: https://api.YOUR-SYSTEM-DOMAIN (API version: 2.54.0)
   Not logged in. Use `cf login` to log in.
   ```

2. Log in with your UAA administrator credentials. To retrieve these credentials, navigate to the Pivotal Elastic Runtime tile in the Ops Manager Installation Dashboard and click Credentials. Under UAA, click Link to Credential next to Admin Credentials and record the password.

   ```bash
   $ cf login
   API endpoint: https://api.YOUR-SYSTEM-DOMAIN
   Email: admin
   Password: [hidden]
   Authenticating...
   OK
   ```

3. To configure proxy access for applications that are staging, run the following command, replacing the placeholder values:

   ```bash
   ```

   - **http_proxy**: Set this value to the proxy to use for HTTP requests.
   - **https_proxy**: Set this value to the proxy to use for HTTPS requests. In most cases, this will be the same as `http_proxy`.
   - **no_proxy**: Set this value to a comma-separated list of DNS names or IP addresses that can be accessed without passing through the proxy. This value may not be needed, because it depends on your proxy configuration. From now on, the proxy settings are applied to staging applications.

4. To configure proxy access for applications that are running, run the following command, replacing the placeholder values as above:

   ```bash
   ```

   To configure proxy settings for Java-based applications, use the following command instead, replacing the placeholder values. For `http.nonProxyHosts`, use a pipe-delimited list rather than a comma-separated list.

   ```bash
   $ cf set-running-environment-variable-group "{"JAVA_OPTS": "-Dhttp.proxyHost=YOUR-PROXY -Dhttp.proxyPort=8080 -Dhttp.nonProxyHosts=NO-PROXY.EXAMPLE.COM"}"
   ```

   For more information about these Java proxy settings, see Java Networking and Proxies.

5. To apply the proxy configuration for the running environment variable group, you must restart each application that you want to use the new configuration.
Troubleshooting

If an application fails after you apply the global proxy settings, try the following solutions.

Exclude an App From Global Proxy Settings

If your application fails, try instructing the application to ignore the global proxy settings. Perform the following commands to manually unset the proxy environment variables for the failing application:

1. Set the proxy environment variables for `http_proxy` to an empty value:

   ```bash
   $ cf set-env YOUR-APP http_proxy ''
   ``

2. Set the proxy environment variables for `https_proxy` to an empty value:

   ```bash
   $ cf set-env YOUR-APP https_proxy ''
   ``

3. Set the proxy environment variables for `no_proxy` to an empty value:

   ```bash
   $ cf set-env YOUR-APP no_proxy ''
   ``

Change Case of HTTP

Your application and language runtime may be case-sensitive. Try performing the steps in the Set Environment Variables section using uppercase for `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` instead of lowercase. Refer to the following example.

```bash
```

Check Proxy Settings

If you have set up your proxy so that it can only send traffic to the Internet, then a request to an internal resource like PCF fails. You must set `no_proxy` so that traffic destined for PCF and other internal resources is sent directly and does not go through the proxy. For instance, setting `no_proxy` to include your system and application domains will ensure that requests destined for those domains are sent directly.

Verify Interpretation

The interpretation of `no_proxy` depends on the application and the language runtime. Most support `no_proxy`, but the specific implementation may vary. For example, some match DNS names that end with the value set in `no_proxy`: `example.com` would match `test.example.com`. Others support the use of the asterisk as a wildcard to provide basic pattern matching in DNS names: `*.example.com` would match `test.example.com`. Most applications and language runtimes do not support pattern matching and wildcards for IP addresses.
Restricting App Access to Internal PCF Components

This topic describes how to secure the component VMs of your Pivotal Cloud Foundry (PCF) deployment from being accessed by apps.

Introduction

See the following list to understand the concepts for this topic:

- **How PCF determines where apps can send traffic:**
  - PCF uses Application Security Groups (ASGs), which are network policy rules specifying protocols, ports, and IP ranges that apply to outbound network connections initiated from apps. See [Understanding ASGs](#).

- **Why you must create new rules for outbound app traffic:**
  - PCF installs with a default ASG that allows apps running on your deployment to send traffic to almost any IP address. This means apps are not blocked from initiating connections to most network destinations unless an administrator takes action to update the ASGs with a more restrictive policy.

- **How you can set up new rules:**
  - To help secure your component VMs against apps while ensuring your apps can access the services they need, follow the procedure below, which includes these steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determine Your Network Layout: The procedure for securing your deployment with ASGs varies depending on your network layout, which you can determine using Ops Manager.</td>
</tr>
<tr>
<td>2</td>
<td>Ensure Access for PCF System Apps: Bind the default ASG to the system org so that PCF system apps can continue accessing the system components they need after you remove the deployment-wide default ASG in Step 4.</td>
</tr>
<tr>
<td>3</td>
<td>Create New ASGs: Block apps from sending traffic to system components, but allow them to send traffic to the services they need.</td>
</tr>
<tr>
<td>4</td>
<td>Remove the Default ASG: After you create and bind new ASGs, you no longer need the deployment-wide default ASG bindings that allow apps to send traffic to any IP.</td>
</tr>
<tr>
<td>5</td>
<td>Restart your Apps: To apply the ASG changes, you must restart all of the apps in your deployment.</td>
</tr>
</tbody>
</table>

- **When to set up new rules:**
  - Pivotal recommends that you complete this procedure directly after installing PCF, prior to developers pushing apps to the platform. If you complete the procedure after apps have been pushed to the platform, you must restart all the apps in your deployment.

Prerequisites

The procedure below requires that you have the latest release of [ASG Creator](https://github.com/cloudfoundry-incubator/ASG-Creator) from the Cloud Foundry incubator repository on Github. See [About the ASG Creator Tool](#).

Procedure

Follow these steps to apply ASGs that prevent apps running on your deployment from accessing internal PCF components.

**Step 1: Determine Your Network Layout**

The procedure for securing your deployment with ASGs varies depending on your network layout, which you can determine by following these steps:

1. Log in to Ops Manager.
2. For each tile, click Assign AZs and Networks and record the selected Network that the tile is installed on.

3. Based on the information you gathered, determine which of the following network layouts you have:

<table>
<thead>
<tr>
<th>Layout Name</th>
<th>Layout Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Network</td>
<td>One network for Ops Manager and the Ops Manager Director, Elastic Runtime, and services.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: You cannot secure your deployment with ASGs if you have this network layout. Because PCF dynamically allocates IPs, they cannot be easily excluded in the case of a single network.</td>
</tr>
<tr>
<td>Two Networks</td>
<td>One network for Ops Manager and the Ops Manager Director.</td>
</tr>
<tr>
<td></td>
<td>One network for Elastic Runtime and Services.</td>
</tr>
<tr>
<td>Three Networks</td>
<td>One network for Ops Manager and the Ops Manager Director.</td>
</tr>
<tr>
<td></td>
<td>One network for Elastic Runtime.</td>
</tr>
<tr>
<td></td>
<td>One network for all services.</td>
</tr>
<tr>
<td>Three or More Networks</td>
<td>One network for Ops Manager and the Ops Manager Director.</td>
</tr>
<tr>
<td></td>
<td>One network for Elastic Runtime.</td>
</tr>
<tr>
<td></td>
<td>One network for each service.</td>
</tr>
</tbody>
</table>

4. If your network layout includes two or more networks, continue Step 2: Ensure Access for PCF System Apps.

**Step 2: Ensure Access for PCF System Apps**

Follow these steps to apply the default ASG to the staging set in the system org. This provides network access to PCF system apps without restrictions, which enables them to continue functioning properly after you perform Step 4: Remove the Deployment-wide Default ASG Binding.

1. Bind the default ASG to the staging set in the system org:

   ```
   $ cf bind-staging-security-group default_security_group
   ```

2. Bind the default ASG to the running set in the system org:

   ```
   $ cf bind-running-security-group default_security_group
   ```

**Step 3: Create New ASGs**

Follow these steps to create ASGs that block apps from accessing PCF components and create any additional ASGs that allow apps to access the services they require.

**Part A: Record CIDRs**

Gather the CIDRs for each network in your deployment:

1. From the Ops Manager Director tile, click Create Networks within the Settings tab.
2. In the Networks section, expand each network in your deployment by clicking its name.
3. Record the CIDR for each network.

**Part B: Create and Bind ASGs that Block Network Access**

Create ASGs that block apps from sending traffic to the networks that host Ops Manager, Elastic Runtime, and (optional) any services installed.

1. Create a `config.yml` containing the appropriate content for your network layout and replace the indicated values with the CIDRs you gathered:

© Copyright Pivotal Software Inc, 2013-2018
Two Network Layout:

excluded_networks:
- YOUR-OPS-MANAGER-CIDR
- YOUR-ELASTIC-RUNTIME-AND-SERVICES-CIDR

Three Network Layout:

Note: If you only want to secure the Ops Manager and Elastic Runtime components, you can optionally exclude the services CIDR.

excluded_networks:
- YOUR-OPS-MANAGER-CIDR
- YOUR-ELASTIC-RUNTIME-CIDR
- YOUR-SERVICES-CIDR

etc...

Three or More Network Layout:

Note: If you only want to secure the Ops Manager and Elastic Runtime components, you can optionally exclude the services CIDRs.

excluded_networks:
- YOUR-OPS-MANAGER-CIDR
- YOUR-ELASTIC-RUNTIME-CIDR
- YOUR-SERVICE-CIDR-1
- YOUR-SERVICE-CIDR-2

etc...

2. Run the following command to create a JSON that contains ASG rules, using your config.yml as input:

   $ asg-creator create --config config.yml --output OUTPUT-FILE-NAME.json

   Replace OUTPUT-FILE-NAME with a name of your choice.

3. Create an ASG by running the following command:
   a. Replace SECURITY-GROUP-NAME with a name of your choice.
   b. Replace OUTPUT-FILE-NAME with the name of the generated file from the previous step.

   $ create-security-group SECURITY-GROUP-NAME OUTPUT-FILE-NAME.json

4. Bind the ASG to the default staging set:

   $ cf bind-staging-security-group SECURITY-GROUP-NAME

5. Bind the ASG to the default running set:

   $ cf bind-running-security-group SECURITY-GROUP-NAME

Part C: Create and Bind ASGs for Service Access

Note: This part is only necessary if you blocked apps from a network that hosts services in the previous part. If you did not block apps from a network that hosts services, proceed to Step 4: Remove the Default ASG.

WARNING: In the two network layout, Elastic Runtime and services share the same network. This means that each time you create an ASG that allows apps to access a new port/protocol within the network, you further expose the Elastic Runtime component VMs. This is a limitation of a two network layout.

Now that you have created ASGs to secure the Ops Man, Elastic Runtime, and service components, work with developers to create additional ASGs that give apps access to the services they need.

For example, in any space where apps need to access the MySQL for PCF service, follow the steps in Creating Application Security Groups for MySQL.

For more information on creating and binding ASGs, see the following:
Step 4: Remove the Default ASG

Now that you have bound new ASGs to determine outbound traffic rules, you no longer need the default ASG bindings that allow apps to send traffic to any IP.

1. Unbind the default ASG from the staging set:

```bash
$ cf unbind-staging-security-group default_security_group
```

2. Unbind the default ASG from the running set:

```bash
$ cf unbind-running-security-group default_security_group
```

Step 5: Restart your Apps

To apply the ASG changes, you must restart all of the apps in your deployment. To mitigate app downtime during the restart, Pivotal recommends a blue-green deployment strategy.

**Notes:** You do not need to restart the apps in the system org.

1. Work with developers to restart a few of their apps individually and test that they still work correctly with the new ASGs in place. If an app does not work as expected, you likely must create another ASG that allows the app to send traffic to a service it requires.

   **Note:** To quickly roll back to the original overly-permissive state, you can re-bind the default_security_group ASG to the default-staging and default-running sets. You must then restart your apps to re-apply the original ASGs.

2. Restart the rest of the apps running on your deployment. Optionally, you can use the app-restarter cf CLI plugin to restart all apps in a particular space, org, or deployment.
Configuring Application Security Groups for Email Notifications

To allow the Notifications Service to have network access you need to create Application Security Groups (ASGs).

Note: Without Application Security Groups the service is not usable.

Prerequisite

Review the Getting Started with the Notifications Service topic to ensure you have setup the service.

Configure Network Connections

The Notifications Service is deployed as a suite of applications to the notifications-with-ui space in the system org, and requires the following outbound network connections:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Ports</th>
<th>Protocol</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMTP_SERVER</td>
<td>587 (default)</td>
<td>tcp</td>
<td>This service is used to send out email notifications</td>
</tr>
<tr>
<td>LOAD_BALANCER_IP</td>
<td>80, 443</td>
<td>tcp</td>
<td>This service will access the load balancer</td>
</tr>
<tr>
<td>ASSIGNED_NETWORK</td>
<td>3306</td>
<td>tcp</td>
<td>This service requires access to internal services.</td>
</tr>
</tbody>
</table>

Note: The SMTP Server port and protocol are dependent on how you configure your server.

Create a SMTP Server ASG

1. Navigate to the Ops Manager Installation Dashboard and click the Pivotal Elastic Runtime tile > Settings tab.
2. Record the information in the Address of SMTP Server and Port of SMTP Server fields.
3. Using the Address of SMTP Server information you obtained in the previous step, find the IP addresses and protocol of your SMTP Server from the service you are using. You might need to contact your service provider for this information.
4. Create a smtp-server.json file. For destination, you must enter the IP address of your SMTP Server.

```json
{
   "protocol": "tcp",
   "destination": "SMTP_SERVER_IPS",
   "ports": "587"
}
```
5. Create a security group called smtp-server:

   cf create-security-group smtp-server smtp-server.json

Create a Load Balancer ASG

Note: If you already have a ASG setup for a Load Balancer, you do not need to perform this step. Review your ASGs to check which groups you have setup.

If you are using the built-in HAProxy as your load balancer, follow this procedure. If you are using an external load balancer, you must obtain your
HAProxy IPs from the service you are using.

1. Record the HAProxy IPs in the Pivotal Elastic Runtime Tile > Settings > Networking tab.

2. Create a `load-balancer-https.json` file. For `destination`, use the HAProxy IPs you recorded above.

```json
[
  {
    "protocol": "tcp",
    "destination": "10.68.196.250",
    "ports": "80,443"
  }
]
```

3. Create a security group called `load-balancer-https`:

```
cf create-security-group load-balancer-https load-balancer-https.json
```

Create an Assigned Network ASG

**Note:** If you use external services, the IP addresses, ports, and protocols depend on the service.

1. Navigate to the Ops Manager Installation Dashboard > Pivotal Elastic Runtime tile > Settings > Assign AZs and Networks section.

2. Navigate to the network selected in the dropdown.

3. Record the Ops Manager Director tile > Settings tab > Create Networks > CIDR for the network identified in the previous step. Ensure the subnet mask allows the space to access p-mysql, p-rabbitmq, and p-redis.

4. Create a file `assigned-network.json`. For the `destination`, enter the CIDR you recorded above.

```json
[
  {
    "protocol": "tcp",
    "destination": "10.68.0.0/20",
    "ports": "3306,5672,6379"
  }
]
```

5. Create a security group called `assigned-network`:

```
cf create-security-group assigned-network assigned-network.json
```

Bind the ASGs

1. Target the `system` org:

```
cf target -o system
```

2. Create a `notifications-with-ui` space:

```
cf create-space notifications-with-ui
```

3. Bind the ASGs you created in this topic to the `notifications-with-ui` space:

```
cf bind-security-group smtp-server system notifications-with-ui
ncf bind-security-group load-balancer-https system notifications-with-ui
ncf bind-security-group assigned-network system notifications-with-ui
```
Configuring SSH Access for PCF

Page last updated:

To help troubleshoot applications hosted by a deployment, Pivotal Cloud Foundry (PCF) supports SSH access into running applications. This document describes how to configure a PCF deployment to allow SSH access to application instances, and how to configure load balancing for those application SSH sessions.

Elastic Runtime Configuration

This section describes how to configure Elastic Runtime to enable or disable deployment-wide SSH access to application instances. Space administrators and app developers and can also control SSH access to the space and app scope, respectively. See Application SSH Overview for details on SSH access permissions.

To configure Elastic Runtime SSH access for application instances:

1. Open the Pivotal Elastic Runtime tile in Ops Manager.
2. Under the Settings tab, select the Application Containers section.
3. Enable or disable the Allow SSH access to app containers checkbox.

SSH Load Balancer Configuration

If you are using HAProxy as a load balancer and SSH access is enabled, SSH requests are load balanced by HAProxy. This configuration relies on the presence of the same consul server cluster that Diego components use for service discovery. This configuration also works well for deployments where all traffic on the system domain and its subdomains is directed towards the HAproxy job, as is the case for a BOSH-Lite Cloud Foundry deployment on the default 192.0.2.34.xip.io domain.

For AWS deployments, where the infrastructure offers load-balancing as a service through ELBs, the deployment operator can provision an ELB to balance load across the SSH proxy instances. You should configure this ELB to listen to TCP traffic on the port given in app_ssh.port and to send it to port 2222.

In order to register the SSH proxies with this ELB, you should then add the ELB identifier to the elbs property in the cloud_properties hash of the Diego manifest access_resource_pools. If you used the spiff-based manifest-generation templates to produce the Diego manifest, specify these cloud_properties hashes in the iaas_settings.resource_pool_cloud_properties section of the iaas-settings.yml stub.
Identifying Elastic Runtime Jobs Using vCenter

To effectively monitor, control, and manage the virtual machines making up your Elastic Runtime deployment, you may need to identify which VM corresponds to a particular job in Elastic Runtime. You can find the CID of a particular VM from Pivotal Cloud Foundry (PCF) Operations Manager by navigating to Elastic Runtime > Status.

If you have deployed Elastic Runtime to VMware vSphere, you can also identify which Elastic Runtime job corresponds to which VM using the vCenter vSphere client. This option is not available if you have deployed Elastic Runtime to VMware vCloud Air / vCloud.

Note: The CID shown in Ops Manager is the name of the machine in vCenter.

Identifying Elastic Runtime Jobs Using vCenter

1. Launch the vSphere client and log in to the vCenter Server system.

2. Select the Inventory > Hosts and Clusters view.


4. Select the Virtual Machines tab.

5. Right-click the column label heading and check job.
6. The job column displays the Elastic Runtime job associated with each virtual machine.
Understanding the Effects of Single Components on a Pivotal Cloud Foundry Upgrade

Page last updated:

The Resource Config page of Pivotal Elastic Runtime tile in the Pivotal Cloud Foundry (PCF) Ops Manager shows the components that the Ops Manager Director installs. You can specify the number of instances for some of the components. We deliver the remaining resources as single components, meaning that they have a preconfigured and unchangeable value of one instance.

In a single-component environment, upgrading can cause the deployment to experience downtime and other limitations because there is no instance redundancy. Although this behavior might be acceptable for a test environment, you should configure the scalable components with editable instance values, such as HAProxy, Router, and Diego cells, for optimal performance in a production environment.

Note: A full Ops Manager upgrade may take close to two hours, and you will have limited ability to deploy an application during this time.

Summary of Component Limitations

The table lists components in the order that Ops Manager upgrades each component and includes the following columns:

- **Scalable?**: Indicates whether the component has an editable value or a preconfigured and unchangeable value of one instance.

  Note: For components marked with a checkmark in this column, we recommend that you change the preconfigured instance value of 1 to a value that best supports your production environment. For more information about scaling a deployment, refer to the Scaling Cloud Foundry topic.

- **Extended Downtime?**: Indicates that if there is only one instance of the component, that component is unavailable for up to five minutes during an Ops Manager upgrade.

- **Other Limitations and Information**: Provides the following information:
  
  - Component availability, behavior, and usage during an upgrade
  - Guidance on disabling the component before an upgrade

  Note: The table does not include the Run Smoke Tests and Run CF Acceptance Tests errands and the Compilation job. Ops Manager runs the errands after it upgrades the components and creates compilation VMs as needed during the upgrade process.

<table>
<thead>
<tr>
<th>Component</th>
<th>Scalable?</th>
<th>Extended Downtime?</th>
<th>Other Limitations and Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAProxy</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATS</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>etcd</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFS Server</td>
<td>✓</td>
<td></td>
<td>You cannot push, stage, or restart an app when an upgrade affects the NFS Server.</td>
</tr>
<tr>
<td>Cloud Controller Database</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud Controller</td>
<td>✓</td>
<td>✓</td>
<td>Your ability to manage an app when an upgrade affects the Cloud Controller depends on the number of instances that you specify for the Cloud Controller and Diego components. If either of these components are single components, you cannot push, stage, or restart an app during the upgrade.</td>
</tr>
<tr>
<td>Clock Global</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud Controller Worker</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Router</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Pivotal Ops Metrics Collector</td>
<td>✓</td>
<td></td>
<td>The Pivotal Ops Metrics tool is a JMX extension for Elastic Runtime that you can install. If you install this tool, Ops Manager operators may experience a 5 minute delay in metrics collection during an upgrade. You can disable this component before an upgrade to reduce the overall system downtime.</td>
</tr>
<tr>
<td>UAA Database</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UAA</td>
<td>✓</td>
<td></td>
<td>If a user has an active authorization token prior to performing an upgrade, the user can still log in using the CLI.</td>
</tr>
</tbody>
</table>

Note: A full Ops Manager upgrade may take close to two hours, and you will have limited ability to deploy an application during this time.

© Copyright Pivotal Software Inc, 2013-2018
Login ✓ ✓ 

Apps Manager Database

You can disable this component before an upgrade to reduce the overall system downtime.

MySQL Server

<table>
<thead>
<tr>
<th>Component</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diego Cells ✓ ✓</td>
<td>Your ability to manage an app when an upgrade affects Diego Cells depends on the number of instances that you specify for the Diego Cells, Cloud Controller, and other Diego components. If any of these components have only one instance, you may fail to push, stage, or restart an app during the upgrade. If you only have one Diego Cell, upgrading it causes downtime for the apps that run on it, including the Apps Manager app and the App Usage Service.</td>
</tr>
<tr>
<td>Diego BBS ✓ ✓</td>
<td>Your ability to manage an app when an upgrade affects the Diego BBS depends on the number of instances that you specify for the Diego BBS, Cloud Controller, and other Diego components. If any of these components have only one instance, you may fail to push, stage, or restart an app during the upgrade.</td>
</tr>
<tr>
<td>Diego Brain ✓ ✓</td>
<td>Your ability to manage an app when an upgrade affects the Diego Brain depends on the number of instances that you specify for the Diego Brain, Cloud Controller, and other Diego components. If any of these components have only one instance, you may fail to push, stage, or restart an app during the upgrade.</td>
</tr>
<tr>
<td>Doppler Server</td>
<td>Ops Manager operators experience 2-5 minute gaps in logging.</td>
</tr>
<tr>
<td>Loggregator Traffic Controller</td>
<td>Ops Manager operators experience 2-5 minute gaps in logging.</td>
</tr>
<tr>
<td>Push Apps Manager and Push App Usage Service errands</td>
<td>These errands run scripts that connect the Apps Manager app and the App Usage Service components to the Apps Manager Database. The Apps Manager app and App Usage Service components run in a single Diego Cell instance and the Apps Manager Database is a single component. If there is an upgrade issue with either the Apps Manager Database instance or the Diego Cell instance, the upgrade fails and Ops Manager does not run the Apps Manager deployment errand again.</td>
</tr>
</tbody>
</table>
Configuring System Logging in Elastic Runtime

Page last updated:

This topic explains how to configure the Pivotal Cloud Foundry Loggregator system to scale its maximum throughput, and to forward logs to an external aggregator service.

Scaling Loggregator

Elastic Runtime system components and apps constantly generate log and metrics data. The Metron agent running on each component or application VM collects and sends this data out to Doppler components, which temporarily buffer the data before periodically forwarding it to the Traffic Controller. The Traffic Controller then serves the aggregated data stream through the Firehose WebSocket endpoint.

When the log and metrics data input to a Doppler exceeds its buffer size for a given interval, data can be lost. You can take several actions to minimize this loss.

Increase buffer size

1. In the Pivotal Cloud Foundry (PCF) Ops Manager Installation Dashboard, click the Elastic Runtime tile.
2. Select System Logging.
3. Increase the drain buffer size to prevent loss of log data.
4. Click Save.

Add additional Doppler instances

1. In the PCF Ops Manager Installation Dashboard, click the Elastic Runtime tile.
2. Select Resource Config.
3. Increase the number in the Instances column and the Doppler Server row.
4. Click Save.

5. Click Apply Changes.

Add additional Traffic Controller instances

1. In the PCF Ops Manager Installation Dashboard, click the Elastic Runtime tile.

2. Select Resource Config.

3. Increase the number in the Instances column and the Loggregator Trafficcontroller row.

4. Click Save.

5. Click Apply Changes.

Enabling System Log Forwarding

Elastic Runtime can forward log data to an external aggregator service instead of routing it to the Loggregator Firehose. System log forwarding for Pivotal Cloud Foundry (PCF) is managed through the PCF Ops Manager Installation Dashboard. To enable syslog forwarding:

1. Click the Elastic Runtime tile.

2. Select System Logging.

3. Enter the Hostname, Port, and Protocol for your third-party log management service.

4. (Optional) Increase the drain buffer size to prevent loss of log data.

5. Click Save.
Configure system logging. Leave the External Syslog fields blank unless you wish to use an external syslog server.

External Syslog Aggregator Hostname

External Syslog Aggregator Port

External Syslog Network Protocol

Syslog Drain Buffer Size (in bytes) *

Loggregator Port

6. Click Apply Changes.
Configuring UAA Password Policy

If your Pivotal Cloud Foundry (PCF) deployment uses the internal user store for authentication, you can configure its password policy within the Pivotal Elastic Runtime tile.

Open the Internal UAA Configuration

1. In a browser, navigate to the fully qualified domain name (FQDN) of your Ops Manager and log in.
2. Click the Pivotal Elastic Runtime tile.
3. Select Authentication and Enterprise SSO on the Settings tab.

4. Confirm that the Internal UAA option is selected.

Set Password Requirements

1. For Minimum Password Length, enter the minimum number of characters for a valid password.
2. For Minimum Uppercase Characters Required for Password, enter the minimum number of uppercase characters required for a valid password.
3. For **Minimum Lowercase Characters Required for Password**, enter the minimum number of lowercase characters required for a valid password.

4. For **Minimum Numerical Digits Required for Password**, enter the minimum number of digits required for a valid password.

5. For **Minimum Special Characters Required for Password**, enter the minimum number of special characters required for a valid password.

**Set Password Expiration and Entry Attempts**

1. For **Number of Months Before Password Expires**, enter the number of months a password remains valid. Enter 0 if you want passwords to never expire.

2. For **Maximum Password Entry Attempts Allowed**, enter the maximum number of failures allowed to enter a password within a five-minute timespan before the account is locked.
Configuring Authentication and Enterprise SSO for Elastic Runtime

This topic describes Pivotal Cloud Foundry (PCF) authentication and single sign-on configuration with Lightweight Directory Access Protocol (LDAP) and Security Assertion Markup Language (SAML).

Refer to the instructions below to configure your deployment with SAML or LDAP.

Connecting Elastic Runtime to either the LDAP or SAML external user store allows the User Account and Authentication (UAA) server to delegate authentication to existing enterprise user stores.

If your enterprise user store is exposed as a SAML or LDAP Identity Provider for single sign-on (SSO), you can configure SSO to allow users to access the Apps Manager and Cloud Foundry Command Line Interface (cf CLI) without creating a new account or, if using SAML, without re-entering credentials.

See the Adding Existing SAML or LDAP Users to a PCF Deployment topic for information about managing user identity and pre-provisioning user roles with SAML or LDAP in PCF.

This Knowledge Base article explains the process used by the UAA Server when it attempts to authenticate a user through LDAP.

Configure PCF to Use a SAML Identity Provider

To connect PCF Elastic Runtime with SAML, you must perform the following tasks:

- Configure PCF as a Service Provider for SAML
- Configure SAML as an Identity Provider for PCF

Configure PCF as a Service Provider for SAML

Follow the instructions below to configure PCF as a service provider for SAML.

1. From the Installation Dashboard, click the Elastic Runtime tile.

2. Select the Domains tab and record your system domain.
3. Select Authentication and Enterprise SSO.

4. Select SAML Identity Provider.
5. Set the **Provider Name**. This is a unique name you create for the Identity Provider. This name can include only alphanumeric characters, +, -, and _. You should not change this name after deployment because all external users use it to link to the provider.

6. Enter a **Display Name**. Your provider display name appears as a link on your Pivotal login page, which you can access at [https://login.YOUR-SYSTEM-DOMAIN](https://login.YOUR-SYSTEM-DOMAIN).
7. Retrieve the metadata from your Identity Provider and copy it into either the Provider Metadata or the Provider Metadata URL fields, depending on whether your Identity Provider exposes a Metadata URL. Refer to the Configure SAML Identity Provider for PCF section of this topic for more information. Pivotal recommends that you use the Provider Metadata URL rather than Provider Metadata because the metadata can change. You can do this in either of the following ways:
   - If your Identity Provider exposes a Metadata URL, provide the Metadata URL.
   - Download your Identity Provider metadata and paste this XML into the Provider Metadata area.

Note: You only need to select one of the above configurations. If you configure both, your Identity Provider defaults to the (OR) Provider Metadata URL.

Note: Refer to the Adding Existing SAML or LDAP Users to a PCF Deployment topic for information on on-boarding SAML users and mapping them to PCF user roles.

8. Select the Name ID Format for your SAML Identity Provider. This translates to username on PCF Elastic Runtime. The default is Email Address.

9. By default, all SAML Authentication Request from PCF are signed. To change this, disable the Sign Authentication Requests checkbox and configure your Identity Provider to verify SAML authentication requests.

10. To validate the signature for the incoming SAML assertions, enable the Required Signed Assertions checkbox and configure your Identity Provider to send signed SAML Assertions.

11. For Signature Algorithm, choose an algorithm from the dropdown menu to use for signed requests and assertions. The default value is SHA1.

12. Click Save.

13. Return to the Installation Dashboard by clicking the link.


Configure SAML as an Identity Provider for PCF

Download the Service Provider Metadata from https://login.YOUR-SYSTEM-DOMAIN/saml/metadata. Consult the documentation from your Identity Provider for configuration instructions.

Refer to the table below for information about certain industry-standard Identity Providers and how to integrate them with PCF:
Configure PCF to Use an LDAP Identity Provider

To integrate the UAA with LDAP, configure Elastic Runtime with your LDAP endpoint information as follows:

1. Log into the Operations Manager web interface.


3. In the left navigation menu, select Authentication and Enterprise SSO.

Note: Some Identity Providers allow uploads of Service Provider Metadata. Other providers require you to manually enter the Service Provider Metadata into a form.

5. Enter the Server URL, a URL pointing to the LDAP server. This URL must include one of the following protocols:
   - `ldap://`: This specifies that the LDAP server uses an unencrypted connection.
   - `ldaps://`: This specifies that the LDAP server uses SSL for an encrypted connection and requires that the LDAP server holds a trusted certificate or that you import a trusted certificate to the JVM truststore.

6. For LDAP Credentials, enter the LDAP Distinguished Name (DN) and password for binding to the LDAP Server. Example DN:
   
   Note: Pivotal recommends that you provide LDAP credentials that grant read-only permissions on the LDAP Search Base and the LDAP Group Search Base.

7. For User Search Base, enter the location in the LDAP directory tree from which any LDAP User search begins. The typical LDAP Search Base matches your domain name.

   For example, a domain named "cloud.example.com" typically uses the following LDAP User Search Base:

   Note: This Knowledge Base article provides instructions for testing and troubleshooting your LDAP search filters.

8. For User Search Filter, enter a string that defines LDAP User search criteria. These search criteria allow LDAP to perform more effective and efficient searches. For example, the standard LDAP search filter `cn=Smith` returns all objects with a common name equal to `Smith`.

   In the LDAP search filter string that you use to configure Elastic Runtime, use `{0}` instead of the username. For example, use `cn={0}` to return all LDAP objects with the same common name as the username.

   In addition to `cn`, other attributes commonly searched for and returned are `mail`, `uid`, and, in the case of Active Directory, `sAMAccountName`.

9. For Group Search Base, enter the location in the LDAP directory tree from which the LDAP Group search begins.
For example, a domain named “cloud.example.com” typically uses the following LDAP Group Search Base:

```
ou=Groups,dc=example,dc=com
```

Follow the instructions in the Grant Admin Permissions to an External Group (SAML or LDAP) section of the Creating and Managing Users with the UAA CLI / UAAC topic to map the groups under this search base to admin roles in PCF.

10. For Group Search Filter, enter string that defines LDAP Group search criteria. The standard value is `member={0}`.

11. For Server SSL Cert, paste in the root certificate from your CA certificate or your self-signed certificate.

12. Click Save.

13. Return to the Installation Dashboard by clicking the link.

Adding Existing SAML or LDAP Users to a Pivotal Cloud Foundry Deployment

This topic describes the procedure for adding existing SAML or LDAP users to a Pivotal Cloud Foundry (PCF) deployment enabled with SAML or LDAP.

Note: You must have admin access to the PCF Ops Manager Installation Dashboard for your deployment to complete the procedure described here.

Step 1: Add SAML or LDAP Users

There are two ways to add existing SAML or LDAP users to your PCF deployment:

- In bulk, using the UAA Bulk Import Tool. See the README for instructions on installing and using the tool.
- Individually, through the CF CLI, as described below:
  1. Each existing SAML or LDAP user must log in to Apps Manager or to the cf CLI using their SAML (by entering `cf login --sso`) or LDAP credentials. Users will not have access to any org or space until these are granted by an Org or Space Manager.
  2. The PCF Admin must log in to the cf CLI and associate the user with the desired org and space roles. See Org and App Space Roles.

(Advanced Option) Integrate with Enterprise Identity Management System

If your organization uses an Enterprise Identity Management System for centralized provisioning and deprovisioning of users, you can use the Users API and Organizations API to write a connector to manage users and permissions in Elastic Runtime.

Step 2: Create User

1. Create the user in UAA by running the following command. Replace 'EXAMPLE-USERNAME' with the username of the SAML or LDAP user you wish to add.
   - For LDAP, set user origin to ldap.
     ```
     $ uaac curl -H "Content-Type: application/json" -k /Users -X POST -d '{"userName":"EXAMPLE-USERNAME",
     "emails": ["EXAMPLE-USERNAME@test.com"]
     , "origin":"ldap",
     "externalId":"cn=EXAMPLE-USERNAME,ou=Users,dc=test,dc=com"}'
     ```
   - For SAML, set user origin to the SAML identity provider name set in the Elastic Runtime tile under Authentication and Enterprise SSO.
     ```
     $ uaac curl -H "Content-Type: application/json" -k /Users -X POST -d '{"userName":"EXAMPLE-USERNAME",
     "emails": ["EXAMPLE-USERNAME@test.com"]
     , "origin":"YOUR-SAML-PROVIDER",
     "externalId":"EXAMPLE-USERNAME"}'
     ```

2. Target the API endpoint for your PCF deployment:
   ```
   $ cf target https://api.YOUR-SYSTEM-DOMAIN
   ```

3. Log in to the cf CLI:
   ```
   $ cf login
   ```

4. Create an environment variable for your OAuth token to use in the next step:
   ```
   $ export OAUTH-TOKEN=$(cf oauth-token | cut -d ' ' -f 2)
   ```

5. Create a User record in the Cloud Controller Database with the SAML or LDAP GUID you created by running the following curl command. The command uses the Users API.

Step 3: Provide User Access to Orgs

Associate the user with the appropriate orgs in your Elastic Runtime deployment, using the Organizations API.

Step 4: Associate User with Space or Org Role

Users can be given Space and Org roles using the following API calls:

- Associate an Auditor with a Space.
- Associate a Developer with a Space.
- Associate a Manager with a Space.
- Associate an Auditor with a Organization.
- Associate a Manager with a Organization.
Switching Application Domains

This topic describes how to change the domain of an existing Pivotal Cloud Foundry (PCF) installation, using an example domain change from myapps.mydomain.com to newapps.mydomain.com.

1. In PCF Ops Manager, select the Pivotal Elastic Runtime tile.
2. Select Domains from the menu to see the current Apps Domain for your Elastic Runtime deployment. In the following example it is myapps.mydomain.com.
3. In the terminal, run `cf login -a YOUR_API_ENDPOINT`. The cf CLI prompts you for your PCF username and password, as well as the org and space you want to access. See Identifying the API Endpoint for your Elastic Runtime Instance if you don’t know your API endpoint.
4. Run `cf domains` to view the domains in the space. If you have more than one shared domain, ensure that the domain you want to change is at the top of the list before you apply the new domain to your Elastic Runtime tile configuration. You can delete and re-create the other shared domains as necessary to push the domain you want to change to the top of the list. If you do this, make sure to re-map the routes for each domain.

```
$ cf domains
Getting domains in org my-org as admin...

  name          status
  myapps.mydomain.com  shared
```

5. Run `cf routes` to confirm that your apps are assigned to the domain you plan to change.

```
$ cf routes
Getting routes as admin ...

  space  host  domain  apps
  my-space  myapp  myapps.mydomain.com  myapp
```

6. Run `cf create-shared-domain YOUR_DESIRED_NEW_DOMAIN` to create the new domain you want to use:

```
$ cf create-shared-domain newapps.mydomain.com
Creating shared domain newapps.mydomain.com as admin...
OK
```

7. Run `cf map-route APP_NAME NEW_DOMAIN -a HOST_NAME` to map the new domain to your app. In this example both the NEW_DOMAIN and HOST_NAME arguments are `myapp`, since this is both the name of the app to which we are mapping a route, and the intended hostname for the URL.

```
$ cf map-route myapp newapps.mydomain.com -a myapp
```
8. Repeat the previous step for each app in this space. Afterwards, check Apps Manager to confirm that the route URL has updated correctly for each app:

```
SPACE
my-space

APPLICATIONS  Learn More

STATUS  APP  INSTANCES

myapp
myapp.newapps.mydomain.com  1
```

9. Repeat the above steps for each space in your PCF installation except for the System org, beginning with logging into the org and space and ending with confirming the URL update.

**Note:** Ordinarily the System org contains only PCF apps that perform utility functions for your installation. Pivotal does not recommend pushing apps to this org. However, if you have pushed apps to System, you must also repeat the above steps for these apps.

10. Once you have confirmed that every app in every space has been mapped to the new domain, delete the old domain by running `cf delete-shared-domain`:

```
S cf delete-shared-domain myapps.mydomain.com
Deleting domain myapps.mydomain.com as admin...

This domain is shared across all orgs.
Deleting it will remove all associated routes, and will make any app with this domain unreachable.
Are you sure you want to delete the domain myapps.mydomain.com?
> yes
OK
```

11. Configure your Elastic Runtime tile to use the new domain, and apply changes. Apps that you push after your update finishes use this new domain.
Elastic Runtime gives each deployed application its own hostname in the app domain, and each system component its own hostname on the system domain. Ensure that you have a wildcard DNS record configured for both domains. You can also use the same domain as your apps and system domains.

System Domain *
my system . my domain . com

Apps Domain *
new apps . my domain . com

This is the default apps domain that pushed apps will use for their hostnames. Use the Cloud Foundry command line interface to add or delete domains. This also requires a wildcard DNS record.

Save
Scaling Elastic Runtime

Page last updated:

This topic discusses how to scale Elastic Runtime for different deployment scenarios. To increase the capacity and availability of the Pivotal Cloud Foundry (PCF) platform, and to decrease the chances of downtime, you can scale a deployment up using the instructions below.

If you want to make a Diego or PCF configuration highly available, see the Zero Downtime Deployment and Scaling in CF topic.

Steps for Scaling Elastic Runtime

1. Navigate to the Pivotal Cloud Foundry Operations Manager Installation Dashboard.

2. Click the Elastic Runtime tile in the Installation Dashboard.

You can scale your deployment horizontally, by increasing the number of Instances of a job. You can also scale your deployment vertically, by adjusting the Persistent Disk Type and VM Type of a job to allocate more disk space and memory. If you choose Automatic from the drop-down menu, Elastic Runtime uses the recommended amount of resources for the job.

If you scale down or delete a job that uses persistent disk, Elastic Runtime marks the disk as “orphaned.” Orphaned disks are not attached to any job, and Elastic Runtime deletes them after five days. You can use the BOSH CLI to list and recover orphaned disks. Follow the instructions in the “Prepare to Use the BOSH CLI” section of the “Advanced Troubleshooting with the BOSH CLI” topic to log in to the BOSH Director, and then follow the procedures in “Orphaned Disks” or in the BOSH documentation.

If you are using one of the following configurations, choose the values in the corresponding table to scale instances for your particular deployment:

- External Databases
- Internal MySQL
- Internal Databases (for Upgrades)
- External Blobstore
- External Load Balancer
External Databases

If you are using an external database, choose the following values in the Resource Config:

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MySQL Proxy</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Apps Manager Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cloud Controller Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UAA Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Internal MySQL

If you are using the internal MySQL database on a clean install, or on an upgrade from a configuration that previously used internal MySQL databases, you do not need to change the default values shown below. If you need to change back to this configuration, choose the values shown below in the Resource Config.

⚠ **Note:** Changing back to this configuration deletes any data written to your other database option.

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MySQL Proxy</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Apps Manager Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cloud Controller Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UAA Database (Postgres)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Internal Databases (for Upgrades)

If you are upgrading from a previous installation that used both Postgres and MySQL databases, you must maintain this configuration to avoid data loss.

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MySQL Proxy</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Apps Manager Database (Postgres)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Cloud Controller Database (Postgres)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>UAA Database (Postgres)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

External Blobstore

If you are using an external Blobstore, choose the following value in the Resource Config:

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
External Load Balancer

If you are using an external load balancer, choose the following values in the Resource Config:

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAPProxy</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Router</td>
<td>≥ 1</td>
<td>For Amazon Web Services, set the Elastic Load Balancer name in the Router’s “External Load Balancer” field.</td>
</tr>
<tr>
<td>Diego Brain</td>
<td>≥ 1</td>
<td>For AWS, if you have the Diego SSH feature enabled, set the SSH ELB name in the Router’s “External Load Balancer” field.</td>
</tr>
</tbody>
</table>

JMX Bridge

If you are using JMX Bridge, choose the following value in the Resource Config:

<table>
<thead>
<tr>
<th>Job</th>
<th>Instance Count</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collector</td>
<td>≥ 1</td>
<td>Must have JMX Bridge tile added to PCF before you scale this instance count up. See Adding and Deleting Products topic.</td>
</tr>
</tbody>
</table>

5. Choose the suggested values outlined in each scenario above, and click Save.

6. Return to the Installation Dashboard and click Apply Changes.
Scaling Down Your MySQL Cluster

This topic describes how to safely scale down your MySQL cluster to a single node.

By default MySQL is a single node. To take advantage of the high availability features of MySQL, you may have scaled the configuration up to three nodes.

Note: If you are only running the MySQL cluster with a single node, you do not need to perform these steps.

Check the Health of Your Cluster

Before scaling down your MySQL cluster, perform the following actions to ensure the cluster is healthy.

1. Use the cf CLI to target the API endpoint of your Pivotal Cloud Foundry (PCF) deployment:

   ```
   cf api YOUR-SYSTEM-DOMAIN
   Setting api endpoint to api.YOUR-SYSTEM-DOMAIN...
   OK
   API endpoint: https://api.YOUR-SYSTEM-DOMAIN (API version: 2.54.0)
   Not logged in. Use 'cf login' to log in.
   ```

2. Log in with your User Account and Authentication (UAA) Administrator user credentials. Obtain these credentials by clicking the Credentials tab of the Elastic Runtime tile, locating the Admin Credentials entry in the UAA section, and clicking Link to Credential.

   ```
   cf login -u admin
   API endpoint: https://api.YOUR-SYSTEM-DOMAIN
   Password>
   Authenticating...
   OK
   ```

3. Create a test organization to verify the database across all nodes:

   ```
   cf create-org data-integrity-test-organization
   Creating org data-integrity-test-organization as admin...
   OK
   Assigning role OrgManager to user admin in org data-integrity-test-organization ...
   OK
   TIP: Use 'cf target -o data-integrity-test-organization' to target new org
   ```

4. Obtain the IP addresses of your MySQL server by performing the following steps:
   a. From the PCF Installation Dashboard, click the Pivotal Elastic Runtime tile.
   b. Click the Status tab.
   c. Record the IP addresses for all instances of the MySQL Server job.

5. Obtain the CCDB credentials for your MySQL server by performing the following steps:
   a. From the Elastic Runtime tile, click the Credentials tab.
   b. Locate the Ccdb Credentials entry in the MySQL Server section and click Link to Credential.
   c. Record the values for identity and password.

6. SSH into the Ops Manager VM. Because the procedures vary by IaaS, review the SSH into Ops Manager section of the Advanced Troubleshooting with the BOSH CLI topic for specific instructions.

7. For each of the MySQL server IP addresses recorded above, perform the following steps from the Ops Manager VM:
   a. Query the new organization with the following command, replacing `YOUR-IP` with the IP address of the MySQL server and `YOUR-IDENTITY` with the `identity` value of the CCDB credentials obtained above:
b. When prompted, provide the password value of the CCDB credentials obtained above.

c. Examine the output of the `mysql` command and verify the `created_at` date is recent.

```
+---------------------+----------------------------------+
| created_at										|	name																						|
+---------------------+----------------------------------+
| 2016-05-28 01:11:42 |	data-integrity-test-organization |
+---------------------+----------------------------------+
```

8. If each MySQL server instance does not return the same `created_at` result, contact Pivotal Support before proceeding further or making any changes to your deployment. If each MySQL server instance does return the same result, then you can safely proceed to scaling down your cluster to a single node by performing the steps in the following section.

## Scale Down Your Cluster

1. From the PCF Installation Dashboard, click the Pivotal Elastic Runtime tile.

2. Select Resource Config.

3. Use the drop-down menu to change the Instances count for MySQL Server to 1.

4. Click Save to apply the changes.

5. Delete your test organization with the following cf CLI command:

```
$ cf delete-org data-integrity-test-organization
```
Using Docker Registries

This topic describes how to configure your Docker registries, such as Docker Hub, with Pivotal Cloud Foundry (PCF). To use Docker registries, you must choose either to submit your root certificate authority (CA) certificate or provide the IP address for your Docker registry. PCF does not support using Docker registries that require user credentials.

**Prerequisite:** Ensure that you have enabled Docker support in PCF with the `cf enable-feature-flag diego_docker` command, as described in the Using Docker in Cloud Foundry topic.

Using a CA Certificate

If you provide your root CA certificate in the Ops Manager configuration, follow this procedure:

1. In the PCF Ops Manager Installation Dashboard, click the Ops Manager Director tile.

2. Click Security.

3. In the Trusted Certificates field paste one or more root CA certificates. The Docker registry does not use the CA certificate itself but uses a certificate that is signed by the CA certificate.

4. Click Save.

5. If you are:
   - Configuring Ops Manager Installation for the first time, return to your specific IAAS configuration to continue the installation process.
   - Modifying an existing Ops Manager installation, return to the PCF Ops Manager Installation Dashboard and click Apply Changes.

After configuration, BOSH propagates this CA certificate to all application containers in your deployment. You can then push and pull images from your Docker registries.

Use an IP Address Whitelist
If you choose not to provide a CA certificate, you must provide the IP address of your Docker registry.

1. Navigate to the PCF Operations Manager Installation Dashboard.

2. Click the Pivotal Elastic Runtime tile, and navigate to the Application Containers tab.

3. Select **Enable Custom Buildpacks** to enable custom-built application runtime buildpacks.

4. Select **Allow SSH access to app containers** to enable app containers to accept SSH connections. If you are using a load balancer instead of HAProxy, you must open port 2222 on your load balancer to enable SSH traffic. In order to open an SSH connection to an app, a user must have Space Developer privileges for that app’s space. Operators can grant those privileges in Apps Manager or via the `cf` CLI.

5. For **Private Docker Insecure Registry Whitelist**, provide the hostname or IP address and port that point to your private Docker registry. For example, enter `198.51.100.1:80` or `mydockerregistry.com:80`. Enter multiple entries in a comma-delimited sequence. SSL validation is ignored for private Docker image registries secured with self-signed certificates at these locations.

6. Under **Docker Images Disk-Cleanup Scheduling on Cell VMs**, choose one of the following:
   - **Never Cleanup Cell Disk-space**
   - ** Routinely Cleanup Cell Disk-space**
   - **Cleanup disk-space once threshold is reached**: If you choose this option, enter the amount of disk space the Cell must reach before disk cleanup initiates under **Threshold of Disk-Used (MB)**.

7. Click **Save**.

8. If you are:
   - Configuring Elastic Runtime for the first time, return to your specific IaaS configuration to continue the installation process.
   - Modifying an existing Elastic Runtime installation, return to the PCF Ops Manager Installation Dashboard and click **Apply Changes**.

After configuration, Elastic Runtime allows Docker images to come through the specified IP address without checking certificates.
Custom Branding Apps Manager

This topic describes how Pivotal Cloud Foundry operators can visually brand Apps Manager by changing certain text, colors, and images of the interface. Developers view the customized interface when logging in, creating an account, resetting a password, or using Apps Manager.

Operators customize Apps Manager by configuring the Custom Branding and Apps Manager Config pages of the Pivotal Elastic Runtime tile.

Custom Branding Page

1. In a browser, navigate to the fully qualified domain name (FQDN) of your Ops Manager and log in.

2. Click Pivotal Elastic Runtime.

3. Click the Custom Branding tab.

4. For Company Name, enter the name of your organization. If left blank, the name defaults to Pivotal.

5. For Accent Color, enter the hexadecimal code for the color used to accent various visual elements. For example, #71ffda.

6. For Main Logo, enter a Base64-encoded URL string for a PNG image to use as your main logo. The image can be square or wide. For example, data:image/png;base64,iVBORw0...

7. For Square Logo/Favicon, enter a Base64-encoded URL string for a PNG image to use as your favicon, in the Apps Manager header, and in places that require a smaller logo. For example, data:image/png;base64,iVBORw0...

8. For Footer Text, enter a string to be displayed as the footer. If left blank, the footer text defaults to Pivotal Software Inc. All rights reserved.

9. To add up to three footer links that appear to the right of the footer text, complete the following steps:
   - Click Add.
   - For Link text, enter a label for the link.
For URL, enter an external or relative URL. For example, http://docs.pivotal.io or /tools.html.

Apps Manager Config Page

1. In a browser, navigate to the fully qualified domain name (FQDN) of your Ops Manager and log in.

2. Click Pivotal Elastic Runtime.

3. Click the Apps Manager Config tab.

4. For Product Name, enter text to replace Apps Manager in the header and the title of Apps Manager. This text defaults to Apps Manager if left blank.

5. For Marketplace Name, enter text to replace the header in the Marketplace pages. This text defaults to Marketplace if left blank.

6. By default, Apps Manager includes three sidebar links: Marketplace, Docs, and Tools. You can edit existing sidebar links by clicking the name of the link and editing the Link text and Url fields. Or, you can remove the link by clicking the trash icon next to its name. If you want to add a new sidebar link, click Add and complete the Link text and Url fields.

Note: Removing any of the default links will remove them from the sidebar for all users.
Monitoring App and Service Instance Usage

Page last updated:

This topic describes how to use the Cloud Foundry Command Line Interface (cf CLI) to retrieve usage information about your app and service instances through the Cloud Controller and Usage service APIs.

You can also access usage information by using Apps Manager. For more information, see the Monitoring Instance Usage with Apps Manager topic.

Obtain System Usage Information

Before you can retrieve any app or service information, you must target the Cloud Controller and log in as admin, as follows:

1. Target the endpoint of your Cloud Controller.

   $ cf api YOUR-DOMAIN

2. Log in with your credentials.

   $ cf login -u admin

   API endpoint: api.YOUR-DOMAIN
   Email: user@example.com
   Password: 
   Authenticating...
   OK
   Targeted org: YOUR-ORG
   Targeted space: development
   API endpoint: https://api.YOUR-DOMAIN (API version: 2.52.0)
   User: user@example.com
   Org: YOUR-ORG
   Space: development

3. Run `curl` for the /system_usage_report on the Usage service.

   $ curl "https://app-usage.YOUR-DOMAIN/system_usage_report" -k -v -H "authorization:"cf oauth-token"

Obtain Usage Information about an Org

To obtain individual org usage information, use the following procedure. You must log in as an admin or as an Org Manager or Org Auditor for the org you want to view.

1. Run `cf login -u USERNAME`.

2. Run `curl` for the /app_usages or /service_usages endpoints on the Usage service.


   Or run the following:


© Copyright Pivotal Software Inc, 2013-2018
Retrieve Apps Information

1. The `apps` endpoint retrieves information about all of your apps.

```
$ cf curl /v2/apps
{
  "total_results": 2,
  "total_pages": 1,
  "prev_url": null,
  "next_url": null,
  "resources": [
    {
      "metadata": {
        "guid": "acf2ce33-ee92-54TY-9adb-55a596a8dcba",
        "url": "/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba",
        "created_at": "2016-02-06T17:40:31Z",
        "updated_at": "2016-02-06T18:09:17Z"
      },
      "entity": {
        "name": "YOUR-APP",
        ...
      }
    },
    {
      "metadata": {
        "guid": "79bb58cc-3737-4540-ac70-39a2843b5178",
        "url": "/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178",
        "created_at": "2016-02-15T23:25:47Z",
        "updated_at": "2016-03-12T21:54:59Z"
      },
      "entity": {
        "name": "ANOTHER-APP",
        ...
      }
    }
  ]
}
```

The output of this command provides the URL endpoints for each app, within the `metadata.url` section. You can use these app-specific endpoints to retrieve more information about that app. In the example above, the endpoints for the two apps are `/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba` and `/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178`.
2. The summary endpoint under each app-specific URL retrieves the instances and any bound services for that app.

```bash
$ cf curl /v2/apps/acf2ce75-e992-fb6-9ad9-fa5df96a8dcb/summary
{
  "guid": "acf2ce75-e992-fb6-9ad9-fa5df96a8dcb",
  "name": "YOUR-APP",
  "routes": [
    {
      "guid": "7421b6af-75cb-4334-a862-bc5e1ab6f6",
      "host": "YOUR-APP",
      "path": 
    },
    "domain": {
      "guid": "f6bd89f-2e9-40d4-9ad1-97951a573135",
      "name": "YOUR-DOMAIN.io"
    }
  ],
  "running_instances": 5,
  "services": [
    {
      "guid": "f6e4d456-3c61-48ba-a307-9b4t836fb2e",
      "name": "YOUR-APP-db",
      "bound_app_count": 1,
      "last_operation": {
        "type": "create",
        "state": "succeeded",
        "description": ",",
        "updated_at": null,
        "created_at": "2016-02-05T04:58:46Z"
      },
      "service_plan": {
        "guid": "fbcec3af-3e8d-4ec7-adfe-3f12a137e6b",
        "name": "turtle",
        "service": {
          "guid": "34dbc753-34ed-4cf1-9a87-a255dfca5339b",
          "label": "elephantsql",
          "provider": null,
          "version": null
        }
      }
    }
  ]
}
```

3. To view the app_usage report that covers app usage within an org during a period of time, see Obtain Usage Information about an Org.

Retrieve Services Information

Use `cf curl` to retrieve service instance information. The `service_instances` endpoint retrieves details about both bound and unbound service instances:
$ curl /v2/service_instances/
{
  "total_results": 4,
  "total_pages": 1,
  "prev_url": null,
  "next_url": null,
  "resources": [
    {
      "metadata": {
        "guid": "b9cdr456-3c61-4f8a-a307-9bby836db2e",
        "url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9bby836db2e",
        "created_at": "2016-02-05T04:58:46Z",
        "updated_at": null
      },
      "entity": {
        "name": "YOUR-BOUND-DB-INSTANCE",
        "credentials": {},
        "service_plan_guid": "fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "space_guid": "a0205ae0-a691-4667-92bc-0b0d4f126d63",
        "gateway_data": null,
        "type": "managed_service_instance",
        "last_operation": {
          "type": "create",
          "state": "succeeded",
          "description": "",
          "updated_at": null,
          "created_at": "2016-02-05T04:58:46Z"
        },
        "tags": [],
        "space_url": "/v2/spaces/a0205ae0-a691-4667-92bc-0b0d4f126d63",
        "service_plan_url": "/v2/service_plans/fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "service_bindings_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9bby836db2e/service_bindings",
        "service_keys_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9bby836db2e/service_keys",
        "routes_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9bby836db2e/routes"
      }
    },
    {
      "metadata": {
        "guid": "78be3399-bdc7-4fbf-a1a4-6858a58d0e03",
        "url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a58d0e03",
        "created_at": "2016-02-15T23:45:30Z",
        "updated_at": null
      },
      "entity": {
        "name": "YOUR-UNBOUND-DB-INSTANCE",
        "credentials": {},
        "service_plan_guid": "fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "space_guid": "a0205ae0-a691-4667-92bc-0b0d4f126d63",
        "gateway_data": null,
        "dashboard_url": "https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/app/start?serviceUuid=78be3399-bdc7-4fbf-a1a4-6858a58d0e03",
        "type": "managed_service_instance",
        "last_operation": {
          "type": "create",
          "state": "succeeded",
          "description": "",
          "updated_at": null,
          "created_at": "2016-02-15T23:45:30Z"
        },
        "tags": [],
        "space_url": "/v2/spaces/a0205ae0-a691-4667-92bc-0b0d4f126d63",
        "service_plan_url": "/v2/service_plans/fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "service_bindings_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a58d0e03/service_bindings",
        "service_keys_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a58d0e03/service_keys",
        "routes_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a58d0e03/routes"
      }
    }
  ]
}
Deploying Diego for Windows

Page last updated:

Note: This topic is deprecated.

This topic contains instructions for setting up a Windows cell in a Diego deployment. For more information about Diego, see the Diego Architecture topic.

A cell is a virtual machine (VM) that stages, hosts, and manages application lifecycles. You can install a Windows cell into your Pivotal Cloud Foundry (PCF) deployment by connecting directly to a Windows VM.

Limitations

Unsupported Features

Diego for Windows does not yet support the following features:

- Guaranteed binary compatibility with Diego BOSH releases
- BOSH rolling updates for PCF or core operating system updates in Windows
- Fair sharing of CPU resources

Note: Diego for Windows does not enforce any CPU limits.

- Container SSH access from the Cloud Foundry Command Line Interface (cf CLI)
- ICMP egress by default. You must explicitly enable ICMP through security groups.
- Emitting firewall logs into the CF log pipeline

Stability and Scalability Expectations

Capacity planning for Windows instances of PCF varies greatly based on the overhead caused by the components you add to the instance.

Supported Applications

The following application types are known to run correctly on Diego Windows:

- ASP.NET MVC

  Note: Twelve-factor ASP.NET MVC apps compiled against .NET 4.5.1 were tested most extensively.

- Windows-compiled executables
- Batch scripts with a manually specified start command
- WCF Applications

Install

Prerequisites

- A working Diego deployment
- A Windows Server 2012R2 VM instance that is routable to your Diego deployment
  
  See recommended instance types in the GitHub Diego release repo for details.
If you are creating a new Windows image, and not using one predefined and supplied by your IaaS, we recommend using this ISO image as a starting point. You must have an MSDN account to download this ISO image.

We recommend at least 50 gigabytes of storage space for your Windows VM instance.

**Note:** The IP address of your Windows cell must not conflict with the IP addresses of VMs managed by BOSH. To prevent a conflict, use separate subnets in your VPC for the Windows cell and BOSH VMs, or assign the Windows cell an IP address from the Excluded IP Range that you declared in Ops Manager.

## Step 1: Retrieve Setup Files

Perform the following steps to download the necessary setup files:

1. From your Windows cell, navigate to the Elastic Runtime product on Pivotal Network (link deprecated).

2. Deprecated: Select the DiegoWindows file group from the table.

3. Download the setup.ps1 and generate.exe files. Keep this window open to complete the steps below.

   **Note:** If you download the generate.exe file using Internet Explorer, Internet Explorer removes the .exe extension. You must rename the file to add the .exe extension.

## Step 2: Configure Windows Cell

Perform the following steps to configure your Windows cell:

1. Using File Explorer, navigate to the location where you downloaded the setup.ps1 and generate.exe files.

2. Right-click on the setup.ps1 file and select Run with PowerShell. The setup.ps1 script configures Windows features, DNS settings, and the firewall for your Windows cell.

   **Note:** Some IaaSes may require elevated Windows privileges to run the setup.ps1 script. If you receive a PSSecurity Unauthorized Access error, use the Set-ExecutionPolicy Unrestricted PowerShell cmdlt before re-running the setup.ps1 script.

## Step 3: Download Your Manifest
Perform the following steps to download your Cloud Foundry manifest:

1. SSH into your Ops Manager VM. The steps vary depending on your IaaS. For more information, see the [SSH into Ops Manager](#) section of the **Advanced Troubleshooting with the BOSH CLI** topic.

2. From your Ops Manager VM, use the BOSH CLI to target and log in to your BOSH Director. The steps vary depending on whether your PCF deployment uses internal authentication or an external user store. For more information, see the [Log into BOSH](#) section of the **Advanced Troubleshooting with the BOSH CLI** topic.

3. Use the `bosh deployments` command to list your deployments:

   ```
   $ bosh deployments
   Acting as user 'director' on 'p-bosh'
   RSA 1024 bit CA certificates are loaded due to old openssl compatibility
   +-------------------------+-------------------------------+-----------------------------------------------+--------------+
   | Name                    | Release(s)                    | Stemed(s)                                    | Cloud Config |
   +-------------------------+-------------------------------+-----------------------------------------------+--------------+
   | cf-222e1e11111111e1111 | [cf-autocoiling/36]           | bosh-google-kvm-ubuntu-trusty-go_agent/3203.7 | none |
   |                         | [cf-mySql/26.6]               |                                               |              |
   |                         | [cf239/0.26]                 |                                               |              |
   |                         | [cf-firewall/1.33.0]          |                                               |              |
   |                         | [console/1.08.0-2]            |                                               |              |
   |                         | [diego/0.1483.1]             |                                               |              |
   |                         | [exec/60.01]                 |                                               |              |
   |                         | [garden-runc/0.9.2]          |                                               |              |
   |                         | [ipv6/1.5.3.7]               |                                               |              |
   |                         | [mysql-backup/1.25.0]        |                                               |              |
   |                         | [mysql-monitoring/6]         |                                               |              |
   |                         | [notifications-in/17]        |                                               |              |
   |                         | [notifications/24]           |                                               |              |
   |                         | [pivotal-account/1]          |                                               |              |
   |                         | [push-apps-manager-release/652] |                                               |              |
   |                         | [router/0.138.6]             |                                               |              |
   |                         | [service-backup/14]          |                                               |              |
   +-------------------------+-------------------------------+-----------------------------------------------+--------------+
   ```

4. Review the output and identify the name of your Cloud Foundry deployment. In the above example, the name is `cf-222e1e11111111e1111`.

5. Use the `bosh download` command to download the manifest of your Cloud Foundry deployment as `cf.yml`:

   ```
   $ bosh download manifest cf-222e1e11111111e1111 cf.yml
   Acting as user 'director' on deployment 'cf-222e1e11111111e1111' on 'p-bosh'
   RSA 1024 bit CA certificates are loaded due to old openssl compatibility
   Deployment manifest saved to 'cf.yml'
   ```

6. Copy the `cf.yml` file from the Ops Manager VM to your Windows cell in one of two ways:
   - From your Windows cell, use **WinSCP** to copy the manifest from the Ops Manager VM.
   - From your local Mac or Linux machine, use **scp** to copy the manifest from the Ops Manager VM, and then use a Remote Desktop Protocol (RDP) client like **Microsoft Remote Desktop** to mount a directory containing the manifest on your local machine as a drive on the Windows cell. For more information, see the [Microsoft documentation](#).

### Step 4: Run Install Script Generator

From your Windows cell, run `generate.exe` with the following arguments:

- `manifest`: The path to the manifest file downloaded from your Ops Manager BOSH Director
- `outputDir`: The directory that will contain the required certificates and a script to run the installers

```
5 generate.exe
-manifest /tmp/cf.yml
-outputDir C:/diego-windows
```

**Note:** The parameters for `generate.exe` are case-sensitive.

### Step 5: Install MSI

1. Download `DiegoWindows.msi` and `GardenWindows.msi` from the same Pivotal Network file group to the `outputDir` that you specified above.
2. From a command prompt, run `install.bat` from the `outputDir`.

Note: By default, Containerizer stores container files at `C:\containerizer`. To modify the default, open `install.bat` and set `CONTAINER_DIRECTORY` to the directory where you want Containerizer to store container files.
Step 6: Confirm Successful Deployment

Follow the steps below to deploy a sample .NET application to one of your Windows cells and exercise basic Elastic Runtime functionality to ensure that your deployment functions properly.

1. Launch Task Manager.

2. Navigate to the Services tab. Confirm that the following five services are running:
   - ConsulService
   - ContainerizerService
   - GardenWindowsService
   - MetronService
   - RepService

3. Clone the CF Smoke Tests repository.

4. Follow the instructions from the CF Smoke Tests README to run the smoke tests against your environment with the `enable_windows_tests` configuration flag set to `true`.
Operating Diego for Windows

This topic describes how to operate a Diego deployment on Windows. For instructions on setting up a Windows Diego deployment, see the Deploying Diego for Windows topic.

Customize Cells

Pivotal recommends that you keep customization of Windows cells to a minimum. If you do customize your cells, you must apply any software or configuration settings to every cell in your cluster.

Reboot Cells

Before rebooting a Windows cell, you must first trigger an evacuation to avoid application downtime.

To trigger an evacuation, execute the following PowerShell script:

```powershell
Set-Service RepService -startuptype "Disabled"
while ($true) {
    try {
        Get-WebRequest "http://localhost:1800/ping"
    } catch {
        [system.exception]
        break;
    }
}
Set-Service RepService -startuptype "Automatic"
```

Retrieve Version Numbers

To retrieve a version number for an executable or MSI, right-click the file and click Properties.

To retrieve the version number for the `setup.ps1` script, pass the version flag on the command line:

```
powershell \setup.ps1 -version
```

Custom CA Certificates

If your applications require custom CA certificates in order to communicate with other components, install the certificates on the Windows cell. Applications running on the cell will trust certificates that the local machine or domain trust.

See the Manage Trusted Root Certificates TechNet article for information.

Upgrade a Cell

Diego retains backwards compatibility with Windows cells, which allows for rolling upgrades. Greenhouse/.NET implements a cell evacuation prior to new releases to support upgrades.
To upgrade a Windows cell, perform the following steps:

1. Spin up a new cell.
2. Trigger an evacuation on an old cell using the PowerShell script from the Rebooting Cells section above.
3. Shut down the old cell when the evacuation completes.
4. Repeat until all cells are updated.
The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment

The Pivotal Cloud Ops team monitors the health of its Cloud Foundry deployments using a customized Datadog dashboard. This topic describes each of the key metrics as they are rendered in the custom dashboard, and why the Cloud Ops team uses them for monitoring the health of a Cloud Foundry deployment.

**Note:** Pivotal does not officially support Datadog.

Cloud Ops' practices are tailored to the specific details of the Cloud Foundry deployments they operate. Therefore, the descriptions here are meant to be informative examples rather than general prescriptions. Pivotal recommends that operators experiment with different combinations of metrics and alerts appropriate to their specific requirements.

The Cloud Ops team's custom configuration of Datadog's dashboards, alerts, and screenboards can be found in the Datadog Config repository.

---

### BOSH Health Monitor

---
Health, broken down by component. Each row displays the average percentage of healthy instances for the relevant component over the last 5 minutes, and over the last 24 hours.

For example, suppose that your Router has ten instances. If one instance becomes unhealthy, the stoplight turns red and shows 90%.

We monitor health for the following components:

- NATS
- Doppler
- Stats
- HM9000
- BOSH
- NAT Box
- ETCD
- Router
- API
- UAA

Why we monitor it
To ensure that all VMs are functioning properly.

System metric
bosh.healthmonitor.system.healthy

Alerts triggered
None

Notes
Alerts generated from this metric are passed to a buffer queue in our alerting system, Pagerduty. Because BOSH restores systems quickly if they fail, we wait two minutes before forwarding any unresolved alerts to our operators.

Requests per Second

Requests per second for each of the following components:

- Router
- API
- UAA
### NATS Traffic Delta

| What we monitor | Delta of average NATS traffic over the last hour.  
The displayed metric is the difference between the average NATS traffic over the last 30 minutes and the average NATS traffic over the interval from 90 to 60 minutes prior. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>To detect significant drops in NATS traffic. A sudden drop might indicate a problem with the health of the NATS VMs.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>aws.ec2.network_in</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>None</td>
</tr>
<tr>
<td>Notes</td>
<td>None</td>
</tr>
</tbody>
</table>

### ETCD Leader Uptime

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>Time since the ETCD leader last was down.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>When the ETCD leader goes down, it usually indicates a push failure.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>cloudops_tools.etcd_leader_health</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>None</td>
</tr>
<tr>
<td>Notes</td>
<td>The <code>cloudops_tools</code> metrics are generated by an internal app that the Pivotal Cloud Ops team developed. These metrics are not available on other Cloud Foundry deployments.</td>
</tr>
</tbody>
</table>

### SSH Attempts

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>Total SSH attempts. We log the count of connection attempts to our systems on the SSH port (port 22).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>A spike in SSH attempts is a good indicator of SSH-cracker attacks.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>cloudops_tools.ssh-abuse-monitor</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>None</td>
</tr>
</tbody>
</table>
| Notes | ✓ Diego cells send their iptables logs to Logsearch. A Cloud Ops internal app polls Logsearch for first packets and pushes the count to Datadog.  
✓ The `cloudops_tools` metrics are generated by an internal app that the Pivotal Cloud Ops team developed. These metrics are not available on other Cloud Foundry deployments. |
## App Instance Count

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>Count of running app instances.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>Unexpected large fluctuations in app count can indicate malicious user behavior or Cloud Foundry component issues.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>avg:cf.collector.HM9000.HM9000.NumberOfAppsWithAllInstancesReporting</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>running app number change rate</td>
</tr>
<tr>
<td>Notes</td>
<td>Spikes in this metric might indicate the need to add more resources.</td>
</tr>
</tbody>
</table>

## Total Routes

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>Route count from the router, indicated as a delta over the last N minutes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>The count on all routers should be the same. If this count differs between routers, it usually indicates a NATS problem.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>cf.collector.router.total_routes</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>prod CF: Number of routes in the router's routing table is too low</td>
</tr>
<tr>
<td>Notes</td>
<td>The router is the only point of access into all Cloud Foundry components and customer apps. Large spikes in this graph typically indicate a problem, and could indicate a denial of service attack. For example, if the router goes down or does not have routes, the system is down and a large dip appears in the graph. However, some large spikes, such as those that would occur during a marketing event, are expected. Small fluctuations are not reflected on the graph.</td>
</tr>
</tbody>
</table>

## Router Dial Errors

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>Separate indicators monitor 5xx codes from the routers to backend CF components and user apps, respectively.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>Indicates failures connecting to components.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>avg:cloudops_tools.app_instance_monitor.router.dial.errors(domain:run.pivotal.io) / avg:cloudops_tools.app_instance_monitor.router.dial.errors(cf_component:false)</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td></td>
</tr>
</tbody>
</table>
  - No data for router dial errors  
  - Router dial errors for console.run.pivotal.io  
  - Too many router dial errors for cf components |
We investigate dial errors to admin domain apps, the Cloud Controller, UAA, Dopplers, and any other BOSH-deployed Cloud Foundry component. We expect dial errors from our large population of customer apps (4000+). 502s occur when customers push flawed apps, or are running dev iterations. 5xx messages in the 500/10 min range are normal. If we saw this number to jump to 1000+/10 min, we would investigate.

The `cloudops_tools` metrics are generated by an internal app that the Pivotal Cloud Ops team developed. These metrics are not available on other Cloud Foundry deployments.

**Router CPU**

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>OS-level CPU usage.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>Routers are multi-threaded and consume a large number of CPU cycles. If the routers are using too much CPU, we use BOSH to scale them.</td>
</tr>
<tr>
<td>System metric</td>
<td><code>bosh.healthmonitor.system.cpu.user{deployment:cf-cfapps-io2,job:}</code></td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>None</td>
</tr>
<tr>
<td>Notes</td>
<td>In general, we add routers whenever doing so may resolve issues.</td>
</tr>
</tbody>
</table>

**AWS Events**

<table>
<thead>
<tr>
<th>What we monitor</th>
<th>The feed from <code>aws ec2 events</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why we monitor it</td>
<td>Contains important or critical information from our IaaS about virtual machines, RDS, etc.</td>
</tr>
<tr>
<td>System metric</td>
<td>N/A</td>
</tr>
<tr>
<td>Alerts triggered</td>
<td>None</td>
</tr>
<tr>
<td>Notes</td>
<td>Only applies to Cloud Foundry deployments on AWS.</td>
</tr>
</tbody>
</table>
Providing a Certificate for your SSL Termination Point

This topic describes the procedure for providing Pivotal Cloud Foundry (PCF) Elastic Runtime with an SSL certificate, as part of the process of configuring Elastic Runtime for deployment. See Getting Started with Pivotal Cloud Foundry for help installing PCF on your IaaS of choice.

Navigate to Elastic Runtime Networking Configuration

1. Navigate to the Pivotal Cloud Foundry Operations Manager Installation Dashboard.
2. Click the Elastic Runtime tile in the Installation Dashboard.

Configure Router or HAProxy SSL Termination

Configure for a Production Deployment

1. Under Configure the point-of-entry to this environment, choose one of the following:
   - **External Load Balancer with Encryption**: Select this option if your deployment uses an external load balancer that can forward encrypted traffic to the Elastic Runtime Router.
   - **HAProxy**: Select this option to use HAProxy as your first point of entry.

2. Enter your PEM encoded certificate and your PEM encoded private key in the fields under SSL Termination Certificate and Private Key. If your deployment is on AWS, this certificate must match the one that you uploaded to AWS earlier in the Upload an SSL Certificate section of the Deploying the CloudFormation Template for PCF on AWS topic.

   ![SSL Certificate and Private Key](image)

   **Note**: Certificates generated in Elastic Runtime are signed by the Operations Manager Certificate Authority. They are not technically self-signed, but they are sometimes referred to as ‘Self-Signed Certificates’ in the Ops Manager GUI and throughout this documentation.

3. (HAProxy Only) Select Disable HTTP traffic to HAProxy if you want HAProxy to only allow HTTPS traffic.

4. Configure SSL Ciphers. Leave these fields blank unless you want to use a specific set of SSL ciphers for the Router or HAProxy. Enter a colon-separated list of custom SSL ciphers to pass to the Router or HAProxy.

Configure for a Development or Testing Deployment

1. Under Configure the point-of-entry to this environment, choose one of the following:
   - **External Load Balancer with Encryption**: Select this option if your deployment uses an external load balancer that can forward encrypted

© Copyright Pivotal Software Inc, 2013-2018
traffic to the Elastic Runtime Router.

- **HAProxy**: Select this option to use HAProxy as your first point of entry.

2. Click **Generate RSA Certificate** for the Pivotal certificate authority to generate a certificate with the RSA certificate wizard. You may need to click **Change** to populate the UI with the **Generate RSA Certificate** text button.

3. Enter your system and app domains in wildcard format. Optionally, also add custom domains in wildcard format. You can generate a single certificate for two domains separated by a comma, such as `.apps.YOUR-DOMAIN.com,.system.YOUR-DOMAIN.com`. The example below uses `.YOUR-DOMAIN.com`.

4. Click **Generate** to populate the **SSL Certificate** fields with RSA certificate and private key information.

5. **(HAProxy Only)** Select **Disable HTTP traffic to HAProxy** if you want HAProxy to only allow HTTPS traffic.

6. Configure **SSL Ciphers**. Leave these fields blank unless you want to use a specific set of SSL ciphers for the Router or HAProxy. Enter a colon-separated list of custom SSL ciphers to pass to the Router or HAProxy.

7. If you expect requests larger than the default maximum of 16 Kbytes, enter a new value (in bytes) for **Request Max Buffer Size**. You may need to do this, for example, to support apps that embed large cookie or query string values in headers.

8. Click **Save**.

**Note**: SSL certificates generated for wildcard DNS records only work for a single domain name component or component fragment. For example, a certificate generated with `.YOUR-DOMAIN.com` does not work for `.apps.YOUR-DOMAIN.com` and `.system.YOUR-DOMAIN.com`. The certificate must have both `.apps.YOUR-DOMAIN.com` and `.system.YOUR-DOMAIN.com` attributed to it.
Administering and Operating Cloud Foundry

For Administrators of a Running Cloud Foundry Deployment

- Managing Custom Buildpacks
- Adding a Custom Stack
- Managing Domains and Routes
- Creating and Managing Users with the cf CLI
- Creating and Managing Users with the UAA CLI (UAAC)
- Creating and Modifying Quota Plans
- Getting Started with the Notifications Service
- Feature Flags

For Operators Deploying Cloud Foundry

- Enabling IPv6 for Hosted Applications
- Securing Traffic into Cloud Foundry
- Migrating Apps to Diego
- Supporting WebSockets
Managing Custom Buildpacks

This topic describes how an admin can manage additional buildpacks in Cloud Foundry. If your application uses a language or framework that the Cloud Foundry system buildpacks do not support, you can:

- Write your own buildpack
- Customize an existing buildpack
- Use a Cloud Foundry Community Buildpack
- Use a Heroku Third-Party Buildpack

Add a Buildpack

💡 Note: You must be an administrator for your Cloud Foundry org to run the commands discussed in this section.

To add a buildpack, run:

```bash
$ cf create-buildpack BUILDPACK PATH POSITION [--enable|--disable]
```

The arguments to `cf create-buildpack` specify the following:

- **buildpack** specifies the buildpack name.
- **path** specifies where to find the buildpack. The path can point to a zip file, the URL of a zip file, or a local directory.
- **position** specifies where to place the buildpack in the detection priority list. See Buildpack Detection.
- **enable or disable** specifies whether to allow apps to be pushed with the buildpack. This argument is optional, and defaults to enable. While a buildpack is disabled, app developers cannot push apps using that buildpack.

To confirm that you have successfully added a buildpack, run:

```bash
$ cf buildpacks
```

The following example shows the output from running the `cf buildpacks` command after the administrator added a Python buildpack:

```bash
$ cf buildpacks
Getting buildpacks...

<table>
<thead>
<tr>
<th>buildpack</th>
<th>position</th>
<th>enabled</th>
<th>locked</th>
<th>filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruby_buildpack</td>
<td>1</td>
<td>true</td>
<td>false</td>
<td>buildpack_ruby_v46-245-g2fc4ad8.zip</td>
</tr>
<tr>
<td>nodejs_buildpack</td>
<td>2</td>
<td>true</td>
<td>false</td>
<td>buildpack_nodejs_v8-177-g2b0a5cf.zip</td>
</tr>
<tr>
<td>java_buildpack</td>
<td>3</td>
<td>true</td>
<td>false</td>
<td>buildpack_java_v2.1.zip</td>
</tr>
<tr>
<td>python_buildpack</td>
<td>4</td>
<td>true</td>
<td>false</td>
<td>buildpack_python_v2.7.6.zip</td>
</tr>
</tbody>
</table>
```

Rename a Buildpack

```bash
$ cf rename-buildpack BUILDPACK_NAME NEW_BUILDPACK_NAME
```

For more information on renaming a buildpack, see the CLI documentation.

Update a Buildpack

```bash
$ cf update-buildpack BUILDPACK [-p PATH] [-i POSITION] [--enable|--disable] [--lock|--unlock]
```

For more information on updating a buildpack, see the CLI documentation.
Delete a Buildpack

$ cf delete-buildpack BUILDPACK [-f]

For more information on deleting a buildpack, see the [CLI documentation](#).

Lock and Unlock a Buildpack

Every new version of Cloud Foundry includes an updated buildpack. By default, your deployment applies the most recent buildpack when you upgrade. In some cases, however, you may want to preserve an existing buildpack, rather than upgrade to the latest version. For example, if an app you deploy depends on a specific component in Buildpack A that is not available in Buildpack B, you may want to continue using Buildpack A.

The `--lock` flag lets you continue to use your existing buildpack even after you upgrade. Locked buildpacks are not updated when PCF updates. You must manually unlock them to update them.

If you elect to use the `--unlock` flag, your deployment will apply the most recent buildpack when you upgrade PCF.

`cf update-buildpack BUILDPACK [-p PATH] [-i POSITION] [--enable|--disable] [--lock|--unlock]`

This feature is also available via API. For more information, see the [API documentation](#).

Disabling Custom Buildpacks

You can disable custom buildpacks using your Ops Manager Elastic Runtime tile. From the Cloud Controller tab, check the Disable Custom Buildpacks checkbox, as shown in the image below.

By default, the cf CLI gives developers the option of using a custom buildpack when they deploy apps to Elastic Runtime. To do so, they use the `-b` option to provide a custom buildpack URL with the `cf push` command. The Disable Custom Buildpacks checkbox disables the `-b` option.

For more information about custom buildpacks, refer to the [buildpacks](#) section of the PCF documentation.
Adding a Custom Stack

This topic outlines how to add a custom stack under Diego architecture. To add a stack, you first build a BOSH job template that installs the stack on the host machine. Then you configure your deployment manifests so that Cloud Foundry can run the job when it creates cells.

The Cloud Foundry cflinuxfs2 repository contains scripts for building your own custom stacks, as well as the available Cloud Foundry stacks.

The following example adds a new Linux-based pancakes stack for use with the garden-linux operating system. This pancakes stack could, for example, support applications that require an old version of CentOS or Ubuntu.

Step 1: Create a BOSH Job Template

Stacks exist in a subdirectory on their host machine, typically under /var/vcap/packages or /var/vcap/data. Your BOSH job template must deploy the stack onto a host machine, and provide lifecycle binaries that work with your stack. The lifecycle binaries for your stack are helper programs that stage and run apps on the stack file system. To create a pancakes-release job template that deploys a custom stack, follow these steps:

1. Create a BOSH release pancakes for a job template that expands a stack into place in its subdirectory. For example, a pancakes-rootfs template might create a full Linux root file system in the directory /var/vcap/packages/pancakes-rootfs/rootfs. See the 'rootfses' job template in diego-release for one way to do this.

2. Create lifecycle binaries for your stack. See the diego-release repo for examples of app lifecycle binary source code:
   - Buildpack App Lifecycle
   - Docker App Lifecycle
   - Windows App Lifecycle

3. Generate a gzipped tar archive of the lifecycle binaries, pancakes-app-lifecycle.tgz.

4. Create a dummy pancakes-app-lifecycle job template as a package within pancakes-release. Include the pancakes-app-lifecycle.tgz file in the job template directory.

   **Note:** The pancakes-app-lifecycle job template does not need to run any process of its own.

5. List the dummy pancakes-app-lifecycle job as a dependency in the pancakes-release spec file. This makes BOSH publish the lifecycle binaries to /var/vcap/packages for inclusion in any cells that use the pancakes stack.

Step 2: Update the Manifests

1. Add the pancakes-rootfs job and release name to the Diego manifest, to the list of job templates defined for the cell object under base_job_templates. This makes the expanded rootsfs available locally on the Diego cell, at /var/vcap/packages/pancakes-rootfs/rootsfs. For example, in the manifests generated with the spiff-based tooling in diego-release, add the lines shown in bold to the following list of cell job templates:

```json
  cell:
  - name: rep
    release: diego
  - name: consul_agent
    release: cf
  - name: garden
    release: garden-linux
  - name: rootfses
    release: diego
  - name: pancakes-rootfs
    release: pancakes
  - name: metron_agent
    release: cf
```

2. Add pancakes-app-lifecycle to the base_job_templates list under the file_server Diego job. In diego-release, the file_server job resides in the access job template group. For example, add the lines shown in bold to the following list of job templates:

```json
  access:
  - name: ssh_proxy
    release: diego
  - name: consul_agent
```

© Copyright Pivotal Software Inc, 2013-2018
3. The `diego.rep.preloaded_rootfses` property of the Cell Rep holds an array associating stacks with their file system root locations. Add a pair to this list to associate the `pancakes` stack with its file system root location, set up on the cell by the `pancakes-rootfs` job. For example, in the property under `{FP}` in your {Diego manifest}, set the array to the following by adding the text shown in **bold**:

```
["cflinuxfs2:/var/vcap/packages/cflinuxfs2/rootfs",
"pancakes:/var/vcap/packages/pancakes-rootfs/rootfs"]
```

4. Configure the stager and nsync components to use the `pancakes` lifecycle binary bundle to start and stop apps running on the `pancakes` stack. For example, in {CAPI release}, add the line shown in **bold** to the `default` list under the manifest definitions for both `diego.nsync.lifecycle_bundles` and `diego.stager.lifecycle.bundles`:

```
description: "List of lifecycle bundles arguments for different stacks in form 'lifecycle-name:path/to/bundle'"
default:
- "buildpack/cflinuxfs2:buildpack_app_lifecycle/buildpack_app_lifecycle.tgz"
- "buildpack/pancakes:pancakes-app-lifecycle/pancakes-app-lifecycle.tgz"
- "buildpack/windows2012R2:windows_app_lifecycle/windows_app_lifecycle.tgz"
- "docker:docker_app_lifecycle/docker_app_lifecycle.tgz"
```

5. Configure the Cloud Controller for the new stack by adding it to the `cc.stacks` property in the CF manifest. For example, in the {diego-release manifest generation stubs for CF}, add the lines shown in **bold**:

```
properties:
 cc:
  stacks:
  - name: "cflinuxfs2"
    description: "Cloud Foundry Linux-based filesystem"
  - name: "windows2012R2"
    description: "Windows Server 2012 R2"
  - name: "pancakes"
    description: "Linux-based filesystem, with delicious pancakes"
```
Managing Domains and Routes

If you are an administrator, you can manage custom shared domains and wildcard routes.

For additional information about managing routes and domains, refer to the following topic:

- Routes and Domains

Creating a Shared Custom Domain

You can use a registered domain of your own and associate it with all organizations in your account. Use the `cf create-shared-domain` command to create a shared custom domain available to all organizations in your account.

For example:

```
$ cf create-shared-domain shared-domain.example.com
```

Deleting a Shared Custom Domain

Use the `cf delete-shared-domain` command to delete a shared domain:

```
$ cf delete-shared-domain shared-domain.example.com
```

Note: Deleting a shared domain removes all associated routes, making any application with this domain unreachable.

Creating a Wildcard Route

Use the `cf create-route` command with a wildcard route by specifying the host as `*`. The star operator, `*`, signals a wildcard route to match any URL that uses your domain, regardless of the host.

Note: You must surround the `*` with quotation marks when referencing it using the CLI.

For example, the following command created the wildcard route `*.example.org` in the “development” space:

```
$ cf create-route development example.org -n "*"
```
Creating and Managing Users with the cf CLI

Page last updated:

Using the Cloud Foundry Command Line Interface (cf CLI), an administrator can create users and manage user roles. Cloud Foundry uses role-based access control, with each role granting permissions in either an organization or an application space.

For more information, see Organizations, Spaces, Roles, and Permissions.

Note: To manage users, organizations, and roles with the cf CLI, you must log in with UAA Administrator user credentials. In Pivotal Operations Manager, refer to Elastic Runtime > Credentials for the UAA admin name and password.

Creating and Deleting Users

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>COMMAND</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a new user</td>
<td>cf create-user USERNAME PASSWORD</td>
<td>cf create-user Alice pa55w0rd</td>
</tr>
<tr>
<td>Delete a user</td>
<td>cf delete-user USERNAME</td>
<td>cf delete-user Alice</td>
</tr>
</tbody>
</table>

Creating Administrator Accounts

To create a new administrator account, use the UAA CLI.

Note: The cf CLI cannot create new administrator accounts.

Org and App Space Roles

A user can have one or more roles. The combination of these roles defines the user’s overall permissions in the org and within specific app spaces in that org.

Org Roles

Valid org roles are OrgManager, BillingManager, and OrgAuditor.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>COMMAND</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>View the organizations belonging to an account</td>
<td>cf orgs</td>
<td>cf orgs</td>
</tr>
<tr>
<td>View all users in an organization by role</td>
<td>cf org-users ORGANIZATION_NAME</td>
<td>cf org-users my-example-org</td>
</tr>
<tr>
<td>Assign an org role to a user</td>
<td>cf set-org-role USERNAME ORGANIZATION_NAME ROLE</td>
<td>cf set-org-role Alice my-example-org OrgManager</td>
</tr>
<tr>
<td>Remove an org role from a user</td>
<td>cf unset-org-role USERNAME ORGANIZATION_NAME ROLE</td>
<td>cf unset-org-role Alice my-example-org OrgManager</td>
</tr>
</tbody>
</table>

App Space Roles

Each app space role applies to a specific app space.

Valid app space roles are SpaceManager, SpaceDeveloper, and SpaceAuditor.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>COMMAND</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>View the spaces in an org</td>
<td>cf spaces</td>
<td>cf spaces</td>
</tr>
<tr>
<td>View all users in a space</td>
<td>cf space-users ORGANIZATION_NAME SPACE_NAME</td>
<td>cf space-users my-example-org development</td>
</tr>
<tr>
<td>by role</td>
<td>command</td>
<td>example</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Assign a space role to a user</td>
<td><code>cf set-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE</code></td>
<td><code>cf set-space-role Alice my-example-org development SpaceAuditor</code></td>
</tr>
<tr>
<td>Remove a space role from a user</td>
<td><code>cf unset-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE</code></td>
<td><code>cf unset-space-role Alice my-example-org development SpaceAuditor</code></td>
</tr>
</tbody>
</table>
Creating and Managing Users with the UAA CLI (UAAC)

Using the UAA Command Line Interface (UAAC), an administrator can create users in the User Account and Authentication (UAA) server.

Note: The UAA only creates users in UAA, and does not assign roles in the Cloud Controller database (CCDB). In general, administrators create users with Cloud Foundry Command Line Interface (cf CLI). The cf CLI both creates user records in the UAA and associates them with org and space roles in the CCDB. Before administrators can assign roles to the user, the user must log in via Apps Manager or the cf CLI in order for the user record to populate the CCDB. Review the Creating and Managing Users with the cf CLI topic for more information.

For additional details and information, refer to the following topics:

- UAA Overview
- UAA Sysadmin Guide
- Other UAA Documentation

Create an Admin User

1. Install the UAA CLI, `uaac`.

   ```
   $ gem install cf-uaac
   ```

2. Use `uaac target uaa.YOUR-DOMAIN` to target your UAA server.

   ```
   $ uaac target uaa.example.com
   ```

3. Record the `uaa:admin:client_secret` from your deployment manifest.

4. Use `uaac token client get admin --client-secret ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAA server. UAAC stores the token in `~/.uaac.yml`.

   ```
   $ uaac token client get admin --client-secret MyAdminPassword
   ```

5. Use `uaac contexts` to display the users and applications authorized by the UAA server, and the permissions granted to each user and application.

   ```
   $ uaac contexts
   ```

6. In the output from `uaac contexts`, search in the `scope` section of the `client_id: admin` user for `scim.write`. The value `scim.write` represents sufficient permissions to create accounts.

7. If the admin user lacks permissions to create accounts:
   - Use `uaac client update admin --authorities "EXISTING_PERMISSIONS scim.write"` to add the necessary permissions to the admin user account on the UAA server. Replace `EXISTING_PERMISSIONS` with the current contents of the `scope` section from `uaac contexts`.
   - Use `uaac token delete` to delete the local token.
   - Use `uaac token client get admin` to obtain an updated access token from the UAA server.
Grant Admin Permissions to an LDAP Group

To grant all users under an LDAP Group admin permissions:

1. Obtain the credentials of an admin client created using UAAC as above, or refer to the `uaa:scim` section of your deployment manifest for the user name and password of an admin user.

2. Use `uaac token client get admin -s ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAA server. UAAC stores the token in `~/.uaac.yml`.

3. Run the following commands to grant all users under the mapped LDAP Group admin permissions:
   - `uaac group map --name scim.read "GROUP-DISTINGUISHED-NAME"`
   - `uaac group map --name cloud_controller.admin "GROUP-DISTINGUISHED-NAME"`

Create Users

1. Obtain the credentials of an admin client created using UAAC as above, or refer to the `uaa:scim` section of your deployment manifest for the user name and password of an admin user.

2. Use `cf login -u NEW-ADMIN-USERNAME -p NEW-ADMIN-PASSWORD` to log in.

3. Use `cf create-user NEW-USER-NAME NEW-USER-PASSWORD` to create a new user.

Change Passwords

1. Obtain the credentials of an admin client created using UAAC as above, or refer to the `uaa:scim` section of your deployment manifest for the user name and password of an admin user.

2. Use `uaac token client get admin -s ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAA server. UAAC stores the token in `~/.uaac.yml`.
$ uaac token client get admin -o MyAdminPassword

3. Use `uaac contexts` to display the users and applications authorized by the UAA server, and the permissions granted to each user and application.

$ uaac contexts

[1]*[admin]
  client_id: admin
  access_token: yJhbGciOiJIUzI1NiJ9.e
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin.clients.secret.password.read
  jti: 91b3-abcd1233

4. In the output from `uaac contexts`, search in the `scope` section of the `client_id: admin` user for `password.write`. The value `password.write` represents sufficient permissions to change passwords.

5. If the admin user lacks permissions to change passwords:
   - Use `uaac client update admin --authorities "EXISTING-PERMISSIONS password.write"` to add the necessary permissions to the admin user account on the UAA server. Replace `EXISTING-PERMISSIONS` with the current contents of the `scope` section from `uaac contexts`.
   - Use `uaac token delete` to delete the local token.
   - Use `uaac token client get admin` to obtain an updated access token from the UAA server.

$ uaac contexts

[1]*[admin]
  client_id: admin
  ...
  scope: uaa.admin.clients.secret.password.read
  ...

$ uaac client update admin --authorities "uaac client get admin | awk '/authorities/ {e=1;e=e+1;if(e==1)print} */" password.write"

$ uaac token delete

$ uaac token client get admin

6. Use `uaac password set USER-NAME -p TEMP-PASSWORD` to change an existing user password to a temporary password.

$ uaac password set Charlie -p ThisIsATempPassword

7. Provide the `TEMP-PASSWORD` to the user. Have the user use `cf target api.YOUR-DOMAIN`, `cf login -u USER-NAME -p TEMP-PASSWORD`, and `cf passwd` to change the temporary password. See the Configuring UAA Password Policy topic to configure the password policy.

$ cf target api.example.com
$ cf login -u Charlie -p ThisIsATempPassword
$ cf passwd

Current Password>ThisIsATempPassword

New Password>******

Verify Password>******

Changing password...

Retrieve User Email Addresses

Some Cloud Foundry components, like Cloud Controller, only use GUIDs for user identification. You can use the UAA to retrieve the emails of your Cloud Foundry instance users either as a list or for a specific user with that user’s GUID.

To retrieve user email addresses:

1. Use `uaac target uaa.YOUR-DOMAIN` to target your UAA server.

$ uaac target uaa.example.com
2. Record the `uaa:admin:client_secret` from your deployment manifest.

3. Use `uaa token client get admin -s ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAA server. UAAC stores the token in `~/.uaac.yml`.

```
$ uaac token client get admin -s MyAdminPassword
```

4. Use `uaac contexts` to display the users and applications authorized by the UAA server, and the permissions granted to each user and application.

```
$ uaac contexts
[1][admin]
  client_id: admin
  access_token: yRbGc10JIIud1NI3N
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin clients.secret
  jti: 9B3-b4d123
```

5. In the output from `uaac contexts`, search in the `scope` section of the `client_id: admin` user for `scim.read`. The value `scim.read` represents sufficient permissions to query the UAA server for user information.

6. If the admin user lacks permissions to query the UAA server for user information:
   - Use `uaac client update admin --authorities "EXISTING-PERMISSIONS scim.write"` to add the necessary permissions to the admin user account on the UAA server. Replace `EXISTING-PERMISSIONS` with the current contents of the `scope` section from `uaac contexts`.
   - Use `uaac token delete` to delete the local token.
   - Use `uaac token client get admin` to obtain an updated access token from the UAA server.

```
$ uaac contexts
[1][admin]
  client_id: admin
  ...
  scope: uaa.admin clients.secret
  ...
$ uaac client update admin --authorities "uaa.admin clients.secret scim.read"
$ uaac token delete
$ uaac token client get admin
```

7. Use `uaac users` to list your Cloud Foundry instance users. By default, the `uaac users` command returns information about each user account including GUID, name, permission groups, activity status, and metadata. Use the `--attributes emails` or `-a emails` flag to limit the output of `uaac users` to email addresses.

```
$ uaac users --attributes emails
resources:
  emails:
    value: user1@example.com
  emails:
    value: user2@example.com
  emails:
    value: user3@example.com
```

8. Use `uaac users "id eq GUID"--attributes emails` with the GUID of a specific user to retrieve that user's email address.

```
$ uaac users "id eq 'aabbcc11-22a5-87-8056-beaf84'"--attributes emails
resources:
  emails:
    value: user1@example.com
```
Creating and Modifying Quota Plans

Quota plans are named sets of memory, service, and instance usage quotas. For example, one quota plan might allow up to 10 services, 10 routes, and 2 GB of RAM, while another might offer 100 services, 100 routes, and 10 GB of RAM. Quota plans have user-friendly names, but are referenced in Cloud Foundry (CF) internal systems by unique GUIDs.

Quota plans are not directly associated with user accounts. Instead, every org has a list of available quota plans, and the account admin assigns a specific quota plan from the list to the org. Everyone in the org shares the quotas described by the plan. There is no limit to the number of defined quota plans an account can have, but only one plan can be assigned at a time.

You must set a quota plan for an org, but you can choose whether to set a space quota.

Org Quota Plan Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Valid Values</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name you use to identify the plan</td>
<td>A sequence of letters, digits, and underscore characters. Quota plan names within an account must be unique.</td>
<td>silver_quota</td>
</tr>
<tr>
<td>memory_limit</td>
<td>Maximum memory usage allowed</td>
<td>An integer and a unit of measurement like M, MB, G, or GB</td>
<td>2048M</td>
</tr>
<tr>
<td>app_instance_limit</td>
<td>Maximum app instances allowed</td>
<td>An integer</td>
<td>25</td>
</tr>
<tr>
<td>non_basic_services_allowed</td>
<td>Determines whether users can provision instances of non-free service plans. Does not control plan visibility. When false, non-free service plans may be visible in the marketplace but instances cannot be provisioned.</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>total_routes</td>
<td>Maximum routes allowed</td>
<td>An integer</td>
<td>500</td>
</tr>
<tr>
<td>total_reserved_route_ports</td>
<td>Maximum routes with reserved ports</td>
<td>An integer not greater than total_routes</td>
<td>60</td>
</tr>
<tr>
<td>total_services</td>
<td>Maximum services allowed</td>
<td>An integer</td>
<td>25</td>
</tr>
<tr>
<td>trial_db_allowed</td>
<td>Legacy Field. Value can be ignored.</td>
<td>true or false</td>
<td>true</td>
</tr>
</tbody>
</table>

Default Quota Plan for an Org

Cloud Foundry installs with a quota plan named default with the following values:

- Memory Limit: 10240 MB
- Total Routes: 1000
- Total Services: 100
- Non-basic Services Allowed: True
- Trial DB Allowed: True

Create a New Quota Plan for an Org

Note: The org manager sets and manages quotas. See the Orgs, Spaces, Roles, and Permissions topic for more information.

You must set an org quota. You can create a new quota plan for an org in one of two ways:

- Directly modify the CF deployment manifest before deploying
- Use cf create-quota after deploying
Modify the CF Deployment Manifest

The CF Deployment manifest specifies the default quota plans applied to orgs. Follow the steps below to modify the default quota plans by locating and editing the manifest.

1. In a terminal window, run `bosh edit deployment` to open the deployment manifest YAML file in your default text editor.

2. Search for `quota_definitions`.

3. Add a new quota definition with values that you specify. Use the default quota definition as a formatting template. The following example shows the `quota_definitions` portion of the `cf.yml` manifest after adding the `silver_quota` plan:

   ```yaml
   quota_definitions:
   default:
     memory_limit: 1024M
     non_basic_services_allowed: true
     total_routes: 1000
     total_services: 100
     trial_db_allowed: true
   silver_quota:
     memory_limit: 2048M
     non_basic_services_allowed: true
     total_routes: 500
     total_services: 25
     trial_db_allowed: true
   ```

4. Save and close the deployment manifest.

5. Run `bosh deploy` to apply the change.

Use `cf create-quota`

In a terminal window, run the following command. Replace the placeholder attributes with the values for this quota plan:

```
cf create-quota QUOTA [-m TOTAL_MEMORY] [-i INSTANCE_MEMORY] [-r ROUTES] [-s SERVICE_INSTANCES] [--allow-paid-service-plans]
```

This command accepts the following flags:

- `-m`: Total amount of memory
- `-i`: Maximum amount of memory an application instance can have (\(-i\) represents an unlimited amount)
- `-r`: Total number of routes
- `-s`: Total number of service instances
- `--allow-paid-service-plans`: Can provision instances of paid service plans

Example:

```
$ cf create-quota small -m 2048M -i 1024M -r 10 -s 10 --allow-paid-service-plans
```

Modify an Existing Quota Plan for an Org

You can modify an existing quota plan for an org in one of two ways:

- Directly modify the CF deployment manifest before deploying.
- Use `cf update-quota` after deploying.

Modify the Manifest

1. In a terminal window, run `bosh edit deployment` to open the deployment manifest YAML file in your default text editor.

2. Search for `quota_definitions`.
3. Modify the value of the attribute.

4. Save and close the deployment manifest.

**Use cf update-quota**

1. Run `cf quotas` to find the names of all quota definitions available to your org. Note the name of the quota plan to be modified.

```
$ cf quotas
Getting quotas as admin@example.com...
OK
```

<table>
<thead>
<tr>
<th>name</th>
<th>total memory limit</th>
<th>instance memory limit</th>
<th>routes</th>
<th>service instances</th>
<th>paid service plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>0</td>
<td>disallowed</td>
</tr>
<tr>
<td>paid</td>
<td>10G</td>
<td>0</td>
<td>1000</td>
<td>-1</td>
<td>allowed</td>
</tr>
<tr>
<td>small</td>
<td>2G</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>allowed</td>
</tr>
<tr>
<td>trial</td>
<td>2G</td>
<td>0</td>
<td>1000</td>
<td>10</td>
<td>disallowed</td>
</tr>
</tbody>
</table>

2. Run the following command, replacing QUOTA with the name of your quota.

```
```

This command accepts the following flags:

- `-i`: Maximum amount of memory an application instance can have (`-1` represents an unlimited amount)
- `-m`: Total amount of memory a space can have
- `-n`: New name
- `-r`: Total number of routes
- `-s`: Total number of service instances
- `--allow-paid-service-plans`: Can provision instances of paid service plans
- `--disallow-paid-service-plans`: Can not provision instances of paid service plans

Example:
```
$ cf update-quota small -i 2048M -m 4096M -r 20 -s 20 --allow-paid-service-plans
```

**Create and Modify Quota Plans for a Space**

For each org, Org Managers create and modify quota plans for spaces in the org. If an Org Manager allocates a space quota, CF verifies that resources do not exceed the allocated space limit. For example, when a Space Developer deploys an application, CF first checks the memory allocation at the space level, then at the org level.

Perform the following procedures to create and modify quota plans for individual spaces within an org.

**Create a New Quota Plan for a Space**

In a terminal window, run the following command to create a quota for a space. Replace the placeholder attributes with the values for this quota plan:

```
cf create-space-quota QUOTA [-i INSTANCE_MEMORY] [-m MEMORY] [-r ROUTES] [-s SERVICE_INSTANCES] [--allow-paid-service-plans]
```

Example:
```
$ cf create-space-quota big -i 1024M -m 4096M -r 20 -s 20 --allow-paid-service-plans
```

**Modify a Quota Plan for a Space**

Run `cf space-quotas` to find the names of all space quota available to your org. Note the name of the quota plan to be modified.
To modify that quota, use the `update-space-quota` command. Replace the placeholder attributes with the values for this quota plan.

Example:

```bash
cf update-space-quota big -i 20 -m 4096M -n bigger -r 20 -s 20 --allow-paid-service-plans
```

Run cf help

For more information regarding quotas, run `cf help` to view a list and brief description of all cf CLI commands. Scroll to view org and space quotas usage and information.
Getting Started with the Notifications Service

Page last updated:

This topic describes how to use the Notifications Service, including how to create a client, obtain a token, register notifications, create a custom template, and send notifications to your users.

Prerequisites

You must have the following setup before using the Notifications Service:

- Install [Elastic Runtime](#).
- You must have `admin` permissions on your Cloud Foundry instance. You also must configure [Application Security Groups (ASGs)](#).
- Install the [cf CLI](#) and [User Account and Authorization Server (UAAC)](#) command line tools.

Create a Client and Get a Token

To interact with the Notifications Service, you need to create [UAAC](#) scopes:

1. Use `uaac target uaa.YOUR-DOMAIN` to target your UAA server.
   
   ```
   $ uaac target uaa.example.com
   ```

2. Record the `uaa:admin:client_secret` from your deployment manifest.

3. Use `uaac token client get admin -s ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAA server. UAAC stores the token in `~/.uaac.yml`.
   
   ```
   $ uaac token client get admin -s MyAdminPassword
   ```

4. Create a `notifications-admin` client with the required scopes.
   
   ```
   $ uaac client add notifications-admin --authorized_grant_types client_credentials --authorities notifications.manage,notifications.write,notification_templates.write,notification_templates.read,critical_notifications.write
   ```
   
   - `notifications.write`: send a notification. For example, you can send notifications to a user, space, or everyone in the system.
   - `notifications.manage`: update notifications and assign templates for that notification.
   - (Optional) `notification_templates.write`: create a custom template for a notification.
   - (Optional) `notification_templates.read`: check which templates are saved in the database.

5. Log in using your newly created client:
   
   ```
   $ uaac token client get notifications-admin
   ```
   
   **Note**: Stay logged in to this client to follow the examples in this topic.

Register Notifications

**Note**: To register notifications, you must have the `notifications.manage` scope on the client. To set critical notifications, you must have the `critical_notifications.write` scope.

You must register a notification before sending it. Using the token `notifications-admin` from the previous step, the following example registers two notifications with the following properties:
Create a Custom Template

A template is made up of a name, a subject, a text representation of the template you are sending for mail clients that do not support HTML, and an HTML version of the template.

The system provides a default template for all notifications, but you can create a custom template using the following curl command.

```
\$ curl https://notifications.user.example.com/templates -X POST --data '{
  "name": "site-maintenance",
  "subject": "Maintenance: \{.Subject\}",
  "text": "The site has gone down for maintenance. More information to follow \{.Text\}",
  "html": "The site has gone down for maintenance. More information to follow \{.HTML\}"}
```

Variables that take the form `{{.}}` interpolate data provided in the send step before a notification is sent. Data that you can insert into a template during the send step include `{{.Text}}`, `{{.HTML}}`, and `{{.Subject}}`.

This curl command returns a unique template ID that can be used in subsequent calls to refer to your custom template. The result looks similar to this:

```
{"template-id": "E3710280-954B-4147-B7E2-AF5BF62772B5"}
```

Check all of your saved templates by running a curl command:

```
\$ curl https://notifications.user.example.com/templates -X GET
```

Associate a Custom Template with a Notification

In this example, the system-going-down notification belonging to the notifications-admin client is associated with the template ID E3710280-954B-4147-B7E2-AF5BF62772B5. This is the template ID of the template we created in the previous section.

Associating a template with a notification requires the notifications.manage scope.

```
\$ curl https://notifications.user.example.com/clients/notifications-admin/notifications/system-going-down/template -X PUT --data '{"template": "E3710280-954B-4147-B7E2-AF5BF62772B5"}"
```

Any notification that does not have a custom template applied, such as system-up, defaults to a system-provided template.
Feature Flags

Page last updated:

Feature flags are switches that you set using the Cloud Controller API. They allow an administrator to turn on or off functional sections of code, or features, of an application without deploying new code. Use feature flags to enable or disable features available to users.

Feature Flags

There are ten feature flags that you can set. They are all enabled by default except user_org_creation, diego_docker, and task_creation. When disabled, these features are only available to administrators.

- **user_org_creation**: Any user can create an organization via the API. Minimum CC API version: 2.12
- **private_domain_creation**: An organization manager can create private domains for that organization. Minimum CC API version: 2.12
- **app_bits_upload**: Space developers can upload app bits. Minimum CC API version: 2.12
- **app_scaling**: Space developers can perform scaling operations (i.e. change memory, disk, or instances). Minimum CC API version: 2.12
- **route_creation**: Space developers can create routes in a space. Minimum CC API version: 2.12
- **service_instance_creation**: Space developers can create service instances in a space. Minimum CC API version: 2.12
- **diego_docker**: Space developers can push docker apps. Minimum CC API version: 2.33
- **set_roles_by_username**: Org Managers and Space Managers can add roles by username. Minimum CC API version: 2.37
- **unset_roles_by_username**: Org Managers and Space Managers can remove roles by username. Minimum CC API version: 2.37
- **task_creation**: Space developers can create tasks on their application. This feature is under development.

Feature Flag Commands

Get All Feature Flags

cf feature-flags

Get status of a Feature Flag

cf feature-flag FEATURE_FLAG_NAME

Enable a Feature Flag

cf enable-feature-flag FEATURE_FLAG_NAME

Disable a Feature Flag

cf disable-feature-flag FEATURE_FLAG_NAME

To view the feature flag commands, review the Feature Flags section of the Cloud Foundry API documentation.
Enabling IPv6 for Hosted Applications

The procedure described below allows apps deployed to Elastic Runtime to be reached using IPv6 addresses.

Note: Amazon Web Services (AWS) EC2 instances currently do not support IPv6.

Elastic Runtime system components use a separate DNS subdomain from hosted applications. These components currently support only IPv4 DNS resolved addresses. This means that although an IPv6 address can be used for application domains, the system domain must resolve to an IPv4 address.

Complete the following steps to enable support for IPv6 application domains:

1. Set up an external load balancer for your Elastic Runtime deployment. See Using Your Own Load Balancer.

2. Configure DNS to resolve application domains to an IPv6 address on your external load balancer.

Note: Your IPv4 interface for the system domain and IPv6 interface for application domain can be configured on the same or different load balancers.

3. Configure the external load balancer to route requests for an IPv6 address to an IPv4 address as follows:
   - If you are using the HAProxy load balancer for SSL termination, route to its IPv4 address.
   - Otherwise, route directly to the IPv4 addresses of the GoRouters.

The following diagram illustrates how a single load balancer can support traffic on both IPv4 and IPv6 addresses for a Elastic Runtime installation.

See Routes and Domains for more information about domains in Elastic Runtime.
Securing Traffic into Cloud Foundry

This topic describes how to secure traffic into your Cloud Foundry (CF) deployment with SSL/TLS certificates. You can configure your deployment to specify where to terminate TLS depending on your needs. You must also configure the load balancer to append the X-Forwarded-For and X-Forwarded-Proto HTTP headers to the HTTP traffic it passes to the router.

The advantages of securing the connection between your load balancer and router instances include:

- A high level of security.
- The ability to secure a deployment that shares a load balancer with other deployments.
- The ability to manage certificates as part of your deployment manifest, which eliminates the need for out-of-band configuration of your load balancer.

For more information on features of HTTP routing handled by the Cloud Foundry router, see the HTTP Routing topic.

Requirements

- You must obtain an SSL/TLS certificate. In a production environment, use a signed SSL/TLS certificate from a known certificate authority (CA). In a development or testing environment, you may use a self-signed certificate generated with `openssl` or a similar tool.
- The CF router only supports HTTP requests. Whatever configuration you choose for securing traffic, it must result in the CF router receiving HTTP requests.
- The CF router currently only supports configuring a single HTTPS certificate. Subject Alternative Name (SAN), an X.509 extension, can be used to associate multiple domains, including wildcard domains, to a single certificate.

Terminate TLS at Router Only

In this configuration, the load balancer does not terminate TLS for CF domains at all. Instead, it passes through the underlying TCP connection to the router. This is the more performant option, establishing and terminating a single TLS connection. The certificate on the router must be associated with the correct hostname so that HTTPS can validate the request, and signed by a trusted CA.

Traffic between the load balancer and the router is encrypted only if the client request is encrypted.

The router appends the X-Forwarded-For and X-Forwarded-Proto headers to requests forwarded to applications and platform system components. X-Forwarded-For is set to the IP address of the client, and X-Forwarded-Proto to the scheme of the client request.

To enable this configuration, perform the following steps:

1. Configure your load balancer to pass through requests from the client to the CF router.

2. Insert the certificate into your deployment manifest for the CF router:
   
   a. Use `bosh edit deployment` to open your release manifest for editing.
   b. Copy the contents of your certificate file into the `properties.router.ssl_cert` field and the contents of the private key file associated with your certificate into the `properties.router.ssl_key` field. Set `enable_ssl` to `true`.

Note: This is the recommended approach.
Terminate TLS at Load Balancer and Router

In this configuration, CF establishes and terminates two TLS connections: one from the client to the load balancer, and another from the load balancer to the CF router. This option is less performant, but it does allow for multiple certificates to be used, meaning multiple domains can be verified when using HTTPS. Certificates for the CF domains must be stored on the load balancer and, if hostname validation is enabled, on the CF router (using a single certificate). The certificate on the CF router only needs to be trusted by the load balancer, and the domain does not need to match the request as long as hostname verification is not enabled on the load balancer.

This configuration secures all traffic between the load balancer and the router.

**Note:** If your CF deployment uses a self-signed certificate, the client needs to install only the certificate stored on the load balancer.

To enable this configuration, perform the following steps:

1. Add your certificate to your load balancer and configure its listening port. The procedures vary depending on your IaaS.

2. Configure your load balancer to append the X-Forwarded-For and X-Forwarded-Proto headers to client requests.

   If the load balancer cannot be configured to provide the X-Forwarded-For header, the CF router will append it in requests forwarded to applications and system components, set to the IP address of the load balancer.

   **Note:** If the load balancer accepts unencrypted requests, it must provide the X-Forwarded-Proto header. Conversely, if the load balancer cannot be configured to send the X-Forwarded-Proto header it should not accept unencrypted requests. Otherwise, applications and platform system components that require encrypted client requests will accept unencrypted requests when they shouldn’t.

3. Insert the certificate into your deployment manifest for the CF router.
   a. Use `bosh edit deployment` to open your release manifest for editing.
   b. Copy the contents of your certificate file into the `properties.router.ssl_cert` field and the contents of the private key file associated with your certificate into the `properties.router.ssl_key` field. Set `enable_ssl` to `true`.

**Note:** Hostname verification between the load balancer and CF router is unnecessary when the load balancer is already configured with the CF router’s IP address to correctly route the request. If DNS resolution is being used by the load balancer to route requests to the CF routers, you should enable hostname verification.
Terminate TLS at Load Balancer Only

In this configuration, your load balancer terminates TLS, and passes unencrypted traffic to the router, which routes it to your app. Traffic between the load balancer and the router is not encrypted.

To enable this configuration, you must perform the following steps:

1. Configure your load balancer to append the X-Forwarded-For and X-Forwarded-Proto headers to client requests.

   If the load balancer cannot be configured to provide the X-Forwarded-For header, the CF router will append it in requests forwarded to applications and system components, set to the IP address of the load balancer.

   **Note:** When terminating TLS at the load balancer only, the load balancer **must** be configured to forward the X-Forwarded-Proto header. Otherwise requests to some applications and system components will result in redirect loops, as they will redirect requests determined to be unencrypted to HTTPS.

2. Add your certificate to your load balancer and configure its listening port. The procedures vary depending on your IaaS.
Migrating Apps to Diego

This topic describes how to migrate the apps in your deployment from the older DEA architecture to the newer Diego architecture. Both operators and app developers have procedures that they need to carry out to migrate apps to the Diego architecture.

For information about the differences between Diego and DEA, see the Differences Between DEA and Diego Architectures topic.

⚠️ WARNING: Before you upgrade to Pivotal Cloud Foundry v1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. Pivotal does not support DEA architecture in Pivotal Cloud Foundry v1.7.

The Stages for the App Migration Process

The following table describes the process and roles required to carry out the migration of apps from the DEA to the Diego architecture. Detailed instructions for each stage of the process are given below.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Who</th>
<th>Does What</th>
<th>Link to Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operators</td>
<td>Plan the migration schedule.</td>
<td>link</td>
</tr>
<tr>
<td>2</td>
<td>Operators</td>
<td>Prepare the deployment for Diego.</td>
<td>link</td>
</tr>
<tr>
<td>3</td>
<td>Developers</td>
<td>Determine if changes need to be made to their environment variables to be compatible with Diego.</td>
<td>link</td>
</tr>
<tr>
<td>4</td>
<td>Developers</td>
<td>Migrate apps to Diego. The method depends on whether downtime for the app is tolerable.</td>
<td>link</td>
</tr>
<tr>
<td>5</td>
<td>Developers</td>
<td>Verify that their apps have been migrated.</td>
<td>link</td>
</tr>
<tr>
<td>6</td>
<td>Operators</td>
<td>Confirm that all apps have been migrated and then scale down the DEA components.</td>
<td>link</td>
</tr>
</tbody>
</table>

Operators: Plan the Migration Schedule

Operators need to plan the migration of the apps on the PCF deployment and communicate the plan to the developers in the organization.

1. Review and understand this topic.

2. Create a migration schedule.
   - For an example of migration schedule, see Migrating Applications from DEAs to Diego.

3. Communicate the schedule to developers and others in the organization who might be affected by the migration.

Operators: Prepare the Deployment for the Migration

In this stage, operators need to enable Diego as the default so that new apps pushed in the future will use the Diego architecture and not the DEA architecture. They also need to add Diego cells to accommodate the migrated apps.

1. In the PCF Ops Manager Installation Dashboard, click the Elastic Runtime tile.

2. In the Settings tab, under Resource Config increase the instances for Diego Cell.

   In general, set the number of Diego cells to be the same as the number of DEA instances.
   - For example, if you have allocated three DEA instances with 32 GB of persistent disk and 32 GB of RAM for each, allocate three Diego Cell instances with 32 GB of persistent disk and 32 GB of RAM for each to have similar capacity on Diego as with DEAs.
   - For more information about scaling VMs, see Scaling Instances in Elastic Runtime.

3. Make sure that Diego Brain and Diego BBS have at least one instance each.
   - For high availability (HA) use at least two Diego brains and at least three Diego BBS instances.

4. Click Save on the Resource Config page.

5. From the Settings tab of the Elastic Runtime tile, click Diego and select the Use Diego by default instead of DEAs checkbox.
6. Click Save.

7. Navigate to the Installation Dashboard and click Apply Changes.

Developers: Check and Prepare Apps for Deployment

Developers need to check their apps to see if any changes need to be made to environment variables that are not used in Diego and to disable port-based health checks for worker apps.

1. If you have any apps that do not serve web traffic (typically, apps that are pushed with the `--no-route` option), disable the port-based health check:

   ```
   $ cf set-health-check APP-NAME none
   ```

   Where `APP-NAME` is the name of your app.

   This prevents the work applications from reporting as unhealthy after they are migrated to Diego.

2. If your app uses any of the following environment variables, you need to modify your code.

<table>
<thead>
<tr>
<th>If your code uses this environment variable...</th>
<th>Then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCAP_APP_HOST</td>
<td>replace it with 0.0.0.0</td>
</tr>
<tr>
<td>VCAP_APP_PORT</td>
<td>replace it with PORT</td>
</tr>
<tr>
<td>VCAP_APPLICATION with any of the following keys:</td>
<td></td>
</tr>
<tr>
<td>users</td>
<td></td>
</tr>
<tr>
<td>started_at_timestamp</td>
<td></td>
</tr>
<tr>
<td>state_timestamp</td>
<td></td>
</tr>
<tr>
<td>started_at</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td></td>
</tr>
</tbody>
</table>

   Then rewrite your code to remove these environment variables.

   For more information about these environment variables that are different in Diego, see [CF-Specific Environment Variables](#).

Developers: Migrate Apps to Diego

Developers migrate their apps from DEAs to the Diego architecture using the Diego-Enabler plugin. The plugin enables Diego in your app manifest.

If your apps can tolerate a brief downtime of (approximately a minute) migrated individually or in batch mode with an overwrite-based migration. Batch mode can perform an overwrite-based migration for all apps in an particular org, space, or deployment. If a zero-downtime migration is required, then a blue-green deployment must be done to temporarily run your apps in parallel using both the DEA and Diego architectures.

1. Use the Cloud Foundry Command Line Interface (cf CLI) to install the Diego-Enabler plugin:

   ```
   $ cf add-plugin-repo CF-Community http://plugins.cloudfoundry.org/
   $ cf install-plugin Diego-Enabler -r CF-Community
   ```

   For more information about installing the Diego-Enabler CLI Plugin, see the [Diego-Enabler repository readme](#).

2. (Optional) Find out what apps you have running on DEAs:

   ```
   $ cf dea-apps
   ```

   For example,
Getting apps on the DEA runtime as example_user...
OK
name  org  space
dea-app1  org1  space1
dea-app2  org1  space2

You can also use the -s flag or the -o to limit the search to a particular space or org, for example:

```
$ cf dea-apps -o MY-ORG
```

3. To perform an overwrite-based migration for one app:

```
$ cf enable-diego APP-NAME
```

**IMPORTANT:** This step might cause downtime.

For example,

```
$ cf enable-diego dea-app
```

Diego support for EXAMPLE-APP is set to true.

4. To perform an overwrite-based migration for multiple apps in a batch job:

```
$ cf migrate-apps diego [-o ORG] [-s SPACE] [-p MAX_IN_FLIGHT]
```

**IMPORTANT:** This step might cause downtime.

Where the options are as follows:
- `-o ORG` migrates all apps in an org.
- `-s SPACE` migrates all apps in a space that exists in your currently targeted org. To migrate apps in a space from different org, target that org before running `cf migrate-apps`.
- `-p MAX_IN_FLIGHT` specifies how many apps to migrate in parallel, up to 100. The default is 1. Pivotal recommends first trying a migration with the default and then increasing `-p` if the first migration is stable. Do not set `-p` to a value greater than the number of Diego cells in the deployment.

For examples of batch migrate commands, see **Batch Migration Examples**.

5. To perform a migration without downtime, follow the steps in **Blue-Green Migration** below.

For troubleshooting information, see **Troubleshoot Migration** below.

### Batch Migration Examples

**Example 1** — The following example migrates all DEA apps in a deployment to Diego. The apps migrate in the sequence specified by running `cf dea-apps`.

```
$ cf migrate-apps diego
```

Migrating apps to Diego as example_user....

Started migrating app my-app to Diego as example_user...

Completed migrating app my-app to Diego in space with guid a045e51-c358-4c26-9d0c-4c7365c987 as example_user

... Migration to Diego completed: 23 apps, 0 warnings

```
$ cf migrate-apps -s space2 -p 10 diego
```

**Example 2** — The following example migrates all DEA apps in `space2` to Diego, ten at a time.

**Blue-Green Migration**

The blue-green deployment method migrates your apps with zero downtime. This method temporarily runs your apps in parallel using both the DEA and Diego architectures.

1. Push your application with a new name and a test route for your application. This app will run on the Diego architecture. Do not use the name or...
route of the existing application.

```
cf push NEW-APP -d MY-DOMAIN.COM -n TEMP-SUBDOMAIN
```

**Note:** The currently deployed application, OLD-APP, and the application you are pushing, NEW-APP, must be the same version. Otherwise, any static assets served from your application, like CSS and Javascript, will not be consistent across both apps. Now is not the time to upgrade your app.

2. Confirm that NEW-APP runs properly.

```
cf app NEW-APP
```

3. Run `has-diego-enabled` to confirm that the application is running on the Diego architecture.

```
cf has-diego-enabled NEW-APP
```

4. If the `has-diego-enabled` command returns false, then set the `diego` boolean to true:

```
cf enable-diego NEW-APP
```

5. Map the route for your application running on the DEA architecture to NEW-APP. This remapping creates a second route that splits your traffic between the DEA and Diego architectures.

```
cf map-route NEW-APP MY-DOMAIN.COM -n MY-SUBDOMAIN
```

6. Unmap the route from the application running on the DEA architecture. This action sends all of the traffic to the application that is running on Diego.

```
cf unmap-route OLD-APP MY-DOMAIN.COM -n MY-SUBDOMAIN
```

7. After confirming that the new application is running correctly on Diego, stop OLD-APP to route all traffic to NEW-APP and complete the blue-green migration.

```
cf stop OLD-APP
```

8. Delete the old application.

```
cf delete OLD-APP
```

9. (Optional) Delete the route to the TEMP-SUBDOMAIN that you specified when pushing the Diego app for testing. See Delete a Route.

**Developers: Verify Migration**

To verify a successful migration of your apps from DEA to Diego, perform the following steps:

1. Run `cf dea-apps` to list any apps running on DEAs. Verify that the list returned by the command is empty.

```
cf dea-apps -o MY-ORG
```

```
Getting apps on the DEA runtime as example_user...
OK
name  org  space
```

2. Run `cf diego-apps` to list any apps running on Diego. Verify that all of your apps are running on Diego.
Getting apps on the DEA runtime as example_user...
OK

3. Perform tests and checks to confirm that your apps are running as expected.
   For example:
   - Run smoke tests.
   - Validate any systems you have to monitor your apps.
   - Tail logs, especially for worker or scheduler apps. For more information, see Tailing Logs.

Operators: Verify Migration and Scale Down VMs

After developers have verified that their applications have migrated successfully, operators should confirm that all applications have been migrated:

1. Run the `cf diego-apps` command to view any apps that are still running on the DEA architecture.

2. Migrate any additional apps using the procedures in Developers: Migrate Apps to Diego.

3. In the PCF Ops Manager Installation Dashboard, click the Elastic Runtime tile.

4. In the Settings tab, under Resource Config scale the DEA and Health Manager VMs to 0 since they are not used in Diego deployments. For more information about scaling VMs, see Scaling Instances in Elastic Runtime.

5. Click Save on the Resource Config page.

6. Navigate to the Installation Dashboard and click Apply Changes.

7. Monitor usage in the Diego cells, in particular, the CapacityRemainingMemory metric over all the Diego cells is a good indicator of usage. For more information, see Diego metrics, origin rep.

Troubleshoot Migration

The following table lists possible errors and their resolutions that may occur when migrating applications from DEA to Diego:

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Explanation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors related to environment variables</td>
<td>Diego does not support the interpolation of environment variables. In the DEA architecture, when you set environment variables with the <code>cf set-env</code> or in your manifest file, you could include one environment variable in the definition of another variable, such as <code>SOMEPATH=$HOME/SOME/PATH</code></td>
<td>Remove interpolation from any environment variables.</td>
</tr>
</tbody>
</table>

| Large applications may fail to stage, typically resulting in the following error: | Large applications may fail to stage because of a known issue where disk usage is over-reported. | By default, apps have a 1GB disk quota. If your application files are close to this size, use the `cf push -k` command to increase your disk quota. |

| Running the `cf files` command results in the error: | Diego does not support the `cf files` command. | Run the `cf ssh APP_NAME` command to list files in your application instances. |

| Error: Runner error: desire app failed: 503 | Diego places a 4KB limit on the maximum size of application routes, space for 40 to 50 medium-sized 50-character routes. This error indicates you have exceeded the limit. | In some cases, you can work around this by using a wildcard route, for example `*.my-domain.com` instead of mapping individual routes. If you see this error and you cannot wildcard the routes, create another |
instance of the same application and bind the remaining routes to the second application instance.
Supporting WebSockets

This topic explains how Cloud Foundry (CF) uses WebSockets, why developers use WebSockets in their applications, and how operators can configure their load balancer to support WebSockets.

Operators who use a load balancer to distribute incoming traffic across CF router CF instances must configure their load balancer for WebSockets. Otherwise, the Loggregator system cannot stream application logs to developers, or application event data and system metrics to third-party aggregation services. Additionally, developers cannot use WebSockets in their applications.

Understand WebSockets

The WebSocket protocol provides full-duplex communication over a single TCP connection. Applications can use WebSockets to perform real-time data exchange between a client and a server more efficiently than HTTP.

CF uses WebSockets for the following metrics and logging purposes:

1. To stream all application event data and system metrics from the Doppler server instances to the Traffic Controller
2. To stream application logs from the Traffic Controller to developers using the cf Command Line Interface (CLI) or Apps Manager
3. To stream all application event data and system metrics from the Traffic Controller over the Firehose endpoint to external applications or services

For more information about these Loggregator components, see the Overview of the Loggregator System topic.

Configure Your Load Balancer for WebSockets

To form a WebSocket connection, the client sends an HTTP request that contains an Upgrade header and other headers required to complete the WebSocket handshake. You must configure your load balancer to not upgrade the HTTP request, but rather to pass the Upgrade header through to the CF router. The procedures required to configure your load balancer depends on your IaaS and load balancer. The following list includes several possible approaches:

- Some load balancers can recognize the Upgrade header and pass these requests through to the CF router without returning the WebSocket handshake response. This may or may not be default behavior, and may require additional configuration.
- Some load balancers do not support passing WebSocket handshake requests containing the Upgrade header to the CF router. For instance, the Amazon Web Services (AWS) Elastic Load Balancer (ELB) does not support this behavior. In this scenario, you must configure your load balancer to forward TCP traffic to your CF router to support WebSockets. If your load balancer does not support TCP pass-through of WebSocket requests on the same port as other HTTP requests, you can do one of the following:
  - Configure an additional port for WebSocket requests, such as port 4443, and follow the steps below in the Modify Your Release Manifest section of this topic
  - Add an additional load balancer with a domain that resolves to it, such as ws.cf.example.com, and ensure that all WebSocket traffic goes through this domain, especially if application clients only support standard ports

Note: Regardless of your IaaS and configuration, you must configure your load balancer to send the X-Forwarded-For and X-Forwarded-Proto headers for non-WebSocket HTTP requests on ports 80 and 443. See the Securing Traffic into Cloud Foundry topic for more information.

Modify Your Release Manifest

By default, the CF release manifest assigns port 4443 for TCP/WebSocket communications. If you have configured your load balancer to use a port other than 4443 for TCP/WebSocket traffic, you must edit your CF manifest to set the value of properties.logger_endpoint.port to the correct port. Locate the following section of your CF manifest and replace YOUR-WEBSOCKET-PORT with the appropriate value:

```properties
properties:
  logger_endpoint:
    port: YOUR-WEBSOCKET-PORT
```
Using Apps Manager

The web-based Apps Manager application helps you manage users, organizations, spaces, and applications.

Apps Manager is compatible with current and recent versions of all major browsers. Pivotal recommends using the current version of Chrome, Firefox, Edge, or Safari for the best Apps Manager experience.

Table of Contents

- Getting Started with Apps Manager
- Managing Orgs and Spaces Using Apps Manager
- Managing User Accounts and Permissions Using Apps Manager
- Managing Apps and Service Instances Using Apps Manager
- Monitoring Instance Usage with Apps Manager
Getting Started with Apps Manager

Overview

Apps Manager is a web-based tool to help manage organizations, spaces, applications, services, and users. Apps Manager provides a visual interface for performing the following subset of functions available through the Cloud Foundry Command Line Interface (cf CLI):

- **Orgs**: You can create and manage orgs.
- **Spaces**: You can create, manage, and delete spaces.
- **Apps**: You can scale apps, bind apps to services, manage environment variables and routes, view logs and usage information, start and stop apps, and delete apps.
- **Services**: You can bind services to apps, unbind services from apps, choose and edit service plans, and rename and delete service instances.
- **Users**: You can invite new users, manage user roles, and delete users.

To access Apps Manager as the Admin user, see the Logging in to Apps Manager topic.

Understanding Permissions

Your ability to perform actions in Apps Manager depends on your user role and the values that the Admin specifies for the Apps Manager environment variables.

The table below shows the relationship between specific org and space management actions and the non-Admin user roles who can perform them within orgs and spaces they are members of. Admin users can perform all of these actions using either the cf CLI or by logging into Apps Manager as an Org Manager, using the UAA Admin credentials.

<table>
<thead>
<tr>
<th>Action</th>
<th>CLI command</th>
<th>Org Manager</th>
<th>Space Manager</th>
<th>Org Auditor, Space Developer, or Space Auditor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an org</td>
<td><code>create-org</code></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Delete an org</td>
<td><code>delete-org</code></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Rename an org</td>
<td><code>rename-org</code></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>View org members</td>
<td><code>org-users</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Assign user a role in org</td>
<td><code>set-org-role</code></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Remove org role from user</td>
<td><code>unset-org-role</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View space members</td>
<td><code>space-users</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Assign user a role in space</td>
<td><code>set-space-role</code></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Remove space role from user</td>
<td><code>unset-space-role</code></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

† No by default, unless feature flag `user_org_creation` is set to true.

*Defaults to no. Yes if Elastic Runtime tile > Settings tab > Apps Manager section > Enable Non Admin Role Management checkbox is selected.

Space Managers assign and remove users from spaces by setting and unsettng their roles within the space. To remove a user from an org, remove them from all spaces within that org.
Managing Orgs and Spaces Using Apps Manager

This topic discusses how to view and manage orgs and spaces in Apps Manager.

Note: To manage a space, you must have Space Manager permissions in that space.

To perform the following steps, log in to Apps Manager with an account that has adequate permissions. See the Understanding Permissions topic for more information.

Manage an Org

The org page displays the spaces associated with the selected org. The left navigation of Apps Manager shows the current org.

To view spaces in a different org, use the drop-down menu to change the org.
To view the page for a particular space, click the space on the org page or on the left navigation. To create a new space, click Add a Space at the bottom of the org page.

Manage a Space

The space page displays the apps and service instances associated with the selected space.

The Apps tab shows the Name, the number of Instances, the amount of Memory available, the time since the Last Push, and the Route for each app.

The Services list shows the Service, the Name, the number of Bound Apps, and the Plan for each service instance. If you want to add a service to your space, click Add Service. For more information on configuring services, see the Services Overview topic.

To delete or rename a space, click Settings on the space page.
To change the space name, enter your new name and click **Save changes**. To delete the space, click **Delete Space**.
Managing User Accounts and Permissions Using Apps Manager

Managing Org Roles

Valid org roles are Organization Manager and Organization Auditor.

To grant or revoke org roles, follow the steps below.

1. In the Apps Manager navigation on the left, the current org is highlighted. Click the drop-down menu to view other orgs belonging to the account.

2. Use the Apps Manager navigation to select an org.

3. Click the Members tab.
4. The Members panel displays all members of the org. Select a checkbox to grant an org role to a user, or deselect a checkbox to revoke a role from a user.

Managing App Space Roles

Valid app space roles are Space Manager, Space Developer, and Space Auditor.

To grant or revoke app space roles, follow the steps below.

1. In the Members tab of an org, click the drop-down menu to view spaces in the org.

2. Use the drop-down menu to select a space.

3. The Members panel displays all members of the org. Select a checkbox to grant an app space role to a user, or deselect a checkbox to revoke a role from a user.
Space Managers can invite and manage users and enable features for a given space. Assign this role to managers or other users who need to administer the account.

Space Developers can create, delete, and manage applications and services, and have full access to all usage reports and logs. Space Developers can also edit applications, including the number of instances and memory footprint. Assign this role to app developers or other users who need to interact with applications and services.

Space Auditors have view-only access to all space information, settings, reports, and logs. Assign this role to users who need to view but not edit the application space.

Inviting New Users

1. On the Org dashboard, click the Members tab.

2. Click Invite New Members. The Invite New Team Member(s) form appears.
3. In the **Add Email Addresses** text field, enter the email addresses of the users that you want to invite. Enter multiple email addresses as a comma-delimited list.

4. The **Assign Org Roles** and **Assign Space Roles** tables list the current org and available spaces with checkboxes corresponding to each possible user role. Select the checkboxes that correspond to the permissions that you want to grant to the invited users.

5. Click **Send Invite**. The Apps Manager sends an email containing an invitation link to each email address that you specified.

### Changing User Permissions

You can also change user permissions for existing users on the account. User permissions are handled on a per-space basis, so you must edit them for each user and for each space that you want to change.

1. On the Org dashboard, click the **Members** tab.

2. Edit the permissions assigned to each user by checking or unchecking the checkboxes under each user role. The Apps Manager saves your changes automatically.

### Removing a User

To remove a user from a space, revoke that user’s permissions for all user roles in that space. The user remains visible in the list unless you remove the user from the org.
Managing Apps and Service Instances Using Apps Manager

This topic discusses how to view and manage apps and service instances associated with a space using Apps Manager.

To perform the following steps, log in to Apps Manager with an account that has adequate permissions. See the Understanding Permissions topic for more information.

Manage an App

On the space page, click the app you want to manage. This directs you to the app page, where you can scale apps, bind apps to services, manage environment variables and routes, view logs and usage information, start and stop apps, and delete apps.

Scale an App

1. Click Scale App.

2. Adjust the number of Instances and the Memory Limit.

3. Click Save.
Bind or Unbind a Service

1. Click Services.

2. To bind your app to a service, click Bind a Service.

3. To bind your app to an an existing service instance, choose the service instance from the dropdown menu, and click Bind. To bind your app to a new service instance, click Add from Marketplace to choose a service from the Marketplace.

4. To unbind your app from a service instance, locate the service instance in the Bound Services list and click Unbind.

View or Add Environment Variables

1. Click Env Variables.

2. The page displays both the User Provided and System Provided environment variables associated with the app.

3. To add a user-provided environment variable, click Add an Env Variable.

   ![Environment Variables](image)

   ```json
   {
   "staging_env.json": {},
   "running_env.json": {},
   "system_env.json": {
   "VCAP_SERVICES": {
   "elephantsql": [
   {
   "credentials": {
   
   
   ```

   Note: Changes to environment variables, as well as to service bindings and unbindings, require restarting the app to take effect. You can restart the app from the Apps Manager or with `cf restage` from the Cloud Foundry Command Line Interface (cf CLI).

Map or Unmap Routes

1. Click Routes.

2. The page displays the routes associated with the app. To add a new route, click Map a Route.

3. Enter the route and click Map.

4. To unmap a route, locate the route from the list and click Unmap.
View Logs

1. Click Logs to view the logs for the app.

> RECENT LOGS

- 2016-05-02T17:25:43.000+00:00 [STG] OUT Downloading ruby_buildpack...
- 2016-05-02T17:25:50.000+00:00 [STG] OUT Expanding Open JDK JRE to java-buildpack/open_jdk_jre (1.1s)
- 2016-05-02T17:25:50.000+00:00 [STG] OUT Downloading Spring Auto Reconfiguration 1.10.0_RELEASE from https://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration-1.10.0_RELEASE.jar (found in cache)
- 2016-05-02T17:25:50.000+00:00 [STG] OUT Downloading Tomcat Logging Support 2.5.0_RELEASE from https://download.run.pivotal.io/tomcat-logging-support/tomcat-logging-support-2.5.0_RELEASE.jar (found in cache)

2. Click the icon next to Recent Logs to refresh the logs. Click Tail Logs to view a live version of the logs.

Start or Stop an App

1. To stop an app, click the stop button under the name of the app. Click Stop in the popup to confirm.

2. To restart a stopped app, click the play button under the name of the app.

3. To restart a running app, click the restart button under the name of the app. Click Restart in the popup to confirm.

Delete an App

1. To delete an app, click Delete App on the right side of the app page.

2. Click Delete in the popup to confirm.

Manage a Service Instance

On the space page, click the service instance you want to manage. This directs you to the service instance page, where you can bind or unbind apps, view or change your service plan, and rename or delete your service instance.

Bind or Unbind an App

1. Click Edit Bindings.

2. Select the checkbox for the apps you want to bind to or unbind from your service instance.

3. Click Save.
View or Change Your Service Plan

1. Click Plan.

   ![MySQL for Pivotal Cloud Foundry](image)

   **App Binding (1) | Plan | Settings**

   100mb free
   1gb free
   5gb free
   20gb free

   **1gb free**
   • Shared MySQL server
   • 1000 MB storage
   • 40 concurrent connections

   ![Select this plan](image)

2. Review your current plan information.

3. To change your plan, select a new plan from the list and click **Select this plan**.

   ![Note: Not all services support upgrading. If your service does not support upgrading, the service plan page only displays the selected plan.](image)

Rename or Delete Your Service Instance

1. Click **Settings**.

   ![MySQL for Pivotal Cloud Foundry](image)

   **App Binding (1) | Plan | Settings**

   **Service Instance Settings**

   **Name**
   mysql-db

   ![Delete Service Instance](image)

2. To change the service instance name, enter your new name and click **Save changes**.

3. To delete the service instance, click **Delete Service Instance**.
Cloud Foundry Command Line Interface (cf CLI)

This guide explains the Cloud Foundry Command Line Interface (cf CLI), a tool you use to deploy and manage your applications.

Contents in this section:

- Installing the cf CLI
- Getting Started with the cf CLI
- Using the cf CLI with an HTTP Proxy Server
- Using the cf CLI with a Self-Signed Certificate
- Using cf CLI Plugins
- Developing cf CLI Plugins
- About Starting Applications
Installing the cf CLI

Page last updated:

This topic describes how to install the Cloud Foundry Command Line Interface (cf CLI). Follow the instructions below for your operating system. If you previously used the cf CLI v5 Ruby gem, uninstall this gem first.

You can install the cf CLI with a package manager, an installer, or a compressed binary.

Use a Package Manager

Mac OS X Installation

For Mac OS X, perform the following steps to install the cf CLI with Homebrew:

1. Tap the Cloud Foundry formula repository:
   
   ```bash
   brew tap cloudfoundry/tap
   ```

2. Install the cf CLI:
   
   ```bash
   brew install cf-cli
   ```

Linux Installation

For Debian and Ubuntu-based Linux distributions, perform the following steps:

1. Add the Cloud Foundry Foundation public key and package repository to your system:
   
   ```bash
   wget -q -O - https://packages.cloudfoundry.org/debian/cli.cloudfoundry.org.key | sudo apt-key add -
   ```

   ```bash
   echo "deb http://packages.cloudfoundry.org/debian stable main" | sudo tee /etc/apt/sources.list.d/cloudfoundry-cli.list
   ```

2. Update your local package index:
   
   ```bash
   sudo apt-get update
   ```

3. Install the cf CLI:
   
   ```bash
   sudo apt-get install cf-cli
   ```

For Enterprise Linux and Fedora systems (RHEL6/CentOS6 and up), perform the following steps:

1. Configure the Cloud Foundry Foundation package repository:
   
   ```bash
   sudo wget -O /etc/yum.repos.d/cloudfoundry-cli.repo https://packages.cloudfoundry.org/fedora/cloudfoundry-cli.repo
   ```

2. Install the cf CLI, which also downloads and adds the public key to your system:
   
   ```bash
   sudo yum install cf-cli
   ```

Use an Installer

Download the installer for Mac OS X, Windows, or Linux from the cf CLI GitHub repository and follow the instructions for your operating system below.
Windows Installation
To use the cf CLI installer for Windows, perform the following steps:

1. Unpack the zip file.
2. Double click the \texttt{cf CLI} executable.
3. When prompted, click \texttt{Install}, then \texttt{Close}.
4. To verify your installation, open a terminal window and type \texttt{cf}. If your installation was successful, the cf CLI help listing appears.

Mac OS X Installation
To use the cf CLI installer for Mac OS X, perform the following steps:

1. Open the \texttt{.pkg} file.
2. In the installer wizard, click \texttt{Continue}.
3. Select an install destination and click \texttt{Continue}.
4. When prompted, click \texttt{Install}.
5. To verify your installation, open a terminal window and type \texttt{cf}. If your installation was successful, the cf CLI help listing appears.

Linux Installation
To use the cf CLI installer for Linux, perform the following steps:

1. Install using your system's package manager. Note these commands may require \texttt{sudo}.
   - For Debian/Ubuntu, run the following command:
     \begin{verbatim}
     sudo dpkg -i path/to/cf-cli-*.deb && apt-get install -f
     \end{verbatim}
   - For Red Hat, run the following command:
     \begin{verbatim}
     rpm -i path/to/cf-cli-*.rpm
     \end{verbatim}
2. To verify your installation, open a terminal window and type \texttt{cf}. If your installation was successful, the cf CLI help listing appears.

Use a Compressed Binary
Download the compressed binary for Mac OS X, Windows, or Linux from the cf CLI GitHub repository \texttt{cf} and install it on your system.

The specific procedures vary by operating system, but the following example illustrates downloading and installing the binary on Mac OS X:

1. Download and extract the Mac OS X binary:
   \begin{verbatim}
   \end{verbatim}
2. Move it to \texttt{/usr/local/bin}, or another location in your \texttt{PATH}:
   \begin{verbatim}
   mv cf/usr/local/bin
   \end{verbatim}
3. Confirm your cf CLI version:
   \begin{verbatim}
   cf --version
   \end{verbatim}
Next Steps

See Getting Started with cf CLI for more information about how to use the cf CLI.

We recommend that you review our CLI releases page for more information about how to use the cf CLI. To learn when updates are released, and download a new binary or a new installer when you want to update to the latest version.

Uninstall the cf CLI

Package Manager

If you previously installed the cf CLI with a package manager, follow the instructions specific to your package manager to uninstall the cf CLI.

The specific procedures vary by package manager, but the following example illustrates uninstalling the cf CLI with Homebrew:

```
$ brew uninstall cf-cli
```

Installer

If you previously installed the cf CLI with an installer, perform the instructions specific to your operating system to uninstall the cf CLI:

- For Mac OS, delete the binary /usr/local/bin/cf and the directory /usr/local/share/doc/cf-cli.
- For Windows, navigate to the Control Panel, click Programs and Features, select Cloud Foundry CLI VERSION and click Uninstall.

Binary

If you previously installed a cf CLI binary, remove the binary from where you copied it.

cf CLI v5

To uninstall, run `gem uninstall cf`.

Note: To ensure that your Ruby environment manager registers the change, close and reopen your terminal.
Getting Started with the cf CLI

This topic describes configuring and getting started with the Cloud Foundry Command Line Interface (cf CLI). This page assumes you have the latest version of the cf CLI. See the Installing the Cloud Foundry Command Line Interface topic for installation instructions.

Locale

The cf CLI translates terminal output into the language that you select. The default language is en-US. The cf CLI supports the following languages:

- Chinese (simplified): zh-Hans
- Chinese (traditional): zh-Hant
- English: en-US
- French: fr-FR
- German: de-DE
- Italian: it-IT
- Japanese: ja-JP
- Korean: ko-KR
- Portuguese (Brazil): pt-BR
- Spanish: es-ES

Use cf config to set the language. To set the language with cf config, use the syntax:

```
$ cf config --locale YOUR_LANGUAGE
```

For example, to set the language to Portuguese and confirm the change by running cf help:

```
$ cf config --locale pt-BR
$ cf help
```

Note: Localization with cf config affects only messages that the cf CLI generates.

Login

Use cf login to log in to Elastic Runtime. The cf login command uses the following syntax to specify a target API endpoint, an org (organization), and a space:

```
$ cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]
```

- API_URL: This is your API endpoint, the URL of the Cloud Controller in your Elastic Runtime instance.
- USERNAME: Your username.
- PASSWORD: Your password. Use of the -p option is discouraged as it may record your password in your shell history.
- ORG: The org where you want to deploy your apps.
- SPACE: The space in the org where you want to deploy your apps.

The cf CLI prompts for credentials as needed. If you are a member of multiple orgs or spaces, cf login prompts you for which ones to log into. Otherwise it targets your org and space automatically.
$ cf login -a https://api.example.com -u username@example.com

API endpoint: https://api.example.com

Password>
Authenticating...
OK

Select an org (or press enter to skip):
1. example-org
2. example-other-org

Org> 1
Targeted org example-org

Select a space (or press enter to skip):
1. development
2. staging
3. production

Space> 1
Targeted space development

Alternatively, you can write a script to log in and set your target using the non-interactive `cf api`, `cf auth`, and `cf target` commands.

Upon successful login, the cf CLI saves a `config.json` file containing your API endpoint, org, space values, and access token. If you change these settings, the `config.json` file is updated accordingly.

By default, `config.json` is located in your `~/.cf` directory. The `CF_HOME` environment variable allows you to locate the `config.json` file wherever you like.

**Users and Roles**

The cf CLI includes commands that list users and assign roles in orgs and spaces. See the Orgs, Spaces, Roles, and Permissions topic.

**Commands for Listing Users**

These commands take an org or space as an argument:

- `cf org-users`
- `cf space-users`

For example, to list the users who are members of an org:

$ cf org-users example-org

Getting users in org example-org as username@example.com...

ORG MANAGER
username@example.com

BILLING MANAGER
huey@example.com
dewey@example.com

ORG AUDITOR
louie@example.com

**Commands for Managing Roles**

These commands require Elastic Runtime admin permissions and take username, org or space, and role as arguments:

- `cf set-org-role`
- `cf unset-org-role`
- `cf set-space-role`
- `cf unset-space-role`

Available roles are “OrgManager”, “BillingManager”, “OrgAuditor”, “SpaceManager”, “SpaceDeveloper”, and “SpaceAuditor”. For example, to grant the
Org Manager role to a user within an org:

```
$ cf set-org-role huey@example.com example-org OrgManager
Assigning role OrgManager to user huey@example.com in org example-org as username@example.com...
OK
```

**Note:** If you are not a Elastic Runtime admin, you see this message when you try to run these commands:

```
error code: 10003, message: You are not authorized to perform the requested action
```

---

**Push**

The `cf push` command pushes a new app or syncs changes to an existing app.

If you do not provide a hostname (also known as subdomain), `cf push` routes your app to a URL of the form `APPNAME.DOMAIN`, based on the name of your app and your default domain. If you want to map a different route to your app, see the [Routes and Domains](https://cloud.run/docs-ref CLI Reference/Guide#routes) topic for information about creating routes.

The `cf push` command supports many options that determine how and where the app instances are deployed. For details about the `cf push` command, see the `push` page in the Cloud Foundry CLI Reference Guide.

The following example pushes an app called `my-awesome-app` to the URL `http://my-awesome-app.example.com` and specifies the Ruby buildpack with the `-b` flag.

```
$ cf push my-awesome-app -b ruby_buildpack
Creating app my-awesome-app in org example-org / space development as username@example.com...
OK
Creating route my-awesome-app.example.com...
OK
...
1 of 1 instances running
App started...
```

**Note:** When you push an app and specify a buildpack with the `-b` flag, the app remains permanently linked to that buildpack. To use the app with a different buildpack, you must delete the app and re-push it.

```
$ cf push my-awesome-app -b ruby_buildpack
Creating app my-awesome-app in org example-org / space development as username@example.com...
OK
Creating route my-awesome-app.example.com...
OK
...
1 of 1 instances running
App started...
```

For more information about available buildpacks, see the [Buildpacks](https://cloud.run/docs-ref CLI Reference/Guide#buildpacks) topic.

---

**User-Provided Service Instances**

To create or update a user-provided service instance, you need to supply basic parameters. For example a database service might require a username, password, host, port, and database name.

The `cf CLI` has three ways of supplying these parameters to create or update an instance of a service: interactively, non-interactively, and in conjunction with third-party log management software as described in [RFC 6587](https://cloud.run/docs-ref CLI Reference/Guide#rncf). When used with third-party logging, the `cf CLI` sends data formatted according to [RFC 5424](https://cloud.run/docs-ref CLI Reference/Guide#rfc5424).

You create a service instance with `cf cups` and update one with `cf uups` as described below.
The cf create-user-provided-service (cups) Command

Use `cf create-user-provided-service` (alias `cf cups`) creates a new service instance.

To supply service instance parameters interactively: Specify parameters in a comma-separated list after the `-p` flag. This example command-line session creates a service instance for a database service.

```
cf cups sql-service-instance -p "host, port, dbname, username, password"
```

host> mysql.example.com
port> 1433
dbname> mysql
username> admin
password> Pa55w0rd

Creating user provided service sql-service-instance in org example-org / space development as username@example.com...
OK

To supply service instance parameters non-interactively: Pass parameters and their values in as a JSON hash, bound by single quotes, after the `-p` tag. This example is a non-interactive version of the `cf cups` session above.

```
cf cups sql-service-instance -p '{"host":"mysql.example.com", "port":1433, "dbname":"mysqldb", "username":"admin", "password":"Pa55w0rd"}'
```

Creating user provided service sql-service-instance in org example-org / space development as username@example.com...
OK

To create a service instance that sends data to a third-party: Use the `-l` option followed by the external destination URL. This example creates a service instance that sends log information to the syslog drain URL of a third-party log management service. For specific log service instructions, see the Service-Specific Instructions for Streaming Application Logs topic.

```
cf cups mylog -l syslog://logs4.example.com:25258
```

Creating user provided service mylog in org example-org / space development as username@example.com...
OK

After you create a user-provided service instance, you bind it to an app with `cf bind-service`, unbind it with `cf unbind-service`, rename it with `cf rename-service`, and delete it with `cf delete-service`.

The cf update-user-provided-service (uups) Command

Use `cf update-user-provided-service` (alias `cf uups`) to update one or more of the parameters for an existing user-provided service instance. The `cf uups` command uses the same syntax as `cf cups` above to set parameter values. The `cf uups` command does not update any parameter values that you do not supply.

The cf CLI Return Codes

The cf CLI uses exit codes, which help with scripting and confirming that a command has run successfully. For example, after you run a cf CLI command, you can retrieve its return code by running `echo $?` (on Windows, `echo %ERRORLEVEL%`). If the return code is 0, the command was successful.

The cf help Command

The `cf help` command lists the cf CLI commands and a brief description of each. Passing the `-a` flag to any command lists detailed help, including any aliases. For example, to see detailed help for `cf delete`, run:
$ cf delete -h
NAME:
  delete - Delete an app

USAGE:
  cf delete APP_NAME [-f -r]

ALIAS:
  d

OPTIONS:
  -f  Force deletion without confirmation
  -r  Also delete any mapped routes
Using the cf CLI with an HTTP Proxy Server

If you have an HTTP proxy server on your network between a host running the cf CLI and your Cloud Foundry API endpoint, you must set `https_proxy` with the hostname or IP address of the proxy server.

The `https_proxy` environment variable holds the hostname or IP address of your proxy server.

`https_proxy` is a standard environment variable. Like any environment variable, the specific steps you use to set it depends on your operating system.

Format of https_proxy

- `https_proxy` is set with hostname or IP address of the proxy server in URL format: `https_proxy=http://proxy.example.com`
- If the proxy server requires a user name and password, include the credentials: `https_proxy=http://username:password@proxy.example.com`
- If the proxy server uses a port other than 80, include the port number: `https_proxy=http://username:password@hostname:port`

Setting https_proxy in Mac OS or Linux

Set the `https_proxy` environment variable using the command specific to your shell. For example, in bash, use the `export` command.

Example:

```bash
$ export https_proxy=http://my.proxyserver.com:8080
```

To make this change persistent, add the command to the appropriate profile file for the shell. For example, in bash, add a line like the following to your `.bash_profile` or `.bashrc` file:

```bash
https_proxy=http://username:password@hostname:port
export $https_proxy
```

Setting https_proxy in Windows

1. Open the Start menu. Right-click Computer and select Properties.

2. In the left pane of the System window, click Advanced system settings.
3. In the System Properties window, select the Advanced tab, then click Environment Variables.

4. In the Environment Variables window, under User variables, click New.
5. In the Variable name field, input `https_proxy`. In the Variable value field, input your proxy server information.

![Edit System Variable dialog box](https://example.com/image.png)

6. Click OK.
Using the cf CLI with a Self-Signed Certificate

This topic describes how developers can use the cf CLI to communicate securely with a Cloud Foundry (CF) deployment without specifying 

```
--skip-ssl-validation
```

under the following circumstances:

- The deployment uses a self-signed certificate.
- The deployment uses a certificate that is signed by a self-signed certificate authority (CA), or a certificate signed by a certificate that’s signed by a self-signed CA.

Before following the procedure below, the developer must obtain either the self-signed certificate or the intermediate and CA certificate(s) used to sign the deployment’s certificate. The developer can obtain these certificates from the CF operator or from the deployment manifest. Review the Securing Traffic into Cloud Foundry topic for more information on how to retrieve certificates from the deployment manifest.

Install the Certificate on Local Machines

The certificates that developers must insert into their local truststore vary depending on the configuration of the deployment.

- If the deployment uses a self-signed certificate, the developer must insert the self-signed certificate into their local truststore.
- If the deployment uses a certificate that is signed by a self-signed certificate authority (CA), or a certificate signed by a certificate that’s signed by a self-signed CA, the developer must insert the self-signed certificate and any intermediate certificates into their local truststore.

Installing the Certificate on Mac OS X

Enter the following command to place a certificate file `server.crt` into your local truststore:

```
$ sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain server.crt
```

Installing the Certificate on Linux

Perform the following steps specific to your distribution to place the certificate file `server.crt` into your truststore:

- Debian/Ubuntu/Gentoo:

```
$ cat server.crt >> /etc/ssl/certs/ca-certificates.crt
```

- Fedora/RHEL:

```
$ cat server.crt >> /etc/pki/tls/certs/ca-bundle.crt
```

Installing the Certificate on Windows

1. Right-click on the certificate file and click Install Certificate.

2. Choose to install the certificate as the Current User or Local Machine. Choose the Trusted Root Certification Authorities as the certification store.
Using cf CLI Plugins

The Cloud Foundry Command Line Interface (cf CLI) includes plugin functionality. These plugins enable developers to add custom commands to the cf CLI. You can install and use plugins that Cloud Foundry developers and third-party developers create. You can review the Cloud Foundry Community CLI Plugin page for a current list of community-supported plugins. You can find information about submitting your own plugin to the community in the Cloud Foundry CLI plugin repository on GitHub.

⚠️ Warning: Plugins are not vetted in any way, including for security or functionality, by Cloud Foundry Foundation or Pivotal. Use plugins at your own risk.

The cf CLI identifies a plugin by its binary filename, its developer-defined plugin name, and the commands that the plugin provides. You use the binary filename only to install a plugin. You use the plugin name or a command for any other action.

⚠️ Note: The cf CLI uses case-sensitive plugin names and commands, but not case-sensitive binary filenames.

Prerequisites

Using plugins requires cf CLI v.6.7 or higher. Refer to the Installing the cf CLI topic for information about downloading, installing, and uninstalling the cf CLI.

Changing the Plugin Directory

By default, the cf CLI stores plugins in $HOME/.cf/plugins on your workstation. To change the root directory of this path from $HOME, set the CF_PLUGIN_HOME environment variable. The cf CLI appends .cf/plugins to the CF_PLUGIN_HOME path that you specify and stores plugins in that location.

For example, if you set CF_PLUGIN_HOME to /my-folder, cf CLI stores plugins in /my-folder/.cf/plugins.

Installing a Plugin

1. Download a binary or the source code for a plugin from a trusted provider.

⚠️ Note: The cf CLI requires a binary file compiled from source code written in Go. If you download source code, you must compile the code to create a binary.

2. Run cf install-plugin BINARY_FILENAME to install a plugin. Replace BINARY_FILENAME with the path to and name of your binary file.

⚠️ Note: You cannot install a plugin that has the same name or that uses the same command as an existing plugin. You must first uninstall the existing plugin.

⚠️ Note: The cf CLI prohibits you from implementing any plugin that uses a native cf CLI command name or alias. For example, if you attempt to install a third-party plugin that includes the command cf push, the cf CLI halts the installation.

Running a Plugin Command

Use the contents of the cf help PLUGIN and PLUGIN COMMANDS sections to manage plugins and run plugin commands.

1. Run cf plugins to list all installed plugins and all commands that the plugins provide.

2. Run cf PLUGIN_COMMAND to execute a plugin command.
Uninstalling a Plugin

Use the `PLUGIN_NAME` to remove a plugin, not the `BINARY_FILENAME`.

1. Run `cf plugins` to view the names of all installed plugins.
2. Run `cf uninstall-plugin PLUGIN_NAME` to remove a plugin.

Adding a Plugin Repo

Run `cf add-plugin-repo REPO_NAME URL` to add a plugin repo.

Example:

```
$ cf add-plugin-repo CF-Community https://plugins.cloudfoundry.org
OK
https://plugins.cloudfoundry.org/list added as 'CF-Community'
```

Listing Available Plugin Repos

Run `cf list-plugin-repos` to view your available plugin repos.

Example:

```
$ cf list-plugin-repos
OK
Repo Name    Url
CF-Community https://plugins.cloudfoundry.org
```

Listing All Plugins by Repo

Run `cf repo-plugins` to show all plugins from all available repos.

Troubleshooting

The cf CLI provides the following error messages to help you troubleshoot installation and usage issues. Third-party plugins can provide their own error messages.

Permission Denied

If you receive a `permission denied` error message, you lack required permissions to the plugin. You must have `read` and `execute` permissions to the plugin binary file.

Plugin Command Collision

Plugin names and commands must be unique. The CLI displays an error message if you attempt to install a plugin with a non-unique name or command.

If the plugin has the same name or command as a currently installed plugin, you must first uninstall the existing plugin to install the new plugin.

If the plugin has a command with the same name as a native cf CLI command or alias, you cannot install the plugin.
Developing cf CLI Plugins

Users can create and install Cloud Foundry Command Line Interface (cf CLI) plugins to provide custom commands. These plugins can be submitted and shared to the CF Community repo.

Requirements

Using plugins requires cf CLI v.6.7 or higher. Refer to the Installing the Cloud Foundry Command Line Interface topic for information about downloading, installing, and uninstalling the cf CLI.

Installing the Architecture

1. Implement the predefined plugin interface.
2. Clone the template repo. You will need the basic GO plugin.

Initializing the Plugin

To initialize a plugin, call `plugin.Start(new(MyPluginStruct))` from within the `main()` method of your plugin. The `plugin.Start(...)` function requires a new reference to the struct that implements the defined interface.

Invoking CLI Commands

Invoke CLI commands with `cliConnection.CliCommand([]args)` from within a plugin's `Run(...)` method. The `Run(...)` method receives the `cliConnection` as its first argument. The `cliConnection.CliCommand([]args)` returns the output printed by the command and an error.

The output is returned as a slice of strings. The error will be present if the call to the CLI command fails.

For more information, see the calling CLI commands example.

Installing a Plugin

To install a plugin, run `cf install-plugin PATH_TO_PLUGIN_BINARY`.

For additional information on developing plugins, see the plugin development guide.
About Starting Applications

The first time you deploy an application, `cf push` uses the buildpack start command by default. After that, `cf push` defaults to whatever start command was used for the previous push.

To override these defaults, provide the `-c` option, or the command attribute in the manifest. When you provide start commands both at the command line and in the manifest, `cf push` ignores the command in the manifest.

Forcing cf push to use the Buildpack Start Command

You can specify a null start command in one of two ways.

1. Using the `-c` command-line option:
   
   `cf push my-app -c "null"`

2. Using the `command` attribute in the application manifest:

   ```
   command: null
   ```

This can be helpful after you have deployed while providing a start command at the command line or the manifest. At this point, a command that you provided, rather than the buildpack start command, has become the default start command. In this situation, if you decide to deploy using the buildpack start command, the `null` command makes that easy.

Start Commands when Migrating a Database

Start commands are used in special ways when you migrate a database as part of an application deployment. See *Migrating a Database in Cloud Foundry*. 
Developer Guide

This guide has instructions for pushing an application to Cloud Foundry and making the application work with any available cloud-based services it uses, such as databases, email, or message servers. The core of this guide is the Deploy an Application process guide, which provides end-to-end instructions for deploying and running applications on Cloud Foundry, including tips for troubleshooting deployment and application health issues.

Before you can use the instructions in this document, you must have an account on your Cloud Foundry instance.

Preparing Applications for the Cloud

- Considerations for Designing and Running an Application in the Cloud

Deploying and Managing Applications

- Understanding Application Deployment
- Deploy an Application
- Deploy a Large Application
- Application Container Lifecycle
- Routes and Domains
- Changing Stacks
- Deploying with Application Manifests
- Scaling an Application Using cf scale
- Cloud Foundry Environment Variables
- Using Blue-Green Deployment to Reduce Downtime and Risk
- Application Logging in Cloud Foundry
- Troubleshooting Application Deployment and Health
- Application SSH Overview
- Accessing Apps with SSH
- Accessing Services with SSH
- Trusted System Certificates
- Cloud Controller API Client Libraries

Services

- Services Overview
- Delivering Service Credentials to an Application
- Managing Service Instances
- Managing Service Keys
- User-Provided Service Instances
- Streaming Application Logs to Log Management Services
- Service-Specific Instructions for Streaming Application Logs
- Streaming Application Logs to Splunk
- Streaming Application Logs with Fluentd
- Configuring Play Framework Service Connections
- Migrating a Database in Cloud Foundry

© Copyright Pivotal Software Inc, 2013-2018
Considerations for Designing and Running an Application in the Cloud

Page last updated:

Application Design for the Cloud

Applications written in supported application frameworks often run unmodified on Cloud Foundry, if the application design follows a few simple guidelines. Following these guidelines makes an application cloud-friendly, and facilitates deployment to Cloud Foundry and other cloud platforms.

The following guidelines represent best practices for developing modern applications for cloud platforms. For more detailed reading about good app design for the cloud, see The Twelve-Factor App.

For more information on features of HTTP routing handled by the Cloud Foundry router, see the HTTP Routing topic. For more information on the lifecycle of application containers, see the Application Container Lifecycle topic.

Avoid Writing to the Local File System

Avoid Writing to the Local File System

Applications running on Cloud Foundry should not write files to the local file system for the following reasons:

- **Local file system storage is short-lived.** When an application instance crashes or stops, the resources assigned to that instance are reclaimed by the platform including any local disk changes made since the app started. When the instance is restarted, the application will start with a new disk image. Although your application can write local files while it is running, the files will disappear after the application restarts.
- **Instances of the same application do not share a local file system.** Each application instance runs in its own isolated container. Thus a file written by one instance is not visible to other instances of the same application. If the files are temporary, this should not be a problem. However, if your application needs the data in the files to persist across application restarts, or the data needs to be shared across all running instances of the application, the local file system should not be used. We recommend using a shared data service like a database or blobstore for this purpose.

For example, instead of using the local file system, you can use a Cloud Foundry service such as the MongoDB document database or a relational database like MySQL or Postgres. Another option is to use cloud storage providers such as Amazon S3, Google Cloud Storage, Dropbox, or Box. If your application needs to communicate across different instances of itself, consider a cache like Redis or a messaging-based architecture with RabbitMQ.

Cookies Accessible across Applications

Cookies Accessible across Applications

In an environment with shared domains, cookies might be accessible across applications.

Many tracking tools such as Google Analytics and Mixpanel use the highest available domain to set their cookies. For an application using a shared domain such as example.com, a cookie set to use the highest domain has a Domain attribute of example.com in its HTTP response header. For example, an application at my-app.shared-domain.example.com might be able to access the cookies for an application at your-app.shared-domain.example.com.

Consider whether you want your applications or tools that use cookies to set and store the cookies at the highest available domain.

Port Limitations

Port Limitations

Clients connect to applications running on Cloud Foundry by making requests to URLs associated with the application. Cloud Foundry allows HTTP requests to applications on ports 80 and 443. For more information, see the Routes and Domains topic.

Cloud Foundry also supports WebSocket handshake requests over HTTP containing the Upgrade header. The Cloud Foundry router handles the upgrade and initiates a TCP connection to the application to form a WebSocket connection.

To support WebSockets, the operator must configure the load balancer correctly. Depending on the configuration, clients may have to use a different port for WebSocket connections, such as port 4443. For more information, see the Supporting WebSockets topic.

Cloud Foundry Updates and Your Application

Cloud Foundry Updates and Your Application

For application management purposes, Cloud Foundry may need to stop and restart your application instances. If this occurs, Cloud Foundry performs the following steps:
1. Cloud Foundry sends a single termination signal to the root process that your start command invokes.

2. Cloud Foundry waits 10 seconds to allow your application to cleanly shut down any child processes and handle any open connections.

3. After 10 seconds, Cloud Foundry forcibly shuts down your application.

Your application should accept and handle the termination signal to ensure that it shuts down gracefully.

Ignore Unnecessary Files When Pushing

By default, when you push an application, all files in the application’s project directory tree are uploaded to your Cloud Foundry instance, except version control or configuration files with the following file extensions:

- .cfignore
- .darcs
- .DS_Store
- .git
- .gitignore
- .hg
- /manifest.yml
- .svn

If the application directory contains other files (such as temp or log files), or complete subdirectories that are not required to build and run your application, the best practice is to exclude them using a .cfignore file. (.cfignore is similar to git’s .gitignore, which allows you to exclude files and directories from git tracking.) Especially with a large application, uploading unnecessary files slows down application deployment.

Specify the files or file types you wish to exclude from upload in a text file, named .cfignore, in the root of your application directory structure. For example, these lines exclude the “tmp” and “log” directories.

```
/tmp/
/log/
```

The file types you will want to exclude vary, based on the application frameworks you use. The .gitignore templates for common frameworks, available at https://github.com/github/gitignore, are a useful starting point.

Run Multiple Instances to Increase Availability

When a DEA is upgraded, the applications running on it are shut down gracefully, then restarted on another DEA. To avoid the risk of an application being unavailable during a Cloud Foundry upgrade processes, you should run more than one instance of the application.

Using Buildpacks

A buildpack consists of bundles of detection and configuration scripts that provide framework and runtime support for your applications. When you deploy an application that needs a buildpack, Cloud Foundry installs the buildpack on the Droplet Execution Agent (DEA) where the application runs.

For more information, see the Buildpacks topic.
Understanding Application Deployment

Page last updated:

Here are things you need to do:

1. Read how to [prepare your application for the cloud](#)

2. Learn about buildpacks and framework-specific considerations:
   - Ruby
   - Node.js
   - Java
   - Build Tool Integration
   - Eclipse Plugin

3. Deploy your application

Additional Information:
- **Routes and Domains**
- **Application Manifests**
- **Cloud Foundry Environment Variables**
- **About Starting Applications**
- **Streaming Logs**
- **Blue-Green Deployment**
- **Troubleshoot Application Deployment and Health**
- **Application SSH Overview**
- **Scaling an Application Using cf scale**
- **Deploying a Large Application**
- **Application Container Lifecycle**
- **Trusted System Certificates**
Deploy an Application

Page last updated:

Note: See the buildpacks documentation for complete deployment guides specific to your app language or framework, such as the Getting Started Deploying Ruby on Rails Apps guide.

Overview of Deployment Process

You deploy an app to Cloud Foundry by running a `push` command from a Cloud Foundry Command Line Interface (cf CLI). Refer to the installing the cf CLI topic for more information. Between the time that you run `push` and the time that the app is available, Cloud Foundry performs the following tasks:

- Uploads and stores app files
- Examines and stores app metadata
- Creates a “droplet” (the Cloud Foundry unit of execution) for the app
- Selects an appropriate droplet execution agent (DEA) to run the droplet
- Starts the app

For more information on the lifecycle of an app, see the Application Container Lifecycle topic.

An app that uses services, such as a database, messaging, or email server, is not fully functional until you provision the service and, if required, bind the service to the app. For more information about services, see the Services Overview topic.

Step 1: Prepare to Deploy

Before you deploy your app to Cloud Foundry, make sure that:

- Your app is cloud-ready. Cloud Foundry behaviors related to file storage, HTTP sessions, and port usage may require modifications to your app.
- All required app resources are uploaded. For example, you may need to include a database driver.
- Extraneous files and artifacts are excluded from upload. You should explicitly exclude extraneous files that reside within your app directory structure, particularly if your app is large.
- An instance of every service that your app needs has been created.
- Your Cloud Foundry instance supports the type of app you are going to deploy, or you have the URL of an externally available buildpack that can stage the app.

For help preparing to deploy your app, see:

- Considerations for Designing and Running an Application in the Cloud
- Buildpacks

Step 2: Know Your Credentials and Target

Before you can push your app to Cloud Foundry you need to know:

- The API endpoint for your Cloud Foundry instance. Also known as the target URL, this is the URL of the Cloud Controller in your Elastic Runtime instance.
- Your username and password for your Cloud Foundry instance.
- The organization and space where you want to deploy your app. A Cloud Foundry workspace is organized into organizations, and within them, spaces. As a Cloud Foundry user, you have access to one or more organizations and spaces.

Step 3: (Optional) Configure Domains

Cloud Foundry directs requests to an app using a route, which is a URL made up of a host and a domain.
The name of an app is the default host for that app, unless you specify the host name with the `-n` flag.

Every app is deployed to an app space that belongs to a domain. Every Cloud Foundry instance has a default domain defined. You can specify a non-default, or custom, domain when deploying, provided that the domain is registered and is mapped to the organization which contains the target app space.

**Note:** CF allows app names, but not app URLs, to include underscores. CF converts underscores to hyphens when setting a default app URL from an app name.

The URL for your app must be unique from other apps hosted by Elastic Runtime. Use the following options with the `cf push` to help create a unique URL:

- `-n` to assign a different HOST name for the app
- `--random-route` to create a URL that includes the app name and random words

**Note:** Use `cf help push` to view other options for this command.

For more information about domains, see Routes and Domains.

---

**Step 4: Determine Deployment Options**

Before you deploy, you need to decide on the following:

- **Name**: You can use any series of alpha-numeric characters, without spaces, as the name of your app.
- **Instances**: Generally speaking, the more instances you run, the less downtime your app will experience. If your app is still in development, running a single instance can simplify troubleshooting. For any production app, we recommend a minimum of two instances.
- **Memory Limit**: The maximum amount of memory that each instance of your app can consume. If an instance exceeds this limit, Cloud Foundry restarts the instance.

**Note:** Initially, Cloud Foundry immediately restarts any instances that exceed the memory limit. If an instance repeatedly exceeds the memory limit in a short period of time, Cloud Foundry delays restarting the instance.

- **Start Command**: This is the command that Cloud Foundry uses to start each instance of your app. This start command varies by app framework.
- **Subdomain (host) and Domain**: The route, which is the combination of subdomain and domain, must be globally unique. This is true whether you specify a portion of the route or allow Cloud Foundry to use defaults.
- **Services**: Apps can bind to services such as databases, messaging, and key-value stores. Apps are deployed into app spaces. An app can only bind to a service that has an existing instance in the target app space.

---

**Define Deployment Options**

You can define deployment options on the command line, in a manifest file, or both together. See Deploying with Application Manifests to learn how app settings change from push to push, and how command-line options, manifests, and commands like `cf scale` interact.

When you deploy an app while it is running, Cloud Foundry stops all instances of that app and then deploys. Users who try to run the app get a “404 not found” message while `cf push` runs. Stopping all instances is necessary to prevent two versions of your code from running at the same time. A worst-case example would be deploying an update that involved a database schema migration, because instances running the old code would not work and users could lose data.

Cloud Foundry uploads all app files except version control files with file extensions `.svn`, `.git`, and `.darcs`. To exclude other files from upload, specify them in a `cfignore` file in the directory where you run the push command. This technique is similar to using a `.gitignore` file. For more information, see the Ignore Unnecessary Files When Pushing section of the Considerations for Designing and Running an Application in the Cloud topic.

For more information about the manifest file, see the Deploying with Application Manifests topic.

---

**Configure Pre-Runtime Hooks**

**Note:** The Java buildpack does not support pre-runtime hooks.
To configure pre-runtime hooks, create a file named `.profile` and place it in the root of your app directory. If the directory includes a `.profile` script, then Cloud Foundry executes it immediately before each instance of your app starts. Because the `.profile` script executes after the buildpack, the script has access to the language runtime environment created by the buildpack.

Note: Your app root directory may also include a `.profile.d` directory that contains bash scripts that perform initialization tasks for the buildpack. Developers should not edit these scripts unless they are using a custom buildpack.

You can use the `.profile` script to perform app-specific initialization tasks, such as setting custom environment variables. Environment variables are key-value pairs defined at the operating system level. These key-value pairs provide a way to configure the apps running on a system. For example, any app can access the LANG environment variable to determine which language to use for error messages and instructions, collating sequences, and date formats.

To set an environment variable, add the appropriate bash commands to your `.profile` file. See the example below.

```
# Set the default LANG for your apps
export LANG=en_US.UTF-8
```

Note: If you are using a PHP buildpack version prior to v4.3.18, the buildpack does not execute your PHP app’s `.profile` script. Your PHP app will host the `.profile` script’s contents. This means that any PHP app staged using the affected PHP Buildpack versions can leak credentials placed in the `.profile` script.

### Step 5: Push the App

Run the following command to deploy an app without a manifest:

```
cf push APP-NAME
```

If you provide the app name in a manifest, you can reduce the command to `cf push`. See Deploying with Application Manifests.

Because all you have provided is the name of your app, `cf push` sets the number of instances, amount of memory, and other attributes of your app to the default values. You can also use command-line options to specify these and additional attributes.

The following transcript illustrates how Cloud Foundry assigns default values to app when given a `cf push` command.

Note: When deploying your own apps, avoid generic names like `my-app`. Cloud Foundry uses the app name to compose the route to the app, and deployment fails unless the app has a globally unique route.
Step 6: (Optional) Configure Service Connections

If you bound a service to the app that you deployed, you might need to configure your app with the service URL and credentials. For more information, see the specific documentation for your app framework:

- **Ruby**
- **Node.js**
- **Spring**
- **Grails**

Step 7: Troubleshoot Deployment Problems

If your app does not start on Cloud Foundry, first ensure that your app can run locally.

You can troubleshoot your app in the cloud using the cf CLI. See [Troubleshoot Application Deployment and Health](#).
Deploying a Large Application

Page last updated:

This topic describes constraints and recommended settings for deploying applications between 750 MB and 1 GB to Elastic Runtime.

Deployment Considerations and Limitations

Elastic Runtime supports application uploads up to 1 GB.

The deployment process involves uploading, staging, and starting the app. See the Deployment section of the Application Container Lifecycle topic for more information on the default time limits for uploading, staging, and starting an app.

To deploy large apps to Elastic Runtime, ensure the following:

- Your network connection speed is sufficient to upload your app within the 15 minute limit. We recommends a minimum speed of 874 KB/s.

- The total size of the files to upload for your app does not exceed 1 GB.

- You allocate enough memory for all instances of your app. Use either the `-m` flag with `cf push` or set an app memory value in your `manifest.yml` file.

- You allocate enough disk space for all instances of your app. Use either the `-k` flag with `cf push` or set a disk space allocation value in your `manifest.yml` file.

- If you use an app manifest file, `manifest.yml`, be sure to specify adequate values for your app for attributes such as app memory, app start timeout, and disk space allocation. For more information about using manifests, refer to the Deploying with Application Manifests topic.

- You push only the files that are necessary for your application. To meet this requirement, push only the directory for your application, and remove unneeded files or use the `.cfignore` file to specify excluded files.

- You configure Cloud Foundry Command Line Interface (cf CLI) staging, startup, and timeout settings to override settings in the manifest, as necessary.

  - `CF_STAGING_TIMEOUT`: Controls the maximum time that the cf CLI waits for an app to stage after Cloud Foundry successfully uploads and packages the app. Value set in minutes.
  
  - `CF_STARTUP_TIMEOUT`: Controls the maximum time that the cf CLI waits for an app to start. Value set in minutes.
  
  - `cf push -t TIMEOUT`: Controls the maximum time that the cf CLI waits for an app to start. When you use this flag, the cf CLI ignores any app start timeout value set in the manifest or in the `CF_STARTUP_TIMEOUT` environment variable. Value set in seconds.

For more information about using the cf CLI to deploy apps, refer to the Push section of the Getting Started with the cf CLI topic.

- Changing the timeout setting for the cf CLI does not change the timeout limit for Cloud Foundry server-side jobs such as staging or starting applications. Server-side timeouts must be changed in the manifest. Because of the differences between the Cloud Foundry and cf CLI timeout values, your app might successfully start even though the cf CLI reports `App failed`. Run `cf apps APP_NAME` to review the actual status of your app.

Default Settings and Limitations Summary Table

This table provides summary information of constraints and default settings to consider when you deploy a large app to Elastic Runtime.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Package Size</td>
<td>Maximum: 1 GB</td>
</tr>
<tr>
<td>Authorization Token Grace Period</td>
<td>Default: 20 minutes, minimum</td>
</tr>
<tr>
<td><code>CF_STAGING_TIMEOUT</code></td>
<td>cf CLI environment variable Default: 15 minutes</td>
</tr>
<tr>
<td><code>CF_STARTUP_TIMEOUT</code></td>
<td>cf CLI environment variable Default: 5 minutes</td>
</tr>
<tr>
<td><code>cf push -t TIMEOUT</code></td>
<td>App start timeout maximum Default: 60 seconds</td>
</tr>
<tr>
<td>Disk Space Allocation</td>
<td>Default: 1024 MB</td>
</tr>
<tr>
<td>Internet Connection Speed</td>
<td>Recommended Minimum: 874 KB/s</td>
</tr>
</tbody>
</table>
Application Container Lifecycle

Page last updated:

This topic describes the lifecycle of an application container for Cloud Foundry (CF) deployments running on the Diego architecture.

Deployment

The application deployment process involves uploading, staging, and starting the app in a container. Your app must successfully complete each of these phases within certain time limits. The default time limits for the phases are as follows:

- Upload: 15 minutes
- Stage: 15 minutes
- Start: 60 seconds

Note: Your administrator can change these defaults. Check with your administrator for the actual time limits set for app deployment.

Developers can change the time limit for starting apps through an application manifest or on the command line. For more information, see The timeout attribute section of the Deploying with Application Manifests topic.

Crash Events

If an app instance crashes, CF automatically restarts it by rescheduling the instance on another container three times. After three failed restarts, CF waits thirty seconds before attempting another restart. The wait time doubles each restart until the ninth restart, and remains at that duration until the 200th restart. After the 200th restart, CF stops trying to restart the app instance.

Evacuation

Certain operator actions require restarting VMs with containers hosting app instances. For example, an operator who updates stemcells or installs a new version of CF must restart all the VMs in a deployment. CF automatically relocates the instances on VMs that are shutting down through a process called evacuation. CF recreates the app instances on another VM, waits until they are healthy, and then shuts down the old instances. During an evacuation, developers may see their app instances in a duplicated state for a brief period.

Shutdown

When PCF requests a shutdown of your app instance, either in response to the command `cf scale APPNAME -i NUMBER-OF-INSTANCES` or because of a system event, CF sends the app process in the container a SIGTERM. The process has ten seconds to shut down gracefully. If the process has not exited after ten seconds, CF sends a SIGKILL.

Apps must finish their in-flight jobs within ten seconds of receiving the SIGTERM before CF terminates the app with a SIGKILL. For instance, a web app must finish processing existing requests and stop accepting new requests.
This topic describes how routes and domains work in Elastic Runtime, and how developers and administrators configure routes and domains for their applications using the Cloud Foundry Command Line Interface (cf CLI).

**Routes**

The Elastic Runtime router routes requests to applications by associating an app with an address, known as a route. This association is called a mapping: for example, the cf CLI command for associating an app and route is `cf map-route`.

The routing tier compares each request with a list of all the routes mapped to apps and attempts to find the best match. For example, the router would make the following matches for the two routes `myapp.shared-domain.example.com` and `myapp.shared-domain.example.com/products`:

<table>
<thead>
<tr>
<th>Request</th>
<th>Matched Route</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://myapp.shared-domain.example.com">http://myapp.shared-domain.example.com</a></td>
<td>myapp.shared-domain.example.com</td>
</tr>
<tr>
<td><a href="http://myapp.shared-domain.example.com/contact">http://myapp.shared-domain.example.com/contact</a></td>
<td>myapp.shared-domain.example.com</td>
</tr>
<tr>
<td><a href="http://myapp.shared-domain.example.com/products">http://myapp.shared-domain.example.com/products</a></td>
<td>myapp.shared-domain.example.com/products</td>
</tr>
<tr>
<td><a href="http://myapp.shared-domain.example.com/products/123">http://myapp.shared-domain.example.com/products/123</a></td>
<td>myapp.shared-domain.example.com/products</td>
</tr>
<tr>
<td><a href="http://products.shared-domain.example.com">http://products.shared-domain.example.com</a></td>
<td>No match; 404</td>
</tr>
</tbody>
</table>

The router does not use a route to match requests until it is mapped to an app. In the above example, `products.shared-domain.example.com` may have been created as a route in Cloud Foundry, but until it is mapped to an app, requests for it receive a 404.

The routing tier knows the location of instances for apps mapped to routes. Once the routing tier determines a route as the best match for a request, it makes a load-balancing calculation using the round-robin algorithm, and forwards the request to an instance of the mapped app.

Developers can map many apps to a single route, resulting in load-balanced requests for the route across all instances of all mapped apps. This approach enables the blue/green zero-downtime deployment strategy. Developers can also map an individual app to multiple routes, enabling access to the app from many URLs.

Routes belong to a space, so that developers can only map apps to a route in the same space. Routes are globally unique, so that developers in one space cannot create a route with the same URL as developers in another space, regardless of which orgs control these spaces.

**HTTP vs TCP Routes**

Note: By default, Elastic Runtime only supports routing of HTTP requests to applications.

Routes are considered HTTP if they are created from HTTP domains, and TCP if the are created from TCP domains. See [HTTP vs TCP Shared Domains](#).

HTTP routes include a domain, an optional hostname, and an optional context path. `shared-domain.example.com`, `myapp.shared-domain.example.com`, and `myapp.shared-domain.example.com/products` are all examples of HTTP routes.

- Requests to HTTP routes must be sent to ports 80 or 443.
- Ports cannot be reserved for HTTP routes.

TCP routes include a domain and a route port. A route port is the port clients make requests to. This is not the same port as what an application pushed to Cloud Foundry listens on; applications should listen to the port defined by the `$PORT` environment variable; 8080 on Diego. `tcp.shared-domain.example.com:60000` is an example of a TCP route.

- Once a port is reserved for a route, it cannot be reserved for another route.
- Hostname and path are not supported for TCP routes.

**Create a Route**

When a developer creates a route using the cf CLI, Elastic Runtime determines whether the route is an HTTP or a TCP route based on the domain. To create
a HTTP route, a developer must choose a HTTP domain. To create a TCP route, a developer must choose a TCP domain.

Domains in Elastic Runtime provide a namespace from which to create routes. To list available domains for a targeted organization, use the `cf domains` command. For more information about domains, see the Domains section.

The following sections describe how developers can create HTTP and TCP routes for different use cases.

Create an HTTP Route with Hostname

In Elastic Runtime, a hostname is the label that indicates a subdomain of the domain associated with the route. Given a domain `shared-domain.example.com`, a developer can create the route `myapp.shared-domain.example.com` in space `my-space` by specifying the hostname `myapp` with the `cf create-route` command:

```
$ cf create-route my-space shared-domain.example.com --hostname myapp
Creating route myapp.shared-domain.example.com for org my-org / space my-space as username@example.com...
OK
```

This command instructs Elastic Runtime to only route requests to apps mapped to this route for the following URLs:

- `http://myapp.shared-domain.example.com`
- `https://myapp.shared-domain.example.com`
- Any path under either of the above URLs, such as `http://myapp.shared-domain.example.com/bar`

Create an HTTP Route without Hostname

This approach creates a route with the same address as the domain itself and is permitted for private domains only. For more information, see the Private Domains section.

A developer can create a route in space `my-space` from the domain `private-domain.example.com` with no hostname with the `cf create-route` command:

```
$ cf create-route my-space private-domain.example.com
Creating route private-domain.example.com for org my-org / space my-space as username@example.com...
OK
```

If DNS has been configured correctly, this command instructs Elastic Runtime to route requests to apps mapped to this route from the following URLs:

- `http://private-domain.example.com`
- `https://private-domain.example.com`
- Any path under either of the above URLs, such as `http://private-domain.example.com/foo`

If there are no other routes for the domain, requests to any subdomain, such as `http://foo.private-domain.example.com`, will fail.

A developer can also create routes for subdomains with no hostnames. The following command creates a route in space `my-space` from the subdomain `foo.private-domain.example.com`:

```
$ cf create-route my-space foo.private-domain.example.com
Creating route foo.private-domain.example.com for org my-org / space my-space as username@example.com...
OK
```

Assuming DNS has been configured for this subdomain, this command instructs Elastic Runtime to route requests to apps mapped to this route from the following URLs:

- `http://foo.private-domain.example.com`
- `https://foo.private-domain.example.com`
- Any path under either of the above URLs, such as `http://foo.private-domain.example.com/foo`

Create an HTTP Route with Wildcard Hostname
An application mapped to a wildcard route acts as a fallback app for route requests if the requested route does not exist. To create a wildcard route, use an asterisk for the hostname.

A developer can create a wildcard route in space `my-space` from the domain `foo.shared-domain.example.com` with the following command:

```
cf create-route my-space foo.shared-domain.example.com --hostname *
```

Creating route `*.foo.shared-domain.example.com` for org `my-org` / space `my-space` as `username@example.com`...
OK

If a client sends a request to `http://app.foo.shared-domain.example.com` by accident, attempting to reach `myapp.foo.shared-domain.example.com`, Elastic Runtime routes the request to the app mapped to the route `*.foo.shared-domain.example.com`.

Create an HTTP Route with a Path

Developers can use paths to route requests for the same hostname and domain to different apps.

A developer can create three routes using the same hostname and domain in the space `my-space` with the following commands:

```
cf create-route my-space shared-domain.example.com --hostname store --path products
Creating route `store.shared-domain.example.com/products` for org `my-org` / space `my-space` as `username@example.com`...
OK

$ cf create-route my-space shared-domain.example.com --hostname store --path orders
Creating route `store.shared-domain.example.com/orders` for org `my-org` / space `my-space` as `username@example.com`...
OK

$ cf create-route my-space shared-domain.example.com --hostname store
Creating route `store.shared-domain.example.com` for org `my-org` / space `my-space` as `username@example.com`...
OK
```

The developer can then map the new routes to different apps by following the steps in the **Map a Route to your Application** section below.

If the developer maps the first route with path `products` to the `products` app, the second route with path `orders` to the `orders` app, and the last route to the `storefront` app, then:

- Elastic Runtime routes requests to `http://store.shared-domain.example.com/products` to the `products` app.
- Elastic Runtime routes requests to `http://store.shared-domain.example.com/orders` to the `orders` app.
- Elastic Runtime routes requests to `http://store.shared-domain.example.com` to the `storefront` app.

Elastic Runtime attempts to match routes with a path first, and then attempts to match host and domain.

**Note:** Routes with the same domain and hostname but different paths can only be created in the same space. Private domains do not have this limitation.

**Note:** Elastic Runtime does not route requests for context paths to the root context of an application. Applications must serve requests on the context path.

Create a TCP Route with a Port

A developer can create a TCP route for `tcp.shared-domain.example.com` on an arbitrary port with the following command. If the clients of the app can accommodate addressing an arbitrary port, then developers should use the `--random-port` flag to instruct Elastic Runtime to pick a port for your route.

```
cf create-route tcp.shared-domain.example.com --random-port
Creating route `tcp.shared-domain.example.com` for org `my-org` / space `my-space` as `username@example.com`...
OK
```

Route `tcp.shared-domain.example.com:60034` has been created

In this example, Elastic Runtime routes requests to `tcp.shared-domain.example.com:60034` to apps mapped to this route.

To request a specific port, a developer can use the `--port` flag, so long as the port is not reserved for another space. The following command creates a TCP route for `tcp.shared-domain.example.com` on port 60035:
List Routes

Developers can list routes for the current space with the `cf routes` command. A route is uniquely identified by the combination of hostname, domain, port, and path.

```
$ cf routes
Getting routes as user@private-domain.example.com ...
```

<table>
<thead>
<tr>
<th>space</th>
<th>host</th>
<th>domain</th>
<th>port</th>
<th>path</th>
<th>type</th>
<th>apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-space</td>
<td>myapp</td>
<td>shared-domain.example.com</td>
<td></td>
<td></td>
<td></td>
<td>myapp</td>
</tr>
<tr>
<td>my-space</td>
<td>myapp</td>
<td>private-domain.example.com</td>
<td></td>
<td></td>
<td></td>
<td>myapp</td>
</tr>
<tr>
<td>my-space</td>
<td>store</td>
<td>shared-domain.example.com</td>
<td></td>
<td>/products</td>
<td></td>
<td>products</td>
</tr>
<tr>
<td>my-space</td>
<td>store</td>
<td>shared-domain.example.com</td>
<td></td>
<td>/orders</td>
<td></td>
<td>orders</td>
</tr>
<tr>
<td>my-space</td>
<td>store</td>
<td>shared-domain.example.com</td>
<td></td>
<td></td>
<td></td>
<td>storefront</td>
</tr>
<tr>
<td>my-space</td>
<td>store</td>
<td>shared-domain.example.com</td>
<td>60000</td>
<td></td>
<td>tcp</td>
<td>tcp-app</td>
</tr>
</tbody>
</table>

Developers can only see routes in spaces where they are members.

Check Routes

Developers cannot create a route that is already taken. To check whether a route is available, developers can use the `cf check-route` command.

The following command checks whether a route with the hostname `store` and the domain `shared-domain.example.com` and the path `/products` exists:

```
$ cf check-route store shared-domain.example.com --path /products
Checking for route...
OK
Route store.shared-domain.example.com/products does exist
```

Map a Route to your Application

For an app to receive requests to a route, developers must map the route to the app with the `cf map-route` command. If the route does not already exist, this command creates it.

Developers can create and reserve routes for later use by following the steps in the Manually Map a Route section. Or they can map routes to their app immediately as part of a push by following the steps in the Map a Route with Application Push section.

Manually Map a Route

Given the following routes and applications:

<table>
<thead>
<tr>
<th>Route</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>store.shared-domain.example.com/products</td>
<td>products</td>
</tr>
<tr>
<td>store.shared-domain.example.com/orders</td>
<td>orders</td>
</tr>
<tr>
<td>store.shared-domain.example.com</td>
<td>storefront</td>
</tr>
<tr>
<td>tcp.shared-domain.example.com:60000</td>
<td>tcp-app</td>
</tr>
</tbody>
</table>

The following commands map the above routes to their respective apps. Developers use hostname, domain, and path to uniquely identify a route to map their apps to.

```
$ cf map-route products shared-domain.example.com --hostname store --path products
$ cf map-route orders shared-domain.example.com --hostname store --path orders
$ cf map-route storefront shared-domain.example.com --hostname store
$ cf map-route tcp-app tcp.shared-domain.example.com --port 60000
```
The following command maps the wildcard route `*.foo.shared-domain.example.com` to the app `myfallbackapp`.

```
cf map-route myfallbackapp foo.shared-domain.example.com --hostname *
```

Map a Route with Application Push

Developers can map a route to their app with the `cf push` command.

If a domain or hostname is not specified, then a route will be created using the app name and the default shared domain (see Shared Domains). The following command pushes the app `myapp`, creating the `myapp.shared-domain.example.com` route with the default shared domain `shared-domain.example.com`. If the route has not already been created in another space this command also maps it to the app.

```
cf push myapp
```

To customize the route during `push`, specify the domain using the `-d` flag and the hostname with the `--hostname` flag. The following command creates the `foo.private-domain.example.com` route for `myapp`:

```
cf push myapp -d private-domain.example.com --hostname foo
```

To map a TCP route during `push`, specify a TCP domain and request a random port using `--random-route`. To specify a port, push the app without a route, then create and map the route manually by following the steps in the Create a TCP Route with a Port section.

```
cf push tcp-app -d tcp.shared-domain.example.com --random-route
```

Map a Route Using Application Manifest

Developers can map a route to their app with a manifest by editing the route attribute to specify the host, domain, port and/or path components of the route. For more information, see the Deploying with Application Manifests topic.

Map a Route to Multiple Apps

Elastic Runtime allows multiple apps, or versions of the same app, to be mapped to the same route. This feature enables Blue-Green deployment. For more information see Using Blue-Green Deployment to Reduce Downtime and Risk.

Routing multiple apps to the same route may cause undesirable behavior in some situations by routing incoming requests randomly to one of the apps on the shared route.

See the Routing Conflict section of the Troubleshooting Application Deployment and Health topic for more information about troubleshooting this problem.

Unmap a Route

Developers can remove a route from an app using the `cf unmap-route` command. The route remains reserved for later use in the space where it was created until the route is deleted.

To unmap an HTTP route from an app, identify the route using the hostname, domain, and path:

```
cf unmap-route tcp-app private-domain.example.com --hostname myapp --path mypath
```

To unmap a TCP route from an app, identify the route using the domain and port:

```
cf unmap-route tcp-app tcp.shared-domain.example.com --port 60000
```

Delete a Route
Developers can delete a route from a space using the `cf delete-route` command.

To delete a HTTP route, identify the route using the hostname, domain, and path:

```
cf delete-route private-domain.example.com --hostname myapp --path mypath
```

To delete a TCP route, identify the route using the domain and port:

```
cf delete-route tcp.private-domain.example.com --port 60000
```

Routing Requests to a specific App Instance

Users can route HTTP requests to a specific application instance using the header `X-CF-APP-INSTANCE`. The format of the header should be `X-CF-APP-INSTANCE: APP_GUID:APP_INDEX`.

`APP_GUID` is an internal identifier for your application. To get it, developers can use the following CLI command.

```
cf myapp --guid
```

`APP_INDEX` (e.g. 0, 1, 2) is an identifier for a particular app instance. Developers can use the CLI command `cf app APP` to get statistics on each instance of a particular app.

```
cf app myapp
```

In the following example, a request is made to instance #9 of the application with GUID `5dc75f95-249b-46b8-8df5-a86b9f2df0e` which is mapped to route `myapp.private-domain.example.com`.

```
curl myapp.private-domain.example.com -H "X-CF-App-Instance: 5dc75f95-249b-46b8-8df5-a86b9f2df0e:9"
```

If the instance cannot be found or the format is wrong, a 404 status code is returned.

Domains

**Note:** The term domain in this topic differs from its common use and is specific to Cloud Foundry. Likewise, shared domain and private domain refer to resources with specific meaning in Cloud Foundry. The use of domain name, root domain, and subdomain refers to DNS records.

Domains indicate to a developer that requests for any route created from the domain will be routed to Elastic Runtime. This requires DNS to be configured out-of-band to resolve the domain name to the IP address of a load balancer configured to forward requests to the CF routers. For more information on configuring DNS, see the [DNS for Domains](#) section.

List Domains for an Org

When creating a route, developers will select from domains available to them. Use the `cf domains` command to view a list of available domains for the targeted org:

```
cf domains
```

This example displays three available domains: a shared HTTP domain `shared-domain.example.com`, a shared TCP domain `tcp.shared-domain.example.com`, and a private domain `private-domain.example.com`. See [Shared Domains](#) and [Private Domains](#).
HTTP vs TCP Domains

HTTP domains indicate to a developer that only requests using the HTTP protocol will be routed to applications mapped to routes created from the domain. Routing for HTTP domains is layer 7 and offers features like custom hostnames, sticky sessions, and TLS termination.

TCP domains indicate to a developer that requests over any TCP protocol, including HTTP, will be routed to applications mapped to routes created from the domain. Routing for TCP domains is layer 4 and protocol agnostic, so many features available to HTTP routing are not available for TCP routing. TCP domains are defined as being associated with the TCP Router Group. The TCP Router Group defines the range of ports available to be reserved with TCP Routes. Currently, only Shared Domains can be TCP.

Note: By default, Elastic Runtime only supports routing of HTTP requests to applications.

Shared Domains

Admins manage shared domains, which are available to users in all orgs of a Elastic Runtime deployment. An admin can offer multiple shared domains to users. For example, an admin may offer developers the choice of creating routes for their apps from shared-domain.example.com and cf.some-company.com.

There is not technically a default shared domain. If a developer pushes an app without specifying a domain (see Map a Route with Application Push), a route will be created for it from the first shared domain created in the system. All other operations involving route require the domain be specified (see Routes).

Shared domains are HTTP by default, but can be configured to be TCP when associated with the TCP Router Group.

Create a Shared Domain

Admins can create an HTTP shared domain with the cf create-shared-domain command:

```bash
$ cf create-shared-domain shared-domain.example.com
```

To create a TCP shared domain, first discover the name of the TCP Router Group.

```bash
$ cf router-groups
Getting router groups as admin ...
```

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-tcp</td>
<td>tcp</td>
</tr>
</tbody>
</table>

Then create the shared domain using the --router-group option to associate the domain with the TCP router group.

```bash
$ cf create-shared-domain tcp.shared-domain.example.com --router-group default-tcp
```

Delete a Shared Domain

Admins can delete a shared domain from Elastic Runtime with the cf delete-shared-domain command:

```bash
$ cf delete-shared-domain example.com
```

Private Domains

Org Managers can add private domains (or custom domains) and give members of the org permission to create routes for privately registered domain names. Private domains can be shared with other orgs, enabling users of those orgs to create routes from the domain.

Private domains can be HTTP only; TCP Routing is supported for Shared Domains only.
Create a Private Domain

Org Managers can create a private domain with the following command:

```
cf create-domain my-org private-domain.example.com
```

Org Managers can create a private domain for a subdomain with the following command:

```
cf create-domain my-org foo.private-domain.example.com
```

Sharing a Private Domain with One or More Orgs

Org Managers can grant or revoke access to a private domain to other orgs if they have permissions for these orgs with the following commands:

```
cf share-private-domain test-org private-domain.example.com
```

```
cf unshare-private-domain test-org private-domain.example.com
```

Delete a Private Domain

Org Managers can delete a domain from Elastic Runtime with the following command:

```
cf delete-domain private-domain.example.com
```

Requirements for Parent and Child Domains

In the domain `myapp.shared-domain.example.com`, `shared-domain.example.com` is the parent domain of subdomain `myapp`. Note the following requirements for domains:

- You can only create a private domain that is parent to a private subdomain.
- You can create a shared domain that is parent to either a shared or a private subdomain.

The domain `foo.myapp.shared-domain.example.com` is the child subdomain of `myapp.shared-domain.example.com`. Note the following requirements for subdomains:

- You can create a private subdomain for a private parent domain only if the domains belong to the same org.
- You can create a private subdomain for a shared parent domain.
- You can only create a shared subdomain for a shared parent domain.
- You cannot create a shared subdomain for a private parent domain.

DNS for Domains

Configuring DNS for a Subdomain of your Registered Domain

To use a subdomain of your registered domain name with apps on Elastic Runtime, configure the subdomain as CNAME record with your DNS provider, pointing at any shared domain offered in Elastic Runtime.

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Target</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNAME</td>
<td>myapp</td>
<td>myapp.shared-domain.example.com</td>
<td>Refer to your DNS provider documentation to determine whether the trailing <code>.</code> is required.</td>
</tr>
<tr>
<td>Wildcard CNAME</td>
<td>foo.myapp.shared-domain.example.com</td>
<td>You can use the wildcard in the CNAME record to point all of your subdomains to your parent domain. Each separately configured subdomain has priority over the wildcard configuration.</td>
<td></td>
</tr>
</tbody>
</table>
Configuring DNS for Your Registered Root Domain

To use your root domain (for example, example.com) for apps on Elastic Runtime you can either use custom DNS record types like ALIAS and ANAME, if your DNS provider offers them, or subdomain redirection.

**Note:** Root domains are also called zone apex domains.

If your DNS provider supports using an ALIAS or ANAME record, configure your root domain with your DNS provider to point at a shared domain in Elastic Runtime.

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Target</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIAS or ANAME</td>
<td>empty or @</td>
<td>private-domain.example.com.</td>
<td>Refer to your DNS provider documentation to determine whether to use an empty or @ value for the Name entry.</td>
</tr>
</tbody>
</table>

If your DNS provider does not support ANAME or ALIAS records you can use subdomain redirection, also known as domain forwarding, to redirect requests for your root domain to a subdomain configured as a CNAME.

**Note:** If you use domain forwarding, SSL requests to the root domain may fail if the SSL certificate only matches the subdomain.

Configure the root domain to point at a subdomain (www), and configure the subdomain as a CNAME record pointing at a shared domain in Elastic Runtime.

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Target</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL or Forward</td>
<td>private-domain.example.com</td>
<td><a href="http://www.private-domain.example.com">www.private-domain.example.com</a></td>
<td>This method results in a 301 permanent redirect to the subdomain you configure.</td>
</tr>
<tr>
<td>CNAME</td>
<td>www</td>
<td>myapp.shared-domain.example.com</td>
<td></td>
</tr>
</tbody>
</table>
Changing Stacks

A stack is a prebuilt root filesystem (rootfs) that supports a specific operating system. For example, Linux-based systems need `/usr` and `/bin` directories at their root, and Windows needs `/windows`. The stack works in tandem with a buildpack to support applications running in compartments. Under Diego architecture, cell VMs can support multiple stacks.

Note: Docker apps do not use stacks.

Available Stacks

- The Linux `cflinuxfs2` stack is derived from Ubuntu Trusty 14.04. Refer to the Github stacks page for supported libraries.
- The Windows stack `windows2012R2` supports .NET apps.

Restaging Applications on a New Stack

For security, stacks receive regular updates to address Common Vulnerabilities and Exposures (CVEs). Apps pick up on these stack changes through new releases of Elastic Runtime. However, if your app links statically to a library provided in the rootfs, you may have to manually restage it to pick up the changes.

It can be difficult to know what libraries an app statically links to, and it depends on the languages you are using. One example is an app that uses a Ruby or Python binary, and links out to part of the C standard library. If the C library requires an update, you may need to recompile the app and restage it as follows:

1. Use the `cf stacks` command to list the stacks available in a deployment.

   ```
   $ cf stacks
   Getting stacks in org MY-ORG / space development as developer@example.com...
   OK
   name       description
   cflinuxfs2  Cloud Foundry Linux-based filesystem
   windows2012R2 Windows Server 2012 R2
   ```

2. To change your stack and restage your application, use the `cf push` command. For example, to restage your app on the default stack `cflinuxfs2` you can run:

   ```
   $ cf push MY-APP
   Using stack cflinuxfs2...
   OK
   Creating app MY-WIN-APP in org MY-ORG / space development as developer@example.com...
   OK
   ...
   requested state: started
   instances: 1/1
   usage: 1G x 1 instances
   urls: MY-APP.cfapps.io
   last uploaded: Wed Apr 8 23:40:57 UTC 2015
   state since 2015-04-08 00:49:54 PM 0.0% 57.3M of 1G 128.3M of 1G
   
   To specify a different stack, append `-s STACKNAME` to the command. For example, you can ensure that Windows application MY-WIN-APP deploys to a Windows-based cell by running:

   ```
   $ cf push MY-WIN-APP -s windows2012R2
   ```

Stacks API

For API information, review the Stacks section of the Cloud Foundry API Documentation.
Deploying with Application Manifests

Page last updated:

Application manifests tell `cf push` what to do with applications. This includes everything from how many instances to create and how much memory to allocate to what services applications should use.

A manifest can help you automate deployment, especially of multiple applications at once.

How cf push Finds the Manifest

By default, the `cf push` command deploys an application using a `manifest.yml` file in the current working directory.

```
$ cf push
Using manifest file /path_to_working_directory/manifest.yml
```

If your manifest is located elsewhere, use the `-f` option to provide the path to the filename.

```
$ cf push -f /some_directory/some_other_directory/alternate_manifest.yml
Using manifest file /path_to_working_directory/some_directory/some_other_directory/alternate_manifest.yml
```

If you provide a path with no filename, the filename must be `manifest.yml`.

```
$ cf push -f /some_directory/some_other_directory/
Using manifest file /path_to_working_directory/some_directory/some_other_directory/manifest.yml
```

Example Manifest

You can deploy applications without ever using a manifest. The benefits manifests may provide include consistency and reproducibility. When you want applications to be portable between different clouds, manifests may prove especially useful.

Manifests are written in YAML. The manifest below illustrates some YAML conventions, as follows:

- The manifest may begin with three dashes.
- The `applications` block begins with a heading followed by a colon.
- The application `name` is preceded by a single dash and one space.
- Subsequent lines in the block are indented two spaces to align with `name`.

```
---
applications:
  - name: nifty-gui
    memory: 512M
    host: nifty
```

A minimal manifest requires only an application `name`. To create a valid minimal manifest, remove the `memory` and `host` properties from this example.

Always Provide an Application Name to cf push

`cf push` requires an application name, which you provide either in a manifest or at the command line.

As described in How cf push Finds the Manifest above, the command `cf push` locates the `manifest.yml` in the current working directory by default, or in the path provided by the `-f` option.

If you do not use a manifest, the minimal push command looks like this:

```
$ cf push my-app
```
How cf push Finds the Application

By default, `cf push` recursively pushes the contents of the current working directory. Alternatively, you can provide a path using either a manifest or a command line option.

- If the path is to a directory, `cf push` recursively pushes the contents of that directory instead of the current working directory.
- If the path is to a file, `cf push` pushes only that file.

### Note
If you want to push more than a single file, but not the entire contents of a directory, consider using a `.cfignore` file to tell `cf push` what to exclude.

Precedence Between Manifests, Command Line Options, and Most Recent Values

When you push an application for the first time, Cloud Foundry applies default values to any attributes that you do not set in a manifest or `cf push` command line options.

- For example, `cf push my-app` with no manifest might deploy one instance of the app with one gigabyte of memory. In this case the default values for instances and memory are "1" and "1G", respectively.

Between one push and another, attribute values can change in other ways.

- For example, the `cf scale` command changes the number of instances.

The attribute values on the server at any one time represent the cumulative result of all settings applied up to that point: defaults, attributes in the manifest, `cf push` command line options, and commands like `cf scale`. There is no special name for this resulting set of values on the server. You can think of them as the most recent values.

`cf push` follows rules of precedence when setting attribute values:

- Manifests override most recent values, including defaults.
- Command line options override manifests.

In general, you can think of manifests as just another input to `cf push`, to be combined with command line options and most recent values.

Optional Attributes

This section explains how to describe optional application attributes in manifests. Each of these attributes can also be specified by a command line option. Command line options override the manifest.

The buildpack attribute

If your application requires a custom buildpack, you can use the `buildpack` attribute to specify it in one of three ways:

- By name: `MY-BUILDPACK`.
- By GitHub URL: `https://github.com/cloudfoundry/java-buildpack.git`.
- By GitHub URL with a branch or tag: `https://github.com/cloudfoundry/java-buildpack.git#v3.3.0` for the `v3.3.0` tag.
The command line option that overrides this attribute is `-b`.

### The command attribute

Some languages and frameworks require that you provide a custom command to start an application. Refer to the `buildpack` documentation to determine if you need to provide a custom start command.

You can provide the custom start command in your application manifest or on the command line. See [About Starting Applications](#) for information on how `cf push` determines its default start command.

To specify the custom start command in your application manifest, add it in the `command: START-COMMAND` format as the following example shows:

```
---
...
command: bundle exec rake VERBOSE=true
---
```

The start command you specify becomes the default for your application. To return to using the original default start command set by your buildpack, you must explicitly set the `null` attribute as follows:

```
---
...
command: null
---
```

On the command line, use the `-c` option to specify the custom start command as the following example shows:

```
$ cf push my-app -c "bundle exec rake VERBOSE=true"
```

### The disk quota attribute

Use the `disk_quota` attribute to allocate the disk space for your app instance. This attribute requires a unit of measurement: `M`, `MB`, `G`, or `GB`, in upper case or lower case.

---

Note: The `cf buildpacks` command lists the buildpacks that you can refer to by name in a manifest or a command line option.

The command line option that overrides this attribute is `-b`.
The domain attribute

Every `cf push` deploys applications to one particular Cloud Foundry instance. Every Cloud Foundry instance may have a shared domain set by an admin. Unless you specify a domain, Cloud Foundry incorporates that shared domain in the route to your application.

You can use the `domain` attribute when you want your application to be served from a domain other than the default shared domain.

```
---
... 
  domain: unique-example.com
---
```

The command line option that overrides this attribute is `-d`.

The domains attribute

Use the `domains` attribute to provide multiple domains. If you define both `domain` and `domains` attributes, Cloud Foundry creates routes for domains defined in both of these fields.

```
---
... 
  domains:
    - domain-example1.com
    - domain-example2.org
---
```

The command line option that overrides this attribute is `-d`.

The stack attribute

Use the `stack` attribute to specify which stack to deploy your application to.

To see a list of available stacks, run `cf stacks` from the cf cli.

```
---
... 
  stack: cflinuxfs2
---
```

The command line option that overrides this attribute is `-s`.

The instances attribute

Use the `instances` attribute to specify the number of app instances that you want to start upon push:

```
---
... 
  instances: 2
---
```

We recommend that you run at least two instances of any apps for which fault tolerance matters.

The command line option that overrides this attribute is `-i`.

The memory attribute

Use the `memory` attribute to specify the memory limit for all instances of an app. This attribute requires a unit of measurement: `M`, `MB`, `G`, or `GB`.
upper case or lower case. For example:

```yaml
---
...  
memory: 1024M
---
```

The default memory limit is 1G. You might want to specify a smaller limit to conserve quota space if you know that your app instances do not require 1G of memory.

The command line option that overrides this attribute is `-m`.

### The health-check-type attribute

Use the `health-check-type` attribute to set the `health_check_type` flag to either `port` or `none`. If you do not provide a `health-check-type` attribute, it defaults to `port`.

```yaml
---
...
  health-check-type: none
---
```

The command line option that overrides this attribute is `-u`.

### The host attribute

Use the `host` attribute to provide a hostname, or subdomain, in the form of a string. This segment of a route helps to ensure that the route is unique. If you do not provide a hostname, the URL for the app takes the form of `APP-NAME.DOMAIN`.

```yaml
---
...
  host: my-app
---
```

The command line option that overrides this attribute is `-n`.

### The hosts attribute

Use the `host` attribute to provide multiple hostnames, or subdomains. Each hostname generates a unique route for the app. `hosts` can be used in conjunction with `host`. If you define both attributes, Cloud Foundry creates routes for hostnames defined in both `host` and `hosts`.

```yaml
---
...
  hosts:
    - app_host1
    - app_host2
---
```

The command line option that overrides this attribute is `-n`.

### The no-hostname attribute

By default, if you do not provide a hostname, the URL for the app takes the form of `APP-NAME.DOMAIN`. If you want to override this and map the root domain to this app then you can set no-hostname as true.

```yaml
---
...
  no-hostname: true
---
```

The command line option that corresponds to this attribute is `--no-hostname`.
The routes attribute

Use the `routes` attribute to provide multiple HTTP and TCP routes. Each route for this app is created if it does not already exist.

This attribute is a combination of `push` options that include `--hostname`, `-d`, and `--route-path`.

```yaml
---
...
routes:
- route: example.com
- route: www.example.com/foo
- route: tcp-example.com:1234
---
```

Manifest Attributes

The `routes` attribute cannot be used in conjunction with the following attributes: `host`, `hosts`, `domain`, `domains`, and `no-hostname`. An error will result.

Push Flag Options

This attribute has unique interactions with different command line options.

<table>
<thead>
<tr>
<th>Push Flag Option</th>
<th>Resulting Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--no-route</code></td>
<td>All declared routes are ignored.</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Overrides DOMAIN part of all declared HTTP and TCP routes.</td>
</tr>
<tr>
<td><code>--hostname</code>, <code>-n</code></td>
<td>Sets or overrides HOSTNAME in all HTTP routes.</td>
</tr>
<tr>
<td></td>
<td>It has no impact on TCP routes.</td>
</tr>
<tr>
<td><code>--route-path</code></td>
<td>Sets or overrides the PATH in all HTTP routes.</td>
</tr>
<tr>
<td></td>
<td>It has no impact on TCP routes.</td>
</tr>
<tr>
<td><code>--random-route</code></td>
<td>Sets or overrides the HOSTNAME in all HTTP routes.</td>
</tr>
<tr>
<td></td>
<td>Sets or overrides the PORT in all TCP routes.</td>
</tr>
<tr>
<td></td>
<td>The PORT and HOSTNAME will be randomly generated.</td>
</tr>
</tbody>
</table>

The random-route attribute

If you push your app without specifying any route-related CLI options or app manifest flags, the cf CLI attempts to generate a route based on the app name, which can cause collisions.

You can use the `random-route` attribute to generate a unique route and avoid name collisions.

When you use `random-route`, the cf CLI generates an HTTP route with a random host (if `host` is not set) or a TCP route with an unused port number.

See the following example use cases:

- You deploy the same app to multiple spaces for testing purposes. In this situation, you can use `random-route` to randomize routes declared with the route attribute in the app manifest.
- You use an app manifest for a classroom training exercise in which multiple users deploy the same app to the same space.

The command line option that corresponds to this attribute is `--random-route`.

```yaml
---
...
random-route: true
---
```

The path attribute

You can use the `path` attribute to tell Cloud Foundry where to find your application. This is generally not necessary when you run `cf push` from the directory where an application is located.

```yaml
---
```
The command line option that overrides this attribute is `p`.

### The timeout attribute

The `timeout` attribute defines the number of seconds that Cloud Foundry allocates for starting your application.

For example:

```yaml
---
...  
timeout: 80
---
```

You can increase the timeout length for very large apps that require more time to start. The `timeout` attribute defaults to 60, but you can set it to any value up to the Cloud Controller's `maximum_health_check_timeout` property.

`maximum_health_check_timeout` defaults to 180, but your Cloud Foundry operator can set to any value.

The command line option that overrides the timeout attribute is `t`.

### The no-route attribute

By default, `cf push` assigns a route to every app. But, some apps process data while running in the background and should not be assigned routes.

You can use the `no-route` attribute with a value of `true` to prevent a route from being created for your app.

```yaml
---
...  
no-route: true
---
```

The command line option that corresponds to this attribute is `--no-route`.

In the newer Diego architecture, `no-route` skips creating and binding a route for the app, but does not specify which type of health check to perform. If your app does not listen on a port because it is a worker or a scheduler app, then it does not satisfy the port-based health check and Cloud Foundry marks it as crashed. To prevent this, disable the port-based health check with `cf set-health-check APP_NAME none`.

In the older Droplet Execution Agent (DEA) architecture, `cf set-health-check APP_NAME none` is unnecessary because `no-route` causes the DEAs to skip the port health-check on app startup.

To remove a route from an existing app, perform the following steps:

1. Remove the route using the `cf unmap-route` command.
2. Push the app again with the `no-route: true` attribute in the manifest or the `--no-route` command line option.

For more information, see [Describing Multiple Applications with One Manifest](#) below.

### Environment Variables

The `env` block consists of a heading, then one or more environment variable/value pairs.

For example:

```yaml
---
...  
env:  
  RAILS_ENV: production  
  RACK_ENV: production
---
```
**cf push** deploys the application to a container on the server. The variables belong to the container environment.

While the application is running, you can modify environment variables.

- View all variables: `cf env my-app`
- Set an individual variable: `cf set-env my-app my-variable_name my-variable_value`
- Unset an individual variable: `cf unset-env my-app my-variable_name my-variable_value`

Environment variables interact with manifests in the following ways:

- When you deploy an application for the first time, Cloud Foundry reads the variables described in the environment block of the manifest, and adds them to the environment of the container where the application is deployed.
- When you stop and then restart an application, its environment variables persist.

**Services**

Applications can bind to services such as databases, messaging, and key-value stores.

Applications are deployed into App Spaces. An application can only bind to services instances that exist in the target App Space before the application is deployed.

The `services` block consists of a heading, then one or more service instance names.

Whoever creates the service chooses the service instance names. These names can convey logical information, as in `backend_queue`, describe the nature of the service, as in `mysql_5.x`, or do neither, as in the example below.

```yaml
---
...
services:
- instance_ABC
- instance_XYZ
```

Binding to a service instance is a special case of setting an environment variable, namely `VCAP_SERVICES`. See the [Bind a Service](#) section of the [Delivering Service Credentials to an Application](#) topic.

### Describing Multiple Applications with One Manifest

You can deploy multiple applications with one `cf push` command by describing them in a single manifest. In doing so, you need to pay extra attention to directory structure and path lines in the manifest.

Suppose you want to deploy two applications called respectively spark and flame, and you want Cloud Foundry to create and start spark before flame. You accomplish this by listing spark first in the manifest.

In this situation there are two sets of bits that you want to push. Let’s say that they are `spark.rb` in the spark directory and `flame.rb` in the flame directory. One level up, the `fireplace` directory contains the spark and the flame directories along with the `manifest.yml` file. Your plan is to run the cf CLI from the `fireplace` directory, where you know it can find the manifest.

Now that you have changed the directory structure and manifest location, `cf push` can no longer find your applications by its default behavior of looking in the current working directory. How can you ensure that `cf push` finds the bits you want to push?

The answer is to add a path line to each application description to lead `cf push` to the correct bits. Assume that `cf push` is run from the `fireplace` directory.

For `spark`:

```yaml
---
...
path: ./spark/
```

For `flame`:

```yaml
---
...
path: ./flame/
```
The manifest now consists of two applications blocks.

```yaml
---
# this manifest deploys two applications
# apps are in flame and spark directories
# flame and spark are in fireplace
# cf push should be run from fireplace
applications:
  - name: spark
    memory: 1G
    instances: 2
    host: flint-99
    domain: shared-domain.example.com
    path: ./spark/
    services:
      - mysql-flint-99
    - name: flame
      memory: 1G
      instances: 2
      host: burnin-77
      domain: shared-domain.example.com
      path: ./flame/
      services:
        - redis-burnin-77
```

Follow these general rules when using a multiple-application manifest:

- Name and completely describe your applications in the manifest.
- Use a `no-route` line in the description of any application that provides background services to another application.
- Do not provide an application name with `cf push`.
- Do not use any command line options with `cf push`.

There are only two narrow exceptions:

- If your manifest is not named `manifest.yml` or not in the current working directory, use the `-f` command line option.
- If you want to push a single application rather than all of the applications described in the manifest, provide the desired application name by running `cf push my-app`.

Minimizing Duplication

In manifests where multiple applications share settings or services, you begin to see content duplicated. While the manifests still work, duplication increases the risk of typographical errors which cause deployment to fail.

The cure for this problem is to “promote” the duplicate content—that is, to move it up above the applications block, where it need appear only once. The promoted content applies to all applications described in the manifest. Note that content in the applications block overrides content above the applications block, if the two conflict.

The manifest becomes shorter, more readable, and more maintainable.

Notice how much content in the manifest below has been promoted in this way.

```yaml
---
...  
# all applications use these settings and services
domain: shared-domain.example.com
memory: 1G
instances: 1
services:
  - clockwork-mysql
applications:
  - name: springtock
    host: tock09876
    path: ./spring-music/build/libs/spring-music.war
  - name: springtick
    host: tick09875
    path: ./spring-music/build/libs/spring-music.war
```

In the next section we carry this principle further by distributing content across multiple manifests.
Multiple Manifests with Inheritance

A single manifest can describe multiple applications. Another powerful technique is to create multiple manifests with inheritance. Here, manifests have parent-child relationships such that children inherit descriptions from a parent. Children can use inherited descriptions as-is, extend them, or override them.

Content in the child manifest overrides content in the parent manifest, if the two conflict.

This technique helps in these and other scenarios:

- An application has a set of different deployment modes, such as debug, local, and public. Each deployment mode is described in child manifests that extend the settings in a base parent manifest.
- An application is packaged with a basic configuration described by a parent manifest. Users can extend the basic configuration by creating child manifests that add new properties or override those in the parent manifest.

The benefits of multiple manifests with inheritance are similar to those of minimizing duplicated content within single manifests. With inheritance, though, we “promote” content by placing it in the parent manifest.

Every child manifest must contain an “inherit” line that points to the parent manifest. Place the inherit line immediately after the three dashes at the top of the child manifest. For example, every child of a parent manifest called `base-manifest.yml` begins like this:

```yaml
---
inherit: base-manifest.yml
...
```

You do not need to add anything to the parent manifest.

In the simple example below, a parent manifest gives each application minimal resources, while a production child manifest scales them up.

**simple-base-manifest.yml**

```yaml
---
path: .
domain: shared-domain.example.com
memory: 256M
instances: 1
services:
  - singular-backend

# app-specific configuration
applications:
  - name: springtock
    host: 765shower
    path: ./april/build/libs/april-weather.war
  - name: wintertick
    host: 321flurry
    path: ./december/target/december-weather.war
```

**simple-prod-manifest.yml**

```yaml
---
inherit: simple-base-manifest.yml
applications:
  - name: springstorm
    memory: 512M
    instances: 1
    host: 765deluge
    path: ./april/build/libs/april-weather.war
  - name: winterblast
    memory: 1G
    instances: 2
    host: 321blizzard
    path: ./december/target/december-weather.war
```

**Note:** Inheritance can add an additional level of complexity to manifest creation and maintenance. Comments that precisely explain how the child manifest extends or overrides the descriptions in the parent manifest can alleviate this complexity.
Scaling an Application Using cf scale

Factors such as user load, or the number and nature of tasks performed by an application, can change the disk space and memory the application uses. For many applications, increasing the available disk space or memory can improve overall performance. Similarly, running additional instances of an application can allow the application to handle increases in user load and concurrent requests. These adjustments are called scaling an application.

Use `cf scale` to scale your application up or down to meet changes in traffic or demand.

Scaling Horizontally

Horizontally scaling an application creates or destroys instances of your application.

Incoming requests to your application are automatically load balanced across all instances of your application, and each instance handles tasks in parallel with every other instance. Adding more instances allows your application to handle increased traffic and demand.

Use `cf scale APP -i INSTANCES` to horizontally scale your application. Cloud Foundry will increase or decrease the number of instances of your application to match `INSTANCES`.

$ cf scale myApp -i 5

Scaling Vertically

Vertically scaling an application changes the disk space limit or memory limit that Cloud Foundry applies to all instances of the application.

Use `cf scale APP -k DISK` to change the disk space limit applied to all instances of your application. `DISK` must be an integer followed by either an M, for megabytes, or G, for gigabytes.

$ cf scale myApp -k 512M

Use `cf scale APP -m MEMORY` to change the memory limit applied to all instances of your application. `MEMORY` must be an integer followed by either an M, for megabytes, or G, for gigabytes.

$ cf scale myApp -m 1G
Cloud Foundry Environment Variables

Page last updated:

Environment variables are the means by which the Cloud Foundry runtime communicates with a deployed application about its environment. This page describes the environment variables that the runtime and buildpacks set for applications.

For information about setting your own application-specific environment variables, refer to the Set Environment Variable in a Manifest section in the Application Manifests topic.

View Environment Variables

Use the `cf env` command to view the Cloud Foundry environment variables for your application. `cf env` displays the following environment variables:

- The `VCAP_APPLICATION` and `VCAP_SERVICES` variables provided in the container environment
- The user-provided variables set using the `cf set-env` command

```bash
$ cf env my-app
Getting env variables for app my-app in org my-org / space my-space as admin...
OK
System-Provided:

```
{
  "VCAP_APPLICATION": {
    "application_id": "fa051a9-0fe1-4bdh-ba1e-139850dec7a3",
    "application_name": "my-app",
    "application_uris": [
      "my-app.192.0.2.34.xip.io"
    ],
    "application_version": "f8f8bec6-8d58-479e-bc7-3b4ec5a7b0ca",
    "limits": {
      "disk": 1024,
      "fds": 16384,
      "mem": 256
    },
    "name": "my-app",
    "space_id": "06450c72-4669-4dc6-8096-45f9777db68a",
    "space_name": "my-space",
    "uris": [
      "my-app.192.0.2.34.xip.io"
    ],
    "users": null,
    "version": "f8f8bec6-8d58-479e-bc7-3b4ec5a7b0ca"
  }
}
```

User-Provided:

```
MY_DRAIN: http://drain.example.com
MY_ENV_VARIABLE: 100
```

Application-Specific System Variables

The subsections that follow describe the environment variables that Cloud Foundry makes available to your application container. Some of these variables are the same across instances of a single application, and some vary from instance to instance.

You can access environment variables programmatically, including variables defined by the buildpack. Refer to the buildpack documentation for Java, Node.js, and Ruby.

CF_INSTANCE_ADDR

The `CF_INSTANCE_IP` and `CF_INSTANCE_PORT` of the app instance in the format `IP:PORT`.

```
CF_INSTANCE_ADDR=1.2.3.4:5678
```
**CF_INSTANCE_GUID**

The UUID of the particular instance of the app. Available only to instances on Diego Cells.

```
CF_INSTANCE_GUID=41653aa4-3a3a-486a-4431-ef258b39f042
```

**CF_INSTANCE_INDEX**

The index number of the app instance.

```
CF_INSTANCE_INDEX=0
```

**CF_INSTANCE_IP**

The external IP address of the host running the app instance.

```
CF_INSTANCE_IP=1.2.3.4
```

**CF_INSTANCE_PORT**

The external (host-side) port corresponding to the internal (container-side) port with value PORT. For instances on Diego, this value is generally not the same as the PORT of the app instance.

```
CF_INSTANCE_PORT=61045
```

**CF_INSTANCE_PORTS**

The list of mappings between internal (container-side) and external (host-side) ports allocated to the instance's container. Note that not all of the internal ports are necessarily available for the application to bind to, as some of them may be used by system-provided services that also run inside the container. On the DEAs, these internal and external values will be the same, but on Diego Cells, they may differ.

```
CF_INSTANCE_PORTS=[{external:61045,internal:5678}, {external:61046,internal:2222}]
```

**HOME**

Root folder for the deployed application.

```
HOME=/home/vcap/app
```

**MEMORY_LIMIT**

The maximum amount of memory that each instance of the application can consume. You specify this value in an application manifest or with the cf CLI when pushing an application. The value is limited by space and org quotas.

If an instance goes over the maximum limit, it will be restarted. If it has to be restarted too often, it will be terminated.

```
MEMORY_LIMIT=512m
```

**PORT**

The port on which the application should listen for requests. The Cloud Foundry runtime allocates a port dynamically for each instance of the application, so code that obtains or uses the application port should refer to it via the PORT environment variable.

```
PORT=61857
```
PWD
Identifies the present working directory, where the buildpack that processed the application ran.

```bash
PWD=/home/vcap/app
```

TMPDIR
Directory location where temporary and staging files are stored.

```bash
TMPDIR=/home/vcap/tmp
```

USER
The user account under which the application runs.

```bash
USER=vcap
```

VCAP_APP_HOST
The IP address of the host. Deprecated: the DEAs set this to be `0.0.0.0`, and Diego Cells do not provide this environment variable.

```bash
VCAP_APP_HOST=0.0.0.0
```

VCAP_APP_PORT
Deprecated name for the `PORT` variable defined above.

VCAP_APPLICATION
This variable contains the associated attributes for a deployed application. Results are returned in JSON format. The table below lists the attributes that are returned.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_id</td>
<td>GUID identifying the application.</td>
</tr>
<tr>
<td>application_name</td>
<td>The name assigned to the application when it was pushed.</td>
</tr>
<tr>
<td>application_apis</td>
<td>The URIs assigned to the application.</td>
</tr>
<tr>
<td>application_version</td>
<td>GUID identifying a version of the application. Each time an application is pushed or restarted, this value is updated.</td>
</tr>
<tr>
<td>host</td>
<td>Deprecated. IP address of the application instance.</td>
</tr>
<tr>
<td>instance_id</td>
<td>Unique ID that identifies the application instance. For instances running on Diego, this is identical to the <code>CF_INSTANCE_GUID</code> variable.</td>
</tr>
<tr>
<td>instance_index</td>
<td>Index number of the instance. Identical to the <code>CF_INSTANCE_INDEX</code> variable.</td>
</tr>
<tr>
<td>limits</td>
<td>The memory, disk, and number of files permitted to the instance. Memory and disk limits are supplied when the application is deployed, either on the command line or in the application manifest. The number of files allowed is operator-defined.</td>
</tr>
<tr>
<td>name</td>
<td>Identical to <code>application_name</code>.</td>
</tr>
<tr>
<td>port</td>
<td>Port of the application instance. Identical to the <code>PORT</code> variable.</td>
</tr>
<tr>
<td>space_id</td>
<td>GUID identifying the application’s space.</td>
</tr>
<tr>
<td>start</td>
<td>Human-readable timestamp for the time the instance was started. Not provided on Diego Cells.</td>
</tr>
<tr>
<td>started_at</td>
<td>Identical to <code>start</code>. Not provided on Diego Cells.</td>
</tr>
<tr>
<td>started_at_timestamp</td>
<td>Unix epoch timestamp for the time the instance was started. Not provided on Diego Cells.</td>
</tr>
<tr>
<td>state_timestamp</td>
<td>Identical to <code>started_at_timestamp</code>. Not provided on Diego Cells.</td>
</tr>
</tbody>
</table>
For example:

```json
VCAP_APPLICATION={"instance_id":"fe98dc76ba549876543210abcd1234",
"instance_index":0,"host":"0.0.0.0","port":61857,"started_at":"2013-08-12 00:05:29 +0000","started_at_timestamp":1376265929,"started_at":null,"state_timestamp":1376265929,"limits":{"mem":512,"disk":1024,"fds":16384},
```

VCAP_SERVICES

For *bindable services* Cloud Foundry will add connection details to the `VCAP_SERVICES` environment variable when you restart your application, after binding a service instance to your application.

The results are returned as a JSON document that contains an object for each service for which one or more instances are bound to the application. The service object contains a child object for each service instance that is bound to the application. The attributes that describe a bound service are defined in the table below.

The key for each service in the JSON document is the same as the value of the "label" attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name assigned to the service instance by the user</td>
</tr>
<tr>
<td>label</td>
<td>The name of the service offering</td>
</tr>
<tr>
<td>tags</td>
<td>An array of strings an app can use to identify a service instance</td>
</tr>
<tr>
<td>plan</td>
<td>The service plan selected when the service instance was created</td>
</tr>
<tr>
<td>credentials</td>
<td>A JSON object containing the service-specific credentials needed to access the service instance.</td>
</tr>
</tbody>
</table>

To see the value of `VCAP_SERVICES` for an application pushed to Cloud Foundry, see [View Environment Variable Values](#).

The example below shows the value of `VCAP_SERVICES` for bound instances of several services available in the [Pivotal Web Services](#) Marketplace.
Environment Variable Groups

Environment variable groups are system-wide variables that enable operators to apply a group of environment variables to all running applications and all staging applications separately.

An environment variable group consists of a single hash of name-value pairs that are later inserted into an application container at runtime or at staging. These values can contain information such as HTTP proxy information. The values for variables set in an environment variable group are case-sensitive.

When creating environment variable groups, consider the following:

- Only the Cloud Foundry operator can set the hash value for each group.
- All authenticated users can get the environment variables assigned to their application.
- All variable changes take effect after the operator restarts or restages the applications.
- Any user-defined variable takes precedence over environment variables provided by these groups.

The table below lists the commands for environment variable groups.

<table>
<thead>
<tr>
<th>CLI Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>running-environment-variable-group</td>
<td>Retrieves the contents of the running environment variable group</td>
</tr>
<tr>
<td>or rrevg</td>
<td></td>
</tr>
<tr>
<td>staging-environment-variable-group</td>
<td>Retrieves the contents of the staging environment variable group</td>
</tr>
<tr>
<td>or sevg</td>
<td></td>
</tr>
<tr>
<td>set-staging-environment-variable-group</td>
<td>Passes parameters as JSON to create a staging environment variable group</td>
</tr>
<tr>
<td>or ssevg</td>
<td></td>
</tr>
<tr>
<td>set-running-environment-variable-group</td>
<td>Passes parameters as JSON to create a running environment variable group</td>
</tr>
<tr>
<td>or srevg</td>
<td></td>
</tr>
</tbody>
</table>

The following examples demonstrate how to retrieve the environment variables:
$ cf revg
Retrieving the contents of the running environment variable group as
sampledeveloper@example.com...
OK
Variable Name Assigned Value
HTTP Proxy 198.51.100.130

$ cf sevg
Retrieving the contents of the staging environment variable group as
sampledeveloper@example.com...
OK
Variable Name Assigned Value
HTTP Proxy 203.0.113.105
EXAMPLE-GROUP 2001

$ cf apps
Getting apps in org SAMPLE-ORG-NAME / space dev as
sampledeveloper@example.com...
OK
name requested state instances memory disk urls
my-app started 1/1 256M 1G my-app.com

$ cf env APP-NAME
Getting env variables for app APP-NAME in org SAMPLE-ORG-NAME / space dev as
sampledeveloper@example.com...
OK
System-Provided:
{
  "VCAP_APPLICATION":{
    "application_name": "APP-NAME",
    "application_uris": ["my-app.example.com"],
    "application_version": "7d0d64be-7f6f-406a-9d21-504643147d63",
    "$limits": {
      "disk": 1024,
      "disk_size": 1024,
      "num": 256
    },
    "name": "APP-NAME",
    "space_id": "37189599-2407-9946-865e-8ebd82c2d87a",
    "space_name": "dev",
    "uris": ["my-app.example.com"],
    "users": null,
    "version": "7d0d64be-78f4-406a-9d21-504643147d63"},
}
Running Environment Variable Groups:
HTTP Proxy: 198.51.100.130
Staging Environment Variable Groups:
EXAMPLE-GROUP: 2001
HTTP Proxy: 203.0.113.105

The following examples demonstrate how to set environment variables:

$ cf sevg '{"test":"198.51.100.130","test2":"203.0.113.105"}'
Setting the contents of the staging environment variable group as admin...
OK
$ cf sevg
Retrieving the contents of the staging environment variable group as admin...
OK
Variable Name Assigned Value
test 198.51.100.130
test2 203.0.113.105

$ cf revg '{"test3":"2001","test4":"2010"}'
Setting the contents of the running environment variable group as admin...
OK
$ cf revg
Retrieving the contents of the running environment variable group as admin...
OK
Variable Name Assigned Value
test3 2001
test4 2010
Using Blue-Green Deployment to Reduce Downtime and Risk

Blue-green deployment is a release technique that reduces downtime and risk by running two identical production environments called Blue and Green.

At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

As you prepare a new release of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

This technique can eliminate downtime due to application deployment. In addition, blue-green deployment reduces risk: if something unexpected happens with your new release on Green, you can immediately roll back to the last version by switching back to Blue.

Blue-Green Deployment with Cloud Foundry Example

For this example, we'll start with a simple application: “demo-time.” This app is a web page that displays the words “Blue time” and the date/time on the server.

Step 1: Push an App

Use the cf CLI to push the application. Name the application “Blue” with the subdomain “demo-time.”

```
cf push Blue -n demo-time
```

As shown in the graphic below:

- Blue is now running on Cloud Foundry.
- The CF Router sends all traffic for demo-time.example.com traffic to Blue.

![Diagram showing Blue running on Cloud Foundry](image)

Step 2: Update App and Push

Now make a change to the application. First, replace the word “Blue” on the web page with “Green,” then rebuild the source file for the application. Run `cf push` again, but use the name “Green” for the application and provide a different subdomain to create a temporary route:

```
cf push Green -n demo-time-temp
```

After this push:

- Two instances of our application are now running on Cloud Foundry: the original Blue and the updated Green.
• The CF Router still sends all traffic for `demo-time.example.com` traffic to Blue. The router now also sends any traffic for `demo-time-temp.example.com` to Green.

Step 3: Map Original Route to Green

Now that both apps are up and running, switch the router so all incoming requests go to the Green app and the Blue app. Do this by mapping the original URL route (`demo-time.example.com`) to the Green application using the `cf map-route` command.

```bash
$ cf map-route Green example.com -n demo-time
Binding demo-time.example.com to Green... OK
```

After the `cf map-route` command:

• The CF Router continues to send traffic for `demo-time-temp.example.com` to Green.
• The CF Router immediately begins to load balance traffic for `demo-time.example.com` between Blue and Green.

Step 4: Unmap Route to Blue

Once you verify Green is running as expected, stop routing requests to Blue using the `cf unmap-route` command:

```bash
$ cf unmap-route Blue example.com -n demo-time
Unbinding demo-time.example.com from blue... OK
```

After `cf unmap-route` command:
The CF Router stops sending traffic to Blue. Instead, it routes all traffic to `demo-time.example.com` to Green:

![Diagram showing traffic routing from CF Router to Blue and Green](image)

**Step 5: Remove Temporary Route to Green**

You can now use `cf unmap-route` to remove the route to `demo-time-temp.example.com`. You can also decommission Blue, or keep it in case you need to roll back your changes.

![Diagram showing traffic routing from CF Router to Green](image)

**Implementations**

Cloud Foundry community members have written plugins to automate the blue-green release process. These include:

- **Autopilot**: Autopilot is a Cloud Foundry Go plugin that provides a subcommand, `zero-downtime-push`, for hands-off, zero-downtime application deploys.

- **BlueGreenDeploy**: `cf-blue-green-deploy` is a plugin, written in Go, for the Cloud Foundry Command Line Interface (cf CLI) that automates a few steps involved in zero-downtime deploys.
Troubleshooting Application Deployment and Health

Refer to this topic for help diagnosing and resolving common issues when you deploy and run applications on Cloud Foundry.

Common Issues

The following sections describe common issues you might encounter when attempting to deploy and run your application, and possible resolutions.

cf push Times Out

If your deployment times out during the upload or staging phase, you may receive one of the following error messages:

- **504 Gateway Timeout**
- **Error uploading application**
- **Timed out waiting for async job JOB-NAME to finish**

If this happens, do the following:

- **Check your network speed.** Depending on the size of your application, your `cf push` could be timing out because the upload is taking too long. We recommended an Internet connection speed of at least 768 KB/s (6 Mb/s) for uploads.
- **Make sure you are pushing only needed files.** By default, `cf push` will push all the contents of the current working directory. Make sure you are pushing only the directory for your application. If your application is too large, or if it has many small files, Cloud Foundry may time out during the upload. To reduce the number of files you are pushing, ensure that you push only the directory for your application, and remove unneeded files or use the `.cfignore` file to specify excluded files.
- **Set the CF_STAGING_TIMEOUT and CF_STARTUP_TIMEOUT environment variables.** By default your app has 15 minutes to stage and 5 minutes to start. You can increase these times by setting `CF_STAGING_TIMEOUT` and `CF_STARTUP_TIMEOUT`. Type `cf help` at the command line for more information.
- **If your app contains a large number of files, try pushing the app repeatedly.** Each push uploads a few more files. Eventually, all files have uploaded and the push succeeds. This is less likely to work if your app has many small files.

App Too Large

If your application is too large, you may receive one of the following error messages on `cf push`:

- **413 Request Entity Too Large**
- **You have exceeded your organization’s memory limit**

If this happens, do the following:

- **Make sure your org has enough memory for all instances of your app.** You will not be able to use more memory than is allocated for your organization. To view the memory quota for your org, use `cf org ORG_NAME`. Your total memory usage is the sum of the memory used by all applications in all spaces within the org. Each application’s memory usage is the memory allocated to it multiplied by the number of instances. To view the memory usage of all the apps in a space, use `cf apps`.
- **Make sure your application is less than 1 GB.** By default, Cloud Foundry deploys all the contents of the current working directory. To reduce the number of files you are pushing, ensure that you push only the directory for your application, and remove unneeded files or use the `.cfignore` file to specify excluded files. The following limits apply:
  - The app files to push cannot exceed 1 GB.
  - The droplet that results from compiling those files cannot exceed 1.5 GB. Droplets are typically a third larger than the pushed files.
  - The combined size of the app files, compiled droplet, and buildpack cache cannot total more than 4 GB of space during staging.

Unable to Detect a Supported Application Type

If Cloud Foundry cannot identify an appropriate buildpack for your app, you will see an error message that states

© Copyright Pivotal Software Inc, 2013-2018
You can view what buildpacks are available with the `cf buildpacks` command.

If you see a buildpack that you believe should support your app, refer to the buildpack documentation for details about how that buildpack detects applications it supports.

If you do not see a buildpack for your app, you may still be able to push your application with a custom buildpack using `cf push -b` with a path to your buildpack.

App Deploy Fails

Even when the deploy fails, the app might exist on Elastic Runtime. Run `cf apps` to review the apps in the currently targeted org and space. You might be able to correct the issue using the CLI or the Apps Manager, or you might have to delete the app and redeploy.

Common reasons deploying an app fails include the following:

- You did not successfully create and bind a needed service instance to the app, such as a PostgreSQL or MongoDB service instance. Refer to Step 3: Create and Bind a Service Instance for a RoR Application.
- You did not successfully create a unique URL for the app. Refer to the troubleshooting tip App Requires Unique URL.

App Requires Unique URL

Elastic Runtime requires that each app that you deploy has a unique URL. Otherwise, the new app URL collides with an existing app URL and Elastic Runtime cannot successfully deploy the app. You can resolve this issue by running `cf push` with either of the following flags to create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words. Using this option might create a long URL, depending on the number of words that the app name includes.

App Fails to Start

After `cf push` stages the app and uploads the droplet, the app may fail to start, commonly with a pattern of starting and crashing similar to the following example:

```
--- app.crash
index: 0, reason: CRASHED, exit_description: app instance exited, exit_status: 1
```

If this happens, try the following:

Find the reason app is failing and modify your code. Run `cf events APP-NAME` and `cf logs APP-NAME --recent` and look for messages similar to this:

```
2014-04-29T17:52:34.00-0700 app.crash: index: 0, reason: CRASHED, exit_description: app instance exited, exit_status: 1
```

These messages may identify a memory or port issue. If they do, take that as a starting point when you re-examine and fix your application code.

- Make sure your application code uses the `PORT` environment variable. Your application may be failing because it is listening on the wrong port. Instead of hard coding the port on which your application listens, use the `PORT` environment variable.

For example, this Ruby snippet assigns the port value to the `listen_here` variable:

```
listen_here = ENV['PORT']
```

For more examples specific to your application framework, see the appropriate buildpack documentation for your app’s language.
• Make sure your app adheres to the principles of the Twelve-Factor App and Prepare to Deploy an Application. These texts explain how to prevent situations where your app builds locally but fails to build in the cloud.

App consumes too much memory, then crashes
An app that `cf push` has uploaded and started can crash later if it uses too much memory.

Make sure your app is not consuming more memory than it should. When you run `cf push` and `cf scale`, that configured a limit on the amount of memory your app should use. Check your app's actual memory usage. If it exceeds the limit, modify the app to use less memory.

Routing Conflict
Elastic Runtime allows multiple apps, or versions of the same app, to be mapped to the same route. This feature enables Blue-Green deployment. For more information see Using Blue-Green Deployment to Reduce Downtime and Risk.

Routing multiple apps to the same route may cause undesirable behavior in some situations by routing incoming requests randomly to one of the apps on the shared route.

If you suspect a routing conflict, run `cf routes` to check the routes in your installation.

If two apps share a route outside of a Blue-Green deploy strategy, choose one app to re-assign to a different route and follow the procedure below:

1. Run `cf unmap-route YOUR-APP-NAME OLD-ROUTE` to remove the existing route from that app.
2. Run `cf map-route YOUR-APP-NAME NEW-ROUTE` to map the app to a new, unique route.

Gathering Diagnostic Information
Use the techniques in this section to gather diagnostic information and troubleshoot app deployment issues.

Examine Environment Variables
`cf push` deploys your application to a container on the server. The environment variables in the container govern your application.

You can set environment variables in a manifest created before you deploy. See Deploying with Application Manifests.

You can also set an environment variable with a `cf set-env` command followed by a `cf push` command. You must run `cf push` for the variable to take effect in the container environment.

Use the `cf env` command to view the environment variables that you have set using the `cf set-env` command and the variables in the container environment:
Getting env variables for app my-app in org My-Org / space development as admin...

OK

System-Provided:

```
{
  "VCAP_SERVICES": {
    "p-mysql-n/a": {
      "credentials": {
        "uri": "postgres://lrra:e6B-X@p-mysqlprovider.example.com:5432/lraa"
      },
      "label": "p-mysql-n/a",
      "name": "p-mysql",
      "syslog_drain_url": "",
      "tags": ["postgres", "postgresql", "relational"]
    }
  }
}
```

User-Provided:

```
my-env-var: 100
my-drain: http://drain.example.com
```

View Logs

To view app logs streamed in real-time, use the `cf logs APP-NAME` command.

To aggregate your app logs to view log history, bind your app to a syslog drain service. For more information, see [Streaming Application Logs to Log Management Services](#).

**Note:** The Diego architecture does not support the `cf files` command.

Trace Cloud Controller REST API Calls

If a command fails or produces unexpected results, re-run it with HTTP tracing enabled to view requests and responses between the cf CLI and the Cloud Controller REST API.

For example:

- Re-run `cf push` with `-v`:
  ```
  cf push APP-NAME -v
  ```

- Re-run `cf push` while appending API request diagnostics to a log file:
  ```
  CF_TRACE=PATH-TO-TRACE.LOG cf push APP-NAME
  ```

These examples enable HTTP tracing for a single command only. To enable it for an entire shell session, set the variable first:

```
export CF_TRACE=true

export CF_TRACE=PATH-TO-TRACE.LOG
```

**Note:** `CF_TRACE` is a local environment variable that modifies the behavior of the cf CLI. Do not confuse `CF_TRACE` with the variables in the container environment where your apps run.

cf Troubleshooting Commands

You can investigate app deployment and health using the cf CLI.

Some cf CLI commands may return connection credentials. Remove credentials and other sensitive information from command output before you post the output to a public forum.
- `cf apps`: Returns a list of the applications deployed to the current space with deployment options, including the name, current state, number of instances, memory and disk allocations, and URLs of each application.

- `cf app APP-NAME`: Returns the health and status of each instance of a specific application in the current space, including instance ID number, current state, how long it has been running, and how much CPU, memory, and disk it is using.

- `cf env APP-NAME`: Returns environment variables set using `cf set-env` and variables existing in the container environment.

- `cf events APP-NAME`: Returns information about application crashes, including error codes. See [https://github.com/cloudfoundry/errors](https://github.com/cloudfoundry/errors) for a list of Cloud Foundry errors. Shows that an app instance exited; for more detail, look in the application logs.

- `cf logs APP-NAME --recent`: Dumps recent logs. See [Viewing Logs in the Command Line Interface](https://github.com/cloudfoundry/docs/tree/master/docs/cli/log).

- `cf logs APP-NAME`: Returns a real-time stream of the application STDOUT and STDERR. Use Ctrl-C (^C) to exit the real-time stream.

- `cf files APP-NAME`: Lists the files in an application directory. Given a path to a file, outputs the contents of that file. Given a path to a subdirectory, lists the files within. Use this to prepare individual logs.

**Note:** Your application should direct its logs to STDOUT and STDERR. The `cf logs` command also returns messages from any `log4j` facility that you configure to send logs to STDOUT.

### Accessing Apps with SSH

If you need to troubleshoot an instance of an app, you can gain SSH access to the app with the Diego proxy and daemon. See the Diego SSH topic for details on [SSH configuration and procedures](https://github.com/cloudfoundry/docs/tree/master/docs/cli/log).
Application SSH Overview

Page last updated:

This topic introduces SSH configuration for applications in your Elastic Runtime deployment.

If you need to troubleshoot an instance of an app, you can gain SSH access to the app using the SSH proxy and daemon.

For example, one of your app instances may be unresponsive, or the log output from the app may be inconsistent or incomplete. You can SSH into the individual VM that runs the problem instance in order to troubleshoot.

SSH Access Control Hierarchy

Operators, space administrators, and developers can configure SSH access for Elastic Runtime, spaces, and apps as described in this table:

<table>
<thead>
<tr>
<th>User Role</th>
<th>Scope of SSH Permissions Control</th>
<th>How They Define SSH Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Entire deployment</td>
<td>Configure the deployment to allow or prohibit SSH access (one-time)</td>
</tr>
<tr>
<td>Space Administrator</td>
<td>Space</td>
<td>cf CLI allow-space-ssh and disallow-space-ssh commands</td>
</tr>
<tr>
<td>Developer</td>
<td>Application</td>
<td>cf CLI enable-ssh and disable-ssh commands</td>
</tr>
</tbody>
</table>

An application is SSH-accessible only if operators, space administrators, and developers all grant SSH access at their respective levels. For example, the image below shows a deployment where:

- An operator allowed SSH access at the deployment level.
- A space administrator allowed SSH access for applications running in spaces “A” and “B” but not “C.”
- A developer enabled SSH access for applications that include “Foo,” “Bar,” and “Baz.”

As a result, apps “Foo,” “Bar,” and “Baz” accept SSH requests.

SSH Access for Apps and Spaces
Administrators and application developers can configure SSH access from the command line. The cf CLI also includes commands to return the value of the SSH access setting. See the Accessing Apps with Diego SSH topic to use and configure SSH at both the application level and the space level.

Configuring SSH Access for Elastic Runtime

Pivotal Cloud Foundry deployments control SSH access to apps at the Elastic Runtime level. Additionally, Cloud Foundry supports load balancing of SSH sessions with your load balancer. The Configuring SSH Access topic describes how to set SSH access for your deployment.

Understanding SSH Access

The SSH system components include the SSH proxy and daemon, and the system also supports authentication, and load balancing of incoming SSH traffic. The Understanding SSH topic provides a conceptual overview.
Accessing Apps with SSH

Page last updated:

This page assumes you are using cf CLI v6.13.0 or later.

The cf CLI lets you securely log into remote host VMs running Elastic Runtime application instances. This topic describes the commands that enable SSH access to applications, and enable, disable, and check permissions for such access.

Under the hood, the cf CLI looks up the \texttt{app_ssh_oauth_client} identifier in the Cloud Controller \texttt{/v2/info} endpoint, and uses this identifier to query the UAA server for an SSH authorization code. On the target VM side, the SSH proxy contacts the Cloud Controller via the \texttt{app_ssh_endpoint} listed in \texttt{/v2/info} to confirm permission for SSH access.

### Application SSH Commands

<table>
<thead>
<tr>
<th>cf CLI Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{cf enable-ssh}</td>
<td>Enable and Disable SSH Access</td>
</tr>
<tr>
<td>\texttt{cf disable-ssh}</td>
<td>Enable and Disable SSH Access</td>
</tr>
<tr>
<td>\texttt{cf allow-space-ssh}</td>
<td>Enable and Disable SSH Access</td>
</tr>
<tr>
<td>\texttt{cf disallow-space-ssh}</td>
<td>Enable and Disable SSH Access</td>
</tr>
<tr>
<td>\texttt{cf ssh-enabled}</td>
<td>Check SSH Access Permissions</td>
</tr>
<tr>
<td>\texttt{cf space-ssh-allowed}</td>
<td>Check SSH Access Permissions</td>
</tr>
<tr>
<td>\texttt{cf ssh}</td>
<td>Securely log into an application container.</td>
</tr>
<tr>
<td>\texttt{cf ssh-code}</td>
<td>Enable secure login to an application container using non-CF SSH tools like ssh, scp, and sftp.</td>
</tr>
</tbody>
</table>

### Enabling and Disabling SSH Access

A cloud operator can deploy Elastic Runtime to either allow or prohibit Application SSH across the entire deployment.

Within a deployment that permits SSH access to applications, developers can enable or disable SSH access to individual applications. Space administrators can blanket allow or disallow SSH access to all apps running within a space.

### Configuring SSH Access at the Application Level

\texttt{cf enable-ssh} enables SSH access to all instances of an app:

```
$ cf enable-ssh MY-AWESOME-APP
```

\texttt{cf disable-ssh} disables SSH access to all instances of an app:

```
$ cf disable-ssh MY-AWESOME-APP
```

### Configuring SSH Access at the Space Level

\texttt{cf allow-space-ssh} allows SSH access into all apps in a space:

```
$ cf allow-space-ssh SPACE-NAME
```

\texttt{cf disallow-space-ssh} disallows SSH access into all apps in a space:

```
$ cf disallow-space-ssh SPACE-NAME
```
Checking SSH Permissions

**cf ssh-enabled** checks whether an app is accessible with SSH:

```
cf ssh-enabled MY-AWESOME-APP
ssh support is disabled for 'MY-AWESOME-APP'
```

**cf space-ssh-allowed** checks whether all apps running within a space are accessible with SSH:

```
cf space-ssh-allowed SPACE-NAME
ssh support is enabled in space 'SPACE-NAME'
```

Logging Into an Application Container with cf SSH

If SSH access is allowed at the deployment, space, and application level, you can run `cf ssh APP-NAME` from the cf CLI to start an interactive SSH session with a VM hosting an application. By default, it accesses the container running first instance of the application, the instance with index 0.

```
$ cf ssh MY-AWESOME-APP
```

Common cf SSH Flags

You can tailor **cf ssh** commands with the following flags, most of which mimic flags for the Unix/Linux `ssh` command. See `cf ssh --help` for more details.

- The `-i` flag targets a specific instance of an application. To log into the VM container hosting the third instance (index=2) of MY-AWESOME-APP, run:

  ```
  $ cf ssh MY-AWESOME-APP -i 2
  ```

- The `-L` flag enables local port forwarding, binding an output port on your machine to an input port on the application VM. Pass in a local port, and your application VM port and port number, all colon delimited. Optionally, you can also prepend your local network interface, or it defaults to `localhost`.

  ```
  ```

- The `-N` flag skips returning a command prompt on the remote machine. This sets up local port forwarding if you do not need to execute commands on the host VM.

- The `-t`, `-tt`, and `-T` flags let you run an SSH session in pseudo-tty mode rather than generate terminal line output.

SSH Session Environment

If you want the environment of your interactive SSH session to match the environment of your buildpack-based app, with the same environment variables and working directory, run the following after starting the session:

```
export HOME=/home/vcap/app
export TMPDIR=/home/vcap/tmp
cd /home/vcap/app
```

Before running the next command, check the contents of the files in both the `/home/vcap/app/profile` and `/home/vcap/app/profile.d` directories to make sure they will not perform any actions that are undesirable for your running app. The `/profile.d` directory contains buildpack-specific initialization tasks, and the `profile` file contains application-specific initialization tasks.

```
source /home/vcap/app/profile.d/*.sh
source /home/vcap/app/profile
```

If the `profile` and `profile.d` scripts do alter your instance in undesirable ways, you should use discretion in running only the commands from them that you need for environmental setup.
Note also that even after running the above commands, the value of the `VCAP_APPLICATION` environment variable will differ slightly from its value in the environment of the app process, as it will not have the `host`, `instance_id`, `instance_index`, or `port` fields set. These fields are available in other environment variables, as described in the `VCAP_APPLICATION` documentation.

Application SSH Access without cf CLI

In addition to `cf ssh`, you can use other SSH clients such as `ssh`, `scp`, or `sftp` to access your application, as long as you have SSH permissions.

To securely connect to an application instance, you log in with a specially-formed username that passes information to the SSH proxy running on the host VM. For the password, you use a one-time SSH authorization code generated by `cf ssh-code`. Here is the full procedure:

1. Run `cf app MY-AWESOME-APP --guid` and record the GUID of your target app.

```
$ cf app MY-AWESOME-APP --guid
d0a2e11d-e6ca-4120-b32d-140c356906a5
```

2. Query the `/v2/info` endpoint of your deployment’s Cloud Controller. Record the domain name and port of the `app_ssh_endpoint` field. Also note the `app_ssh_host_key_fingerprint` field, which you will compare with the fingerprint returned by the SSH proxy on your target VM.

```
$ cf curl /v2/info
...
    "app_ssh_endpoint": "ssh.MY-DOMAIN.com:2222",
    "app_ssh_host_key_fingerprint": "a6:14:c0:ea:42:07:b2:7f:53:2c:0b:66:e0:00:21:6c",
...
```

3. Run `cf ssh-code` to obtain a one-time authorization code that substitutes for an SSH password. Or you can run `cf ssh-code | pbcopy` to automatically copy the code to the clipboard.

```
$ cf ssh-code
E1x89n
```

4. Run your `ssh` or other command to connect to the application instance. For the username, use a string of the form `cf:APP-GUID/APP-INSTANCE-INDEX@SSH-ENDPOINT`, where `APP-GUID` and `SSH-ENDPOINT` come from the previous steps. For the port number, pass in the `SSH-PORT` also recorded above. `APP-INSTANCE-INDEX` is the index of the instance you want to access.

```
With the above example, you `ssh` into the container hosting the first instance of your app by running:

$ ssh -p 2222 cf:d0a2e11d-e6ca-4120-b32d-140c356906a5/0@ssh.MY-DOMAIN.com
```

Or you can use `scp` to transfer files by running:

```
Or you can use `scp` to transfer files by running:

$ scp -P 2222 -o User=cf:d0a2e11d-e6ca-4120-b32d-140c356906a5@ssh.MY-DOMAIN.com:REMOTE-FILE-TO-RETRIEVE LOCAL-FILE-DESTINATION
```

5. When the SSH proxy reports its RSA fingerprint, confirm that it matches the `app_ssh_host_key_fingerprint` recorded above. When prompted for a password, paste in the authorization code that `cf ssh-code` returned.

```
$ ssh -p 2222 cf:d0a2e11d-e6ca-4120-b32d-140c356906a5@ssh.MY-DOMAIN.com
The authenticity of host 'ssh.MY-DOMAIN.com [203.0.113.5] (2222)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ssh.MY-DOMAIN.com [203.0.113.5] (2222)' (RSA) to the list of known hosts.
cf:d0a2e11d-e6ca-4120-b32d-140c356906a5@ssh.MY-DOMAIN.com's password:
vcap@ce4l5164kws:$
```

6. That’s it. You’re in!

Proxy to Container Authentication

A second layer of SSH security runs within each container. When the SSH proxy attempts to handshake with the SSH daemon inside the target container, it uses the following fields associated with the `diego-ssh` key in its route to the application instance. This inner layer works invisibly and requires no user

© Copyright Pivotal Software Inc, 2013-2018

508
action, but is described here to complete the SSH security picture.

CONTAINER_PORT (required)

The `container_port` indicates which port inside the container the SSH daemon is listening on. The proxy attempts to connect to host side mapping of this port after authenticating the client.

HOST_FINGERPRINT (optional)

When present, the `host_fingerprint` declares the expected fingerprint of the SSH daemon's host public key. When the fingerprint of the actual target's host key does not match the expected fingerprint, the connection is terminated. The fingerprint should only contain the hex string generated by `ssh-keygen -l`.

USER (optional)

The `user` declares the user ID to use during authentication with the container's SSH daemon. While this is not a required part of the routing data, it is required for password authentication and may be required for public key authentication.

PASSWORD (optional)

The `password` declares the password to use during password authentication with the container's ssh daemon.

PRIVATE_KEY (optional)

The `private_key` declares the private key to use when authenticating with the container's SSH daemon. If present, the key must be a PEM encoded RSA or DSA public key.

Example Application Process

```json
{
  "process_guid": "ssh-process-guid",
  "domain": "ssh-experiments",
  "rootfs": "preloaded:cflinuxfs2",
  "instances": 1,
  "start_timeout": 30,
  "setup": {
    "download": {
      "artifact": "diego-sshd",
      "from": "http://file-server.service.cf.internal.example.com:8080/v1/static/diego-sshd/diego-sshd.tgz",
      "to": "/tmp",
      "cache_key": "diego-sshd"
    }
  },
  "action": {
    "run": {
      "path": "/tmp/diego-sshd",
      "args": {
        "-address=0.0.0.0:2222",
        "-authorizedKey=ssh-rsa...
      },
      "env": [],
      "resource_limits": {}
    }
  },
  "ports": [2222],
  "routes": {
    "diego-sshd": {
      "container_port": 2222,
      "private_key": "PEM encoded PKCS#1 private key"
    }
  }
}
```
Daemon discovery

To be accessible via the SSH proxy, containers must host an SSH daemon, expose it via a mapped port, and advertise the port in a `diego-ssh` route. If a proxy cannot find the target process or a route, user authentication fails.

```json
"routes": {
  "diego-ssh": { "container_port": 2222 }
}
```

The Diego system generates the appropriate process definitions for Elastic Runtime applications which reflect the policies that are in effect.
Accessing Services with SSH

This page assumes you are using cf CLI v6.15.0 or later.

This topic describes how to gain direct command line access to your deployed service instance. For example, you may need access to your database to execute raw SQL commands to edit the schema, import and export data, or debug application data issues.

To establish direct command line access to a service, you deploy a host app and utilize its SSH and port forwarding features to communicate with the service instance through the app container. The technique outlined below works with any TCP service, such as MySQL or Redis.

Create a Service Instance

In your terminal window, log in to your deployment with cf login.

1. List the marketplace services installed as product tiles on your Pivotal Cloud Foundry (PCF) Ops Manager. See the Adding and Deleting Products topic if you need to add the service as a tile. In this example, we create a p-mysql service instance.

   ```
cf marketplace
   p-mysql 100mb MySQL databases on demand
   ```

2. Create your service instance. As part of the create-service command, indicate the service name, the service plan, and the name you choose for your service instance.

   ```
cf create-service p-mysql 100mb MY-DB
   ```

Push Your Host App

To push an app that will act as the host for the SSH tunnel, push any app that will successfully deploy to Elastic Runtime.

1. Push your app.

   ```
cf push YOUR-HOST-APP
   ```

2. Enable SSH for your app.

   ```
cf enable-ssh YOUR-HOST-APP
   ```

   Note: In order to enable SSH access to your app, SSH access must also be enabled for both the space that contains the app and Elastic Runtime. See the Application SSH Overview topic for further details.

Create Your Service Key

To establish SSH access to your service instance, you need to create a service key that contains critical information for configuring your SSH tunnel.

1. Create a service key for your service instance using the cf create-service-key command.

   ```
cf create-service-key MY-DB EXTERNAL-ACCESS-KEY
   ```

2. Retrieve your new service key using the cf service-key command.
Configure Your SSH Tunnel

Configure an SSH tunnel to your service instance using `cf ssh`. Tailor the example command below with information from your service key.

```
cf ssh -L 63306:us-cdbr-iron-east-01.p-mysql.net:3306 YOUR-HOST-APP
```

- Use any available local port for port forwarding. For example, `63306`.
- Replace `us-cdbr-iron-east-01.p-mysql.net` with the address provided under `hostname` in the service key retrieved above.
- Replace `3306` with the port provided under `port` above.
- Replace `YOUR-HOST-APP` with the name of your host app.

After you enter the command, open another terminal window and perform the steps below in Access Your Service Instance.

Access Your Service Instance

To establish direct command-line access to your service instance, use the relevant command line tool for that service. This example uses the MySQL command line client to access the p-mysql service instance.

```
mysql -u b5136c448be920 -h 0 -p -D ad_b2fca649704585d -P 63306
```

- Replace `b5136c448be920` with the username provided under `username` in your service key.
- `-h 0` indicates to `mysql` to connect to your local machine.
- `-p` indicates to `mysql` to prompt for a password. When prompted, use the password provided under `password` in your service key.
- Replace `ad_b2fca649704585d` with the database name provided under `name` in your service key.
- `-P 63306` indicates to `mysql` to connect on port 63306.
Trusted System Certificates

Page last updated:

The Cloud Foundry Administrator can deploy a set of trusted system certificates to be made available in Linux-based application instances running on the Diego backend. Such instances include buildpack-based apps using the cflinuxfs2 stack and Docker-image-based apps. If the administrator has configured these certificates, they will be available inside the instance containers as files with extension `.crt` in the read-only `/etc/cf-system-certificates` directory. For cflinuxfs2-based apps, these certificates will also be installed directly in the `/etc/ssl/certs` directory, and so will be available automatically to libraries such as `openssl` that respect that trust store.
Cloud Controller API Client Libraries

This topic describes the client libraries available for developers who want to consume the Cloud Controller API (CAPI).

Overview

CAPI is the entry point for most operations within the Cloud Foundry (CF) platform. You can use it to manage orgs, spaces, and apps, which includes user roles and permissions. You can also use CAPI to manage the services provided by your CF deployment, including provisioning, creating, and binding them to apps.

For more information, see the [CAPI documentation](https://www.pivotal.io/)

Client Libraries

While you can develop apps that consume CAPI by calling it directly as in the API documentation, you may want to use an existing client library. See the available client libraries below.

Supported

CF currently supports the following clients for CAPI:

- [Java](https://www.pivotal.io/)
- [Scripting](https://www.pivotal.io/) with the Cloud Foundry Command Line Interface (cf CLI)

Experimental

The following client is experimental and a work in progress:

- [Golang](https://www.pivotal.io/)

Unofficial

CF does not support the following clients, but they may be supported by third-parties:

- [Golang](https://www.pivotal.io/)
- [Golang](https://www.pivotal.io/)
- [Node.js](https://www.pivotal.io/)
Delivering Service Credentials to an Application

Page last updated:

This topic describes binding applications to service instances for the purpose of generating credentials and delivering them to applications. For an overview of services, and documentation on other service management operations, see Using Services. If you are interested in building Services for Cloud Foundry and making them available to end users, see the Custom Services documentation.

Bind a Service Instance

Binding a service instance to your application triggers credentials to be provisioned for the service instance and delivered to the application runtime in the VCAP_SERVICES environment variable. For details on consuming these credentials with your application, see Using Bound Service Instances.

Not all services support binding, as some services deliver value to users directly without integration with an application. In many cases binding credentials are unique to an application, and another app bound to the same service instance would receive different credentials; however this depends on the service.

$ cf bind-service my-app mydb
Binding service mydb to my-app in org my-org / space test as me@example.com... OK
TIP: Use 'cf push' to ensure your env variable changes take effect

$ cf restart my-app

Note: You must restart or in some cases re-push your application for changes to be applied to the VCAP_SERVICES environment variable and for the application to recognize these changes.

Arbitrary Parameters

Arbitrary parameters require cf CLI v6.12.1+

Some services support additional configuration parameters with the bind request. These parameters are passed in a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file. For a list of supported configuration parameters, see documentation for the particular service offering.

$ cf bind-service rails-sample my-db -c '{"role":"read-only"}'
Binding service my-db to app rails-sample in org console / space development as user@example.com... OK

$ cf bind-service rails-sample my-db -c /tmp/config.json
Binding service my-db to app rails-sample in org console / space development as user@example.com... OK

Binding with Application Manifest

As an alternative to binding a service instance after pushing an application, you can use the application manifest to bind the service instance during push. As of cf CLI v6.12.1, Arbitrary Parameters are not supported in application manifests.

The following excerpt from an application manifest would bind a service instance called test-mysql-01 to the application on push.

```yaml
services:
  - test-mysql-01
```

The following excerpt from the cf push command and response demonstrates that the cf CLI reads the manifest and binds the service instance to an app called test-mysql-app.
Using Bound Service Instances

Once you have a service instance created and bound to your application, you need to configure the application to dynamically fetch the credentials for your service instance. The `VCAP_SERVICES` environment variable contains credentials and additional metadata for all bound service instances. There are two methods developers can leverage to have their applications consume binding credentials.

- **Parse the JSON yourself**: See the documentation for `VCAP_SERVICES`. Helper libraries are available for some frameworks.
- **Auto-configuration**: Some buildpacks create a service connection for you by creating additional environment variables, updating config files, or passing system parameters to the JVM.

For details on consuming credentials specific to your development framework, refer to the Service Binding section in the documentation for your framework's buildpack.

Update Service Credentials

To update your service credentials, perform the following steps:

1. **Unbind the service instance** using the credentials you are updating with the following command:

   ```bash
   $ cf unbind-service YOUR-APP YOUR-SERVICE-INSTANCE
   ```

2. **Bind the service instance** with the following command. This adds your credentials to the `VCAP_SERVICES` environment variable.

   ```bash
   $ cf bind-service YOUR-APP YOUR-SERVICE-INSTANCE
   ```

3. Restart or re-push the application bound to the service instance so that the application recognizes your environment variable updates.

Unbind a Service Instance

Unbinding a service removes the credentials created for your application from the `VCAP_SERVICES` environment variable.

```bash
$ cf unbind-service my-app mydb
Unbinding app my-app from service mydb in org my-org / space test as me@example.com...
OK
```

**Note:** You must restart or in some cases re-push your application for changes to be applied to the `VCAP_SERVICES` environment variable and for the application to recognize these changes.
Managing Service Instances with the CLI

This topic describes lifecycle operations for service instances, including creating, updating, and deleting. For an overview of services, and documentation on other service management operations, see Using Services CF. If you are interested in building Services for Cloud Foundry and making them available to end users, see the Custom Services CF documentation.

List Marketplace Services

After targeting and logging into Cloud Foundry, you can view what services are available to your targeted organization with the command cf marketplace.

Available services may differ between organizations and between Cloud Foundry marketplaces.

Creating Service Instances

You can create a service instance with the command:

cf create-service SERVICE PLAN

SERVICE
The service you choose.

PLAN
Service plans are a way for providers to offer varying levels of resources or features for the same service.

SERVICE_INSTANCE
A name you provide for your service instance. This is an alias for the instance which is meaningful to you. Use any series of alphanumeric characters, hyphens (-), and underscores (_). You can rename the instance at any time.

Arbitrary Parameters

Arbitrary parameters require cf CLI v6.12.1+

Some services support additional configuration parameters with the provision request. These parameters are passed in a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file. For a list of supported configuration parameters, see documentation for the particular service offering.

Note: User Provided Service Instances provide a way for developers to bind applications with services that are not available in their Cloud Foundry marketplace.
Instance Tags

*Instance tags require cf CLI v6.12.1+*

Some services provide a list of tags that Cloud Foundry delivers in the `VCAP_SERVICES` Environment Variable. These tags provide developers with a more generic way for applications to parse `VCAP_SERVICES` for credentials. Developers may provide their own tags when creating a service instance by including a comma-separated list of tags with the `-t` flag.

```bash
$ cf create-service my-db-service small-plan my-db -t "prod, workers"
Creating service my-db in org console / space development as user@example.com...
OK
```

List Service Instances

You can list the service instances in your targeted space with the command `cf services`. The output includes any bound apps, along with the state of the last requested operation for the service instance.

```bash
$ cf services
Getting services in org my-org / space test as user@example.com...
OK

<table>
<thead>
<tr>
<th>name</th>
<th>service</th>
<th>plan</th>
<th>bound apps</th>
<th>last operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mybucket</td>
<td>p-riakcs</td>
<td></td>
<td>developer</td>
<td>myapp</td>
</tr>
<tr>
<td>mydb</td>
<td>p-mysql</td>
<td>100mb</td>
<td></td>
<td>create succeeded</td>
</tr>
</tbody>
</table>
```

Get Details for a Particular Service Instance

Details include dashboard urls, if applicable, and operation start and last updated timestamps.

```bash
$ cf service mydb
Service instance: mydb
Service: p-mysql
Plan: 100mb
Description: MySQL databases on demand
Documentation url: https://p-mysql.example.com/manage/instances/cc4eab9d-aff4-4beb-bc46-123f2a02dcf1

Last Operation
Status: create succeeded
Updated: 2015-05-18T22:01:26Z
```

Bind a Service Instance

Depending on the service, you can bind service instances to applications and/or routes.

Not all services support binding, as some services deliver value to users directly without integration with Cloud Foundry, such as SaaS applications.

Bind a Service Instance to an Application

Depending on the service, binding a service instance to your application may deliver credentials for the service instance to the application. See the [Delivering Service Credentials to an Application](http://docs.cloudfoundry.org/guides/delivering-service-credentials.html) topic for more information. Binding a service instance to an application may also trigger application logs to be streamed to the service instance. For more information, see [Streaming Application Logs to Log Management Services](http://docs.cloudfoundry.org/guides/streaming-app-logs.html).
Binding with Application Manifest

As an alternative to binding a service instance to an application after pushing an application, you can use the application manifest to bind the service instance during push. As of cf CLI v6.12.1, Arbitrary Parameters are not supported in application manifests. Using the manifest to bind service instances to routes is also not supported.

The following excerpt from an application manifest binds a service instance called `test-mysql-01` to the application on push.

```
services:
  - test-mysql-01
```

The following excerpt from the `cf push` command and response demonstrates that the cf CLI reads the manifest and binds the service instance to an app called `test-msg-app`.

```
$ cf push
Using manifest file /Users/Bob/test-apps/test-msg-app/manifest.yml

...  

Binding service test-mysql-01 to test-msg-app in org My-Org / space development as Bob@example.com OK

For more information about application manifests, see Deploying with Application Manifests.

Bind a Service Instance to a Route

Binding a service instance to a route will cause application requests and responses to be proxied through the service instance, where it may be used to transform or intermediate requests. For more information, see Manage Application Requests with Route Services.

```
$ cf bind-route-service shared-domain.example.com --hostname my-app my-service-instance
Binding route my-app.shared-domain.example.com to service instance my-service-instance in org my-org / space test as me@example.com... OK
```

Restaging your application is not required.

Arbitrary Parameters

Arbitrary parameters require cf CLI v6.12.1+

Some services support additional configuration parameters with the bind request. These parameters are passed in a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file. For a list of supported configuration parameters, see documentation for the particular service offering.

```
$ cf bind-service rails-sample my-db -c '{"role":"read-only"}'
Binding service my-db to app rails-sample in org console / space development as user@example.com... OK
```

```
$ cf bind-service rails-sample my-db -c /tmp/config.json
Binding service my-db to app rails-sample in org console / space development as user@example.com... OK
```
Unbind a Service Instance

Unbind a Service Instance from an Application

Unbinding a service instance from an application removes the credentials created for your application from the `VCAP_SERVICES` environment variable.

```
cf unbind-service my-app mydb
Unbinding app my-app from service mydb in org my-org / space test as me@example.com...
OK
```

**Note:** You must restart or in some cases re-push your application for changes to be applied to the `VCAP_SERVICES` environment variable and for the application to recognize these changes.

Unbind a Service Instance from a Route

Unbinding a service instance from a route will result in requests and responses no longer being proxied through the service instance. For more information, see Manage Application Requests with Route Services.

```
cf unbind-route-service shared-domain.example.com --hostname my-app my-service-instance
Unbinding may leave apps mapped to route my-app.shared-domain.example.com vulnerable; e.g. if service instance my-service-instance provides authentication. Do you want to proceed? [y]...
Unbinding route my-app.shared-domain.example.com from service instance my-service-instance in org my-org / space test as me@example.com...
OK
```

Restaging your application is not required.

Rename a Service Instance

You can change the name given to a service instance. Keep in mind that upon restarting any bound applications, the name of the instance will change in the `VCAP_SERVICES` environment variable. If your application depends on the instance name for discovering credentials, changing the name could break your application’s use of the service instance.

```
cf rename-service mydb mydb1
Renaming service mydb to mydb1 in org my-org / space test as me@example.com...
OK
```

Update a Service Instance

Upgrade/Downgrade Service Plan

Changing a plan requires cf CLI v6.7+ and cf-release v192+

By updating the service plan for an instance, users can effectively upgrade and downgrade their service instance to other service plans. Though the platform and CLI now support this feature, services must expressly implement support for it so not all services will. Further, a service might support updating between some plans but not others. For instance, a service might support updating a plan where only a logical change is required, but not where data migration is necessary. In either case, users can expect to see a meaningful error when plan update is not supported.

```
cf update-service mydb -p new-plan
Updating service instance mydb as me@example.com...
OK
```

© Copyright Pivotal Software Inc, 2013-2018
Arbitrary Parameters

Arbitrary parameters require cf CLI v6.12.1+

Some services support additional configuration parameters with the update request. These parameters are passed in a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file. For a list of supported configuration parameters, see documentation for the particular service offering.

```
$ cf update-service mydb -c "{"storage_gb":4}"
Updating service instance mydb as me@example.com...

$ cf update-service mydb -c /tmp/config.json
Updating service instance mydb as me@example.com...
```

Instance Tags

Instance tags require cf CLI v6.12.1+

Some services provide a list of tags that Cloud Foundry delivers in the `VCAP_SERVICES` Environment Variable. These tags provide developers with a more generic way for applications to parse `VCAP_SERVICES` for credentials. Developers may provide their own tags when creating a service instance by including a comma-separated list of tags with the `-t` flag.

```
$ cf update-service my-db -t "staging, web"
Updating service my-db in org console / space development as user@example.com...
OK
```

Delete a Service Instance

Deleting a service instance deprovisions the service instance and deletes all data associated with the service instance.

```
$ cf delete-service mydb
Are you sure you want to delete the service mydb ? y
Deleting service mydb in org my-org / space test as me@example.com...
OK
```
Managing Service Keys

This topic describes managing service instance credentials manually with service keys. You can use service keys to connect to a service instance from a local client, or from an app that is not deployed by Cloud Foundry. Create, list, retrieve, and delete service keys using the Cloud Foundry Command Line Interface (cf CLI).

Note: Not all services support service keys. Some services support credentials through application binding only.

Create a Service Key

To generate credentials for a service instance, use the `cf create-service-key` command:

```
$ cf create-service-key MY-SERVICE MY-KEY
Creating service key MY-KEY for service instance MY as me@example.com...
OK
```

Use the `-c` flag to provide service-specific configuration parameters in a valid JSON object, either in-line or in a file.

To provide the JSON object in-line, use the following format:

```
$ cf create-service-key MY-SERVICE MY-KEY -c "{"permissions":"read-only"}"
Creating service key MY-KEY for service instance MY-SERVICE as me@example.com...
OK
```

To provide the JSON object as a file, give the absolute or relative path to your JSON file:

```
$ cf create-service-key MY-SERVICE MY-KEY -c PATH-TO-JSON-FILE
Creating service key MY-KEY for service instance MY-SERVICE as me@example.com...
OK
```

List Service Keys for a Service Instance

To list service keys for a service instance, use the `cf service-keys` command:

```
$ cf service-keys MY-SERVICE
Getting service keys for service instance MY-SERVICE as me@example.com...

name
mykey1
mykey2
```

Get Credentials for a Service Key

To retrieve credentials for a service key, use the `cf service-key` command:

```
$ cf service-key MY-SERVICE MY-KEY
Getting key MY-KEY for service instance MY-SERVICE as me@example.com...

{
  uri: foo://user2:pass2@example.com/mydb,
  servicename: mydb
}
```

Use the `--guid` flag to display the API GUID for the service key:
Delete Service Key

To delete a service key, use the `cf delete-service-key` command:

```bash
cf delete-service-key MY-SERVICE MY-KEY
```

Are you sure you want to delete the service key MY-KEY? y

Deleting service key MY-KEY for service instance MY-SERVICE as me@example.com...

OK

Add option `cf` to force deletion without confirmation.

```bash
cf delete-service-key -f MY-SERVICE MY-KEY
```

Deleting service key MY-KEY for service instance MY-SERVICE as me@example.com...

OK
User-Provided Service Instances

User-provided service instances enable developers to use services that are not available in the marketplace with their applications running on Cloud Foundry.

User-provided service instances can be used to deliver service credentials to an application, and/or to trigger streaming of application logs to a syslog compatible consumer. These two functions can be used alone or at the same time.

Once created, user-provided service instances behave like service instances created through the marketplace; see Managing Service Instances and Application Binding for details on listing, renaming, deleting, binding, and unbinding.

Create a User-Provided Service Instance

The alias for `cf create-user-provided-service` is `cf cups`.

Deliver Service Credentials to an Application

Suppose a developer obtains a URL, port, username, and password for communicating with an Oracle database managed outside of Cloud Foundry. The developer could manually create custom environment variables to configure their application with these credentials (of course you would never hard code these credentials in your application!).

User-provided service instances enable developers to configure their applications with these using the familiar Application Binding operation and the same application runtime environment variable used by Cloud Foundry to automatically deliver credentials for marketplace services (`VCAP_SERVICES`).

```
cf cups SERVICE_INSTANCE -p "{'username":"admin","password":"pa55woRD'"
```

To create a service instance in interactive mode, use the `-p` option with a comma-separated list of parameter names. The cf CLI will prompt you for each parameter value.

```
cf cups SERVICE_INSTANCE -p "host, port, dbname, username, password"
```

Once the user-provided service instance is created, to deliver the credentials to one or more applications see Application Binding.

Stream Application Logs to a Service

User-provided service instances enable developers to stream applications logs to a syslog compatible aggregation or analytics service that isn't available in the marketplace. For more information on the syslog protocol see RFC 5424 and RFC 6587.

Create the user-provided service instance, specifying the URL of the service with the `-l` option.

```
cf cups SERVICE_INSTANCE -l syslog://example.log-aggregator.com
```

To stream application logs to the service, bind the user-provided service instance to your app.

Proxy Application Requests to a Route Service

User-provided service instances enable developers to proxy application requests to `route services` for preprocessing. To create a user-provided service instance for a route service, specify the url for the route service using the `-r` option.

```
$ cf create-user-provided-service my-user-provided-route-service -r https://my-route-service.example.com
Creating user-provided service my-user-provided-route-service in org my-org / space my-space as user@example.com... OK
```

**Note:** When creating the user-provided service, the route service url specified must be https.
In order to proxy requests to the user-provided route service, you will need to bind the service instance to the route. For more information, see Manage Application Requests with Route Services.

Update a User-provided Service Instance

You can use `cf update-user-provided-service` to update the attributes of an instance of a user-provided service. New credentials overwrite old credentials, and parameters not provided are deleted.

The alias for `update-user-provided-service` is `uups`.
Streaming Application Logs to Log Management Services

This topic describes how to drain logs from Cloud Foundry to a third party log management service.

Cloud Foundry aggregates logs for all instances of your applications as well as for requests made to your applications through internal components of Cloud Foundry. For example, when the Cloud Foundry Router forwards a request to an application, the Router records that event in the log stream for that app. Run the following command to access the log stream for an app in the terminal:

```
$ cf logs YOUR-APP-NAME
```

If you want to persist more than the limited amount of logging information that Cloud Foundry can buffer, drain these logs to a log management service. For more information about the systems responsible for log aggregation and streaming in Cloud Foundry, see Application Logging in Cloud Foundry.

Using Services from the Cloud Foundry Marketplace

Your Cloud Foundry marketplace may offer one or more log management services. To use one of these services, create an instance of the service and bind it to your application with the following commands:

```
$ cf create-service SERVICE PLAN SERVICE-INSTANCE
$ cf bind-service YOUR-APP YOUR-LOG-STORE
```

For more information on service instance lifecycle management, see Managing Service Instances.

Using Services Not Available in your Marketplace

If a compatible log management service is not available in your Cloud Foundry marketplace, you can use User-provided Service Instances to stream application logs to a service of your choice.

Your service may require some preparation before application logs can be streamed to it from Cloud Foundry. For specific instructions for several popular services, see Service-Specific Instructions for Streaming Application Logs. If you cannot find instructions for your service, follow the generic instructions below.

Step 1: Configure the Log Management Service

Complete the following steps to set up a communication channel between the log management service and your Cloud Foundry deployment:

1. Obtain the external IP addresses that your Cloud Foundry administrator assigns to outbound traffic.
2. Provide these IP addresses to the log management service. The specific steps to configure a third-party log management service depend on the service.
3. Whitelist these IP addresses to ensure unrestricted log routing to your log management service.
4. Record the syslog URL provided by the third-party service. Third-party services typically provide a syslog URL to use as an endpoint for incoming log data. You use this syslog URL in Step 2: Create a User-provided Service Instance.

Cloud Foundry uses the syslog URL to route messages to the service. The syslog URL has a scheme of syslog, syslog-tls, or https, and can include a port number. For example:

```
syslog://logs.example.com:1234
```

Note: Elastic Runtime does not support using syslog-tls with self-signed certificates. If you are running your own syslog server and want to...
Step 2: Create a User-provided Service Instance

Create a user-provided service instance using the `cf` CLI `create-user-provided-service` command with the `-l` flag and the syslog URL that you obtained in Step 1: Configure the Log Management Service. The `-l` flag configures the syslog drain.

```
cf create-user-provided-service SERVICE-INSTANCE -l SYSLOG-URL
```

Refer to User-Provided Service Instances for more information.

Step 3: Bind the Service Instance

You have two options for binding the service instance to an application:

- Run `cf push` with a manifest. The services block in the manifest must specify the service instance that you want to bind.
- Run `cf bind-service`

```
cf bind-service YOUR-APP-NAME SERVICE-INSTANCE
```

After a short delay, logs begin to flow automatically. Refer to Managing Service Instances with the CLI for more information.

Step 4: Verify Logs are Draining

To verify that logs are draining correctly to a third-party log management service:

1. Take actions that produce log messages, such as making requests of your app.
2. Compare the logs displayed in the CLI against those displayed by the log management service.

For example, if your application serves web pages, you can send HTTP requests to the application. In Cloud Foundry, these generate Router log messages, which you can view in the CLI. Your third-party log management service should display corresponding messages.

**Note:** For security reasons, Cloud Foundry applications do not respond to `ping`. You cannot use `ping` to generate log entries.
Service-Specific Instructions for Streaming Application Logs

This topic provides instructions for configuring some third-party log management services.

Once you have configured a service, refer to the Third-Party Log Management Services topic for instructions on binding your application to the service.

Logit.io

From your Logit.io dashboard:

1. Identify the Logit stack you want to use.
2. Click Logstash Configuration.
3. Note your Logstash Endpoint.
4. Note your TCP or UDP Port (not the syslog port).
5. Create the log drain service in Cloud Foundry.
   
   ```
   cf cups logit-drain -l syslog://ENDPOINT:PORT
   ```
6. Bind the service to an app.
   
   ```
   cf bind-service YOUR-CF-APP-NAME logit-drain
   ```
7. Restage or push the app using one of the following commands:
   
   ```
   cf restage YOUR-CF-APP-NAME
   cf push YOUR-CF-APP-NAME
   ```

   After a short delay, logs begin to appear in Kibana.

Papertrail

From your Papertrail account:

1. Click Add System.
2. Click the Other link.
3. Select I use Cloud Foundry, enter a name, and click Save.
4. Record the URL with port that is displayed after creating the system.

CloudFoundry will log to logs.papertrailapp.com:36129.

5. Create the log drain service in Cloud Foundry.

```bash
$ cf cups my-logs -l syslog-tls://logs.papertrailapp.com:PORT
```

6. Bind the service to an app.

```bash
$ cf bind-service APPLICATION-NAME my-logs
```

7. Restage the app.

```bash
$ cf restage APPLICATION-NAME
```

After a short delay, logs begin to flow automatically.

8. Once Papertrail starts receiving log entries, the view automatically updates to the logs viewing page.
Splunk Storm

From your Splunk Storm account:

1. Click Add project.

2. Enter the project details.

3. Create a new input for Network data.

4. Manually enter the external IP addresses your Cloud Foundry administrator assigns to outbound traffic.

5. Note the host and port provided for TCP input.

6. Create the log drain service in Cloud Foundry using the displayed TCP host and port.

7. Bind the service to an app.
8. Restage the app

```
$ cf restage APPLICATION-NAME
```

After a short delay, logs begin to flow automatically.

9. Wait for some events to appear, then click **Data Summary**.

![Data Summary](image)

10. Click the **loggregator** link to view all incoming log entries from Cloud Foundry.

![loggregator](image)

**SumoLogic**

- **Note:** SumoLogic uses HTTPS for communication. HTTPS is supported in Cloud Foundry v158 and above.

From your SumoLogic account:

1. Click the **Add Collector** link.

2. Choose **Hosted Collector** and fill in the details.
3. In the new collector's row of the collectors view, click the Add Source link.

4. Select HTTP source and fill in the details. Note that you'll be provided an HTTPS url

5. Once the source is created, a URL should be displayed. You can also view the URL by clicking the Show URL link beside the created source.

6. Create the log drain service in Cloud Foundry using the displayed URL.

   $ cf cups my-logs -d HTTPS-SOURCE-URL

7. Bind the service to an app.

   $ cf bind-service APPLICATION-NAME my-logs

8. Restage the app.

   $ cf restage APPLICATION-NAME

After a short delay, logs begin to flow automatically.
9. In the SumoLogic dashboard, click Manage, then click Status to see a view of log messages received over time.

![SumoLogic Status](image)

10. In the SumoLogic dashboard, click on Search. Place the cursor in the search box, then press Enter to submit an empty search query.

![SumoLogic Search](image)

---

**Logsene**

*Note: Logsene uses HTTPS for communication. HTTPS is supported in Cloud Foundry v158 and above.*

From your Sematext account:

1. Click the Create App / Logsene App menu item. Enter a name and click the Add Application button to create the Logsene App.

2. Create the log drain service in Cloud Foundry using the displayed URL.

   ```
   $ cf cups logsene-log-drain -l https://logsene-cf-receiver.sematext.com/YOUR_LOGSENE_TOKEN
   ```

3. Bind the log drain to an app. You could optionally bind multiple apps to one log drain.

   ```
   $ cf bind-service YOUR-CF-APP-NAME logsene-log-drain
   ```

4. Restage the app.

   ```
   $ cf restage APPLICATION-NAME
   ```

   After a short delay, logs begin to flow automatically and appear in the Logsene UI.

---

**Logentries is Not Supported**
Cloud Foundry distributes log messages over multiple servers in order to handle load. Currently, we do not recommend using Logentries as it does not support multiple syslog sources.
Streaming Application Logs to Splunk

Page last updated:

To integrate Cloud Foundry with Splunk Enterprise, complete the following process.

1. Create a Cloud Foundry Syslog Drain for Splunk

In Cloud Foundry, create a syslog drain user-provided service instance as described in Using Third-Party Log Management Services.

Choose one or more applications whose logs you want to drain to Splunk through the service.

Bind each app to the service instance and restart the app.

Note the GUID for each app, the IP address of the Loggregator host, and the port number for the service. Locate the port number in the syslog URL. For example:

```
syslog://logs.example.com:1234
```

2. Prepare Splunk for Cloud Foundry

For detailed information about the following tasks, see the Splunk documentation.

Install the RFC5424 Syslog Technology Add-On

The Cloud Foundry Loggregator component formats logs according to the Syslog Protocol defined in RFC 5424. Splunk does not parse log fields according to this protocol. To allow Splunk to correctly parse RFC 5424 log fields, install the Splunk RFC5424 Syslog Technical Add-On.

Patch the RFC5424 Syslog Technology Add-On

1. SSH into the Splunk VM

2. Replace `/opt/splunk/etc/apps/rfc5424/default/transforms.conf` with a new `transforms.conf` file that consists of the following text:

   ```
   [rfc5424_host]
   DEST_KEY = MetaData:Host
   REGEX = <\d+>\d{1}\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+
   FORMAT = host::$1
   
   [rfc5424_header]
   REGEX = <\d+>\d{1}\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+
   MV_ADD = true
   ```

   3. Restart Splunk

Create a TCP Syslog Data Input

Create a TCP Syslog Data Input in Splunk, with the following settings:

- **TCP port** is the port number you assigned to your log drain service
- **Set sourcetype** is [Manual](#)
- **Source type** is rfc5424_syslog (type this value into text field)
- **Index** is the index you created for your log drain service

Your Cloud Foundry syslog drain service is now integrated with Splunk.
3. Verify that Integration was Successful

Use Splunk to execute a query of the form:

```
sourcetype=rfc5424_syslog index=<the_index_you_created> appname=<app_guid>
```

To view logs from all apps at once, you can omit the `appname` field.

Verify that results rows contain the three Cloud Foundry-specific fields:

- **appname** — the GUID for the Cloud Foundry application
- **host** — the IP address of the Loggregator host
- **procid** — the Cloud Foundry component emitting the log

If the Cloud Foundry-specific fields appear in the log search results, integration is successful.

If logs from an app are missing, make sure that:

- The app is bound to the service and was restarted after binding
- The service port number matches the TCP port number in Splunk
Streaming Application Logs with Fluentd

Fluentd is an open source collector that allows you to implement unified logging layers. With Fluentd, you can stream application logs to different backends or services like Elasticsearch, HDFS and Amazon S3. This topic explains how to integrate Fluentd with Cloud Foundry applications.

Step 1: Create a Cloud Foundry Syslog Drain for Fluentd

1. In Cloud Foundry, create a syslog drain user-provided service instance as described in Using Third-Party Log Management Services.

2. Choose one or more applications whose logs you want to drain to Fluentd through the service.

3. Bind each app to the service instance, and restart the app.

4. Note the GUID for each app, the IP address of the Loggregator host, and the port number for the service.

5. Locate the port number in the syslog URL. For example:

   ```
syslog://logs.example.com:5140
   ```

Step 2: Set up Fluentd for Cloud Foundry

This section assumes you have an active Fluentd instance running. If you do not have an active Fluentd instance, refer to the Fluentd Documentation/Install steps for more details.

Fluentd comes with native support for syslog protocol. To set up Fluentd for Cloud Foundry, configure the syslog input of Fluentd as follows.

1. In your main Fluentd configuration file, add the following source entry:

   ```
   <source>
   @type syslog
   port 5140
   bind 0.0.0.0
   tag cf.app
   protocol_type udp
   </source>
   ```

2. Restart the Fluentd service.

   ```
   Note: The Fluentd syslog input plugin supports udp and tcp options. Make sure to use the same transport that Cloud Foundry is using.
   ```

   Fluentd will start listening for Syslog message on port 5140 and tagging the messages with `cf.app`, which can be used later for data routing. For more details about the full setup for the service, refer to the Config File article.

   If your goal is to use an Elasticsearch or Amazon S3 backend, read the following guide: http://www.fluentd.org/guides/recipes/elasticsearch-and-s3

© Copyright Pivotal Software Inc, 2013-2018
Configuring Play Framework Service Connections

Page last updated:

Cloud Foundry provides support for connecting a Play Framework application to services such as MySQL, and Postgres. In many cases, a Play Framework application running on Cloud Foundry can automatically detect and configure connections to services.

Auto-Configuration

By default, Cloud Foundry will detect service connections in a Play Framework application and configure them to use the credentials provided in the Cloud Foundry environment. Auto-configuration will only happen if there is a single service of any of the supported types - MySQL or Postgres.
Migrating a Database in Cloud Foundry

Page last updated:

Application development and maintenance often requires changing a database schema, known as migrating the database. This topic describes three ways to migrate a database on Cloud Foundry.

Migrate Once

This method executes SQL commands directly on the database, bypassing Cloud Foundry. This is the fastest option for a single migration. However, this method is less efficient for multiple migrations because it requires manually accessing the database every time.

Note: Use this method if you expect your database migration to take longer than the timeout that `cf push` applies to your application. The timeout defaults to 60 seconds, but you can extend it up to 180 seconds with the `-t` command line option.

1. Run `cf env` and obtain your database credentials by searching in the `VCAP_SERVICES` environment variable:

   ```
   $ cf env db-app
   Getting env variables for app my-db-app in org My-Org / space development as admin...
   OK
   System-Provided:
   
   "VCAP_SERVICES": {"example-db-n/a": [{"name": "test-777", "label": "example-db-n", "tags": ["mysql", "relational"], "plan": "basic", "credentials": {"jdbcUrl": "jdbc:mysql://aa11:2b@cdbr-05.example.net:3306/ad_01", "uri": "mysql://aa11:2b@cdbr-05.example.net:3306/ad_01?reconnect=true", "name": "ad_01", "hostname": "cdbr-05.example.net", "port": "1234", "username": "aa11", "password": "2b" }}],
   }
   ```

2. Connect to the database using your database credentials.

3. Migrate the database using SQL commands.

4. Update the application using `cf push`.

Migrate Occasionally

This method requires you to:

- Create a schema migration command or script.
- Run the migration command when deploying a single instance of the application.
- Re-deploy the application with the original start command and number of instances.

This method is efficient for occasional use because you can re-use the schema migration command or script.

1. Create a schema migration command or SQL script to migrate the database. For example:

   ```rake db:migrate```

2. Deploy a single instance of your application with the database migration command as the start command. For example:

   ```cf push APP -c 'rake db:migrate' -i 1```
3. Deploy your application again with the normal start command and desired number of instances. For example:

```
cf push APP -c 'null' -i 4
```

**Note:** This example assumes that the normal start command for your application is the one provided by the buildpack, which the `-c 'null'` option forces Cloud Foundry to use.

---

**Migrate Frequently**

This method uses an idempotent script to partially automate migrations. The script runs on the first application instance only.

This option takes the most effort to implement, but becomes more efficient with frequent migrations.

1. Create a script that:
   - Examines the `instance_index` of the `VCAP_APPLICATION` environment variable. The first deployed instance of an application always has an `instance_index` of "0". For example, this code uses Ruby to extract the `instance_index` from `VCAP_APPLICATION`:
     ```ruby
     instance_index = JSON.parse(ENV['VCAP_APPLICATION'])['instance_index']
     ```
   - Determines whether or not the `instance_index` is "0".
   - If and only if the `instance_index` is "0", runs a script or uses an existing command to migrate the database. The script or command must be idempotent.

2. Create a manifest that provides:
   - The application name
   - The `command` attribute with a value of the schema migration script chained with a start command.

   Example partial manifest:

   ```yaml
   ---
   applications:
   - name: my-rails-app
     command: bundle exec rake cf:on_first_instance db:migrate && bundle exec rails s -p $PORT -e $RAILS_ENV
   ```

3. Update the application using `cf push`.

For an example of the migrate frequently method used with Rails, see Running Rake Tasks.
Buildpacks

Page last updated:

Buildpacks provide framework and runtime support for your applications. Buildpacks typically examine user-provided artifacts to determine what dependencies to download and how to configure applications to communicate with bound services.

When you push an application, Cloud Foundry automatically detects which buildpack is required and installs it on the Diego cell or Droplet Execution Agent (DEA) where the application needs to run.

**Note:** Cloud Foundry deployments often have limited access to dependencies. This limitation occurs when the deployment is behind a firewall, or when administrators want to use local mirrors and proxies. In these circumstances, Cloud Foundry provides a [Buildpack Packager](#) application.

System Buildpacks

Cloud Foundry includes a set of system buildpacks for common languages and frameworks. This table lists the system buildpacks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Supported Languages, Frameworks, and Technologies</th>
<th>GitHub Repo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Grails, Play, Spring, or any other JVM-based language or framework</td>
<td><a href="#">Java source</a></td>
</tr>
<tr>
<td>Ruby</td>
<td>Ruby, JRuby, Rack, Rails, or Sinatra</td>
<td><a href="#">Ruby source</a></td>
</tr>
<tr>
<td>Node.js</td>
<td>Node or JavaScript</td>
<td><a href="#">Node.js source</a></td>
</tr>
<tr>
<td>Binary</td>
<td>n/a</td>
<td><a href="#">Binary source</a></td>
</tr>
<tr>
<td>Go</td>
<td>Go</td>
<td><a href="#">Go source</a></td>
</tr>
<tr>
<td>PHP</td>
<td>Cake, Symfony, Zend, Nginx, or HTTPD</td>
<td><a href="#">PHP source</a></td>
</tr>
<tr>
<td>Python</td>
<td>Django or Flask</td>
<td><a href="#">Python source</a></td>
</tr>
<tr>
<td>Staticfile</td>
<td>HTML, CSS, JavaScript, or Nginx</td>
<td><a href="#">Staticfile source</a></td>
</tr>
</tbody>
</table>

Using, Developing, and Customizing Buildpacks

For general information about using buildpacks, see [Using Buildpacks](#).

For general information about customizing existing buildpacks and developing new buildpacks, see [Developing Buildpacks](#).

© Copyright Pivotal Software Inc, 2013-2018
Java Buildpack

Page last updated:

Use the Java buildpack with applications written in Grails, Play, Spring, or any other JVM-based language or framework.

See the following topics:

- Getting Started Deploying Grails Apps
- Getting Started Deploying Ratpack Apps
- Getting Started Deploying Spring Apps
- Tips for Java Developers
- Configuring Service Connections for Grails
- Configuring Service Connections for Play
- Configuring Service Connections for Spring
- Cloud Foundry Eclipse Plugin
- Cloud Foundry Java Client Library
- Build Tool Integration
- BOSH Configured Custom Trusted Certificate Support

You can find the source for the buildpack on GitHub: [https://github.com/cloudfoundry/java-buildpack](https://github.com/cloudfoundry/java-buildpack)

Buildpack Logging and Application Logging

The buildpack only runs during the staging process, and therefore only logs staging information such as the downloaded components, configuration data, and work performed on your application by the buildpack.

The Java buildpack source documentation states the following:

- The Java buildpack logs all messages, regardless of severity, to `APP-DIRECTORY/.java-buildpack.log`. The buildpack also logs messages to `Stderr`, filtered by a configured severity level.
- If the buildpack fails with an exception, the exception message is logged with a log level of `ERROR`. The exception stack trace is logged with a log level of `DEBUG`. This prevents users from seeing stack traces by default.

Once staging completes, the buildpack stops logging. The Loggregator handles application logging.

Your application must write to STDOUT or STDERR for its logs to be included in the Loggregator stream. For more information, see the Application Logging in Cloud Foundry topic.
Getting Started Deploying Grails Apps

Page last updated:

This guide is intended to walk you through deploying a Grails app to Elastic Runtime. If you experience a problem following the steps below, check the Known Issues topic or refer to the Troubleshooting Application Deployment and Health topic.

Sample App Step
If you want to go through this tutorial using the sample app, run `git clone https://github.com/cloudfoundry-samples/pong_matcher_grails.git` to clone the app from GitHub, and follow the instructions in the Sample App Step sections.

Note: Ensure that your Grails app runs locally before continuing with this procedure.

Deploy a Grails Application

This section describes how to deploy a Grails application to Elastic Runtime.

Prerequisites
- A Grails app that runs locally on your workstation
- Intermediate to advanced Grails knowledge
- The Cloud Foundry Command Line Interface (cf CLI)
- JDK 1.7 or 1.8 for Java 7 or 8 configured on your workstation

Note: You can develop Grails applications in Groovy, Java 7 or 8, or any JVM language. The Cloud Foundry Java buildpack uses JDK 1.8, but you can modify the buildpack and the manifest for your app to compile to JDK 1.7. Refer to Step 8: Configure the Deployment Manifest.

Step 1: Declare App Dependencies

Declare all the dependency tasks for your app in the build script of your chosen build tool. The table lists build script information for Gradle, Grails, and Maven and provides documentation links for each build tool.

<table>
<thead>
<tr>
<th>Build Tool</th>
<th>Build Script</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradle</td>
<td><code>build.gradle</code></td>
<td>Gradle User Guide</td>
</tr>
<tr>
<td>Grails</td>
<td><code>BuildConfig.groovy</code></td>
<td>Grails: Configuration - Reference Documentation</td>
</tr>
<tr>
<td>Maven</td>
<td><code>pom.xml</code></td>
<td>Apache Maven Project Documentation</td>
</tr>
</tbody>
</table>

Sample App Step
You can skip this step. The `pong_matcher_grails/app/grails-app/conf/BuildConfig.groovy` file contains the dependencies for the sample app, as the example below shows.

```groovy
dependencies {
    // specify dependencies here under either 'build', 'compile', 'runtime', 'test' or 'provided' scopes e.g.
    // runtime 'mysql:mysql-connector-java:5.1.29'
    // runtime 'org.postgresql:postgresql:9.3-1101-jdbc41'
    // test 'org.grails.grails-datasources-test-support:1.0-grails-2.4'
    runtime 'mysql:mysql-connector-java:5.1.33'
}
```

Step 2: Allocate Sufficient Memory

© Copyright Pivotal Software Inc, 2013-2018
Use the `cf push -m` command to specify the amount of memory that should be allocated to the application. Memory allocated this way is done in preset amounts of 64M, 128M, 256M, 512M, 1G, or 2G. For example:

```
$ cf push -m 128M
```

When your app is running, you can use the `cf app APP_NAME` command to see memory utilization.

**Sample App Step**

You can skip this step. In the `manifest.yml` of the `pong_matcher_grails` sample app, the `memory` sub-block of the `applications` block allocates 1 GB to the app.

**Step 3: Provide a JDBC Driver**

The Java buildpack does not bundle a JDBC driver with your application. If your application accesses a SQL RDBMS, you must do the following:

- Include the appropriate driver in your application.
- Create a dependency task for the driver in the build script for your build tool or IDE.

**Sample App Step**

You can skip this step. The `pong_matcher_grails` sample app declares a MySQL JDBC driver in the `pong_matcher_grails/app/grails-app/conf/DataSource.groovy` file because the app uses ClearDB, which is a database-as-service for MySQL-powered apps. The example below shows this declaration.

```groovy
dataSource {
    pooled = true
    jmxExport = true
    driverClassName = "com.mysql.jdbc.Driver"
    dialect = org.hibernate.dialect.MySQL5InnoDBDialect
    url = new URI(System.env.DATABASE_URL ?: "mysql://foo:bar@localhost")
    username = uri.userInfo ? uri.userInfo.split(':')[0]: ''
    password = uri.userInfo ? uri.userInfo.split(':')[1]: ''
    url = "jdbc:mysql://" + uri.host + uri.path

    properties {
        dbProperties {
            autoReconnect = true
        }
    }
}
```

**Step 4: (Optional) Configure a Procfile**

Use a Procfile to declare required runtime processes for your web app and to specify your web server. For more information, see the Configuring a Production Server topic.

**Sample App Step**

You can skip this step. The `pong_matcher_grails` app does not require a Procfile.

**Step 5: Create and Bind a Service Instance for a Grails Application**

This section describes using the CLI to configure a ClearDB managed service instance for an app. You can use either the CLI or the Apps Manager to perform this task.

Elastic Runtime supports two types of service instances:
Managed services integrate with Elastic Runtime through service brokers that offer services and plans and manage the service calls between Elastic Runtime and a service provider.

User-provided service instances enable you to connect your application to pre-provisioned external service instances.

For more information about creating and using service instances, refer to the Services Overview topic.

Create a Service Instance

Run `cf marketplace` to view managed and user-provided services and plans available to you.

The example shows two of the available managed database-as-a-service providers and their offered plans: `cleardb` database-as-a-service for MySQL-powered apps and `elephantsql` PostgreSQL as a Service.

```
$ cf marketplace
Getting services from marketplace in org Cloud-Apps / space development as clouduser@example.com... OK

<table>
<thead>
<tr>
<th>service</th>
<th>plans</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardb</td>
<td>spark, boost, amp</td>
<td>Highly available MySQL for your apps</td>
</tr>
<tr>
<td>elephantsql</td>
<td>turtle, panda, elephant</td>
<td>PostgreSQL as a Service</td>
</tr>
</tbody>
</table>
```

Run `cf create-service SERVICE PLAN SERVICE_INSTANCE` to create a service instance for your app. Choose a SERVICE and PLAN from the list, and provide a unique name for the SERVICE_INSTANCE.

```
$ cf create-service cleardb spark mysql
Creating service mysql in org Cloud-Apps / space development as clouduser@example.com... OK
```

Bind a Service Instance

When you bind an app to a service instance, Elastic Runtime writes information about the service instance to the VCAP_SERVICES app environment variable. The app can use this information to integrate with the service instance.

Most services support bindable service instances. Refer to your service provider’s documentation to confirm if they support this functionality.

You can bind a service to an application with the command `cf bind-service APPLICATION SERVICE_INSTANCE`.

Alternately, you can configure the deployment manifest file by adding a `services` sub-block to the `applications` block and specifying the service instance.

For more information and an example on service binding using a manifest, see the Sample App step.

You can also bind a service using the Apps Manager.

```
... applications:
```
Step 6: Configure the Deployment Manifest

You can specify deployment options in the `manifest.yml` that the `cf push` command uses when deploying your app.

Refer to the Deploying with Application Manifests topic for more information.

Sample App Step
You can skip this step. The `manifest.yml` file for the `pong_matcher_grails` sample app does not require any additional configuration to deploy the app.

Step 7: Log in and Target the API Endpoint

Run `cf login -a API_ENDPOINT`, enter your login credentials, and select a space and org. The API endpoint is the URL of the Cloud Controller in your Elastic Runtime instance.

Sample App Step
You must do this step to run the sample app.

Step 8: Deploy the Application

**Note:** You must use the cf CLI to deploy apps.

From the root directory of your application, run `cf push APP-NAME -p PATH-TO-FILE.war` to deploy your application.

**Note:** You must deploy the `.war` artifact for a Grails app, and you must include the path to the `.war` file in the `cf push` command using the `-p` option if you do not declare the path in the `applications` block of the manifest file. For more information, refer to the Grails section in the Tips for Java Developers topic.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs APP-NAME`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step
1. Change to the `app` directory, and run `./grailsw to build the app.

2. Run `cf push pong_matcher_grails -n HOST_NAME` to push the app.
The example below shows the terminal output of deploying the `pong_matcher_grails` app. `cf push` uses the instructions in the manifest file to create the app, create and bind the route, and upload the app. It then binds the app to the `mysql` service and follows the instructions in the manifest to start two instances of the app, allocating 1 GB of memory between the instances. After the instances start, the output displays their health and status.

```bash
$ cf push pong_matcher_grails -n my-grails-app
Using manifest file /Users/example/workspace/pong_matcher_grails/app/manifest.yml
Creating app pong_matcher_grails in org Cloud-Apps / space development as clouduser@example.com... OK
Creating route my-grails-app.cfapps.io... OK
Binding my-grails-app.cfapps.io to pong_matcher_grails... OK
Uploading pong_matcher_grails... Uploading app files from: /Users/example/workspace/pong_matcher_grails/app/target/pong_matcher_grails-0.1.war Uploading 4.8M, 704 files OK
Binding service mysql to app pong_matcher_grails in org Cloud-Apps / space development as clouduser@example.com... OK
Starting app pong_matcher_grails in org Cloud-Apps / space development as clouduser@example.com... OK

----> Downloaded app package (38M)
----> Java Buildpack Version: v2.5 / https://github.com/cloudfoundry/java-buildpack.git#840500e
----> Downloading Open Jdk JRE 18.0.25 from https://download.run.pivotal.io/openjdk/86-64/openjdk-1.8.0_25.tar.gz (1.5s)
Expanding Open Jdk JRE to /java-buildpack/open_jdk_jre (1.1s)
----> Downloading Spring Auto Reconfiguration 1.5.0_RELEASE from https://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration-1.5.0_RELEASE.jar (0.0s)
Modifying /WEB-INF/web.xml for Auto Reconfiguration
----> Downloading Tomcat Instance 8.0.14 from https://download.run.pivotal.io/tomcat/tomcat-8.0.14.tar.gz (0.4s)
Expanding Tomcat to /java-buildpack/tomcat (0.1s)
----> Downloading Tomcat Lifecycle Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-lifecycle-support/tomcat-lifecycle-support-2.4.0_RELEASE.jar (0.0s)
----> Downloading Tomcat Logging Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-logging-support/tomcat-logging-support-2.4.0_RELEASE.jar (0.0s)
----> Downloading Tomcat Access Logging Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-access-logging-support/tomcat-access-logging-support-2.4.0_RELEASE.jar (0.0s)

----> Uploading droplet (83M)
0 of 2 instances running, 2 starting
0 of 2 instances running, 2 starting
0 of 2 instances running, 2 starting
2 of 2 instances running

App started

Showing health and status for app pong_matcher_grails in org Cloud-Apps / space development as clouduser@example.com... OK

requested state: started instances: 2/2
usage: 1G x 2 instances
urls: my-grails-app.cfapps.io

   state since   cpu      memory    disk
00 running 2014-11-10 05:07:33 PM 0.0%  866.4M of 1G 153.6M of 1G
01 running 2014-11-10 05:07:36 PM 0.0%  677.2M of 1G 153.6M of 1G
```

Step 9: Test Your Deployed App

You've deployed an app to Elastic Runtime!

Use the cf CLI or the Apps Manager to review information and administer your app and your Elastic Runtime account. For example, you can edit the `manifest.yml` to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

See the Manage Your Application with the cf CLI section for more information. See also Using the Apps Manager.

© Copyright Pivotal Software Inc, 2013-2018
Alternative Methods for Pushing Apps

Integrate a Plugin for Your Build Tool

Elastic Runtime provides plugins for Maven and Gradle. You can deploy and manage your apps using Maven or Gradle command-line syntax and configure security credentials.

For more information, refer to the Build Tool Integration topic.

Manage Your Application with the cf CLI

Run `cf help` to view a complete list of commands, grouped by task categories, and run `cf help COMMAND` for detailed information about a specific command. For more information about using the cf CLI, refer to the Cloud Foundry Command Line Interface (cf CLI) topics, especially the Getting Started with cf CLI v6 topic.

Note: You cannot perform certain tasks in the CLI or the Apps Manager because these are commands that only a Elastic Runtime administrator can run. If you are not a Elastic Runtime administrator, the following message displays for these types of commands:

error code: 10003, message: You are not authorized to perform the requested action

For more information about specific Admin commands you can perform with the Apps Manager, depending on your user role, refer to the Getting Started with the Apps Manager topic.

Troubleshooting

If your application fails to start, verify that the application starts in your local environment. Refer to the Troubleshooting Application Deployment and
Health topic to learn more about troubleshooting.

App Deploy Fails

Even when the deploy fails, the app might exist on Elastic Runtime. Run `cf apps` to review the apps in the currently targeted org and space. You might be able to correct the issue using the CLI or the Apps Manager, or you might have to delete the app and redeploy.

App Requires a Unique URL

Elastic Runtime requires that each app that you deploy have a unique URL. Otherwise, the new app URL collides with an existing app URL and Elastic Runtime cannot successfully deploy the app. You can fix this issue by running `cf push` with either of the following flags to create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words. Using this option might create a long URL, depending on the number of words that the app name includes.
Getting Started Deploying Ratpack Apps

This guide is intended to walk you through deploying a Ratpack app to Elastic Runtime. If you experience a problem following the steps below, check the Known Issues topic or refer to the Troubleshooting Application Deployment and Health topic.

Sample App Step
If you want to go through this tutorial using the sample app, run `git clone https://github.com/cloudfoundry-samples/pong_matcher_groovy.git` to clone the app from GitHub, and follow the instructions in the Sample App Step sections.

Note: Ensure that your Ratpack app runs locally before continuing with this procedure.

Deploy a Ratpack Application

This section describes how to deploy a Ratpack application to Elastic Runtime.

Prerequisites

- A Ratpack app that runs locally on your workstation
- Intermediate to advanced Ratpack knowledge
- The Cloud Foundry Command Line Interface (cf CLI)
- JDK 1.7 or 1.8 for Java 7 or 8 configured on your workstation

Note: You can develop Ratpack applications in Java 7 or 8 or any JVM language. The Cloud Foundry Java buildpack uses JDK 1.8, but you can modify the buildpack and the manifest for your app to compile to JDK 1.7. Refer to Step 8: Configure the Deployment Manifest.

Step 1: Declare App Dependencies

Declare all the dependency tasks for your app in the build script of your chosen build tool. The table lists build script information for Gradle and Maven and provides documentation links for each build tool.

<table>
<thead>
<tr>
<th>Build Tool</th>
<th>Build Script</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradle</td>
<td>build.gradle</td>
<td>Gradle User Guide</td>
</tr>
<tr>
<td>Maven</td>
<td>pom.xml</td>
<td>Apache Maven Project Documentation</td>
</tr>
</tbody>
</table>

Sample App Step
You can skip this step. The `build.gradle` file contains the dependencies for the `pong_matcher_groovy` sample app, as the example below shows.

```java
dependencies {
    // SpringLoaded enables runtime hot reloading.
    // It is not part of the app runtime and is not shipped in the distribution.
    springloaded "org.springframework:springloaded:1.2.0.RELEASE"

    // Default SLF4J binding. Note that this is a blocking implementation.
    // See here for a non blocking adderpper http://logging.apache.org/log4j/2.x/manual/async.html
    runtime "org.slf4j.slf4j-simple:1.7.7"

    compile group: 'redis.clients', name: 'jedis', version: '2.5.2', transitive: true

    testCompile "org.spockframework:spock-core:0.7-groovy-2.0"
}
```
Step 2: Allocate Sufficient Memory

Use the `cf push -m` command to specify the amount of memory that should be allocated to the application. Memory allocated this way is done in preset amounts of 64M, 128M, 256M, 512M, 1G, or 2G. For example:

5 cf push -m 128M

When your app is running, you can use the `cf app APP_NAME` command to see memory utilization.

Sample App Step
You can skip this step. In the `manifest.yml` of the `pong_matcher_groovy` sample app, the `memory` sub-block of the `applications` block allocates 512 MB to the app.

Step 3: Provide a JDBC Driver

The Java buildpack does not bundle a JDBC driver with your application. If your application accesses a SQL RDBMS, you must do the following:

- Include the appropriate driver in your application.
- Create a dependency task for the driver in the build script for your build tool or IDE.

Sample App Step
You can skip this step. The `pong_matcher_groovy` sample app does not require a JDBC driver.

Step 4: (Optional) Configure a Procfile

Use a Procfile to declare required runtime processes for your web app and to specify your web server. For more information, see the Configuring a Production Server topic.

Sample App Step
You can skip this step. The `pong_matcher_groovy` app does not require a Procfile.

Step 5: Create and Bind a Service Instance for a Ratpack Application

This section describes using the CLI to configure a Redis managed service instance for an app. You can use either the CLI or the Apps Manager to perform this task.

Elastic Runtime supports two types of service instances:

- Managed services integrate with Elastic Runtime through service brokers that offer services and plans and manage the service calls between Elastic Runtime and a service provider.
- User-provided service instances enable you to connect your application to pre-provisioned external service instances.

For more information about creating and using service instances, refer to the Services Overview topic.

Create a Service Instance

Run `cf marketplace` to view managed and user-provided services and plans available to you.

The example shows two of the available managed database-as-a-service providers and their offered plans: elephantsql PostgreSQL as a Service and rediscloud Enterprise-Class Redis for Developers.

© Copyright Pivotal Software Inc, 2013-2018
$ cf marketplace
Getting services from marketplace in org Cloud-Apps / space development as clouduser@example.com...
OK

<table>
<thead>
<tr>
<th>service</th>
<th>plans</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>elephantsql</td>
<td>turtle, panda, elephant</td>
<td>PostgreSQL as a Service</td>
</tr>
<tr>
<td>rediscloud</td>
<td>30mb, 100mb, 1gb, 10gb, 50gb</td>
<td>Enterprise-Class Redis for Developers</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Run `cf create-service SERVICE PLAN SERVICE_INSTANCE` to create a service instance for your app. Choose a SERVICE and PLAN from the list, and provide a unique name for the SERVICE_INSTANCE.

**Sample App Step**
Run `cf create-service rediscloud 30mb baby-redis` to create a service instance named `baby-redis` that uses the `rediscloud` service and the `30mb` plan, as the example below shows.

$ cf create-service rediscloud 30mb baby-redis
Creating service baby-redis in org Cloud-Apps / space development as clouduser@example.com....
OK

**Bind a Service Instance**
When you bind an app to a service instance, Elastic Runtime writes information about the service instance to the VCAP_SERVICES app environment variable. The app can use this information to integrate with the service instance.

Most services support bindable service instances. Refer to your service provider’s documentation to confirm if they support this functionality.

You can bind a service to an application with the command `cf bind-service APPLICATION SERVICE_INSTANCE`.

Alternately, you can configure the deployment manifest file by adding a `services` sub-block to the `applications` block and specifying the service instance. For more information and an example on service binding using a manifest, see the Sample App step.

You can also bind a service using the Apps Manager.

**Sample App Step**
You can skip this step because the service instance is already bound. Open the `manifest.yml` file in a text editor to view the bound service instance information. Locate the file in the app root directory and search for the `services` sub-block in the `applications` block, as the example below shows.

```yaml
applications:
  ...
  services:
    - baby-redis
```

**Step 6: Configure the Deployment Manifest**
You can specify deployment options in the `manifest.yml` that the `cf push` command uses when deploying your app.

Refer to the Deploying with Application Manifests topic for more information.

**Sample App Step**
You can skip this step. The `manifest.yml` file for the `pong_matcher_groovy` sample app does not require any additional configuration to deploy the
Step 7: Log in and Target the API Endpoint

Run `cf login -a API_ENDPOINT`, enter your login credentials, and select a space and org. The API endpoint is the URL of the Cloud Controller in your Elastic Runtime instance.

**Sample App Step**
You must do this step to run the sample app.

Step 8: Deploy the Application

**Note:** You must use the cf CLI to deploy apps.

From the root directory of your application, run `cf push APP-NAME -p PATH-TO-FILE.distZip` to deploy your application.

**Note:** You must deploy the `distZip` artifact for a Ratpack app, and you must include the path to the `distZip` file in the `cf push` command using the `-p` option if you do not declare the path in the `applications` block of the manifest file. For more information, refer to the Tips for Java Developers topic.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs APP-NAME`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

**Sample App Step**
1. Change to the `app` directory, and run `./gradlew distZip` to build the app.
2. Run `cf push pong_matcher_groovy -a HOST_NAME` to push the app.

**Example:**
```bash
cf push pong_matcher_groovy -a groovy-ratpack-app
```

**Note:** You do not have to include the `-p` flag when you deploy the sample app. The sample app manifest declares the path to the archive that `cf push` uses to upload the app files.

The example below shows the terminal output of deploying the `pong_matcher_groovy` app. `cf push` uses the instructions in the manifest file to create the app, create and bind the route, and upload the app. It then binds the app to the `baby-redis` service and follows the instructions in the manifest to start one instance of the app with 512 MB. After the app starts, the output displays the health and status of the app.
Step 9: Test Your Deployed App

You’ve deployed an app to Elastic Runtime!

Use the cf CLI or the [Apps Manager ➤](#) to review information and administer your app and your Elastic Runtime account. For example, you can edit the `manifest.yml` to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

See the [Manage Your Application with the cf CLI](#) section for more information. See also [Using the Apps Manager ➤](#).

---

**Sample App Step**

To test the sample app, do the following:

1. To export the test host, run `export HOST=YOUR_APP_URL`, substituting the URL for your app for `SAMPLE_APP_URL`.

2. To clear the database from any previous tests, run:

   ```bash
   curl -v -X DELETE $HOST/all
   ```

   You should get a response of 200.

3. To request a match as “andrew”, run:

   ```bash
   curl -v -H "Content-Type: application/json" -X PUT $HOST/match_requests/firstrequest -d '{"player": "andrew"}'
   ```

   You should again get a response of 200.

4. To request a match as a different player, run:

   ```bash
   curl -v -H "Content-Type: application/json" -X PUT $HOST/match_requests/secondrequest -d '{"player": "navratilova"}"
   ```

© Copyright Pivotal Software Inc, 2013-2018

554 1.7
Alternative Methods for Pushing Apps

Integrate a Plugin for Your Build Tool

Elastic Runtime provides plugins for Maven and Gradle. You can deploy and manage your apps using Maven or Gradle command-line syntax and configure security credentials.

For more information, refer to the Build Tool Integration topic.

Manage Your Application with the cf CLI

Run `cf help` to view a complete list of commands, grouped by task categories, and run `cf help COMMAND` for detailed information about a specific command. For more information about using the cf CLI, refer to the Cloud Foundry Command Line Interface (cf CLI) topics, especially the Getting Started with cf CLI v6 topic.

Note: You cannot perform certain tasks in the CLI or the Apps Manager because these are commands that only an Elastic Runtime administrator can run. If you are not an Elastic Runtime administrator, the following message displays for these types of commands:

```
error code: 10003, message: You are not authorized to perform the requested action
```
For more information about specific Admin commands you can perform with the Apps Manager, depending on your user role, refer to the Getting Started with the Apps Manager topic.

Troubleshooting

If your application fails to start, verify that the application starts in your local environment. Refer to the Troubleshooting Application Deployment and Health topic to learn more about troubleshooting.

App Deploy Fails

Even when the deploy fails, the app might exist on Elastic Runtime. Run `cf apps` to review the apps in the currently targeted org and space. You might be able to correct the issue using the CLI or the Apps Manager, or you might have to delete the app and redeploy.

App Requires a Unique URL

Elastic Runtime requires that each app you deploy have a unique URL. Otherwise, the new app URL collides with an existing app URL and Elastic Runtime cannot successfully deploy the app. You can fix this issue by running `cf push` with either of the following flags to create a unique URL:

- `--hostname` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words. Using this option might create a long URL, depending on the number of words that the app name includes.
Getting Started Deploying Spring Apps

Page last updated:

This guide is intended to walk you through deploying a Spring app to Elastic Runtime. You can choose whether to push a sample app, your own app, or both.

If at any time you experience a problem following the steps below, try checking the Known Issues topic, or refer to the Troubleshooting Application Deployment and Health topic for more tips.

Sample App Step
If you want to go through this tutorial using the sample app, run `git clone https://github.com/cloudfoundry-samples/pong_matcher_spring` to clone the `pong_matcher_spring` app from GitHub, and follow the instructions in the Sample App Step sections.

Note: Ensure that your Spring app runs locally before continuing with this procedure.

Deploy a Spring Application

This section describes how to deploy your Spring application to Elastic Runtime.

Prerequisites

- A Spring app that runs locally on your workstation
- Intermediate to advanced Spring knowledge
- The Cloud Foundry Command Line Interface (cf CLI)
- JDK 1.6, 1.7, or 1.8 for Java 6, 7, or 8 configured on your workstation

Note: The Cloud Foundry Java buildpack uses JDK 1.8, but you can modify the buildpack and the manifest for your app to compile to an earlier version. For more information, refer to the Custom Buildpacks topic.

Step 1: Declare App Dependencies

Be sure to declare all the dependency tasks for your app in the build script of your chosen build tool.

The Spring Getting Started Guides demonstrate features and functionality you can add to your app, such as consuming RESTful services or integrating data. These guides contain Gradle and Maven build script examples with dependencies. You can copy the code for the dependencies into your build script.

The table lists build script information for Gradle and Maven and provides documentation links for each build tool.

<table>
<thead>
<tr>
<th>Build Tool</th>
<th>Build Script</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradle</td>
<td>build.gradle</td>
<td>Gradle User Guide</td>
</tr>
<tr>
<td>Maven</td>
<td>pom.xml</td>
<td>Apache Maven Project Documentation</td>
</tr>
</tbody>
</table>

Sample App Step
You can skip this step. The `pom.xml` file contains the dependencies for the `pong_matcher_spring` sample app, as the example below shows.
Step 2: Allocate Sufficient Memory

Use the `cf push -m` command to specify the amount of memory that should be allocated to the application. Memory allocated this way is done in preset amounts of 64M, 128M, 256M, 512M, or 1G, or 2G. For example:

5 cf push -m 128M

When your app is running, you can use the `cf app APP-NAME` command to see memory utilization.

Sample App Step
You can skip this step. The Cloud Foundry Java buildpack uses settings declared in the sample app to allocate 1 GB of memory to the app.

Step 3: Provide a JDBC Driver

The Java buildpack does not bundle a JDBC driver with your application. If your application accesses a SQL RDBMS, you must do the following:

- Include the appropriate driver in your application.
- Create a dependency task for the driver in the build script for your build tool or IDE.

Sample App Step
You can skip this step. In the pong_matcher_spring sample app, the `src/main/resources/application.yml` file declares the JDBC driver, and the `pom.xml` file includes the JDBC driver as a dependency.

Step 4: Configure Service Connections for a Spring App

Elastic Runtime provides extensive support for creating and binding a Spring application to services such as MySQL, PostgreSQL, MongoDB, Redis, and RabbitMQ. For more information about creating and binding a service connection for your app, refer to the Configure Service Connections for Spring.
Sample App Step: Create a Service Instance

Run

```
cf create-service cleardb spark mysql
```

This creates a service instance named `mysql` that uses the `cleardb` service and the `mysql` plan, as the example below shows.

```
$ cf create-service cleardb spark mysql
Creating service mysql in org Cloud-Apps / space development as a user@example.com... OK
```

Sample App Step: Bind a Service Instance

You can skip this step because the service instance is already bound. Open the `manifest.yml` file in a text editor to view the bound service instance information. Locate the file in the app root directory and search for the `services` sub-block in the `applications` block, as the example below shows.

```
---
applications:
  ...
  services:
    - mysql
```

Step 5: Configure the Deployment Manifest

You can specify deployment options in a manifest file `manifest.yml` that the `cf push` command uses when deploying your app.

Refer to the [Deploying with Application Manifests](#) topic for more information.

Sample App Step

You can skip this step. The `manifest.yml` file for the `pong_matcher_spring` sample app does not require any additional configuration to deploy the app.

Step 6: Log in and Target the API Endpoint

Run

```
cf login -a API-ENDPOINT
```

Enter your login credentials, and select a space and org. The API endpoint is the URL of the Cloud Controller in your Elastic Runtime instance.

```
Sample App Step
You must do this step to run the sample app.
```

Step 7: Deploy Your Application

Note: You must use the cf CLI to deploy apps.

From the root directory of your application, run

```
cf push APP-NAME -p PATH-TO-FILE.war
```

to deploy your application.

Note: Most Spring apps include an artifact, such as a `.jar`, `.war`, or `.zip` file. You must include the path to this file in the `cf push` command using the `-p` option if you do not declare the path in the `applications` block of the manifest file. The example shows how to specify a path to the `.war` file for a Spring app. Refer to the [Tips for Java Developers](#) topic for CLI examples for specific build tools, frameworks, and languages that create...
The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs APP-NAME`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

The example below shows the terminal output of deploying the `pong_matcher_spring` app. `cf push` uses the instructions in the manifest file to create the app, create and bind the route, and upload the app. It then binds the app to the `mysql` service and starts one instance of the app with 1 GB of memory. After the app starts, the output displays the health and status of the app.

Sample App Step

1. Run `brew install maven`.
2. Change to the `app` directory, and run `mvn package` to build the app.
3. Run `cf push pong_matcher_spring -n HOSTNAME` to push the app.

Example: `cf push pong_matcher_spring -n my-spring-app`

Note: You do not have to include the `-p` flag when you deploy the sample app. The sample app manifest declares the path to the archive that `cf push` uses to upload the app files.
$ cf push pong_matcher_spring -s spring119
Using manifest file /Users/example/workspace/pong_matcher_spring/manifest.yml
Creating app pong_matcher_spring in org Cloud-Apps / space development as a user@example.com...
OK
Creating route spring119.cfapps.io...
OK
Binding spring119.cfapps.io to pong_matcher_spring...
OK
Uploading pong_matcher_spring...
Uploading app files from: /Users/example/workspace/pong_matcher_spring/target/pong-matcher-spring-1.0.0.BUILD-SNAPSHOT.jar
Uploading 797.5K, 116 files
OK
Binding service mysql to app pong_matcher_spring in org Cloud-Apps / space development as a user@example.com...
OK
Starting app pong_matcher_spring in org Cloud-Apps / space development as a user@example.com...
OK
-----> Downloaded app package (25M)
-----> Downloading Open Jdk JRE 1.8.0_25 from https://download.run.pivotal.io/openjdk/lucid/x86_64/openjdk-1.8.0_25.tar.gz (1.2s)
Expanding Open Jdk JRE to /java-buildpack/open_jdk_jre (1.1s)
-----> Downloading Spring Auto Reconfiguration 1.5.0_RELEASE from https://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration-1.5.0_RELEASE.jar (0.1s)
-----> Uploading droplet (63M)
0 of 1 instances running, 1 starting
1 of 1 instances running
App started
Showing health and status for app pong_matcher_spring in org Cloud-Apps / space development as a user@example.com...
OK
requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: spring119.cfapps.io

<table>
<thead>
<tr>
<th>state</th>
<th>since</th>
<th>cpu</th>
<th>memory</th>
<th>disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>running</td>
<td>2014-11-19 12:29:27 PM</td>
<td>0.0%</td>
<td>553.6M of 1G</td>
<td>127.4M of 1G</td>
</tr>
</tbody>
</table>

Step 8: Test Your Deployed App

You've deployed an app to Elastic Runtime!

Use the cf CLI or the Apps Manager to review information and administer your app and your Elastic Runtime account. For example, you can edit the manifest.yml to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

See the Manage Your Application with the cf CLI section for more information. See also Using the Apps Manager.

Sample App Step

To test the sample app, do the following:

1. To export the test host, run `export HOST=SAMPLE-APP-URL`, substituting the URL for your app for SAMPLE-APP-URL.

2. To clear the database from any previous tests, run:
   
   `curl -v -X DELETE $HOST/all`
   You should get a response of 200.

3. To request a match as "andrew", run:
   
   `curl -v -H "Content-Type: application/json" -X PUT $HOST/match_requests/firstrequest -d '{"player": "andrew"}'}
   You should again get a response of 200.

4. To request a match as a different player, run:
   
   `curl -v -H "Content-Type: application/json" -X PUT $HOST/match_requests/secondrequest -d '{"player": "navratilova"}'}

© Copyright Pivotal Software Inc, 2013-2018
Alternative Methods for Pushing Apps

Alternative Method 1: Integrate a Plugin for Your Build Tool

Elastic Runtime provides plugins for Maven and Gradle. You can deploy and manage your apps using Maven or Gradle command-line syntax and configure security credentials.

For more information, refer to the Build Tool Integration topic.

Alternative Method 2: Integrate the Cloud Foundry Eclipse Plugin for STS

Elastic Runtime provides an Eclipse plugin extension that enables you to deploy and manage Spring applications on a Cloud Foundry instance from the Spring Tool Suite (STS), version 3.0.0 and later. For more information, refer to the Cloud Foundry Eclipse Plugin topic. You must follow the instructions in the Install to STS from the IDE Extensions Tab and Create a Cloud Foundry Server sections before you can deploy and manage your apps with the plugin.

Manage Your Application with the cf CLI

Run cf help to view a complete list of commands, grouped by task categories, and run cf help COMMAND for detailed information about a specific command. For more information about using the cf CLI, refer to the Cloud Foundry Command Line Interface (cf CLI) topics, especially the Getting Started with cf CLI v6 topic.

Note: You cannot perform certain tasks in the CLI or the Apps Manager because these are commands that only a Elastic Runtime administrator can run. If you are not a Elastic Runtime administrator, the following message displays for these types of commands:

```
error code: 10003, message: You are not authorized to perform the requested action
```

For more information about specific Admin commands you can perform with the Apps Manager, depending on your user role, refer to the Getting Started with the Apps Manager topic.

Troubleshooting

If your application fails to start, verify that the application starts in your local environment. Refer to the Troubleshooting Application Deployment and Health topic to learn more about troubleshooting.

App Deploy Fails

Even when the deploy fails, the app might exist on Elastic Runtime. Run cf apps to view the apps in the currently targeted org and space. You might be able to correct the issue using the CLI or the Apps Manager, or you might have to delete the app and redeploy.

App Requires a Content-Type

If you specify a Content-Encoding header of gzip but do not specify a Content-Type within your application, Elastic Runtime might send a Content-Type of
application/x-gzip to the browser. This scenario might cause the deploy to fail if it conflicts with the actual encoded content of your app. To avoid this issue, be sure to explicitly set `Content-Type` within your app.

App Requires a Unique URL

Elastic Runtime requires that each app that you deploy have a unique URL. Otherwise, the new app URL collides with an existing app URL and Elastic Runtime cannot successfully deploy the app. You can fix this issue by running `cf push` with either of the following flags to create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words. Using this option might create a long URL, depending on the number of words that the app name includes.
Tips for Java Developers

Page last updated:

Cloud Foundry can deploy a number of different JVM-based artifact types. For a more detailed explanation of what it supports, see the [Java Buildpack documentation](#).

Java Client Library

The Cloud Foundry Client Library provides a Java API for interacting with a Cloud Foundry instance. This library, `cloudfoundry-client-lib`, is used by the Cloud Foundry Maven plugin, the Cloud Foundry Gradle plugin, the [Cloud Foundry STS integration](#), and other Java-based tools.

For information about using this library, see the [Java Cloud Foundry Library](#) page.

Grails

Grails packages applications into WAR files for deployment into a Servlet container. To build the WAR file and deploy it, run the following:

```
grails prod war
cf push my-application -p target/my-application-version.war
```

Groovy

Groovy applications based on both [Ratpack](#) and a simple collection of files are supported.

Ratpack

Ratpack packages applications into two different styles; Cloud Foundry supports the `distZip` style. To build the ZIP and deploy it, run the following:

```
gradle distZip
cf push my-application -p build/distributions/my-application.zip
```

Raw Groovy

Groovy applications that are made up of a [single entry point](#) plus any supporting files can be run without any other work. To deploy them, run the following:

```
cf push my-application
```

Java Main

Java applications with a `main()` method can be run provided that they are packaged as [self-executable JARs](#).

**Note:** If your application is not web-enabled, you must suppress route creation to avoid a “failed to start accepting connections” error. To suppress route creation, add `no-route: true` to the application manifest or use the `--no-route` flag with the `cf push` command.

For more information about the `no-route` attribute, see the [Deploying with Application Manifests](#) topic.

Maven
A Maven build can create a self-executable JAR. To build and deploy the JAR, run the following:

```
mvn package
cf push my-application -p target/my-application-version.jar
```

Gradle

A Gradle build can create a self-executable JAR. To build and deploy the JAR, run the following:

```
grade build
cf push my-application -p build/libs/my-application-version.jar
```

Play Framework

The Play Framework packages applications into two different styles. Cloud Foundry supports both the staged and dist styles. To build the dist style and deploy it, run the following:

```
play dist
cf push my-application -p target/universal/my-application-version.zip
```

Spring Boot CLI

Spring Boot can run applications comprised entirely of POGOs. To deploy then, run the following:

```
spring grab *.groovy
cf push my-application
```

Servlet

Java applications can be packaged as Servlet applications.

Maven

A Maven build can create a Servlet WAR. To build and deploy the WAR, run the following:

```
mvn package
cf push my-application -p target/my-application-version.war
```

Gradle

A Gradle build can create a Servlet WAR. To build and deploy the JAR, run the following:

```
grade build
cf push my-application -p build/libs/my-application-version.war
```

Binding to Services

Information about binding apps to services can be found on the following pages:

- Service Bindings for Grails Applications

© Copyright Pivotal Software Inc, 2013-2018
Service Bindings for Play Framework Applications

Service Bindings for Spring Applications

Java Buildpack

For detailed information about using, configuring, and extending the Cloud Foundry Java buildpack, see https://github.com/cloudfoundry/java-buildpack.

Design

The Java Buildpack is designed to convert artifacts that run on the JVM into executable applications. It does this by identifying one of the supported artifact types (Grails, Groovy, Java, Play Framework, Spring Boot, and Servlet) and downloading all additional dependencies needed to run. The collection of services bound to the application is also analyzed and any dependencies related to those services are also downloaded.

As an example, pushing a WAR file that is bound to a PostgreSQL database and New Relic for performance monitoring would result in the following:

```
Initialized empty Git repository in /tmp/buildpacks/java-buildpack/.git/
-->
Java Buildpack source: https://github.com/cloudfoundry/java-buildpack#0928916a2dd78e9fa9469c558046ec09180e5d
-->
Downloading OpenJdk JRE 1.7.0_51 from http://.../openjdk/jdk-17.0.51.tar.gz (0.0s)
Expanding OpenJdk JRE to .java-buildpack/open_jdk_jre (1.9s)
-->
Downloading New Relic Agent 3.4.1 from http://.../new-relic/new-relic-3.4.1.jar (0.4s)
-->
Downloading PostgreSQL JDBC 9.3.1100 from http://.../postgresql-jdbc/postgresql-jdbc-9.3.1100.jar (0.0s)
-->
Downloading Spring Auto Reconfiguration 0.8.7 from http://.../auto-reconfiguration/auto-reconfiguration-0.8.7.jar (0.0s)
Modifying /WEB-INF/web.xml for Auto Reconfiguration
-->
Downloading Tomcat 7.0.50 from http://.../tomcat/tomcat-7.0.50.tar.gz (0.0s)
Expanding Tomcat to .java-buildpack/tomcat (0.1s)
-->
Downloading Buildpack Tomcat Support 1.1.1 from http://.../tomcat-buildpack-support/tomcat-buildpack-support-1.1.1.jar (0.1s)
-->
Uploading droplet (57M)
```

Configuration

In most cases, the buildpack should work without any configuration. If you are new to Cloud Foundry, we recommend that you make your first attempts without modifying the buildpack configuration. If the buildpack requires some configuration, use a fork of the buildpack.

Java and Grails Best Practices

Provide JDBC driver

The Java buildpack does not bundle a JDBC driver with your application. If your application will access a SQL RDBMS, include the appropriate driver in your application.

Allocate Sufficient Memory

If you do not allocate sufficient memory to a Java application when you deploy it, it may fail to start, or Elastic Runtime may terminate it. You must allocate enough memory to allow for the following:

- Java heap
- Metaspace, if using Java 8
- PermGen, if using Java 7 or earlier
- Thread stacks
JVM overhead

The `config/open_jdk_jre.yml` file of the Cloud Foundry Java buildpack contains default memory size and weighting settings for the JRE. See the Open JDK JRE README on GitHub for an explanation of JRE memory sizes and weightings and how the Java buildpack calculates and allocates memory to the JRE for your app.

To configure memory-related JRE options for your app, you create a custom buildpack and specify this buildpack in your deployment manifest. For more information about configuring custom buildpacks and manifests, refer to the Custom Buildpacks and Deploying with Application Manifests topics.

When your app is running, you can use the `cf app APP-NAME` command to see memory utilization.

Troubleshoot Failed Upload

If your application fails to upload when you push it to Cloud Foundry, it may be for one of the following reasons:

- WAR is too large: An upload may fail due to the size of the WAR file. Cloud Foundry testing indicates WAR files as large as 250 MB upload successfully. If a WAR file larger than that fails to upload, it may be a result of the file size.
- Connection issues: Application uploads can fail if you have a slow Internet connection, or if you upload from a location that is very remote from the target Cloud Foundry instance. If an application upload takes a long time, your authorization token can expire before the upload completes. A workaround is to copy the WAR to a server that is closer to the Cloud Foundry instance, and push it from there.
- Out-of-date cf CLI client: Upload of a large WAR is faster and hence less likely to fail if you are using a recent version of the cf CLI. If you are using an older version of the cf CLI client to upload a large WAR, and having problems, try updating to the latest version of the cf CLI.
- Incorrect WAR targeting: By default, `cf push` uploads everything in the current directory. For a Java application, a plain `cf push` will upload source code and other unnecessary files, in addition to the WAR. When you push a Java application, specify the path to the WAR:

```
cf push MY-APP -p PATH/TO/WAR-FILE
```

You can determine whether or not the path was specified for a previously pushed application by looking at the application deployment manifest, `manifest.yml`. If the `push` attribute specifies the current directory, the manifest will include a line like this:

```
push: .
```

To re-push just the WAR, either:

- Delete `manifest.yml` and push again, specifying the location of the WAR using the `-p` flag, or
- Edit the `push` argument in `manifest.yml` to point to the WAR, and re-push the application.

Debug Java Apps on Cloud Foundry

Because of the way that Cloud Foundry deploys your applications and isolates them, it is not possible to connect to your application with the remote Java debugger. Instead, instruct the application to connect to the Java debugger on your local machine.

Here are the instructions for setting up remote debugging when using BOSH Lite or a CloudFoundry installation.

1. Open your project in Eclipse.
2. Right-click on your project, go to Debug as and pick Debug Configurations.
3. Create a new Remote Java Application.
4. Make sure your project is selected, pick Standard (Socket Listen) from the Connection Type drop down and set a port. Make sure this port is open if you are running a firewall.
5. Click Debug.

The debugger should now be running. If you switch to the Debug perspective, you should see your application listed in the Debug panel and it should say

```
Waiting for vm to connect at port
```

Next, push your application to Cloud Foundry and instruct Cloud Foundry to connect to the debugger running on your local machine using the following instructions:

1. Edit your `manifest.yml` file. Set the instances count to 1. If you set this greater than one, multiple applications try to connect to your debugger.
2. Also in `manifest.yml`, add the `env` section and create a variable called `JAVA_OPTS`.

3. Add the remote debugger configuration to the `JAVA_OPTS` variable:

   ```
   -agentlib:jdwp=transport=dt_socket,address=YOUR-IP-ADDRESS:YOUR-PORT
   ```

4. Save the `manifest.yml` file.

5. Run `cf push`.

Upon completion, you should see that your application has started and is now connected to the debugger running in your IDE. You can now add breakpoints and interrogate the application just as you would if it were running locally.

### Slow Starting Java or Grails Apps

Some Java and Grails applications do not start quickly, and the DEA health check for an application can fail if an application starts too slowly.

The current Java buildpack implementation sets the Tomcat `bindOnInit` property to `false`. This prevents Tomcat from listening for HTTP requests until an application has fully deployed.

If your application does not start quickly, the DEA health check may fail because it checks the health of the application before the application can accept requests. By default, the DEA health check fails after a timeout threshold of 60 seconds.

To resolve this issue, use `cf push APP-NAME` with the `-t TIMEOUT-THRESHOLD` option to increase the timeout threshold. Specify TIMEOUT-THRESHOLD in seconds.

```bash
$ cf push my-app -t 180
```

**Note:** The timeout threshold cannot exceed 180 seconds. Specifying a timeout threshold greater than 180 seconds results in the following error:

```
Server error, status code: 400, error code: 100001, message: The app is invalid: health_check_timeout
maximum_exceeded
```

If your Java or Grails application requires more than 180 seconds to start, you can fork the Java buildpack and change the value of `bindOnInit` to `true` in `resources/tomcat/conf/server.xml`. This change allows Tomcat to listen for HTTP requests before your application has fully deployed.

**Note:** Changing the value of `bindOnInit` to `true` allows the DEA health check of your application to pass even before your application has fully deployed. In this state, Cloud Foundry might pass requests to your application before your application can serve them.

### Extension

The Java Buildpack is also designed to be easily extended. It creates abstractions for three types of components (containers, frameworks, and JREs) in order to allow users to easily add functionality. In addition to these abstractions, there are a number of utility classes for simplifying typical buildpack behaviors.

As an example, the New Relic framework looks like the following:
```ruby
class NewRelicAgent < JavaBuildpack::Component::VersionedDependencyComponent

# @macro base_component_compile
def compile
  FileUtils.mkdir_p logs_dir
  @droplet.copy_resources
end

# @macro base_component_release
def release
  @droplet.java_opts.
  .add_javaagent(@droplet.sandbox + jar_name)
  .add_system_property('newrelic.home', @droplet.sandbox)
  .add_system_property('newrelic.config.license_key', license_key)
  .add_system_property('newrelic.config.log_file_path', logs_dir)
end

protected

# @macro versioned_dependency_component_supports
def supports?
  @application.services.one_service? FILTER.'licenseKey'
end

private

FILTER = /newrelic/.freeze

def application_name
  @application.details['application_name']
end

def license_key
  @application.services.find_service(FILTER['credentials']['licenseKey'])
end

def logs_dir
  @droplet.sandbox + 'logs'
end
end
```

### Environment Variables

You can access environments variable programmatically.

For example, you can obtain `VCAP_SERVICES` as follows:

```ruby
System.getenv("VCAP_SERVICES");
```

See the [Cloud Foundry Environment Variables](https://www.pivotalcf.com/docs) topic for more information.
Configuring Service Connections for Grails

Cloud Foundry provides extensive support for connecting a Grails application to services such as MySQL, Postgres, MongoDB, Redis, and RabbitMQ. In many cases, a Grails application running on Cloud Foundry can automatically detect and configure connections to services. For more advanced cases, you can control service connection parameters yourself.

Auto-Configuration

Grails provides plugins for accessing SQL (using Hibernate, MongoDB, and Redis services. If you install any of these plugins and configure them in your Config.groovy or DataSource.groovy file, Cloud Foundry re-configures the plugin with connection information when your app starts.

If you were using all three types of services, your configuration might look like this:

```groovy
environments {
  production {
    dataSource {
      url = 'jdbc:mysql://localhost/db?useUnicode=true&characterEncoding=utf8'
      dialect = org.hibernate.dialect.MySQLInnoDBDialect
      driverClassName = 'com.mysql.jdbc.Driver'
      username = 'user'
      password = 'password'
    }
    grails {
      mongo {
        host = 'localhost'
        port = 27017
        databaseName = 'foo'
        username = 'user'
        password = 'password'
      }
      redis {
        host = 'localhost'
        port = 6379
        password = 'password'
        timeout = 2000
      }
    }
  }
}
```

The url, host, port, databaseName, username, and password fields in this configuration will be overridden by the Cloud Foundry auto-reconfiguration if it detects that the application is running in a Cloud Foundry environment. If you want to test the application locally against your own services, you can put real values in these fields. If the application will only be run against Cloud Foundry services, you can put placeholder values as shown here, but the fields must exist in the configuration.

Manual Configuration

If you do not want to use auto-configuration, you can configure the Cloud Foundry service connections manually.

Follow the steps below to manually configure a service connection.

1. Add the `spring-cloud` library to the dependencies section of your `BuildConfig.groovy` file.

```groovy
repositories {
  grailsHome()
  mavenCentral()
  grailsCentral()
  mavenRepo "http://repo.spring.io/milestone"
}
dependencies {
  compile "org.springframework.cloud:spring-cloud-cloudfoundry-connector:1.0.0.RELEASE"
  compile "org.springframework.cloud:spring-cloud-spring-service-connector:1.0.0.RELEASE"
}
```

Adding the `spring-cloud` library allows you to disable auto-configuration and use the `spring-cloud` API in your `DataSource.groovy` file to set the...
connection parameters.

2. Add the following to your `grails-app/conf/spring/resources.groovy` file to disable auto-configuration:

```groovy
beans = {
    cloudFactory = org.springframework.cloud.CloudFactory
}
```

3. Add the following imports to your `DataSource.groovy` file to allow spring-cloud API commands:

```groovy
import org.springframework.cloud.CloudFactory
import org.springframework.cloud.CloudException
```

4. Add the following code to your `DataSource.groovy` file to enable Cloud Foundry's `getCloud` method to function locally or in other environments outside of a cloud.

```groovy
def cloud = null
try {
    cloud = new CloudFactory().cloud
} catch (CloudException) { }
```

5. Use code like the following to access the cloud object:

```groovy
def dbInfo = cloud?.getServiceInfo('myapp-mysql')
url = dbInfo??jdbcUrl
username = dbInfo??userName
password = dbInfo??password
```

The example `DataSource.groovy` file below contains the following:

- The imports that allow spring-cloud API commands
- The code that enables the `getCloud` method to function locally or in other environments outside of a cloud
- Code to access the cloud object for SQL, MongoDB, and Redis services
import org.springframework.cloud.CloudFactory
import org.springframework.cloud.CloudException

def cloud = null
try {
    cloud = new CloudFactory().cloud
} catch (CloudException) {}
Configuring Service Connections for Play Framework

Page last updated:

Cloud Foundry supports running Play Framework applications and the Play JPA plugin for auto-configuration for Play versions up to and including v2.4.x.

Cloud Foundry provides support for connecting a Play Framework application to services such as MySQL and Postgres. In many cases, a Play Framework application running on Cloud Foundry can automatically detect and configure connections to services.

Auto-Configuration

By default, Cloud Foundry detects service connections in a Play Framework application and configures them to use the credentials provided in the Cloud Foundry environment. Note that auto-configuration happens only if there is a single service of either of the supported types—MySQL or Postgres.
Configuring Service Connections for Spring

Page last updated:

Cloud Foundry provides extensive support for connecting a Spring application to services such as MySQL, PostgreSQL, MongoDB, Redis, and RabbitMQ. In many cases, Cloud Foundry can automatically configure a Spring application without any code changes. For more advanced cases, you can control service connection parameters yourself.

Auto-Reconfiguration

If your Spring application requires services such as a relational database or messaging system, you might be able to deploy your application to Cloud Foundry without changing any code. In this case, Cloud Foundry automatically re-configures the relevant bean definitions to bind them to cloud services.

Cloud Foundry auto-reconfigures applications only if the following items are true for your application:

- Only one service instance of a given service type is bound to the application. In this context, MySQL and PostgreSQL are considered the same service type, relational database, so if both a MySQL and a PostgreSQL service are bound to the application, auto-reconfiguration will not occur.
- Only one bean of a matching type is in the Spring application context. For example, you can have only one bean of type `javax.sql.DataSource`.

With auto-reconfiguration, Cloud Foundry creates the database or connection factory bean itself, using its own values for properties such as host, port, username and so on. For example, if you have a single `javax.sql.DataSource` bean in your application context that Cloud Foundry auto-reconfigures and binds to its own database service, Cloud Foundry does not use the username, password and driver URL you originally specified. Instead, it uses its own internal values. This is transparent to the application, which really only cares about having a relational database to which it can write data but does not really care what the specific properties are that created the database. Also note that if you have customized the configuration of a service, such as the pool size or connection properties, Cloud Foundry auto-reconfiguration ignores the customizations.

For more information about auto-reconfiguration of specific services types, see the Service-Specific Details section.

Manual Configuration

Use manual configuration if you have multiple services of a given type or you want to have more control over the configuration than auto-reconfiguration provides.

To use manual configuration, include the `spring-cloud` library in your list of application dependencies. Update your application Maven `pom.xml` or Gradle `build.gradle` file to include dependencies on the `org.springframework.cloud:spring-cloud-spring-service-connector` and `org.springframework.cloud:spring-cloud-cloudfoundry-connector` artifacts.

For example, if you use Maven to build your application, the following `pom.xml` snippet shows how to add this dependency.

```xml
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-spring-service-connector</artifactId>
    <version>1.1.0.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-cloudfoundry-connector</artifactId>
    <version>1.1.0.RELEASE</version>
  </dependency>
</dependencies>
```

You also need to update your application build file to include the Spring Framework Milestone repository. The following `pom.xml` snippet shows how to do this for Maven:

```xml
<repositories>
  <repository>
    <id>repository.springsource.milestone</id>
    <name>SpringSource Milestone Repository</name>
    <url>http://repo.springsource.org/milestone</url>
  </repository>
</repositories>
```
Java Configuration

Typical use of Java config involves extending the AbstractCloudConfig class and adding methods with the @Bean annotation to create beans for services. Apps migrating from auto-reconfiguration might first try Scanning for Services until they need more explicit control. Java config also offers a way to expose application and service properties. Use this for debugging or to create service connectors using a lower-level access.

Create a Service Bean

In the following example, the configuration creates a DataSource bean connecting to the only relational database service bound to the app. It also creates a MongoDbFactory bean, again, connecting to the only MongoDB service bound to the app. Check Javadoc for AbstractCloudConfig for ways to connect to other services.

```java
class CloudConfig extends AbstractCloudConfig {
    @Bean
    public DataSource inventoryDataSource() {
        return connectionFactory().dataSource();
    }
    // ... more beans to obtain service connectors
}
```

The bean names match the method names unless you specify an explicit value to the annotation such as @Bean("inventory-service") , following Spring's Java configuration standards.

If you have more than one service of a type bound to the app or want to have an explicit control over the services to which a bean is bound, you can pass the service names to methods such as dataSource() and mongoDbFactory() as follows:

```java
class CloudConfig extends AbstractCloudConfig {
    @Bean
    public DataSource inventoryDataSource() {
        return connectionFactory().dataSource("inventory-db-service");
    }
    @Bean
    public MongoDbFactory documentMongoDbFactory() {
        return connectionFactory().mongoDbFactory("document-service");
    }
    // ... more beans to obtain service connectors
}
```

Method such as dataSource() come in an additional overloaded variant that offer specifying configuration options such as the pooling parameters. See Javadoc for more details.

Connect to Generic Services

Java config supports access to generic services through the service() method. Generic services do not have a directly mapped method. This is typical for a newly introduced service or when connecting to a private service in private PaaS. The generic service() method follows the same pattern as the dataSource(), except it allows supplying the connector type as an additional parameters.

Scan for Services

You can scan for each bound service using the @ServiceScan annotation as shown below. This is conceptually similar to the @ComponentScan annotation in Spring:

```java
@Configuration
@ServiceScan
class CloudConfig {
}
```

Here, one bean of the appropriate type (DataSource for a relational database service, for example) is created. Each created bean will have the id matching the corresponding service name. You can then inject such beans using auto-wiring.
If the app is bound to more than one services of a type, you can use the \@Qualifier annotation supplying it the name of the service as in the following code:

```java
@Autowired
@Qualifier("inventory-db") DataSource inventoryDb;
@Autowired
@Qualifier("shipping-db") DataSource shippingDb;
```

**Access Service Properties**

You can expose raw properties for all services and the app through a bean as follows:

```java
class CloudPropertiesConfig extends AbstractCloudConfig {
    @Bean
    public Properties cloudProperties() {
        return properties();
    }
}
```

**Cloud Profile**

Spring Framework versions 3.1 and above support bean definition profiles as a way to conditionallyize the application configuration so that only specific bean definitions are activated when a certain condition is true. Setting up such profiles makes your application portable to many different environments so that you do not have to manually change the configuration when you deploy it to, for example, your local environment and then to Cloud Foundry.

See the Spring Framework documentation for additional information about using Spring bean definition profiles.

When you deploy a Spring application to Cloud Foundry, Cloud Foundry automatically enables the `cloud` profile.

**Note:** Cloud Foundry auto-reconfiguration requires the Spring application to be version 3.1 or later and include the Spring context JAR. If you are using an earlier version, update your framework or use a custom buildpack.

**Profiles in Java Configuration**

The \@Profile annotation can be placed on \@Configuration classes in a Spring application to set conditions under which configuration classes are invoked. By using the `default` and `cloud` profiles to determine whether the application is running on Cloud Foundry or not, your Java configuration can support both local and cloud deployments using Java configuration classes like these:
public class Configuration {
    @Configuration
    @Profile("cloud")
    static class CloudConfiguration {
        @Bean
        public DataSource dataSource() {
            CloudFactory cloudFactory = new CloudFactory();
            Cloud cloud = cloudFactory.getCloud();
            String serviceID = cloud.getServiceID();
            return cloud.getServiceConnector(serviceID, DataSource.class, null);
        }
    }
}

public class Configuration {
    @Configuration
    @Profile("default")
    static class LocalConfiguration {
        @Bean
        public DataSource dataSource() {
            BasicDataSource dataSource = new BasicDataSource();
            dataSource.setUrl("jdbc:postgresql://localhost/db");
            dataSource.setDriverClassName("org.postgresql.Driver");
            dataSource.setUsername("postgres");
            dataSource.setPassword("postgres");
            return dataSource;
        }
    }
}

Property Placeholders

Cloud Foundry exposes a number of application and service properties directly into its deployed applications. The properties exposed by Cloud Foundry include basic information about the application, such as its name and the cloud provider, and detailed connection information for all services currently bound to the application.

Service properties generally take one of the following forms:

cloud.services.[service-name].connection.[property]
cloud.services.[service-name].[property]

In this form, [service-name] refers to the name you gave the service when you bound it to your application at deploy time, and [property] is a field in the credentials section of the VCAP_SERVICES environment variable.

For example, assume that you created a Postgres service called my-postgres and then bound it to your application. Assume also that this service exposes credentials in VCAP_SERVICES as discrete fields. Cloud Foundry exposes the following properties about this service:

cloud.services.my-postgres.connection.hostname
cloud.services.my-postgres.connection.name
cloud.services.my-postgres.connection.password
cloud.services.my-postgres.connection.port
cloud.services.my-postgres.connection.username
cloud.services.my-postgres.plan
cloud.services.my-postgres.type

If the service exposed the credentials as a single uri field, then the following properties would be set up:

cloud.services.my-postgres.connection.uri
cloud.services.my-postgres.plan
cloud.services.my-postgres.type

For convenience, if you have bound just one service of a given type to your application, Cloud Foundry creates an alias based on the service type instead of the service name. For example, if only one MySQL service is bound to an application, the properties take the form cloud.services.mysql.connection.[property]. Cloud Foundry uses the following aliases in this case:

+ mysql
+ postgresql
+ mongodb
A Spring application can take advantage of these Cloud Foundry properties using the property placeholder mechanism. For example, assume that you have bound a MySQL service called \texttt{spring-mysql} to your application. Your application requires a \texttt{c3p0} connection pool instead of the connection pool provided by Cloud Foundry, but you want to use the same connection properties defined by Cloud Foundry for the MySQL service - in particular the username, password and JDBC URL.

The following table lists all the application properties that Cloud Foundry exposes to deployed applications.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{cloud.application.name}</td>
<td>The name provided when the application was pushed to Cloud Foundry.</td>
</tr>
<tr>
<td>\texttt{cloud.provider.url}</td>
<td>The URL of the cloud hosting the application, such as \texttt{cloudfoundry.com}.</td>
</tr>
</tbody>
</table>

The service properties that are exposed for each type of service are listed in the Service-Specific Details section.

### Service-Specific Details

The following sections describe Spring auto-reconfiguration and manual configuration for the services supported by Cloud Foundry.

#### MySQL and Postgres

**Auto-Reconfiguration**

Auto-reconfiguration occurs if Cloud Foundry detects a \texttt{javax.sql.DataSource} bean in the Spring application context. The following snippet of a Spring application context file shows an example of defining this type of bean which Cloud Foundry will detect and potentially auto-reconfigure:

```xml
<bean class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" id="dataSource">
  <property name="driverClassName" value="org.h2.Driver"/>
  <property name="url" value="jdbc:h2:mem:">
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>
```

The relational database that Cloud Foundry actually uses depends on the service instance you explicitly bind to your application when you deploy it: MySQL or Postgres. Cloud Foundry creates either a commons DBCP or Tomcat datasource depending on which datasource implementation it finds on the classpath.

Cloud Foundry internally generates values for the following properties: \texttt{driverClassName}, \texttt{url}, \texttt{username}, \texttt{password}, \texttt{validationQuery}.

**Manual Configuration in Java**

To configure a database service in Java configuration, create a \texttt{@Configuration} class with a \texttt{@Bean} method to return a \texttt{javax.sql.DataSource} bean. The bean can be created by helper classes in the \texttt{spring-cloud} library, as shown here:

```java
@Configuration
public class DataSourceConfig {
    @Bean
    public DataSource dataSource() {
        CloudFactory cloudFactory = new CloudFactory();
        Cloud cloud = cloudFactory.getCloud();
        String serviceID = cloud.getServiceID();
        CloudServiceConnector serviceConnector = cloud.getServiceConnector(serviceID, DataSource.class, null);
        return serviceConnector.getDataSource(serviceID, null);
    }
}
```

### MongoDB
Auto-Reconfiguration

You must use Spring Data MongoDB 1.0 M4 or later for auto-reconfiguration to work.

Auto-reconfiguration occurs if Cloud Foundry detects an \texttt{org.springframework.data.document.mongodb.MongoDbFactory} bean in the Spring application context. The following snippet of a Spring XML application context file shows an example of defining this type of bean which Cloud Foundry will detect and potentially auto-reconfigure:

```xml
<mongo:db-factory
    id="mongoDbFactory"
    dbname="pwdtest"
    host="127.0.0.1"
    port="1234"
    username="test_user"
    password="test_pass"/>
```

Cloud Foundry creates a \texttt{SimpleMongoDbFactory} with its own values for the following properties: \texttt{host}, \texttt{port}, \texttt{username}, \texttt{password}, \texttt{dbname}.

Manual Configuration in Java

To configure a MongoDB service in Java configuration, create a \texttt{@Configuration} class with a \texttt{@Bean} method to return an \texttt{org.springframework.data.mongodb.MongoDbFactory} bean from Spring Data MongoDB. The bean can be created by helper classes in the \texttt{spring-cloud} library, as shown here:

```java
@Configuration
public class MongoConfig {

    @Bean
    public MongoDbFactory mongoDbFactory() {
        CloudFactory cloudFactory = new CloudFactory();
        Cloud cloud = cloudFactory.getCloud();
        MongoServiceInfo serviceInfo = (MongoServiceInfo) cloud.getServiceInfo("my-mongodb");
        String serviceID = serviceInfo.getID();
        return cloud.getServiceConnector(serviceID, DataSource.class, null);
    }

    @Bean
    public MongoTemplate mongoTemplate() {
        return new MongoTemplate(mongoDbFactory());
    }
}
```

Redis

Auto-Configuration

You must be using Spring Data Redis 1.0 M4 or later for auto-configuration to work.

Auto-configuration occurs if Cloud Foundry detects a \texttt{org.springframework.data.redis.connection.RedisConnectionFactory} bean in the Spring application context. The following snippet of a Spring XML application context file shows an example of defining this type of bean which Cloud Foundry will detect and potentially auto-configure:

```xml
<bean
    id="redis"
    class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
    p:hostname="localhost" p:port="6379"/>
```

Cloud Foundry creates a \texttt{JedisConnectionFactory} with its own values for the following properties: \texttt{host}, \texttt{port}, \texttt{password}. This means that you must package the Jedis JAR in your application. Cloud Foundry does not currently support the JRedis and RJC implementations.

Manual Configuration in Java

To configure a Redis service in Java configuration, create a \texttt{@Configuration} class with a \texttt{@Bean} method to return an \texttt{org.springframework.data.redis.connection.RedisConnectionFactory} bean from Spring Data Redis. The bean can be created by helper classes in the \texttt{spring-cloud} library.
library, as shown here:

```java
@Configuration
public class RedisConfig {
    @Bean
    public RedisConnectionFactory redisConnectionFactory() {
        CloudFactory cloudFactory = new CloudFactory();
        Cloud cloud = cloudFactory.getCloud();
        RedisServiceInfo serviceInfo = (RedisServiceInfo) cloud.getServiceInfo("my-redis");
        String serviceID = serviceInfo.getID();
        return cloud.getServiceConnector(serviceID, RedisConnectionFactory.class, null);
    }

    @Bean
    public RedisTemplate redisTemplate() {
        return new StringRedisTemplate(redisConnectionFactory());
    }
}
```

RabbitMQ

Auto-Configuration

You must be using Spring AMQP 1.0 or later for auto-configuration to work. Spring AMQP provides publishing, multi-threaded consumer generation, and message conversion. It also facilitates management of AMQP resources while promoting dependency injection and declarative configuration.

Auto-configuration occurs if Cloud Foundry detects an `org.springframework.amqp.rabbit.connection.ConnectionFactory` bean in the Spring application context. The following snippet of a Spring application context file shows an example of defining this type of bean which Cloud Foundry will detect and potentially auto-configure:

```xml
<rabbit:connection-factory
    id="rabbitConnectionFactory"
    host="localhost"
    password="testpwd"
    port="1234"
    username="testuser"
    virtual-host="/"
/>
```

Cloud Foundry creates an `org.springframework.amqp.rabbit.connection.CachingConnectionFactory` with its own values for the following properties: `host`, `virtual-host`, `port`, `username`, `password`.

Manual Configuration in Java

To configure a RabbitMQ service in Java configuration, create a `@Configuration` class with a `@Bean` method to return an `org.springframework.amqp.rabbit.connection.ConnectionFactory` bean from the Spring AMQP library. The bean can be created by helper classes in the `spring-cloud` library, as shown here:

```java
@Configuration
public class RabbitConfig {
    @Bean
    public ConnectionFactory rabbitConnectionFactory() {
        CloudFactory cloudFactory = new CloudFactory();
        Cloud cloud = cloudFactory.getCloud();
        AmqpServiceInfo serviceInfo = (AmqpServiceInfo) cloud.getServiceInfo("my-rabbit");
        String serviceID = serviceInfo.getID();
        return cloud.getServiceConnector(serviceID, ConnectionFactory.class, null);
    }

    @Bean
    public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
        return new RabbitTemplate(connectionFactory);
    }
}
```
Cloud Foundry Eclipse Plugin

Page last updated:

The Cloud Foundry Eclipse Plugin is an extension that enables Cloud Foundry users to deploy and manage Java and Spring applications on a Cloud Foundry instance from Eclipse or Spring Tool Suite (STS).

The plugin supports Eclipse v3.8 and v4.3 (a Java EE version is recommended), and STS 3.0.0 and later.

This page has instructions for installing and using v1.7.2 of the plugin.

You can use the plugin to perform the following actions:

- Deploy applications from an Eclipse or STS workspace to a running Cloud Foundry instance. The Cloud Foundry Eclipse plugin supports the following application types:
  - Spring Boot
  - Spring
  - Java Web
  - Java standalone
  - Grails
- Create, bind, and unbind services.
- View and manage deployed applications and services.
- Start and stop applications.

v1.7.2 of this plugin provides the following updates and changes:

- Cloud Foundry Eclipse is now enabled for NLS and Internationalization.
- A “New Service Binding” wizard that allows service instances to be bound to applications. This wizard serves as an alternative to binding through the existing drag-and-drop feature.
- Improvements in Loggregator streaming to the console.
- Improvements in deploying Spring Boot and Getting Started projects with templates.

Install Cloud Foundry Eclipse Plugin

If you have a previous version of the Cloud Foundry Eclipse Plugin installed, uninstall it before installing the new version. To uninstall the plugin:

1. Choose About Eclipse (or About Spring Tool Suite) from the Eclipse (or Spring Tool Suite) menu and click Installation Details.
2. In Installation Details, select the previous version of the plugin and click Uninstall.

Follow the installation instructions appropriate for your environment:

- Install to Eclipse from Marketplace
- Install to STS from IDE Extensions Tab
- Install from a Local Repository

Install to Eclipse from Marketplace

Follow the instructions below to install the Cloud Foundry Eclipse Plugin to Eclipse from the Eclipse Marketplace.

1. Start Eclipse.
2. From the Eclipse Help menu, select Eclipse Marketplace.
3. In the Eclipse Marketplace window, enter “Cloud Foundry” in the Find field. Click Go.
4. In the search results, next to the listing for Cloud Foundry Integration, click Install.
5. In the Confirm Selected Features window, click Confirm.

6. The Review Licenses window appears. Select “I accept the terms of the license agreement” and click Finish.
7. The Software Updates window appears. Click Yes to restart Eclipse.

Install to STS from IDE Extensions Tab

Follow these instructions to install the Cloud Foundry Eclipse Plugin to Spring Tool Suite (STS) from the IDE Extensions tab.

1. Start STS.
2. On the STS Dashboard, click IDE Extensions.
3. Enter “Cloud Foundry” in the Find field.
4. Select Cloud Foundry Integration for Eclipse and click Install.
5. In the Install window, click Next.
6. In the Install Details window, click Next.

7. The Review Licenses window appears. Select “I accept the terms of the license agreement” and click Finish.

8. The Software Updates window appears. Click Yes to restart Spring Tool Suite.
Install a Release Version Offline

If you need to install a release version of Cloud Foundry Eclipse Plugin in offline mode, you can download a release update site zip file and transfer it to the offline environment.

To install a Release Version offline, follow the steps below on a computer running Eclipse or Spring Tool Suite (STS).

2. In Eclipse or STS, select Install New Software from the Help menu.
3. In the Available Software window, to the right of the Work with field, click Add.
4. In the Add Repository window, enter Cloud Foundry Integration or a name of your choice for the repository. Click Archive.
5. In the Open window, browse to the location of the update site zip file and click Open.
6. In the Add Repository window, click OK.
7. In the Available Software window, select Core/Cloud Foundry Integration and, optionally, Resources/Cloud Foundry Integration. Click Next.
8. In the Review Licenses window, select “I accept the terms of the license agreement” and click Finish.

Install from a Local Build

If you need to install the Cloud Foundry Eclipse Plugin from a local build, rather than from a release version, you can download and build the source, create a repository and copy it to the target machine, then install from the copied repository.

1. Obtain the plugin source from GitHub in one of the following ways:
   - Download archived source code for released versions of the plugin from https://github.com/SpringSource/eclipse-integration-cloudfoundry/releases
   - Clone the project repository:
     ```sh
git clone https://github.com/SpringSource/eclipse-integration-cloudfoundry
```
2. Unzip the downloaded archive. In a terminal, run the following command:
   ```sh
mvn -P e37 package
```
3. Copy the org.cloudfoundry.ide.eclipse.server.site/target/site directory to the machine where you want to install the plugin.
4. On the machine where you want to install the plugin, launch Eclipse or Spring Tool Suite (STS).
5. Select Install New Software from the Help menu.
6. In the Available Software window, to the right of the Work with field, click Add.
7. In the Add Repository window, enter Cloud Foundry Integration or a name of your choice for the repository. Click Local.
8. In the Open window, browse to the org.cloudfoundry.ide.eclipse.server.site/target/site directory. Click Open.
9. In the Add Repository window, click OK.
10. In the **Available Software** window, select **Core/Cloud Foundry Integration** and, optionally, **Resources/Cloud Foundry Integration**. Click **Next**.

11. In the **Review Licenses** window, select "I accept the terms of the license agreement" and click **Finish**.

### About the Plugin User Interface

The sections below describe the Cloud Foundry Eclipse plugin user interface. If you do not see the tabs described below, select the Pivotal Cloud Foundry server in the **Servers** view. To expose the **Servers** view, ensure that you are using the Java perspective, then select **Window > Show View > Other > Server > Servers**.

The Cloud Foundry editor, outlined in red in the screenshot below, is the primary plugin user interface. Some workflows involve interacting with standard elements of the Eclipse user interface, such as the **Project Explorer** and the **Console** and **Servers** views.

Note that the Cloud Foundry editor allows you to work with a single Cloud Foundry space. Each space is represented by a distinct server instance in the **Servers** view (B). Multiple editors, each targeting a different space, can be open simultaneously. However, only one editor targeting a particular Cloud Foundry server instance can be open at a time.

### Overview Tab

The follow panes and views are present when the **Overview** tab is selected:

- **A** — The **Package Explorer** view lists the projects in the current workspace.
- **B** — The **Servers** view lists server instances configured in the current workspace. A server of type **Pivotal Cloud Foundry** represents a targeted space in a Cloud Foundry instance.
- **C** — The **General Information** pane.
- **D** — The **Account Information** pane lists your Cloud Foundry credentials and the target organization and space. The pane includes these controls:
  - **Clone Server** — Use to create additional Pivotal Cloud Foundry server instances. You must configure a server instance for each Cloud Foundry space that you wish to target. For more information, see [Create Additional Server Instances](#).
  - **Change Password** — Use to change your Cloud Foundry password.
  - **Validate Account** — Use to verify your currently configured Cloud Foundry credentials.
- **E** — The **Server Status** pane shows whether or not you are connected to the target Cloud Foundry space, and the **Disconnect** and **Connect** controls.
- **F** — The **Console** view displays status messages when you perform an action such as deploying an application.
- **G** — The **Remote Systems** view allows you to view the contents of a file that is part of a deployed application. For more information, see [View an Application File](#).
Applications and Services Tab

The follow panes are present when the Applications and Services tab is selected:

- **H** — The Applications pane lists the applications deployed to the target space.
- **I** — The Services pane lists the services provisioned in the targeted space.
- **J** — The General pane displays the following information for the application currently selected in the Applications pane:
  - Name
  - **Mapped URLs** – Lists URLs mapped to the application. You can click a URL to open a browser to the application within Eclipse or STS, and click the pencil icon to add or remove mapped URLs. See Manage Application URLs.
  - **Memory Limit** – The amount of memory allocated to the application. You can use the pull-down to change the memory limit.
  - **Instances** – The number of instances of the application that are deployed. You can use the pull-down to change number of instances.
  - **Start, Stop, Restart, Update and Restart** — The controls that appear depend on the current state of the application. The Update and Restart command will attempt an incremental push of only those application resources that have changed. It will not perform a full application push. See Push Application Changes below.
- **K** — The Services pane lists services that are bound to the application currently selected in the Applications pane. The icon in the upper right corner of the pane allows you to create a service, as described in Create a Service.
Create a Cloud Foundry Server

This section contains instructions for configuring a server resource that will represent a target Cloud Foundry space. You will create a server for each space in Cloud Foundry to which you will deploy applications. Once you create your first Cloud Foundry service instances using the instructions below, you can create additional instances using the Clone Server feature.

1. Right-click the Servers view and select New > Server.

2. In the Define a New Server window, expand the Pivotal folder, select Cloud Foundry, and click Next.

   ![Diagram showing the Define a New Server window](image)

   **Note:** Do not modify default values for Server host name or Server Runtime Environment. These fields are not used.
3. In the Cloud Foundry Account window, if you already have a Pivotal Cloud Foundry Hosted Developer Edition account, enter your email account and password credentials and click Validate Account.

Note: By default, the URL field points to the Pivotal Cloud Foundry Hosted Developer Edition URL of https://api.run.pivotal.io. If you have a Pivotal Elastic Runtime account, refer to the Logging in to Apps Manager topic to determine your Pivotal Elastic Runtime URL. Click Manage Cloud... to add this URL to your Cloud Foundry account. Validate the account and continue through the wizard.

If you do not have a Cloud Foundry account and want to register a new Pivotal Cloud Foundry Hosted Developer Edition account, click Sign Up. After you create the account, you can complete this procedure.

Note: The Register Account button is inactive.
4. The **Cloud Foundry Account** window is refreshed and displays a message indicating whether or not your credentials were valid. Click **Next**.

5. In the **Organizations and Spaces** window, select the space that you want to target, and click **Finish**.

**Note:** If you do not select a space, the server will be configured to connect to the default space, which is the first encountered in a list of your spaces.
6. Once you have successfully configured the Pivotal Cloud Foundry server, it will appear in the Servers view of the Eclipse or STS user interface. To familiarize yourself with the plugin user interface, see About the Plugin User Interface. Following this step, proceed to Deploy an Application.

**Deploy an Application**

To deploy an application to Cloud Foundry using the plugin:

1. To initiate deployment either:
   - Drag the application from the Package Explorer view onto the Pivotal Cloud Foundry server in the Servers view, or
   - Right-click the Pivotal Cloud Foundry server in the Servers view, select Add and Remove from the server context menu, and move the application from the Available to the Configured column.

2. In the Application Details window:
   - By default, the Name field is populated with the application project name. You can enter a different name. The name is assigned to the deployed application, but does not rename the project.
   - If you want to use an external buildpack to stage the application, enter the URL of the buildpack.

You can deploy the application without further configuration by clicking Finish. Note that because the application default values may take a second or two to load, the Finish button might not be enabled immediately. A progress indicator will indicate when the application default values have been loaded, and the “Finish” button will be enabled. Click Next to continue.
3. In the Launch Deployment window:
   - **Host** — By default, contains the name of the application. You can enter a different value if desired. If you push the same application to multiple spaces in the same organization, you must assign a unique Host to each.
   - **Domain** — Contains the default domain. If you have mapped custom domains to the target space, they appear in the pull-down list.

   **Note:** This version of the Cloud Foundry Eclipse plugin does not provide a mechanism for mapping a custom domain to a space. You must use the `cf map domain` command to do so.

   - **Deployed URL** — By default, contains the value of the **Host** and **Domain** fields, separated by a period (.) character.
   - **Memory Reservation** — Select the amount of memory to allocate to the application from the pull-down list.
   - **Start application on deployment** — If you do not want the application to be started on deployment, uncheck the box.

4. The Services Selection window lists services provisioned in the target space. Checkmark the services, if any, that you want to bind to the application, and click Finish. You can bind services to the application after deployment, as described in Bind and Unbind Services.
As the deployment proceeds, progress messages appear in the Console view. When deployment is complete, the application is listed in the Applications pane.

Create a Service

Before you can bind a service to an application, you must create it.

To create a service:

1. Select the Applications and Services tab.
2. Click the icon in the upper right corner of the Services pane.
3. In the Service Configuration window, enter a text pattern to Filter for a service. Matches are made against both service name and description.
4. Select a service from the Service List. The list automatically updates based on the filter text.
5. Enter a Name for the service and select a service Plan from the drop-down list.
6. Click Finish. The new service appears in the Services pane.

**Bind and Unbind Services**

You can bind a service to an application when you deploy it. To bind a service to an application that is already deployed, drag the service from the Services pane to the Application Services pane. (See the area labelled “G” in the screenshot in the Applications and Services above.)

To unbind a service, right-click the service in the Application Services pane, and select Unbind from Application.

**View an Application File**

You can view the contents of a file in a deployed application by selecting it the Remote Systems View. (See the areas labelled “I” and “J” in the screenshot in the Applications and Services Tab above.)

1. If the Remote Systems View is not visible:
   - Select the Applications and Services tab.
   - Select the application of interest from the Applications pane.
   - In the Instances pane, click the Remote Systems View link.

2. In the Remote Systems View, browse to the application and application file of interest, and double-click the file. A new tab appears in the editor area with the contents of the selected file.
Undeploy an Application

To undeploy an application, right click the application in either the Servers or the Applications pane and click Remove.

Scale an Application

You can change the memory allocation for an application and the number of instances deployed in the General pane when the Applications and Services tab is selected. Use the Memory Limit and Instances selector lists.

Although the scaling can be performed while the application is running, if the scaling has not taken effect, restart the application. If necessary, the application statistics can be manually refreshed by clicking the Refresh button in the top, right corner of the “Applications” pane, labelled “H” in the screenshot in Applications and Services Tab.

Push Application Changes

The Cloud Foundry editor supports these application operations:

- **Start** and **Stop** — When you Start an application, the plugin pushes all application files to the Cloud Foundry instance before starting the application, regardless of whether there are changes to the files or not.
- **Restart** — When you Restart a deployed application, the plugin does not push any resources to the Cloud Foundry instance.
- **Update and Restart** — When you run this command, the plugin pushes only the changes that were made to the application since last update, not the entire application. This is useful for performing incremental updates to large applications.

Manage Application URLs

You add, edit, and remove URLs mapped to the currently selected application in the General pane when the Applications and Services tab is selected. Click the pencil icon to display the Mapped URIs Configuration window.
Information in the Console View

When you start, restart, or update and restart an application, application output will generally be streamed to the Console view (labelled “F” in the screenshot in Overview Tab). The information shown in the Console view for a running application instance includes staging information, and the application’s `std.out` and `std.error` logs.

If multiple instances of the application are running, only the output of the first instance appears in the Console view. To view the output of another running instance, or to refresh the output that is currently displayed:

1. In the Applications and Services tab, select the deployed application in the Applications pane.
2. Click Refresh on the top right corner of the Applications pane.
3. In the Instances pane, wait for the application instances to be updated.
4. Once non-zero health is shown for an application instance, right-click on that instance to open the context menu and select Show Console.

Clone a Cloud Foundry Server Instance

Each space in Cloud Foundry to which you want to deploy applications must be represented by a Cloud Foundry server instance in the Servers view. After you have created a Cloud Foundry server instance, as described in Create a Cloud Foundry Server, you can clone it to create another.

Follow the step below to clone a server:

1. Perform one of the following actions:
   - In the Cloud Foundry server instance editor “Overview” tab, click Clone Server.
   - Right-click a Cloud Foundry server instance in the Servers view, and select Clone Server from the context menu.
2. In the Organizations and Spaces window, select the space that you want to target.
3. The name field will be filled with the name of the space that you selected. If desired, edit the server name before clicking finish Finish.

Add a Cloud Foundry Instance URL

You can configure the plugin to work with any Cloud Foundry instances to which you have access. To do so:
1. Perform steps 1 and 2 of Create a Cloud Foundry Server.

2. In the Cloud Foundry Account window, enter the email account and password that you use to log on to the target instance, then click Manage Cloud URLs.

3. In the Manage Cloud URLs window, click Add.

4. In the Add a Cloud URL window, enter the name and URL of the target cloud instance and click Finish.
5. The new cloud instance should appear in the list on the Manage Cloud URLs window. Click OK to proceed.

6. In the Cloud Foundry Account window, click Validate Account.

7. The Cloud Foundry Account window is refreshed and displays a message indicating whether or not your credentials were valid. Click Next.

8. In the Organizations and Spaces window, select the space that you want to target, and click Finish.

Note: If you do not select a space, the server will be configured to connect to the default space, which is the first encountered in a list of your spaces.
9. Once you have successfully configured the Pivotal Cloud Foundry server, it will appear in the Servers view of the Eclipse or STS user interface. To familiarize yourself with the plugin user interface, see About the Plugin User Interface. Following this step, proceed to Deploy an Application.
Cloud Foundry Java Client Library

Page last updated:

Introduction

This is a guide to using the Cloud Foundry Java Client Library to manage an account on a Cloud Foundry instance.

Note: The 1.1.x versions of the Cloud Foundry Java Client Library work with apps using Spring 4.x, and the 1.0.x versions of the Cloud Foundry Java Client Library work with apps using Spring 3.x. Both versions are available in the source repository on GitHub.

Adding the Library

Visit the Cloud Foundry Java Client Library GitHub page to obtain the correct components.

Most projects need two dependencies: the Operations API and an implementation of the Client API. Refer to the following sections for more information on how to add the Cloud Foundry Java Client Library as dependencies to a Maven or Gradle project.

Maven

Add the cloudfoundry-client-reactor dependency (formerly known as cloudfoundry-client-spring) to your pom.xml as follows:

```xml
<dependencies>
  <dependency>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cloudfoundry-client-reactor</artifactId>
    <version>2.0.0.BUILD-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cloudfoundry-operations</artifactId>
    <version>2.0.0.BUILD-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
    <version>2.5.0.BUILD-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-netty</artifactId>
    <version>2.5.0.BUILD-SNAPSHOT</version>
  </dependency>
  ...
</dependencies>
```

The artifacts can be found in the Spring release and snapshot repositories:

```xml
<repositories>
  <repository>
    <id>spring-releases</id>
    <name>Spring Releases</name>
    <url>http://repo.spring.io/release</url>
  </repository>
  ...
</repositories>
```
Gradle

Add the `cloudfoundry-client-reactor` dependency to your `build.gradle` file as follows:

```gradle
dependencies {
    compile 'org.cloudfoundry:cloudfoundry-client-reactor:2.0.0.BUILD-SNAPSHOT'
    compile 'org.cloudfoundry:cloudfoundry-operations:2.0.0.BUILD-SNAPSHOT'
    compile 'io.projectreactor:reactor-core:2.5.0.BUILD-SNAPSHOT'
    compile 'io.projectreactor:reactor-netty:2.5.0.BUILD-SNAPSHOT'
    ...
}
```

The artifacts can be found in the Spring release and snapshot repositories:

```gradle
repositories {
    maven { url 'http://repo.spring.io/release' }
    ...
}

repositories {
    maven { url 'http://repo.spring.io/snapshot' }
    ...
}
```

Sample Code

The following is a very simple sample application that connects to a Cloud Foundry instance, logs in, and displays some information about the Cloud Foundry account. When running the program, provide the Cloud Foundry target (e.g. `https://api.run.pivotal.io`) along with a valid user name and password as command-line parameters.
import org.cloudfoundry.client.lib.CloudCredentials;
import org.cloudfoundry.client.lib.CloudFoundryClient;
import org.cloudfoundry.client.lib.domain.CloudApplication;
import org.cloudfoundry.client.lib.domain.CloudService;
import org.cloudfoundry.client.lib.domain.CloudSpace;
import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;

public final class JavaSample {
    public static void main(String[] args) {
        String target = args[0];
        String user = args[1];
        String password = args[2];
        CloudCredentials credentials = new CloudCredentials(user, password);
        CloudFoundryClient client = new CloudFoundryClient(credentials, getTargetURL(target));
        client.login();
        System.out.printf("%nSpaces:%n");
        for (CloudSpace space : client.getSpaces()) {
            System.out.printf("%s\t(%s)%n", space.getName(), space.getOrganization().getName());
        }
        System.out.printf("%nApplications:%n");
        for (CloudApplication application : client.getApplications()) {
            System.out.printf("%s\n", application.getName());
        }
        System.out.printf("%nServices%n");
        for (CloudService service : client.getServices()) {
            System.out.printf("%s\t(%s)%n", service.getName(), service.getLabel());
        }
    }
    private static URL getTargetURL(String target) {
        try {
            return URI.create(target).toURL();
        } catch (MalformedURLException e) {
            throw new RuntimeException("The target URL is not valid: " + e.getMessage());
        }
    }
}

For more details about the Cloud Foundry Java Client Library, visit the source repository on GitHub. The domain package shows the objects that you can query and inspect.
Build Tool Integration

This page assumes you are using version 1.1.2 of either the Cloud Foundry Maven plugin or the Cloud Foundry Gradle plugin.

Maven Plugin

The Cloud Foundry Maven plugin allows you to deploy and manage applications with Maven goals. This plugin provides Maven users with access to the core functionality of the Cloud Foundry cf command-line tool.

Basic Configuration

To install the Cloud Foundry Maven plugin, add the `<cf-maven-plugin>` to the `<plugins>` section of the `pom.xml` file:

```xml
<plugins>
  <plugin>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cf-maven-plugin</artifactId>
    <version>1.1.2</version>
  </plugin>
</plugins>
```

This minimal configuration is sufficient to execute many of the Maven goals provided by the plugin, as long as you provide all other necessary configuration information through command-line parameters.

Additional Configuration

Instead of relying on command-line parameters, you can include additional configuration information in the `pom.xml` by nesting a `<configuration>` section within the `<cf-maven-plugin>` section.

Example:

```xml
<plugins>
  <plugin>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cf-maven-plugin</artifactId>
    <version>1.1.2</version>
    <configuration>
      <target>http://api.run.pivotal.io</target>
      <org>mycloudfoundry-org</org>
      <space>development</space>
      <appname>my-app</appname>
      <url>my-app.shared-domain.example.com</url>
      <memory>512</memory>
      <instances>2</instances>
      <env>
        <ENV-VAR-NAME>env-var-value</ENV-VAR-NAME>
      </env>
      <services>
        <service>
          <name>my-rabbitmq</name>
          <label>rabbitmq</label>
          <provider>rabbitmq</provider>
          <version>n/a</version>
          <plan>small_plan</plan>
        </service>
      </services>
    </configuration>
  </plugin>
</plugins>
```

After adding and configuring the plugin you can build and push the application to Cloud Foundry with the following command:

```
mvn clean package cf:push
```
Security Credentials

While you can include Cloud Foundry security credentials in the `pom.xml` file, a more secure method is to store the credentials in the Maven `settings.xml` file, using the server XML configuration element (http://maven.apache.org/settings.html#Servers). The default location for this configuration file is `~/.m2/settings.xml`.

To implement this:

1. Add a server to the servers section of the `settings.xml` file. Include the Cloud Foundry security credentials (username and password) and an ID tag. The `pom.xml` references this ID to access the security credentials.

   ```xml
   <settings>
     ...
     <servers>
       ...
       <server>
         <id>cloud-foundry-credentials</id>
         <username>my-name@example.com</username>
         <password>s3cr3t</password>
       </server>
       ...
     </servers>
     ...
   </settings>
   ``

2. Add a server configuration element referencing the ID to the `pom.xml` file:

   ```xml
   <plugins>
     <plugin>
       <groupId>org.cloudfoundry</groupId>
       <artifactId>cf-maven-plugin</artifactId>
       <version>1.1.2</version>
       <configuration>
         <server>cloud-foundry-credentials</server>
       </configuration>
     </plugin>
   </plugins>
   ```

Command-Line Usage

Key functionality available with the Cloud Foundry Maven plugin:

<table>
<thead>
<tr>
<th>Maven Goal</th>
<th>Cloud Foundry Command</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>cflogin</td>
<td>login -u USERNAME</td>
<td>$ mvn cflogin</td>
</tr>
<tr>
<td>cflogout</td>
<td>logout</td>
<td>$ mvn cflogout</td>
</tr>
<tr>
<td>cfapp</td>
<td>app APPNAME</td>
<td>$ mvn cfapp [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cfapps</td>
<td>apps</td>
<td>$ mvn cfapps</td>
</tr>
<tr>
<td>cf:target</td>
<td>api</td>
<td>$ mvn cf:target</td>
</tr>
<tr>
<td>cf:start</td>
<td>start APPNAME</td>
<td>$ mvn cf:start [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cf:stop</td>
<td>stop APPNAME</td>
<td>$ mvn cf:stop [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cf:restart</td>
<td>restart APPNAME</td>
<td>$ mvn cf:restart [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cf:delete</td>
<td>delete APPNAME</td>
<td>$ mvn cf:delete [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cf:scale</td>
<td>scale APPNAME -i INSTANCES</td>
<td>$ mvn cf:scale [-Dcf.appname=APPNAME] [-Dcf.instances=INTEGER]</td>
</tr>
<tr>
<td>cf:env</td>
<td>env APPNAME</td>
<td>$ mvn cf:env [-Dcf.appname=APPNAME]</td>
</tr>
<tr>
<td>cf:services</td>
<td>services</td>
<td>$ mvn cf:services</td>
</tr>
<tr>
<td>cf:create-services</td>
<td>create-service SERVICE PLAN SERVICE_INSTANCE</td>
<td>$ mvn cf:create-services</td>
</tr>
<tr>
<td>cf:delete-services</td>
<td>delete-service SERVICE_INSTANCE</td>
<td>$ mvn cf:delete-service</td>
</tr>
</tbody>
</table>
Gradle Plugin

The Cloud Foundry Gradle plugin allows you to deploy and manage applications with Gradle tasks. This plugin provides Gradle users with access to the core functionality of the Cloud Foundry cf command-line tool.

Basic Configuration

To install the Cloud Foundry Gradle plugin, add the cf-gradle-plugin as a dependency in the buildscript section of the build.gradle file:

buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'org.cloudfoundry:cf-gradle-plugin:1.1.2'
    }
}

apply plugin: 'cloudfoundry'

This minimal configuration is sufficient to execute many of the Gradle tasks provided by the plugin, as long as you provide all other necessary configuration information through command-line parameters.

Additional Configuration

Instead of relying on command-line parameters, you can add additional configuration information to build.gradle in a cloudfoundry configuration section:

cloudfoundry {
    target = "https://api.run.pivotal.io"
    space = "deployment"
    file = file("path/to/my/file.war")
    uri = "my-app.shared-domain.example.com"
    memory = 512
    instances = 1
    env = {
        key: "value"
    }
    services {
        my_rabbitmq {
            label = "rabbitmq"
            plan = "small_plan"
            bind = true
        }
    }
}

After adding and configuring the plugin you can build and push the application to Cloud Foundry with the following command:

$ gradle clean assemble cfPush

Security Credentials

While you can include Cloud Foundry security credentials in the build.gradle file, a more secure method is to store the credentials in a gradle.properties file. This file can be placed in either the project directory or in the gradle directory.

To implement this, add cfUsername and cfPassword with the Cloud Foundry security credentials parameters to the gradle.properties file as follows:

cfUsername=user@example.com
cfPassword=examplePassword
### Command-Line Usage

Key functionality available with the Cloud Foundry Gradle plugin:

<table>
<thead>
<tr>
<th>Gradle Task</th>
<th>Cloud Foundry Command</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfLogin</td>
<td>login -u USERNAME</td>
<td><code>$ gradle cfLogin</code></td>
</tr>
<tr>
<td>cfLogout</td>
<td>logout</td>
<td><code>$ gradle cfLogout</code></td>
</tr>
<tr>
<td>cfApp</td>
<td>app APPNAME</td>
<td><code>$ gradle cfApp [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfApps</td>
<td>apps</td>
<td><code>$ gradle cfApps</code></td>
</tr>
<tr>
<td>cfTarget</td>
<td>api</td>
<td><code>$ gradle cfTarget</code></td>
</tr>
<tr>
<td>cfStart</td>
<td>start APPNAME</td>
<td><code>$ gradle cfStart [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfStop</td>
<td>stop APPNAME</td>
<td><code>$ gradle cfStop [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfRestart</td>
<td>restart APPNAME</td>
<td><code>$ gradle cfStop [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfDelete</td>
<td>delete APPNAME</td>
<td><code>$ gradle cfDelete [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfScale</td>
<td>scale APPNAME -i INSTANCES</td>
<td><code>$ gradle cfScale [-PcfApplication=APPNAME] [-PcfInstances=INTEGER]</code></td>
</tr>
<tr>
<td>cfEnv</td>
<td>env APPNAME</td>
<td><code>$ gradle cfEnv [-PcfApplication=APPNAME]</code></td>
</tr>
<tr>
<td>cfServices</td>
<td>services</td>
<td><code>$ gradle cfServices</code></td>
</tr>
<tr>
<td>cfCreateService</td>
<td>create-service SERVICE PLAN SERVICE_INSTANCE</td>
<td><code>$ gradle cfCreateService</code></td>
</tr>
<tr>
<td>cfDeleteServices</td>
<td>delete-service SERVICE_INSTANCE</td>
<td><code>$ gradle cfDeleteServices</code></td>
</tr>
<tr>
<td>cfBind</td>
<td>bind-service APPNAME SERVICE_INSTANCE</td>
<td><code>$ gradle cfBind</code></td>
</tr>
<tr>
<td>cfUnbind</td>
<td>unbind-service APPNAME SERVICE_INSTANCE</td>
<td><code>$ gradle cfUnbind</code></td>
</tr>
</tbody>
</table>
BOSH Custom Trusted Certificate Support

Page last updated:

Configure

Java Buildpack versions 3.7 and later support BOSH configured custom trusted certificates.

Run the following command to configure support for this feature:

```bash
cf set-env JBP_CONFIG_CONTAINER_CERTIFICATE_TRUST_STORE '{enabled: true}'
```

Alternatively, you can modify the buildpack by setting the `enabled` property to true in `config/container_certificate_trust_store.yml`.

For more information see the official Java Buildpack documentation for this feature.
Ruby Buildpack

Page last updated:

Push Apps

This buildpack will be used if your app has a `Gemfile` and `Gemfile.lock` in the root directory. It will then use Bundler to install your dependencies.

If your Cloud Foundry deployment does not have the Ruby Buildpack installed, or the installed version is out of date, you can use the latest version with the command:

cf push my_app -b https://github.com/cloudfoundry/ruby-buildpack.git

For more detailed information on deploying Ruby applications see the following topics:

- Getting Started Deploying Ruby Apps
- Getting Started Deploying Ruby on Rails Apps
- Deploying a Sample Ruby on Rails App
- Configuring Rake Tasks for Deployed Apps
- Tips for Ruby Developers
- Environment Variables Defined by the Ruby Buildpack
- Configuring Service Connections for Ruby

You can find the source for the buildpack on GitHub: https://github.com/cloudfoundry/cf-buildpack-ruby

Supported Versions

Supported Ruby versions can be found in the release notes.

Specify a Ruby Version

Specific versions of the Ruby runtime can be specified in the `Gemfile`:

**MRI**

For MRI you can specify the version of Ruby by doing the following:

`ruby '2.2.3'`

Beginning in Ruby Buildpack v1.6.18, Rubygem's version operators are supported for the `ruby` directive. For example, the `~>` pessimistic operator is also supported:

`ruby '~> 2.2.3'`

With this example declaration in the `Gemfile`, if Ruby versions 2.2.4, 2.2.5, and 2.3.0 are present in the Ruby buildpack, the app will use Ruby 2.2.5.

For more information on the `ruby` directive for Bundler Gemfiles, see Bundler’s documentation.

**JRuby**

For JRuby you can specify the version of ruby by doing the following:
JRuby version 1.7.x supports either 1.9 mode, e.g.:

```
ruby '1.9.3', :engine => 'jruby', :engine_version => '1.7.25'
```

or 2.0 mode, e.g.:

```
ruby '2.0.0', :engine => 'jruby', :engine_version => '1.7.25'
```

For Jruby version >= 9.0:

```
ruby '2.2.3', :engine => 'jruby', :engine_version => '9.0.5.0'
```

The buildpack only supports the stable Ruby versions, which are listed in the `manifest.yml` and releases page.

If you try to use a binary that is not currently supported, staging your app will fail and you will see the following error message:

```
Could not get translated url, exited with: DEPENDENCY_MISSING_IN_MANIFEST: ...
```

Additionally, note that the pessimistic version operator (~>) on the Gemfile `ruby` directive for JRuby is not supported by the Ruby buildpack.

### Vendor App Dependencies

As stated in the Disconnected Environments documentation, your application must ‘vendor’ its dependencies.

For the Ruby buildpack, use bundler:

```
cd <your app dir>
bundle package --all
```

`cf push` uploads your vendored dependencies. The buildpack will compile any dependencies requiring compilation while staging your application.

### Buildpack Logging and Application Logging

The buildpack only runs during the staging process, and only logs what is important to staging, such as what is being downloaded, what the configuration is, and work that the buildpack does on your application.

The buildpack stops logging when the staging process finishes. The Loggregator handles application logging.

Your application must write to STDOUT or STDERR for its logs to be included in the Loggregator stream. For more information, see the Application Logging in Cloud Foundry topic.

If you are deploying a Rails application, the buildpack may or may not automatically install the necessary plugin or gem for logging, depending on the Rails version of the application:

- **Rails 2.x**: The buildpack automatically installs the `rails_log_stdout` plugin into the application. For more information about the `rails_log_stdout` plugin, refer to the Github README.
- **Rails 3.x**: The buildpack automatically installs the `rails_12factor` gem if it is not present and issues a warning message. You must add the `rails_12factor` gem to your `Gemfile` to quiet the warning message. For more information about the `rails_12factor` gem, refer to the Github README.
- **Rails 4.x**: The buildpack only issues a warning message that the `rails_12factor` gem is not present, but does not install the gem. You must add the `rails_12factor` gem to your `Gemfile` to quiet the warning message. For more information about the `rails_12factor` gem, refer to the Github README.

For more information about the `rails_12factor` gem, refer to the Github README.
Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the Proxy Usage Docs.

BOSH Configured Custom Trusted Certificate Support

Ruby uses certificates stored in `/etc/ssl/certs` and supports BOSH configured custom trusted certificates out of the box.

Help and Support

Join the #buildpacks channel in our Slack community if you need any further assistance.

For more information about using and extending the Ruby buildpack in Cloud Foundry, see the ruby-buildpack GitHub repo.

You can find current information about this buildpack on the Ruby buildpack release page in GitHub.
Getting Started Deploying Ruby Apps

This guide is intended to walk you through deploying a Ruby app to Elastic Runtime. If you experience a problem following the steps below, check the Known Issues topic, or refer to the Troubleshooting Application Deployment and Health topic.

**Sample App Step**

If you want to go through this tutorial using the sample app, run `git clone https://github.com/cloudfoundry-samples/pong_matcher_ruby.git` to clone the `pong_matcher_ruby` app from GitHub, and follow the instructions in the Sample App Step sections.

**Note:** Ensure that your Ruby app runs locally before continuing with this procedure.

Deploy a Ruby Application

This section describes how to deploy a Ruby application to Elastic Runtime, and uses output from a sample app to show specific steps of the deployment process.

**Prerequisites**

- A Ruby 2.x application that runs locally on your workstation
- Bundler configured on your workstation
- Basic to intermediate Ruby knowledge
- The [Cloud Foundry Command Line Interface (cf CLI)](https://github.com/cloudfoundry/cli) installed on your workstation

**Step 1: Create and Bind a Service Instance for a Ruby Application**

This section describes using the CLI to configure a Redis Cloud managed service instance for an app. You can use either the CLI or the [Apps Manager](https://github.com/cloudfoundry/apps-manager-cli) to perform this task.

Elastic Runtime supports two types of service instances:

- Managed services integrate with Elastic Runtime through service brokers that offer services and plans and manage the service calls between Elastic Runtime and a service provider.
- User-provided service instances enable you to connect your application to pre-provisioned external service instances.

For more information about creating and using service instances, refer to the [Services Overview](https://github.com/cloudfoundry/apps-manager-cli) topic.

**Create a Service Instance**

Run `cf marketplace` to view managed and user-provided services and plans that are available to you.

The example shows three of the available managed database-as-a-service providers and the plans that they offer: `cleardb` MySQL, `elephantsql` PostgreSQL as a Service, and `mongolab` MongoDB-as-a-Service.
$ cf marketplace
Getting services from marketplace in org Cloud-Apps / space development as clouduser@example.com...
OK

<table>
<thead>
<tr>
<th>service</th>
<th>plans</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardb</td>
<td>spark, boost, amp, shock</td>
<td>Highly available MySQL for your Apps</td>
</tr>
<tr>
<td>elephantsql</td>
<td>turtle, panda, hippo, elephant</td>
<td>PostgreSQL as a Service</td>
</tr>
<tr>
<td>mongolab</td>
<td>sandbox</td>
<td>Fully-managed MongoDB-as-a-Service</td>
</tr>
</tbody>
</table>

Run `cf create-service SERVICE PLAN SERVICE_INSTANCE` to create a service instance for your app. Choose a SERVICE and PLAN from the list, and provide a unique name for the SERVICE_INSTANCE.

Sample App Step

Run `cf create-service rediscloud 30mb redis`.

Creating service redis in org Cloud-Apps / space development as clouduser@example.com...
OK

Bind a Service Instance

When you bind an app to a service instance, Elastic Runtime writes information about the service instance to the VCAP_SERVICES app environment variable. The app can use this information to integrate with the service instance.

Most services support bindable service instances. Refer to your service provider’s documentation to confirm if they support this functionality.

You can bind a service to an application with the command `cf bind-service APPLICATION SERVICE_INSTANCE`.

Alternately, you can configure the deployment manifest file by adding a `services` block to the `applications` block and specifying the service instance. For more information and an example on service binding using a manifest, see the Sample App Step.

You can also bind a service using the Apps Manager.

Sample App Step

You can skip this step. The manifest for the sample app contains a `services` sub-block in the `applications` block, as the example below shows. This binds the `redis` service instance that you created in the previous step.

services:
- redis

Step 2: Configure Deployment Options

Configure the Deployment Manifest

You can specify app deployment options in a manifest that the `cf push` command uses. For more information about application manifests and supported attributes, refer to the Deploying with Application Manifests topic.

Configure a Production Server
Elastic Runtime uses the default standard Ruby web server library, WEBrick, for Ruby and RoR apps. However, Elastic Runtime can support a more robust production web server, such as Phusion Passenger, Puma, Thin, or Unicorn. If your app requires a more robust web server, refer to the "Configuring a Production Server" topic for help configuring a server other than WEBrick.

Sample App Step
You can skip this step. The `manifest.yml` file for `pong_matcher_ruby` does not require any additional configuration to deploy the app.

Step 3: Log in and Target the API Endpoint

Run `cf login -a API_ENDPOINT`, enter your login credentials, and select a space and org. The API endpoint is the URL of the Cloud Controller in your Elastic Runtime instance.

Sample App Step
You must do this step to run the sample app.

Step 4: Deploy an App

Note: You must use the cf CLI to deploy apps.

From the root directory of your application, run `cf push APP_NAME` to deploy your application.

`cf push APP_NAME` creates a URL route to your application in the form HOST.DOMAIN, where HOST is your APP_NAME and DOMAIN is specified by your administrator. Your DOMAIN is `shared-domain.example.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words.
- `cf help push` to view other options for this command.

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs APP_NAME`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step
Run `cf push pong_matcher_ruby -n HOST_NAME`.

Example: `cf push pong_matcher_ruby -n pongmatcher-ex12`

The example below shows the terminal output of deploying the `pong_matcher_ruby` app. `cf push` uses the instructions in the manifest file to create the app, create and bind the route, and upload the app. It then binds the app to the `redis` service and follows the instructions in the manifest to start one instance of the app with 256M. After the app starts, the output displays the health and status of the app.

Note: The `pong_matcher_ruby` app does not include a web interface. To interact with the `pong_matcher_ruby` app, see the interaction instructions on Github: `https://github.com/cloudfoundry-samples/pong_matcher_ruby`.
Step 5: Test a Deployed App

You've deployed an app to Elastic Runtime!

Use the cf CLI or the Apps Manager to review information and administer your app and your Elastic Runtime account. For example, you could edit the `manifest.yml` to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

See the Manage Your Application with the cf CLI section for more information. See also Using the Apps Manager.

Manage Your Application with the cf CLI

Run `cf help` to view a complete list of commands, grouped by task categories, and run `cf help COMMAND` for detailed information about a specific command. For more information about using the cf CLI, refer to the Cloud Foundry Command Line Interface (cf CLI) topics, especially the Getting Started with cf CLI v6 topic.

Note: You cannot perform certain tasks in the CLI or the Apps Manager because these are commands that only a Elastic Runtime administrator can run. If you are not a Elastic Runtime administrator, the following message displays for these types of commands:

```
error code: 10003, message: You are not authorized to perform the requested action
```

For more information about specific Admin commands you can perform with the Apps Manager, depending on your user role, refer to the Getting Started with the Apps Manager topic.

Troubleshooting

If your application fails to start, verify that the application starts in your local environment. Refer to the Troubleshooting Application Deployment and Health topic to learn more about troubleshooting.
App Deploy Fails

Even when deploying an app fails, the app might exist on Elastic Runtime. Run `cf apps` to review the apps in the currently targeted org and space. You might be able to correct the issue using the CLI or the Apps Manager, or you might have to delete the app and redeploy.

Common reasons deploying an app fails include:

- You did not successfully create and bind a needed service instance to the app, such as a PostgreSQL or MongoDB service instance. Refer to Step 2: Create and Bind a Service Instance for a Ruby Application.
- You did not successfully create a unique URL for the app. Refer to the troubleshooting tip App Requires Unique URL.

App Requires Unique URL

Elastic Runtime requires that each app that you deploy has a unique URL. Otherwise, the new app URL collides with an existing app URL and Elastic Runtime cannot successfully deploy the app. You can resolve this issue by running `cf push` with either of the following flags to create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words. Using this option might create a long URL, depending on the number of words that the app name includes.
Getting Started Deploying Ruby on Rails Apps

This guide walks you through deploying a Ruby on Rails (RoR) app to Elastic Runtime. To deploy a sample RoR app, refer to the Deploy a Sample Ruby on Rails App topic.

Note: Ensure that your RoR app runs locally before continuing with this procedure.

Prerequisites

- A Rails 4.x app that runs locally
- Bundler configured on your workstation
- Intermediate to advanced RoR knowledge
- The Cloud Foundry Command Line Interface (cf CLI)

Step 1: Create and Bind a Service Instance for a RoR Application

This section describes using the CLI to configure a PostgreSQL managed service instance for an app. For more information about creating and using service instances, refer to the Services Overview topic.

Create a Service Instance

Run `cf marketplace` to view managed and user-provided services and plans available to you.

Run `cf create-service SERVICE PLAN SERVICE_INSTANCE` to create a service instance for your app. Choose a SERVICE and PLAN from the list, and provide a unique name for the SERVICE_INSTANCE.

Bind a Service Instance

When you bind an app to a service instance, Elastic Runtime writes information about the service instance to the VCAP_SERVICES app environment variable. The app can use this information to integrate with the service instance.

Most services support bindable service instances. Refer to your service provider's documentation to confirm whether they support this functionality.

To bind a service to an application, run `cf bind-service APPLICATION SERVICE_INSTANCE`.

Step 2: Configure Deployment Options

Configure the Deployment Manifest

You can specify app deployment options in a manifest that the `cf push` command uses. For more information about application manifests and supported attributes, refer to the Deploying with Application Manifests topic.

Configure a Production Server

Elastic Runtime uses the default standard Ruby web server library, WEBrick, for Ruby and RoR apps. However, Elastic Runtime can support a more robust production web server, such as Phusion Passenger, Puma, Thin, or Unicorn. If your app requires a more robust web server, refer to the Configuring a Production Server topic.

© Copyright Pivotal Software Inc, 2013-2018
Step 3: Log in and Target the API Endpoint

Run `cf login -a API_ENDPOINT`, enter your login credentials, and select a space and org. The API endpoint is the URL of the Cloud Controller in your Elastic Runtime instance.

Step 4: Deploy Your App

From the root directory of your application, run `cf push APP-NAME --random-route` to deploy your application.

`cf push APP-NAME` creates a URL route to your application in the form HOST.DOMAIN, where HOST is your APP-NAME and DOMAIN is specified by your administrator. Your DOMAIN is shared-domain.example.com. For example: `cf push my-app` creates the URL my-app.shared-domain.example.com.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words.
- `cf help push` to view other options for this command.

To view log activity while the app deploys, launch a new terminal window and run `cf logs APP-NAME`.

In the terminal output of the `cf push command, the urls field of the App started block contains the app URL. This is the HOST_NAME you specified with the `-n` flag plus the domain shared-domain.example.com. Once your app deploys, use this URL to access the app online.

Next Steps

You’ve deployed an app to Elastic Runtime! Consult the sections below for information about what to do next.

Test a Deployed App

Use the cf CLI or the Apps Manager to review information and administer your app and your Elastic Runtime account. For example, you could edit the `manifest.yml` to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

Manage Your Application with the cf CLI

Run `cf help` to view a complete list of commands, grouped by task categories, and run `cf help COMMAND` for detailed information about a specific command. For more information about using the cf CLI, refer to the Cloud Foundry Command Line Interface (cf CLI) topics, especially the Getting Started with cf CLI v6 topic.

Troubleshooting

If your application fails to start, verify that the application starts in your local environment. Refer to the Troubleshooting Application Deployment and Health topic to learn more about troubleshooting.
Deploy a Sample Ruby on Rails Application

This topic guides the reader through deploying a sample Ruby on Rails app to Elastic Runtime.

Prerequisites

In order to deploy a sample Ruby on Rails app, you must have the following:

- A working PCF deployment
- Cloud Foundry CLI
- Cloud Foundry username and password with Space Developer permissions. See your Org Manager if you require permissions.

Step 1: Clone the App

Run the following terminal command to create a local copy of the rails_sample_app.

```bash
$ git clone https://github.com/cloudfoundry-samples/rails_sample_app
```

The newly created directory contains a `manifest.yml` file, which assists CF with deploying the app. See Deploying with Application Manifests for more information.

Step 2: Log in and Target the API Endpoint

1. Run the following terminal command to log in and target the API endpoint of your deployment. For more information, see the Identifying the API Endpoint for your Elastic Runtime Instance topic.

   ```bash
   $ cf login -a YOUR-API-ENDPOINT
   
   $ cf target
   
   $ cf app rails_sample
   
   $ cf services
   
   $ cf_applications
   
   $ cf manifests
   ```

2. Use your credentials to log in, and to select a Space and Org.

   - Note: The API endpoint must be entered in the format `https://api.IP-ADDRESS`, where IP-ADDRESS is the IP address of your API endpoint.

Step 3: Create a Service Instance

Run the following terminal command to create a PostgreSQL service instance for the sample app. Our service instance is `rails-postgres`. It uses the `elephantsql` service and the `turtle` plan.

```bash
$ cf create-service elephantsql turtle rails-postgres
```

The manifest for the rails_sample_app contains a `services` sub-block in the `applications` block. The Cloud Foundry Command Line Interface tool (cf CLI) binds the service to the app.

```yaml
---

applications:
  - name: rails-sample
    memory: 256M
    instances: 1
    path:
      command: bundle exec rake db:migrate && bundle exec rails s -p $PORT
    services:
      - rails-postgres
```
Step 4: Deploy the App

Make sure you are in the rails_sample_app directory. Run the following terminal command to deploy the app:

```
cf push rails_sample_app
```

cf push rails_sample_app creates a URL route to your application in the form HOST.DOMAIN. In this example, HOST is rails_sample_app. Administrators specify the DOMAIN. For example, for the DOMAIN shared-domain.example.com, running cf push rails_sample_app creates the URL rails_sample_app.shared-domain.example.com.

The example below shows the terminal output when deploying the rails_sample_app. cf push uses the instructions in the manifest file to create the app, create and bind the route, and upload the app. It then binds the app to the rails-postgres service and follows the information in the manifest to start one instance of the app with 256M of RAM. After the app starts, the output displays the health and status of the app.

```
cf push rails_sample_app
Using manifest file ~/workspace/rails_sample_app/manifest.yml
Updating app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...OK
Using route rails_sample_app.shared-domain.example.com
Uploading rails_sample_app...
Uploading app files from: ~/workspace/rails_sample_app
Uploading 445.7K, 217 files
OK
Binding service rails-postgres to app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...OK
Starting app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...OK

0 of 1 instances running, 1 starting
1 of 1 instances running

App started

Showing health and status for app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: rails_sample_app.shared-domain.example.com

state since cpu memory disk
  running 2014-08-25 03:32:10 PM 0.0% 68.4M of 256M 73.4M of 1G
```

Note: If you want to view log activity while the app deploys, launch a new terminal window and run cf logs rails_sample_app.

Step 5: Verify the App

Verify that the app is running by browsing to the URL generated in the output of the previous step. In this example, navigating to rails_sample_app.shared-domain.example.com verifies that the app is running.

You’ve now pushed an app to Elastic Runtime! For more information on this topic, see the Deploy an Application topic.
Configure Rake Tasks for Deployed Apps

Page last updated:

For Elastic Runtime to automatically invoke a Rake task while a Ruby or Ruby on Rails app is deployed, you must:

- Include the Rake task in your app.
- Configure the application start command using the `command` attribute in the application manifest.

The following is an example of how to invoke a Rake database migration task at application startup.

1. Create a file with the Rake task name and the extension `.rake`, and store it in the `lib/tasks` directory of your application.

2. Add the following code to your rake file:

```ruby
namespace :cf do
desc "only run on the first application instance"
task :on_first_instance do
  instance_index = JSON.parse(ENV['VCAP_APPLICATION'])['instance_index'] rescue nil
  exit(0) unless instance_index == 0
end
end
```

This Rake task limits an idempotent command to the first instance of a deployed application.

3. Add the task to the `manifest.yml` file with the `command` attribute, referencing the idempotent command `rake db:migrate` chained with a start command.

```yaml
applications:
  - name: my-rails-app
    command: bundle exec rake cf:on_first_instance db:migrate && rails s -p $PORT
```
Tips for Ruby Developers

Page last updated:

This page has information specific to deploying Rack, Rails, or Sinatra applications.

Application Bundling

You must run Bundler to create a Gemfile and a Gemfile.lock. These files must be in your application before you push to Cloud Foundry.

Rack Config File

For Rack and Sinatra, you must have a config.ru file. For example:

```ruby
require './hello_world'
run HelloWorld.new
```

Asset Precompilation

Cloud Foundry supports the Rails asset pipeline. If you do not precompile assets before deploying your application, Cloud Foundry will precompile them when staging the application. Precompiling before deploying reduces the time it takes to stage an application.

Use the following command to precompile assets before deployment:

```
rake assets:precompile
```

Note that the Rake precompile task reinitializes the Rails application. This could pose a problem if initialization requires service connections or environment checks that are unavailable during staging. To prevent reinitialization during precompilation, add the following line to application.rb:

```ruby
config.assets.initialize_on_precompile = false
```

If the assets:precompile task fails, Cloud Foundry uses live compilation mode, the alternative to asset precompilation. In this mode, assets are compiled when they are loaded for the first time. You can force live compilation by adding the following line to application.rb:

```ruby
Rails.application.config.assets.compile = true
```

Running Rake Tasks

Cloud Foundry does not provide a mechanism for running a Rake task on a deployed application. If you need to run a Rake task that must be performed in the Cloud Foundry environment, rather than locally before deploying or redeploying, you can configure the command that Cloud Foundry uses to start the application to invoke the Rake task.

An application's start command is configured in the application's manifest file, manifest.yml, using the command attribute.

If you have previously deployed the application, the application manifest should already exist. There are two ways to create a manifest. You can manually create the file and save it in the application's root directory before you deploy the application for the first time. If you do not manually create the manifest file, the cf CLI will prompt you to supply deployment settings when you first push the application, and will create and save the manifest file for you, with the settings you specified interactively. For more information about application manifests, and supported attributes, see Deploying with Application Manifests.

Example: Invoking a Rake database migration task at application startup

The following is an example of the “migrate frequently” method described in the Migrating a Database in Cloud Foundry topic.
1. Create a Rakefile if one does not already exist, and add it to your application directory.

2. In your Rakefile, add a Rake task to limit an idempotent command to the first instance of a deployed application:

```ruby
namespace :cf do
desc "Only run on the first application instance"
task :on_first_instance do
  instance_index = JSON.parse(ENV['VCAP_APPLICATION'])['instance_index']
  rescue nil
  exit(0) unless instance_index == 0
end
end
```

3. Add the task to the `manifest.yml` file, referencing the idempotent command ` rake db:migrate` with the `command` attribute.

```yaml
applications:
- name: my-rails-app
  command: bundle exec rake cf:on_first_instance db:migrate && bundle exec rails s -p $PORT -e $RAILS_ENV
```

4. Update the application using `cf push`.

Rails 3 Worker Tasks

This section shows you how to create and deploy an example Rails application that uses a worker library to defer a task that a separate application executes.

The guide also describes how to scale the resources available to the worker application.

**Note:** Most worker tasks do not serve external requests. Use the `--no-route` flag with the `cf push` command, or `no-route: true` in the application manifest, to suppress route creation and remove existing routes.

Choose a Worker Task Library

You must choose a worker task library. The table below summarizes the three main libraries available for Ruby / Rails:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delayed::Job</td>
<td>A direct extraction from <a href="https://www.shopify.com">Shopify</a> where the job table is responsible for a multitude of core tasks.</td>
</tr>
<tr>
<td>Resque</td>
<td>A Redis-backed library for creating background jobs, placing those jobs on multiple queues, and processing them later.</td>
</tr>
<tr>
<td>Sidekiq</td>
<td>Uses threads to handle many messages at the same time in the same process. It does not require Rails but will integrate tightly with Rails 3 to make background message processing dead simple. This library is also Redis-backed and is actually somewhat compatible with Resque messaging.</td>
</tr>
</tbody>
</table>

For other alternatives, see [https://www.ruby-toolbox.com/categories/Background_Jobs](https://www.ruby-toolbox.com/categories/Background_Jobs).

Create an Example Application

For the purposes of the example application, we will use Sidekiq.

First, create a Rails application with an arbitrary model called "Things":

```
$ rails create rails-sidekiq
$ cd rails-sidekiq
$ rails g model Thing title:string description:string
```

Add `sidekiq` and `uuidtools` to the Gemfile:
source https://rubygems.org/

gem 'rails', '3.2.9'
gem 'mysql2'

group :assets do
  gem 'sass-rails', '~> 3.2.3'
gem 'coffee-rails', '~> 3.2.1'
gem 'uglifier', '>= 1.0.3'
end

gem 'jquery-rails'
gem 'sidekiq'
gem 'uuidtools'

Install the bundle.

$ bundle install

Create a worker (in app/workers) for Sidekiq to carry out its tasks:

$ touch app/workers/thing_worker.rb

```ruby
class ThingWorker
  include Sidekiq::Worker

  def perform(count)
    count.times do
      thing_uuid = UUIDTools::UUID.random_create.to_s
      Thing.create(title: "New Thing (#{thing_uuid})", description: "Description for thing #{thing_uuid}")
    end
  end
end
```

This worker will create n number of things, where n is the value passed to the worker.

Create a controller for "Things":

$ rails g controller Thing

```ruby
class ThingController < ApplicationController
  def new
    ThingWorker.perform_async(2)
    redirect_to '/thing'
  end

  def index
    @things = Thing.all
  end
end
```

Add a view to inspect our collection of "Things":

$ mkdir app/views/things
$ touch app/views/things/index.html.erb

```
```

Deploy the Application

This application needs to be deployed twice for it to work, once as a Rails web application and once as a standalone Ruby application. The easiest way to
do this is to keep separate Cloud Foundry manifests for each application type:

Web Manifest: Save this as `web-manifest.yml`:

```yaml
---
applications:
- name: sidekiq
  memory: 256M
  instances: 1
  host: sidekiq
  domain: $target-base
  path:
  - services:
    - sidekiq-redis:
      - sidekiq-mysql:

Worker Manifest: Save this as `worker-manifest.yml`:

```yaml
---
applications:
- name: sidekiq-worker
  memory: 256M
  instances: 1
  path:
  command: bundle exec sidekiq
  no-route: true
  services:
  - sidekiq-redis:
    - sidekiq-mysql:

Since the url “sidekiq.cloudfoundry.com” is probably already taken, change it in `web-manifest.yml` first, then push the application with both manifest files:

```
cf push -f web-manifest.yml
```
```
cf push -f worker-manifest.yml
```

If the cf CLI asks for a URL for the worker application, select “none”.

Test the Application

Test the application by visiting the new action on the “Thing” controller at the assigned url. In this example, the URL would be

http://sidekiq.cloudfoundry.com/thing/new

This will create a new Sidekiq job which will be queued in Redis, then picked up by the worker application. The browser is then redirected to `thing` which will show the collection of “Things”.

Scale Workers

Use the `cf scale` command to change the number of Sidekiq workers.

Example:

```
cf scale sidekiq-worker +2
```

Use railsServeStaticAssets on Rails 4

By default Rails 4 returns a 404 if an asset is not handled via an external proxy such as Nginx. The `railsServeStaticAssets` gem enables your Rails server to deliver static assets directly, instead of returning a 404. You can use this capability to populate an edge cache CDN or serve files directly from your web application. The gem enables this behavior by setting the `config.serve_static_assets` option to `true`, so you do not need to configure it manually.
For information about using and extending the Ruby buildpack in Cloud Foundry, see the [ruby-buildpack GitHub repo](https://github.com/cloudfoundry/ruby-buildpack).

You can find current information about this buildpack on the [Ruby buildpack release page](https://github.com/cloudfoundry/ruby-buildpack) in GitHub.

The buildpack uses a default Ruby version of 2.2.2. To override this value for your app, add a Ruby declaration in the Gemfile. This also applies to using a JRuby interpreter.

## Environment Variables

You can access environments variable programmatically. For example, you can obtain `VCAP_SERVICES` as follows:

```
ENV['VCAP_SERVICES']
```

Environment variables available to you include both those defined by the system and those defined by the Ruby buildpack, as described below.

### BUNDLE_BIN_PATH

Location where Bundler installs binaries.

```
BUNDLE_BIN_PATH:/home/vcap/app/vendor/bundle/ruby/1.9.1/gems/bundler-1.3.2/bin/bundle
```

### BUNDLE_GEMFILE

Path to application’s Gemfile.

```
BUNDLE_GEMFILE:/home/vcap/app/Gemfile
```

### BUNDLE_WITHOUT

The `BUNDLE_WITHOUT` environment variable causes Cloud Foundry to skip installation of gems in excluded groups. `BUNDLE_WITHOUT` is particularly useful for Rails applications, where there are typically “assets” and “development” gem groups containing gems that are not needed when the app runs in production.

For information about using this variable, see [http://blog.cloudfoundry.com/2012/10/02/polishing-cloud-foundrys-ruby-gem-support](http://blog.cloudfoundry.com/2012/10/02/polishing-cloud-foundrys-ruby-gem-support).

```
BUNDLE_WITHOUT=assets
```

### DATABASE_URL

The Ruby buildpack looks at the database_uri for bound services to see if they match known database types. If there are known relational database services bound to the application, the buildpack sets up the DATABASE_URL environment variable with the first one in the list.

If your application depends on DATABASE_URL being set to the connection string for your service, and Cloud Foundry does not set it, you can set this variable manually.

```
$ cf set-env my_app_name DATABASE_URL mysql://b5d435f40dd2b2c6bf00ac@us-cdbr-east-03.cleardb.com:3306/ad_c6f444632610ab
```

### GEM_HOME

Location where gems are installed.

```
GEM_HOME:/home/vcap/app/vendor/bundle/ruby/1.9.1
```

### GEM_PATH

© Copyright Pivotal Software Inc, 2013-2018

626 1.7
Location where gems can be found.

```
GEM_PATH=/home/vcap/app/vendor/bundle/ruby/1.9.1:
```

**RACK_ENV**

This variable specifies the Rack deployment environment: development, deployment, or none. This governs what middleware is loaded to run the application.

```
RACK_ENV=production
```

**RAILS_ENV**

This variable specifies the Rails deployment environment: development, test, or production. This controls which of the environment-specific configuration files will govern how the application will be executed.

```
RAILS_ENV=production
```

**RUBYOPT**

This Ruby environment variable defines command-line options passed to Ruby interpreter.

```
RUBYOPT: -I/home/vcap/app/vendor/bundle/ruby/1.9.1/gems/bundler-1.3.2/lib -rbundler/setup
```
Environment Variables Defined by the Ruby Buildpack

When you use the Ruby buildpack, you get three Ruby-specific environment variables in addition to the regular Cloud Foundry environment variables.

- **BUNDLE_BIN_PATH** — Location where Bundler installs binaries.
  - `BUNDLE_BIN_PATH:/home/vcap/app/vendor/bundle/ruby/1.9.1/gems/bundler-1.3.2/bin/bundle`

- **BUNDLE_GEMFILE** — Path to application’s gemfile.
  - `BUNDLE_GEMFILE:/home/vcap/app/Gemfile`

- **BUNDLE_WITHOUT** — This variable causes Cloud Foundry to skip installation of gems in excluded groups. Use this with Rails applications, where “assets” and “development” gem groups typically contain gems that are not needed when the app runs in production.
  - `BUNDLE_WITHOUT=assets`

- **DATABASE_URL** — The Ruby buildpack looks at the database_uri for bound services to see if they match known database types. If known relational database services are bound to the application, the buildpack sets up the `DATABASE_URL` environment variable with the first one in the list. If your application depends on `DATABASE_URL` being set to the connection string for your service, and Cloud Foundry does not set it, you can set this variable manually.
  - `cf set-env my_app_name DATABASE_URL mysql://b5d435f40dd2b2:ebfc00ac@us-cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab`

- **GEM_HOME** — Location where gems are installed.
  - `GEM_HOME:/home/vcap/app/vendor/bundle/ruby/1.9.1`

- **GEM_PATH** — Location where gems can be found.
  - `GEM_PATH=/home/vcap/app/vendor/bundle/ruby/1.9.1:`

- **RACK_ENV** — This variable specifies the Rack deployment environment: development, deployment, or none. This governs what middleware is loaded to run the application.
  - `RACK_ENV=production`

- **RAILS_ENV** — This variable specifies the Rails deployment environment: development, test, or production. This controls which of the environment-specific configuration files will govern how the application will be executed.
  - `RAILS_ENV=production`

- **RUBYOPT** — This Ruby environment variable defines command-line options passed to Ruby interpreter.
  - `RUBYOPT: -I/home/vcap/app/vendor/bundle/ruby/1.9.1/gems/bundler-1.3.2/lib:`
Configure Service Connections for Ruby

Page last updated:

After you create a service instance and bind it to an application, you must configure the application to connect to the service.

Query VCAP_SERVICES with cf-app-utils

The cf-apps-utils gem allows your application to search for credentials from the VCAP_SERVICES environment variable by name, tag, or label.

- cf-app-utils-ruby

VCAP_SERVICES defines DATABASE_URL

At runtime, the Ruby buildpack creates a DATABASE_URL environment variable for every Ruby application based on the VCAP_SERVICES environment variable.

Example VCAP_SERVICES:

```
VCAP_SERVICES = {
  "elephantsql": {
    "name": "elephantsql-c6c60",
    "label": "elephantsql",
    "credentials": {
      "uri": "postgres://exampleuser:examplepass@babar.elephantsql.com:5432/exampledb"
    }
  }
}
```

Based on this VCAP_SERVICES, the Ruby buildpack creates the following DATABASE_URL environment variable:

```
DATABASE_URL = postgres://exampleuser:examplepass@babar.elephantsql.com:5432/exampledb
```

The Ruby buildpack uses the structure of the VCAP_SERVICES environment variable to populate DATABASE_URL. Any service containing a JSON object with the following form will be recognized by Cloud Foundry as a candidate for DATABASE_URL:

```
{
  "some-service": {
    "credentials": {
      "uri": "<some database URL>"
    }
  }
}
```

Cloud Foundry uses the first candidate found to populate DATABASE_URL.

Older Rails Applications Have Auto-Configured database.yml

On Rails versions 4 or before, the Ruby buildpack replaces your database.yml with one based on the DATABASE_URL variable.

**Note:** On Rails versions 4 or before, Ruby buildpack ignores the contents of any database.yml that you provide and overwrites it during staging.

Configure Non-Rails Applications

© Copyright Pivotal Software Inc, 2013-2018 629 1.7
Non-Rails applications can also access the \texttt{DATABASE\_URL} variable.

If you have more than one service with credentials, only the first will be populated into \texttt{DATABASE\_URL}. To access other credentials, you can inspect the \texttt{VCAP\_SERVICES} environment variable.

\begin{verbatim}
vcap_services = JSON.parse(ENV['VCAP\_SERVICES'])
\end{verbatim}

Use the hash key for the service to obtain the connection credentials from \texttt{VCAP\_SERVICES}.

- For services that use the \texttt{v2 format}, the hash key is the name of the service.
- For services that use the \texttt{v1 format}, the hash key is formed by combining the service provider and version, in the format \texttt{PROVIDER-VERSION}.

For example, the service provider “p-mysql” with version “n/a” forms the hash key \texttt{p-mysql-n/a}.

### Seed or Migrate Database

Before you can use your database the first time, you must create and populate or migrate it. For more information, see \url{Migrating a Database in Cloud Foundry}.

### Troubleshooting

To aid in troubleshooting issues connecting to your service, you can examine the environment variables and log messages Cloud Foundry records for your application.

#### View Environment Variables

Use the \texttt{cf env} command to view the Cloud Foundry environment variables for your application. \texttt{cf env} displays the following environment variables:

- The \texttt{VCAP\_SERVICES} variables existing in the container environment
- The user-provided variables set using the \texttt{cf set-env} command

\begin{verbatim}
$ cf env my-app
Getting env variables for app my-app in org My-Org / space development as admin... OK.

System-Provided:

  "VCAP\_SERVICES": {
    "p-mysql-n/a": [
      {
        "credentials": {
          "uri": "postgres://lrra:e6B-X@p-mysqlprovider.example.com:5432/lraa",
          "label": "p-mysql-n/a",
          "name": "p-mysql",
          "syslog\_drain\_url": "",
          "tags": ["postgres", "postgresql", "relational"]
        }
      }
    ]
  }

User-Provided:

my-env-var: 100
my-drain: http://drain.example.com
\end{verbatim}

#### View Logs

Use the \texttt{cf logs} command to view the Cloud Foundry log messages for your application. You can direct current logging to standard output, or you can dump the most recent logs to standard output.

Run \texttt{cf logs APPNAME} to direct current logging to standard output:
Run `cf logs APPNAME --recent` to dump the most recent logs to standard output:

```
$ cf logs my-app
Connected, tailing logs for app my-app in org My-Org / space development as admin...
```

If you encounter the error, “A fatal error has occurred. Please see the Bundler troubleshooting documentation,” update your version of bundler and run `bundle install`.

```
$ gem update bundler
$ gem update --system
$ bundle install
```
Node.js Buildpack

Page last updated:

Push Node.js Apps

This buildpack is used automatically if there is a `package.json` file in the root directory of your project.

If your Cloud Foundry deployment does not have the Node.js buildpack installed or the installed version is out of date, you can use the latest version with the command:

```
bash
cf push my_app -b https://github.com/cloudfoundry/buildpack-nodejs.git
```

For more detailed information on deploying Node.js apps see the following topics:

- Tips for Node.js Developers
- Environment Variables Defined by the Node Buildpack
- Configuring Service Connections for Node

You can find the source for the buildpack on GitHub: [https://github.com/cloudfoundry/heroku-buildpack-nodejs](https://github.com/cloudfoundry/heroku-buildpack-nodejs)

Supported Versions

Supported Node versions can be found in the release notes.

Specify a Node.js Version

Set `engines.node` in `package.json` to the semver range or the specific version of node you are using:

```
"engines": {
  "node": "4.4.x"
}
```

or

```
"engines": {
  "node": "4.4.3"
}
```

If you try to use a version that is not currently supported, staging your app fails with the following error message:

```
$ Could not get translated url, exited with: DEPENDENCY_MISSING_IN_MANIFEST:...
$ ! exit
$ !
$ Staging failed: Buildpack compilation step failed
```

Specify an npm Version

Set `engines.npm` in `package.json` to the semver range, or specific version, of npm you are using:

```
"engines": {
  "node": "4.4.x",
  "npm": "2.15.x"
}
```

or

```
"engines": {
  "node": "4.4.3",
  "npm": "2.15.3"
}
```
If you do not specify an npm version, your app uses the default npm packaged with your app’s Node.js version.

If your environment cannot connect to the internet and a non-default version of npm is specified, the buildpack fails to download npm. You see the following error message:

```
$ We're unable to download the version of npm you've provided (...).
$ Please remove the npm version specification in package.json (...)
$ Staging failed: Buildpack compilation step failed
```

Vendor App Dependencies

As stated in the [Disconnected Environments documentation](#), your app must ‘vendor’ its dependencies.

For the Node.js buildpack, use `npm`:

```
cd <your app dir>
npm install
```

`cf push` uploads your vendored dependencies.

OpenSSL Support

Since November 2015, the [nodejs-buildpack](#) has been packaging binaries of Node.js with OpenSSL that are statically linked. With [community approval](#), it was decided to support Node.js 4.x and greater, which relied on the Node.js release cycle to provide OpenSSL updates.

The buildpack’s team [binary-builder](#) was updated to [enable the static OpenSSL compilation](#). All versions of Node.js compiled since have been statically linked with OpenSSL.

Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the [Proxy Usage Docs](#).

BOSH Configured Custom Trusted Certificate Support

Nodejs hardcodes root CA certs in its source code. To use [BOSH configured custom trusted certificates](#), a developer must pass the specified CAs to the `tls.connect` function as extra arguments.

Help and Support

Join the #buildpacks channel in our [Slack community](#) if you need any further assistance.

For more information about using and extending the Node.js buildpack in Cloud Foundry, see the [nodejs-buildpack GitHub repo](#).

You can find current information about this buildpack on the Node.js buildpack [release page](#) in GitHub.
Tips for Node.js Applications

Application Package File

Cloud Foundry expects a `package.json` in your Node.js app. You can specify the version of Node.js you want to use in the `engine` node of your `package.json` file.

In general, Cloud Foundry supports the two most recent versions of Node.js. See the GitHub Node.js buildpack page for current information.

Example `package.json` file:

```json
{
  "name": "first",
  "version": "0.0.1",
  "author": "Demo",
  "dependencies": {
    "express": "3.4.8",
    "consolidate": "1.0.0",
    "swig": "1.3.2"
  },
  "engines": {
    "node": "0.12.7",
    "npm": "2.7.4"
  }
}
```

Application Port

You must use the `PORT` environment variable to determine which port your app should listen on. In order to also run your app locally, you may want to make port 3000 the default:

```javascript
app.listen(process.env.PORT || 3000);
```

Application Start Command

Node.js apps require a start command. You can specify the web start command for a Node.js app in a Procfile or in the app deployment manifest. For more information about Procfiles, see the Configuring a Production Server topic.

The first time you deploy, you are asked if you want to save your configuration. This saves a `manifest.yml` in your app with the settings you entered during the initial push. Edit the `manifest.yml` file and create a start command as follows:

```yaml
---
applications:
  - name: my-app
    command: node my-app.js
    ... the rest of your settings ...
```

Alternately, specify the start command with `cf push`.

```
cf push -c "node my-app.js"
```
Application Bundling

You do not need to run `npm install` before deploying your app. Cloud Foundry runs it for you when your app is pushed. If you prefer to run `npm install` and create a `node_modules` folder inside of your app, this is also supported.

Solve Discovery Problems

If Cloud Foundry does not automatically detect that your app is a Node.js app, you can override the auto-detection by specifying the Node.js buildpack.

Add the buildpack into your `manifest.yml` and re-run `cf push` with your manifest:

```yaml
---
aplications:
  - name: my-app
    buildpack: https://github.com/cloudfoundry/nodejs-buildpack
    ... the rest of your settings ...
```

Alternately, specify the buildpack on the command line with `cf push -b`:

```
$ cf push my-app -b https://github.com/cloudfoundry/nodejs-buildpack
```

Bind Services

Refer to Configure Service Connections for Node.js.

About the Node.js Buildpack

For information about using and extending the Node.js buildpack in Cloud Foundry, see the nodejs-buildpack repo.

You can find current information about this buildpack on the Node.js buildpack release page in GitHub.

The buildpack uses a default Node.js version. To specify the versions of Node.js and npm an app requires, edit the app’s `package.json`, as described in “node.js and npm versions” in the nodejs-buildpack repo.

Environment Variables

You can access environments variable programmatically.

For example, you can obtain `VCAP_SERVICES` like this:

```
process.env.VCAP_SERVICES
```

Environment variables available to you include both those defined by the system and those defined by the Node.js buildpack, as described below.

**BUILD_DIR**

Directory into which Node.js is copied each time a Node.js app is run.

**CACHE_DIR**

Directory that Node.js uses for caching.
PATH

The system path used by Node.js.

PATH=/home/vcap/app/bin:/home/vcap/app/node_modules/bin:/bin:/usr/bin
Environment Variables Defined by the Node Buildpack

When you use the Node buildpack, you get three Node-specific environment variables in addition to the regular Cloud Foundry environment variables.

- **BUILD_DIR** — The directory into which Node.js is copied each time a Node.js application is run.
- **CACHE_DIR** — The directory that Node.js uses for caching.
- **PATH** — The system path used by Node.js:

  PATH=/home/vcap/app/bin:/home/vcap/app/node_modules/.bin:/bin:/usr/bin
Configuring Service Connections for Node.js

This guide is for developers who wish to bind a data source to a Node.js application deployed and running on Cloud Foundry.

Parse VCAP_SERVICES for Credentials

You must parse the VCAP_SERVICES environment variable in your code to get the required connection details such as host address, port, user name, and password.

For example, if you are using PostgreSQL, your VCAP_SERVICES environment variable might look something like this:

```
{
  "mypostgres": {
    "name": "myinstance",
    "credentials": {
      "uri": "postgres://myusername:mypassword@host.example.com:5432/serviceinstance"
    }
  }
}
```

This example JSON is simplified; yours may contain additional properties.

Parse with cfenv

The `cfenv` package provides access to Cloud Foundry application environment settings by parsing all the relevant environment. The settings are returned as JavaScript objects. `cfenv` provides reasonable defaults when running locally, as well as when running as a Cloud Foundry application.

- [https://www.npmjs.org/package/cfenv](https://www.npmjs.org/package/cfenv)

Manual Parsing

First, parse the VCAP_SERVICES environment variable.

For example:

```
var vcap_services = JSON.parse(process.env.VCAP_SERVICES)
```

Then pull out the credential information required to connect to your service. Each service packages requires different information. If you are working with Postgres, for example, you will need a `uri` to connect. You can assign the value of the `uri` to a variable as follows:

```
var uri = vcap_services.mypostgres[0].credentials.uri
```

Once assigned, you can use your credentials as you would normally in your program to connect to your database.

Connecting to a Service

You must include the appropriate package for the type of services your application uses. For example:

- Rabbit MQ via the [amqp](https://www.npmjs.org/package/amqp) module
- Mongo via the [mongodb](https://www.npmjs.org/package/mongodb) and [mongoose](https://www.npmjs.org/package/mongoose) modules
- MySQL via the [mysql](https://www.npmjs.org/package/mysql) module
- Postgres via the [pg](https://www.npmjs.org/package/pg) module
- Redis via the [redis](https://www.npmjs.org/package/redis) module
Add the Dependency to package.json

Edit `package.json` and add the intended module to the `dependencies` section. Normally, only one would be necessary, but for the sake of the example we will add all of them:

```json
{
  "name": "hello-node",
  "version": "0.0.1",
  "dependencies": {
    "express": "*",
    "mongodb": "*",
    "mongoose": "*",
    "mysql": "*",
    "pg": "*",
    "redis": "*",
    "amqp": "*"
  },
  "engines": {
    "node": "0.8.x"
  }
}
```

You must run `npm shrinkwrap` to regenerate your `npm-shrinkwrap.json` file after you edit `package.json`.
Binary Buildpack

Page last updated:

Use the binary buildpack for running arbitrary binary web servers.

Pushing Apps

Unlike most other Cloud Foundry buildpacks, you must specify the binary buildpack to use it when staging your binary file. On a command line, use `cf push APP-NAME` with the `-b` option to specify the buildpack.

For example:

```
$ cf push my_app -b https://github.com/cloudfoundry/binary-buildpack.git
```

You can provide Cloud Foundry with the shell command to execute your binary in the following two ways:

- **Procfile**: In the root directory of your app, add a [Procfile](#) that specifies a `web` task:

  ```
  web: ./app
  ```

- **Command line**: Use `cf push APP-NAME` with the `-c` option:

  ```
  $ cf push my_app -c '/app' -b binary-buildpack
  ```

Compiling your Binary

Cloud Foundry expects your binary to bind to the port specified by the `PORT` environment variable.

The following example in Go binds a binary to the PORT environment variable:

```go
package main

import (
    "fmt"
    "net/http"
    "os"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, %s", "world!")
}

func main() {
    http.HandleFunc('/', handler)
    http.ListenAndServe("*:", nil)
}
```

Your binary should run without any additional runtime dependencies on the cflinuxfs2 or lucid64 root filesystem (roots). Any such dependencies should be statically linked to the binary.

To boot a docker container running the cflinuxfs2 filesystem, run the following command:

```
$ docker run -it cloudfoundry/cflinuxfs2 bash
```

To boot a docker container running the lucid64 filesystem, run the following command:

```
$ docker run -it cloudfoundry/lucid64 bash
```

To compile the above Go application on the roots, golang must be installed. `apt-get install golang` and `go build app.go` will produce an `app` binary.
When deploying your binary to Cloud Foundry, use `cf push` with the `-s` option to specify the root filesystem it should run against.

```
cf push my_app -s (cflinuxfs2|lucid64)
```

To run docker on Mac OS X, we recommend boot2docker.

**BOSH Configured Custom Trusted Certificate Support**

Certificates deployed through BOSH configured custom trusted certificates exist in the `/etc/ssl/certs` directory and can be used by binary applications.

**Help and Support**

Join the #buildpacks channel in our Slack community if you need any further assistance.

For more information about using and extending the binary buildpack in Cloud Foundry, see the binary-buildpack GitHub repo.

You can find current information about this buildpack on the binary buildpack release page in GitHub.
Go Buildpack

Page last updated:

Supported Versions

Supported Go versions can be found in the release notes.

Pushing Apps

The Go buildpack will be automatically detected if:

- Your app has been packaged with `godep` using `godep save`.
- Your app has a `vendor/` directory and has any files ending with `.go`.
- Your app has a `GOPACKAGENAME` environment variable specified and has any files ending with `.go`.
- Your app has a `glide.yaml` file and is using `glide` starting in buildpack version 1.7.9.

If your Cloud Foundry deployment does not have the Go Buildpack installed, or the installed version is out of date, you can use the latest version with the command:

```
$ cf push my_app -b https://github.com/cloudfoundry/go-buildpack.git
```

When specifying versions, specify only major/minor versions, eg. go1.6, rather than go1.6.0. This will ensure you receive the most recent patches.

Start Command

When pushing go apps, you can specify a start command for the app. The start command can be placed in the file `Procfile` in your app’s root directory. For example, if the binary generated by your go project is `my-go-server`, your `Procfile` could be:

```
web: my-go-server
```

For more information about `Procfiles`, see the Configuring a Production Server topic.

You can also specify your app’s start command in the `manifest.yml` file in the root directory, for example:

```
---
applications:
  - name: my-app-name
    command: my-go-server
```

If you do not specify a start command via a `Procfile`, in the manifest, or via the `-c` flag for `cf push`, the generated binary will be used as the start command. (ex. `my-go-server`)

Pushing Apps with godep

If you are using `godep` to package your dependencies, make sure that you have created a valid `Godeps/Godeps.json` file in the root directory of your app by running `godep save`.

When using godep, you can fix your Go version in `GoVersion` key of the `Godeps/Godeps.json` file.

```
Go 1.6
```

- `go 1.6 sample app`  

**NOTE:** if you are using godep with Go 1.6, you must set the `GO15VENDOREXPERIMENT` environment variable to 0, otherwise your app will not stage.
An example Godeps/Godeps.json:

```json
{
  "ImportPath": "go_app",
  "GoVersion": "go1.6",
  "Dep": []
}
```

An example manifest.yml:

```yaml
---
applications:
  - name: my-app-name
    env:
      GO15VENDOREXPERIMENT: 0
```

Pushing Apps with Glide

If you are using glide to specify and/or package your dependencies, make sure that you have created a valid glide.yaml file in the root directory of your app by running

```
glide init
```

To vendor your dependencies before pushing, run `glide install`. This will generate a `vendor` directory and a `glide.lock` file specifying the latest compatible versions of your dependencies. A `glide.lock` is not required when deploying a non-vendored app. A `glide.lock` is required when pushing a vendored app.

Glide

- glide sample app

An example glide.yaml:

```yaml
package: go_app_with_glide
import:
  - package: github.com/ZiCog/shiny-thing

subpackages:
  - foo
```

Pushing Apps with Native Go Vendoring

If you are using the native Go vendoring system, which packages all local dependencies in the `vendor` directory, you must specify your app’s package name in the `GOPACKAGENAME` environment variable. An example manifest.yml:

```yaml
---
applications:
  - name: my-app-name
    command: go-online
    env:
      GOPACKAGENAME: go-online
```

Go 1.6

- sample app

An example manifest.yml:

```yaml
---
applications:
  - name: my-app-name
    command: example-project
    env:
      GOVERSION: go1.6
      GOPACKAGENAME: github.com/example-org/example-project
```

© Copyright Pivotal Software Inc, 2013-2018
Passing a Symbol and String to the Linker

This buildpack supports the go linker's ability (-X symbol=value) to set the value of a string at link time. This can be done by setting `GO_LINKER_SYMBOL` and `GO_LINKER_VALUE` in the application's config before pushing code.

This can be used to embed the commit sha, or other build specific data directly into the compiled executable.

For an example usage, see the relevant fixture app.

C Dependencies

This buildpack supports building with C dependencies via cgo. You can set config vars to specify CGO flags to, e.g., specify paths for vendored dependencies. E.g., to build gopgsqldriver, add the config var `CGO_CFLAGS` with the value `-I/app/code/vendor/include/postgresql` and include the relevant Postgres header files in `/vendor/include/postgresql/` in your app.

Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the Proxy Usage Docs.

BOSH Configured Custom Trusted Certificate Support

Go uses certificates stored in `/etc/ssl/certs` and supports BOSH configured custom trusted certificates out of the box.

Help and Support

Join the #buildpacks channel in our Slack community if you need any further assistance.

For more information about using and extending the Go buildpack in Cloud Foundry, see the go-buildpack GitHub repo.

You can find current information about this buildpack on the Go buildpack release page in GitHub.
PHP Buildpack

Page last updated:

Use the PHP buildpack with PHP or HHVM runtimes.

Supported Software and Versions

The release notes page has a list of currently supported modules and packages.

- **PHP Runtimes**
  - php-cli
  - php-cgi
  - php-fpm

- **Third-Party Modules**
  - New Relic, in connected environments only.

Pushing Apps

30 Second Tutorial

Getting started with this buildpack is easy. With the `cf` command line utility installed, open a shell, change directories to the root of your PHP files and push your application using the argument `cf push -m 128M -b https://github.com/cloudfoundry/php-buildpack.git my-php-app`.

Example:

```shell
mkdir my-php-app
cd my-php-app
cat <! EOF > index.php
<?php
    phpinfo();
?>
EOF
cf push -m 128M -b https://github.com/cloudfoundry/php-buildpack.git my-php-app
```

Please note that you should change `my-php-app` to some unique name on your target Cloud Foundry instance, otherwise you’ll get a hostname conflict error and the push will fail.

The example above will create and push a test application, `my-php-app`, to Cloud Foundry. The `-b` argument instructs CF to use this buildpack. The remainder of the options and arguments are not specific to the buildpack, for questions on those consult the output of `cf push`.

Here’s a breakdown of what happens when you run the example above.

- **On your PC:**
  - It will create a new directory and one PHP file, which just invokes `phpinfo()`.
  - Run `cf` to push your application. This will create a new application with a memory limit of 128M (more than enough here) and upload our test file.

- **Within Cloud Foundry:**
  - The buildpack is executed.
  - Application files are copied to the `htdocs` folder.
  - Apache HTTPD & PHP are downloaded, configured with the buildpack defaults and run.
  - Your application is accessible at the URL `http://my-php-app.example.com` (Replacing `example.com` with the domain of your public CF provider or private instance).
More information about deploying

While the 30 Second Tutorial shows how quick and easy it is to get started using the buildpack, it skips over quite a bit of what you can do to adjust, configure and extend the buildpack. The following sections and links provide a more in-depth look at the buildpack.

Features

Here are some special features of the buildpack.

- Supports running commands or migration scripts prior to application startup.
- Supports an extension mechanism that allows the buildpack to provide additional functionality.
- Allows for application developers to provide custom extensions.
- Easy troubleshooting with the `BP_DEBUG` environment variable.
- Download location is configurable, allowing users to host binaries on the same network (i.e. run without an Internet connection)
- Smart session storage, defaults to file w/sticky sessions but can also use redis for storage.

Examples

Here are some example applications that can be used with this buildpack.

- **php-info** This app has a basic index page and shows the output of `phpinfo()`.
- **PHPMyAdmin** A deployment of PHPMyAdmin that uses bound MySQL services.
- **PHPPgAdmin** A deployment of PHPPgAdmin that uses bound PostgreSQL services.
- **Drupal** A deployment of Drupal that uses bound MySQL service.
- **CodeIgniter** CodeIgniter tutorial application running on CF.
- **Stand Alone** An example which runs a standalone PHP script.
- **pgbouncer** An example which runs the PgBouncer process in the container to pool database connections.
- **phalcon** An example which runs a Phalcon based application.
- **composer** An example which uses Composer.

Advanced Topics

See the following topics:

- **Composer**
- **Sessions**
- **New Relic**
- **Configuration**
- **Deploying and Developing PHP Apps**
- **Tips for PHP Developers**

You can find the source for the buildpack on GitHub: [https://github.com/cloudfoundry/php-buildpack](https://github.com/cloudfoundry/php-buildpack)

Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the [Proxy Usage Docs](https://github.com/cloudfoundry/php-buildpack#proxy-configuration).

BOSH Configured Custom Trusted Certificate Support

For versions of PHP 5.6.0 and later, the default cert location is `/usr/lib/ssl/certs`, which symlinks to `/etc/ssl/certs` and supports BOSH configured custom.
trusted certificates out of the box.

Help and Support

Join the #buildpacks channel in our Slack community if you need any further assistance.

For more information about using and extending the PHP buildpack in Cloud Foundry, see the php-buildpack GitHub repo.

You can find current information about this buildpack on the PHP buildpack release page in GitHub.

License

The Cloud Foundry PHP Buildpack is released under version 2.0 of the Apache License.
Composer

Composer is activated when you supply a `composer.json` or `composer.lock` file. A `composer.lock` is not required, but is strongly recommended for consistent deployments.

You can require dependencies for packages and extensions. Extensions must be prefixed with the standard `ext-`. If you reference an extension that is available to the buildpack, it will automatically be installed. See the main README for a list of supported extensions.

The buildpack uses the version of PHP specified in your `composer.json` or `composer.lock` file. Composer settings override the version set in the `options.json` file.

The PHP buildpack supports a subset of the version formats supported by Composer. The buildpack supported formats are:

<table>
<thead>
<tr>
<th>Example</th>
<th>Expected Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.*</td>
<td>latest 5.4.x release (5.3 is not supported)</td>
</tr>
<tr>
<td>&gt;=5.3</td>
<td>latest 5.4.x release (5.3 is not supported)</td>
</tr>
<tr>
<td>5.4.*</td>
<td>latest 5.4.x release</td>
</tr>
<tr>
<td>&gt;=5.4</td>
<td>latest 5.4.x release</td>
</tr>
<tr>
<td>5.5.*</td>
<td>latest 5.5.x release</td>
</tr>
<tr>
<td>&gt;=5.5</td>
<td>latest 5.5.x release</td>
</tr>
<tr>
<td>5.4.x</td>
<td>specific 5.4.x release that is listed</td>
</tr>
<tr>
<td>5.5.x</td>
<td>specific 5.5.x release that is listed</td>
</tr>
</tbody>
</table>

**Configuration**

The buildpack runs with a set of default values for Composer. You can adjust these values by adding a `hp-config/options.json` file to your application and setting any of the following values in it.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPOSER_VERSION</td>
<td>The version of Composer to use. It defaults to the latest bundled with the buildpack.</td>
</tr>
<tr>
<td>COMPOSER_INSTALL_OPTIONS</td>
<td>A list of options that should be passed to <code>composer install</code>. This defaults to <code>--no-interaction --no-dev --no-progress</code>. The <code>--no-progress</code> option must be used due to the way the buildpack calls Composer.</td>
</tr>
<tr>
<td>COMPOSER_VENDOR_DIR</td>
<td>Allows you to override the default value used by the buildpack. This is passed through to Composer and instructs it where to create the <code>vendor</code> directory. Defaults to <code>{BUILD_DIR}/{LIBDIR}/vendor</code>.</td>
</tr>
<tr>
<td>COMPOSER_BIN_DIR</td>
<td>Allows you to override the default value used by the buildpack. This is passed through to Composer and instructs it where to place executables from packages. Defaults to <code>{BUILD_DIR}/php/bin</code>.</td>
</tr>
<tr>
<td>COMPOSER_CACHE_DIR</td>
<td>Allows you to override the default value used by the buildpack. This is passed through to Composer and instructs it where to place its cache files. Generally you should not change this value. The default is <code>{CACHE_DIR}/composer</code> which is a subdirectory of the cache folder passed in to the buildpack. Composer cache files will be restored on subsequent application pushes.</td>
</tr>
</tbody>
</table>

By default, the PHP buildpack uses the `composer.json` and `composer.lock` files that reside inside the root directory, or in the directory specified as `WEBDIR` in your `options.json`. If you have composer files inside your app, but not in the default directories, use a `COMPOSER_PATH` environment variable for your app to specify this custom location, relative to the app root directory. Note that the `composer.json` and `composer.lock` files must be in the same directory.

**Github API Request Limits**

Composer uses Github’s API to retrieve zip files for installation into the application folder. If you do not vendor dependencies before pushing an app, Composer will fetch dependencies during staging using the Github API.

Github’s API is request-limited. If you reach your daily allowance of API requests (typically 60), Github’s API returns a 403 error and staging fails.
There are two ways to avoid the request limit:

1. Vendor dependencies before pushing your application.
2. Supply a Github OAuth API token.

### Vendoring Dependencies

To vendor dependencies, you must run `composer install` before you push your application. You might also need to configure `COMPOSER_VENDOR_DIR` to "vendor".

### Supply a Github Token

Composer can use Github API OAuth tokens, which increase your request limit, typically to 5000 per day.

During staging, the buildpack looks for this token in the environment variable `COMPOSER_GITHUB_OAUTH_TOKEN`. If you supply a valid token, Composer uses it. This mechanism does not work if the token is invalid.

To supply the token, you can use either of the following methods:

1. `cf set-env YOUR_APP_NAME COMPOSER_GITHUB_OAUTH_TOKEN "OAUTH_TOKEN_VALUE"`
2. Add the token to the `env` block of your application manifest.

### Staging Environment

Composer runs in the buildpack staging environment. Variables set with `cf set-env` or with a manifest.yml `env` block are visible to Composer.

For example:

```
$ cf push a_symfony_app --no-start
$ cf set-env a_symfony_app SYMFONY_ENV "prod"
$ cf start a_symfony_app
```

In this example, `a_symfony_app` is supplied with an environment variable, `SYMFONY_ENV`, which is visible to Composer and any scripts started by Composer.

### Non-configurable Environment Variables

User-assigned environment variables are applied to staging and runtime. Unfortunately, `LD_LIBRARY_PATH` and `PHPRC` must be different for staging and runtime. The buildpack takes care of setting these variables, which means user values for these variables are ignored.

© Copyright Pivotal Software Inc, 2013-2018
Sessions

Usage

When your application has one instance, it’s mostly safe to use the default session storage, which is the local file system. You would only see problems if your single instance crashes as the local file system would go away and you’d lose your sessions. For many applications, this will work just fine but please consider how this will impact your application.

If you have multiple application instances or you need a more robust solution for your application, then you’ll want to use Redis or Memcached as a backing store for your session data. The build pack supports both and when one is bound to your application it will detected it and automatically configure PHP to use it for session storage.

By default, there’s no configuration necessary. Create a Redis or Memcached service, make sure the service name contains `redis` or `memcached` and then bind the service to the application.

Ex:

```
cf create-service redis some-plan app-redis-sessions
cf bind-service app app-redis-sessions
cf restage app
```

If you want to use a specific service instance or change the search key, you can do that by setting either `REDIS_SESSION_STORE_SERVICE_NAME` or `MEMCACHED_SESSION_STORE_SERVICE_NAME` in `.bp-config/options.json` to the new search key. The session configuration extension will then search the bound services by name for the new session key.

Configuration Changes

When detected, the following changes will be made.

Redis

- the `redis` PHP extension will be installed, which provides the session save handler
- `session.name` will be set to `PHPSESSIONID` this disables sticky sessions
- `session.save_handler` is configured to `redis`
- `session.save_path` is configured based on the bound credentials (i.e. `tcp://host:port?auth=pass`)

Memcached

- the `memcached` PHP extension will be installed, which provides the session save handler
- `session.name` will be set to `PHPSESSIONID` this disables sticky sessions
- `session.save_handler` is configured to `memcached`
- `session.save_path` is configured based on the bound credentials (i.e. `PERSISTENT=app_sessions host:port`)
- `memcached.sess_binary` is set to `On`
- `memcached.use_sasl` is set to `On`, which enables authentication
- `memcached.sess_sasl_username` and `memcached.sess_sasl_password` are set with the service credentials.
New Relic

Page last updated:

New Relic collects analytics about application and client-side performance.

Configuration

There are two ways to configure New Relic for the PHP buildpack.

With a CF service

Bind a NewRelic service to the app. The buildpack will automatically detect and set up NewRelic.

This should work as long as the VCAP_SERVICES environment variable contains a service called `newrelic`. That service has a key called `credentials` and that, in turn, has a key called `licenseKey`.

**WARNING:** This will not work with user provided services.

With a License Key

If you already have a New Relic account, use this method.

1. Go to the New Relic website to find your `license key`.

2. Set the value of the environment variable `NEWRELIC_LICENSE` to your NewRelic license key, either through the `manifest.yml` file or with the `cf set-env` command.

For more information, see https://github.com/cloudfoundry/php-buildpack#supported-software
**PHP Buildpack Configuration**

**Page last updated:**

## Defaults

The buildpack stores all of its default configuration settings in the `defaults` directory.

### options.json

The `options.json` file is the configuration file for the buildpack itself. It instructs the buildpack what to download, where to download it from, and how to install it. It allows you to configure package names and versions (i.e. PHP, HTTPD, or Nginx versions), the web server to use (HTTPD, Nginx, or None), and the PHP extensions that are enabled.

The buildpack overrides the default `options.json` file with any configuration it finds in the `.bp-config/options.json` file of your application.

Below is an explanation of the common options you might need to change.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEB_SERVER</td>
<td>Sets the web server to use. Must be one of <code>httpd</code>, <code>nginx</code>, or <code>none</code>. This value defaults to <code>httpd</code>.</td>
</tr>
<tr>
<td>HTTPD_VERSION</td>
<td>Sets the version of Apache HTTPD to use. Currently the buildpack supports the latest stable version. This value will default to the latest release that is supported by the build pack.</td>
</tr>
<tr>
<td>ADMIN_EMAIL</td>
<td>The value used in HTTPD's configuration for <code>ServerAdmin</code>.</td>
</tr>
<tr>
<td>NGINX_VERSION</td>
<td>Sets the version of Nginx to use. By default, the buildpack uses the latest stable version.</td>
</tr>
<tr>
<td>PHP_VERSION</td>
<td>Sets the version of PHP to use.</td>
</tr>
<tr>
<td>PHP_EXTENSIONS</td>
<td>A list of the extensions to enable. <code>bz2</code>, <code>zlib</code>, <code>curl</code>, and <code>mcrypt</code> are enabled by default.</td>
</tr>
<tr>
<td>PHP_MODULES</td>
<td>A list of the modules to enable. No modules are explicitly enabled by default, however the buildpack automatically chooses <code>fpm</code> or <code>cli</code>. You can explicitly enable any or all of: <code>fpm</code>, <code>cli</code>, <code>cwl</code>, and <code>pear</code>.</td>
</tr>
<tr>
<td>ZEND_EXTENSIONS</td>
<td>A list of the Zend extensions to enable. Nothing is enabled by default.</td>
</tr>
<tr>
<td>APP_START_CMD</td>
<td>When the <code>WEB_SERVER</code> option is set to 'none', this command is used to start your app. If <code>WEB_SERVER</code> and <code>APP_START_CMD</code> are not set, then the buildpack searches for <code>app.php</code>, <code>main.php</code>, <code>run.php</code>, or <code>start.php</code> (in that order). This option accepts arguments.</td>
</tr>
<tr>
<td>WEBDIR</td>
<td>The root directory of the files served by the web server specified in <code>WEB_SERVER</code>. Defaults to <code>htdocs</code>. Other common settings are <code>public</code>, <code>static</code>, or <code>html</code>. Path is relative to the root of your application.</td>
</tr>
<tr>
<td>LIBDIR</td>
<td>This path is added to PHP's <code>include_path</code>. Defaults to <code>lib</code>. Path is relative to the root of your application.</td>
</tr>
<tr>
<td>HTTP_PROXY</td>
<td>The buildpack downloads uncached dependencies using HTTP. If you are using a proxy for HTTP access, set its URL here.</td>
</tr>
<tr>
<td>HTTPS_PROXY</td>
<td>The buildpack downloads uncached dependencies using HTTPS. If you are using a proxy for HTTPS access, set its URL here.</td>
</tr>
<tr>
<td>ADDITIONAL_PREPROCESS_CMDS</td>
<td>A list of additional commands that will run prior to the application starting. For example, you might use this command to run migration scripts or static caching tools before the application launches.</td>
</tr>
</tbody>
</table>

For details about supported versions, please read the release notes for your buildpack version.

## HTTPD, Nginx and PHP configuration

The buildpack automatically configures HTTPD, Nginx and PHP for your application. This section explains how to modify the configuration.

The `.bp-config` directory in your application can contain configuration overrides for these components. Name the directories `httpd`, `nginx`, and `php`.

For example: `.bp-config httpd nginx php`

Each directory can contain configuration files that the component understands.
For example, to change HTTPD logging configuration:

```
$ ls -l .bp-config/httpd/extra/
 total 8
 -rw-r--r-- 1 daniel staff 396 Jan 3 08:31 httpd-logging.conf
```

In this example, the `httpd-logging.conf` file overrides the one provided by the buildpack. We recommend that you copy the default from the buildpack and modify it.

The default configuration files are found in the [PHP Buildpack `defaults/config` directory](https://github.com/Buildpacks/php/tree/master/defaults/config).

Take care when modifying configurations, as it might cause your application to fail, or cause Cloud Foundry to fail to stage your application.

You can add your own configuration files. The components will not know about these, so you must ensure that they are included. For example, you can add an include directive to the [httpd configuration](https://github.com/Buildpacks/php/tree/master/defaults/config) to include your file:

```
ServerRoot "${HOME}/httpd"
Listen ${PORT}
ServerAdmin "${HTTPD_SERVER_ADMIN}"
ServerName "0.0.0.0"
DocumentRoot "${HOME}/#{WEBDIR}"
Include conf/extra/httpd-modules.conf
Include conf/extra/httpd-directories.conf
Include conf/extra/httpd-mime.conf
Include conf/extra/httpd-logging.conf
Include conf/extra/httpd-mpm.conf
Include conf/extra/httpd-default.conf
Include conf/extra/httpd-remoteip.conf
Include conf/extra/httpd-php.conf
Include conf/extra/httpd-my-special-config.conf  # This line includes your additional file.
```

### PHP Extensions

PHP extensions are easily enabled by setting the `PHP_EXTENSIONS` or `ZEND_EXTENSIONS` option in `.bp-config/options.json`. Use these options to install bundled PHP extensions.

Please note that if an extension is already present and enabled in the compiled php (ex. `intl`), you do not need to explicitly enable it via `PHP_EXTENSIONS` or `ZEND_EXTENSIONS` in `.bp-config/options.json` to use that extension.

### PHP Modules

The following modules can be included by adding it to the `PHP_MODULES` list:

- `cli`, installs `php-cgi`
- `phar`, installs `phar`
- `fpm`, installs `PHP-FPM`
- `cgi`, installs `php-cgi`

`php-cgi` installs Pear

By default, the buildpack installs the `cli` module when you push a standalone application, and it installs the `fpm` module when you run a web application. You must specify `cgi` and `pear` if you want them installed.

### Buildpack Extensions

The buildpack comes with extensions for its default behavior. These are the [HTTPD](https://github.com/Buildpacks/httpd), [Nginx](https://github.com/Buildpacks/nginx), [PHP](https://github.com/Buildpacks/php), and [NewRelic](https://github.com/Buildpacks/newrelic) extensions.

The buildpack is designed with an extension mechanism, allowing application developers to add behavior to the buildpack without modifying the buildpack code.

When an application is pushed, the buildpack runs any extensions found in the `extensions` directory of your application.

Deploying and Developing PHP Apps

This document is intended to guide you through the process of deploying PHP applications to Elastic Runtime. If you experience a problem with deploying PHP apps, check the Troubleshooting section below.

Getting Started

Prerequisites

- Basic PHP knowledge
- The Cloud Foundry Command Line Interface (cf CLI) installed on your workstation

A First PHP Application

```
$ mkdir my-php-app
$ cd my-php-app
$ cat << EOF > index.php
<?php
phpinfo();
?>
EOF
$ cf push my-php-app -m 128M
```

Change “my-php-app” to a unique name, otherwise you get an error and the push fails.

The example above creates and pushes a test application, “my-php-app”, to Cloud Foundry.

Here is a breakdown of what happens when you run the example above:

- On your workstation…
  - It creates a new directory and one PHP file, which calls `phpinfo();`
  - Run `cf push` to push your application. This will create a new application with a memory limit of 128M and upload our test file.

- On Cloud Foundry…
  - The buildpack detects that your app is a php app
  - The buildpack is executed.
  - Application files are copied to the `htdocs` folder.
  - Apache HTTPD & PHP are downloaded, configured with the buildpack defaults, and run.
  - Your application is accessible at the default route. Use `cf app my-php-app` to view the url of your new app.

Folder Structure

The easiest way to use the buildpack is to put your assets and PHP files into a directory and push it to Elastic Runtime. This way, the buildpack will take your files and automatically move them into the `WEBDIR` (defaults to `htdocs`) folder, which is the directory where your chosen web server looks for the files.

URL Rewriting

If you select Apache as your web server, you can include `.htaccess` files with your application.

Alternatively, you can provide your own Apache or Nginx configurations.
Preventing Access To PHP Files

The buildpack will put all of your files into a publicly accessible directory. In some cases, you might want to have PHP files that are not publicly accessible but are on the include_path. To do that, create a `lib` directory in your project folder and place your protected files there.

For example:

```
$ ls -lRh
total 0
-rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:40 images
-rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:39 index.php
drwxr-xr-x  3 daniel  staff 102B 08 Feb 27 21:40 lib
.
/lib:
  total 0
-rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:40 my.class.php -- not public, http://app.cfapps.io/lib/my.class.php == 404
```

This comes with a catch. If your project legitimately has a `lib` directory, these files will not be publicly available because the buildpack does not copy a top-level `lib` directory into the `WEBDIR` folder. If your project has a `lib` directory that needs to be publicly available, then you have two options as follows:

Option #1

In your project folder, create an `htdocs` folder (or whatever you’ve set for `WEBDIR`). Then move any files that should be publicly accessible into this directory. In the example below, the `lib/controller.php` file is publicly accessible.

Example:

```
$ ls -lRh
Total 0
drwxr-xr-x  7 daniel  staff  238B 08 Feb 27 21:48 htdocs
  .htdocs: <-- create the htdocs directory and put your files there
    total 0
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:40 images
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:39 index.php
    drwxr-xr-x  3 daniel  staff 102B 08 Feb 27 21:48 lib
  ./htdocs/lib: <-- anything under htdocs is public, including a lib directory
    total 0
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:48 controller.php
```

Given this setup, it is possible to have both a public `lib` directory and a protected `lib` directory. The following example demonstrates this setup:

Example:

```
$ ls -lRh
Total 0
drwxr-xr-x  7 daniel  staff  238B 08 Feb 27 21:48 htdocs
drwxr-xr-x  3 daniel  staff 102B 08 Feb 27 21:51 lib
  .htdocs:
    total 0
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:40 images
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:39 index.php
    drwxr-xr-x  3 daniel  staff 102B 08 Feb 27 21:48 lib
  ./htdocs/lib: <-- public lib directory
    total 0
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:48 controller.php
  ./lib: <-- protected lib directory
    total 0
    -rw-r--r--  1 daniel  staff  0B 08 Feb 27 21:51 my.class.php
```

Option #2

The second option is to pick a different name for the `LIBDIR`. This is a configuration option that you can set (it defaults to `lib`). Thus if you set it to something else such as `include`, your application’s `lib` directory would no longer be treated as a special directory and it would be placed into `WEBDIR`.

© Copyright Pivotal Software Inc, 2013-2018

655
Other Folders

Beyond the WEBDIR and LIBDIR directories, the buildpack also supports a .bp-config directory and a .extensions directory.

The .bp-config directory should exist at the root of your project directory and it is the location of application-specific configuration files. Application-specific configuration files override the default settings used by the buildpack. This link explains application configuration files in depth.

The .extensions directory should also exist at the root of your project directory and it is the location of application-specific custom extensions. Application-specific custom extensions allow you, the developer, to override or enhance the behavior of the buildpack. This link explains extensions in more detail.

Troubleshooting

There are a couple of easy ways to debug the buildpack:

1. Check the output from the buildpack. It writes some basic information to stdout, like the files that are being downloaded. It writes information should something fail, specifically, stack traces.

2. Check the logs from the buildpack. The buildpack writes logs to disk. Retrieve them with the command, as the following example shows:

   DEA release
   ```
   $ cf files APP path/activities/bp/logs/bp.log
   ```

   Diego release
   ```
   $ cf ssh APP
   $ cat bp/logs/bp.log
   ```

   This log is more detailed than the stdout output, however it is still terse.

   1. Set the BP_DEBUG environment variable to true for more verbose logging. This instructs the buildpack to set its log level to DEBUG and it writes to stdout. Follow Environment Variables documentation to set BP_DEBUG.
Tips for PHP Developers

Page last updated:

About the PHP Buildpack

For information about using and extending the PHP buildpack in Cloud Foundry, see the [php-buildpack Github repo](https://github.com/pivotal-cf/php-buildpack).

You can find current information about this buildpack on the [PHP buildpack release page](https://github.com/pivotal-cf/php-buildpack/releases) in GitHub.

The buildpack uses a default PHP version specified in `.defaults/options.json` under the `PHP_VERSION` key.

To change the default version, specify the `PHP_VERSION` key in your app's `.bp-config/options.json` file.
Python Buildpack

Page last updated:

Pushing Apps

This buildpack will be automatically used if there is a `requirements.txt` or `setup.py` file in the root directory of your project.

If your Cloud Foundry deployment does not have the Python Buildpack installed, or the installed version is out of date, you can use the latest version with the command:

```
bash cf push my_app -b https://github.com/cloudfoundry/buildpack-python.git
```

Supported Versions

Supported Python versions can be found in the release notes.

Specifying a Python Version

Specific versions of the Python runtime can be specified with a `runtime.txt` file:

```
$ cat runtime.txt
python-3.5.2
```

The buildpack only supports the stable Python versions, which are listed in the `manifest.yml` and releases page.

If you try to use a binary that is not currently supported, staging your app will fail and you will see the following error message:

```
Could not get translated url, exited with: DEPENDENCY_MISSING_IN_MANIFEST: ...
```

Specifying a Start Command

The Python buildpack does not generate a default start command for your applications.

To stage with the Python buildpack and start an application, do one of the following:

- Supply a Procfile. For more information about Procfiles, see the Configuring a Production Server topic. The following example Procfile specifies `gunicorn` as the start command for a web app running on Gunicorn:

```
web: gunicorn SERVER-NAME:APP-NAME
```

- Specify a start command with `-c`. The following example specifies `waitress-serve` as the start command for a web app running on Waitress:

```
cf push python-app -c "waitress-serve --port=$PORT DJANGO-APP.WSGI:MY-APP"
```

- Specify a start command in the application manifest by setting the `command` attribute. For more information, see the Deploying with Application Manifests topic.

Vendoring App Dependencies

As stated in the Disconnected Environments documentation, your application must ‘vendor’ it’s dependencies.
For the Python buildpack, use `pip`:

```
    cd <your app dir>
    mkdir -p vendor

    # Vendors all the pip *.tar.gz into vendor/
    pip install --download vendor -r requirements.txt
```

cf push uploads your vendored dependencies. The buildpack will install them directly from the `vendor/`.

### Using Miniconda (starting in buildpack version 1.5.6)

To use miniconda instead of pip for installing dependencies, place an `environment.yml` file in the root directory.

For examples, see our sample apps [using Python 2 with miniconda](https://pivotal.io/2015/03/06/miniconda-python-buildpack) and [using Python 3 with miniconda](https://pivotal.io/2015/03/06/miniconda-python-buildpack).

### Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the [Proxy Usage Docs](https://pivotal.io/2015/03/06/miniconda-python-buildpack).

### BOSH Configured Custom Trusted Certificate Support

Versions of Python 2.7.9 and later use certificates stored in `/etc/ssl/certs` and support [BOSH configured custom trusted certificates](https://pivotal.io/2015/03/06/miniconda-python-buildpack) out of the box.

### Help and Support

Join the `#buildpacks` channel in our [Slack community](https://pivotal.io/2015/03/06/miniconda-python-buildpack) if you need any further assistance.

For more information about using and extending the Python buildpack in Cloud Foundry, see the [python-buildpack GitHub repo](https://pivotal.io/2015/03/06/miniconda-python-buildpack).

You can find current information about this buildpack on the Python buildpack [release page](https://pivotal.io/2015/03/06/miniconda-python-buildpack) in GitHub.
Staticfile Buildpack

Page last updated:

The Staticfile buildpack works for apps and content that require no backend code besides an Nginx webserver, which the buildpack provides. Examples include front-end JavaScript apps, static HTML content, and HTML/JavaScript forms. It also supports apps with backends hosted elsewhere.

To find which version of Nginx the current Staticfile buildpack uses, see the release notes.

Pushing Apps

This buildpack will be used if your directory has a Staticfile file in the root directory.

For example, in an empty directory:

```bash
echo '<html>hello</html>' > index.html
touch Staticfile
cf push APPNAME -m 64M
```

If your Cloud Foundry deployment does not have the Staticfile Buildpack installed, or the installed version is out of date, you can use the latest version with the command:

```bash
cf push APPNAME -b https://github.com/cloudfoundry/staticfile-buildpack.git -m 64M
```

Why `-m 64M`? Your static assets will be served by Nginx and it only requires 20M [reference]. The `-m 64M` reduces the RAM allocation from the default 1G allocated to Cloud Foundry containers. In the future there may be a way for a buildpack to indicate its default RAM requirements; but not as of writing.

Configuration

You can configure how the Staticfile buildpack accesses and serves apps by editing the Staticfile or nginx.conf files.

Specify an Alternate Root Folder

By default, the buildpack will serve `index.html` and all other assets from the root folder of your project.

In many cases, you may have an alternate folder where your HTML/CSS/JavaScript files are to be served from, such as `dist` or `public`.

To configure the buildpack add the following line to your Staticfile:

```plaintext
root: dist
```

Enable or Disable Basic Authentication

Basic authentication for your app or website depends on whether you have a Staticfile.auth file in its root directory. If Staticfile.auth is present and contains hashed user/password pairs, basic auth is enabled. If this file is absent, the app is unprotected.
Follow these steps to enable basic auth:


2. Create a file in the root of your application named `Staticfile.auth`. This becomes the `htpasswd` file for Nginx to project your site. It can include one or more user/password lines.

   ```
   bob:$apr1$DuUQEQp8$ZccZCHQIUNSp9gywF0
   ```

3. Push your application to apply changes to basic auth.

   To disable basic auth, remove `Staticfile.auth` and push.

**Directory List Instead of 404 If Missing index.html**

If your site doesn’t have a nice `index.html`, you can configure `Staticfile` to display a Directory Index of other files; rather than show a relatively unhelpful 404 error.

Add a line to your `Staticfile` that begins with `directory`:

```
directory: visible
```

**Enable Server Side Includes (SSI)**

Add the following line to your `Staticfile` to enable support for SSI:

```
ssi: enabled
```

**Enable Pushstate Routing**

With client-side JavaScript apps that serve multiple routes, pushstate routing keeps browser-visible URLs clean. For example, pushstate routing allows a single JavaScript file route to multiple anchor-tagged URLs that look like `/some/path1` instead of `/some/path1#`.

To enable support for pushstate routing, add the following line to your `Staticfile`:

```
pushstate: enabled
```
Disable Serving and Decompressing GZip Files

The `gzip_static` and `gunzip` modules are set to `on` by default. `gzip_static` enables Nginx to serve files stored in compressed .gz format, and `gunzip` uncompresses them for clients that do not support compressed content or responses.

To disable these modules, specify the following in your custom `nginx.conf`:

```
gunzip off;
gzip_static off;
```

Force HTTPS

To only allow requests sent via HTTPS, set the `FORCE_HTTPS` environment variable. This redirects non-HTTPS requests as HTTPS requests.

The value that `FORCE_HTTPS` is set to does not matter, only that the environment variable is set. For example, `FORCE_HTTPS: true` and `FORCE_HTTPS: enabled` both force HTTPS.

**Note:** If you are using a reverse proxy like CloudFlare and have rules that redirect HTTP to HTTPS, these can cause redirect loops with your app if `FORCE_HTTPS` is also enabled. In the example of CloudFlare, when attempting to make an HTTPS request to the app, the connection between the user and CloudFlare is HTTPS but the connection between CloudFlare and your app is via HTTP, which your app then redirects as HTTPS.

Custom Nginx Configuration

You can customize the Nginx configuration further, by adding `nginx.conf` and/or `mime.types` to your root folder. (Note that if you specified an alternate root folder, the files will need to be placed there.)

If the buildpack detects either of these files, they will be used in place of the built-in versions.

Proxy Support

If you need to use a proxy to download dependencies during staging, you can set the `http_proxy` and/or `https_proxy` environment variables. For more information, see the Proxy Usage Docs.

BOSH Configured Custom Trusted Certificate Support

The staticfile-buildpack, based on Nginx, does not make outgoing requests and does not benefit from BOSH configured custom trusted certificates.

Help and Support

Join the #buildpacks channel in our Slack community if you need any further assistance.

For more information about using and extending the Staticfile buildpack in Cloud Foundry, see the staticfile-buildpack GitHub repo.

You can find current information about this buildpack on the Staticfile buildpack release page in GitHub.

© Copyright Pivotal Software Inc, 2013-2018
Using Buildpacks

This topic provides links to additional information about using buildpacks. Each of the following are applicable to all supported buildpack languages and frameworks:

- Buildpack Detection
- Proxy Usage
- Supported Binary Dependencies
- Configuring a Production Server
Buildpack Detection

When you push an app, Cloud Foundry determines which buildpack to use by running each enabled buildpacks' detect script during staging.

During staging, each buildpack has a position in a priority list (identified by running `cf buildpacks`). Cloud Foundry checks if the buildpack in position 1 is a compatible buildpack. If the position 1 buildpack is not compatible, Cloud Foundry moves on to the buildpack in position 2. Cloud Foundry continues this process until the correct buildpack is found. If no buildpack is compatible, `cf push` fails with the error:

<table>
<thead>
<tr>
<th>None of the buildpacks detected a compatible application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit status 222</td>
</tr>
<tr>
<td>Staging failed: Exited with status 222</td>
</tr>
</tbody>
</table>

FAILED

NoAppDetectedError
Proxy Usage

Page last updated:

Buildpacks can use proxies via the `http_proxy` and/or `https_proxy` environment variables. These should be set to the proxy hostname and/or port.

These can be set via the `manifest.yml` file or `cf set-env`.

All of the buildpacks will automatically utilize these proxy environment variables correctly. If any of them contact the internet during staging, it will be through the proxy host. The binary buildpack will not use a proxy because it does not use the internet at all during staging.

```yaml
---
env:
  http_proxy: http://proxy-site.com:3000
  https_proxy: https://proxy-site.com:3003
```

Note: Whilst many applications will use the `http_proxy` and `https_proxy` environment variables at runtime, this is entirely dependent on your application, the buildpack does not add any extra functionality to make proxies work at runtime.
Supported Binary Dependencies

Page last updated:

Each buildpack only supports the stable patches for each dependency listed in the buildpack's `manifest.yml` and also in its GitHub releases page. For example, see the [php-buildpack releases page](https://example.com).

If you try to use a binary that is not currently supported, staging your app fails with the following error message:

```
Could not get translated url, exited with: DEPENDENCY_MISSING_IN_MANIFEST:

!! exit

Staging failed: Buildpack compilation step failed
```
Configuring a Production Server

This topic describes how to configure a production server for your apps.

When you deploy an app, Elastic Runtime determines the command used to start the app through the following process:

1. If the developer uses the command `cf push -c COMMAND`, then Elastic Runtime uses `COMMAND` to start the app.

2. If the developer creates a file called a Procfile, Elastic Runtime uses the Procfile to configure the command that launches the app. See the About Procfiles section below for more information.

3. If the developer does not use `cf push -c COMMAND` and does not create a Procfile, then Elastic Runtime uses the command provided in the `default_process_types` property of the `release` script of the buildpack used by the app. See the Custom Buildpacks topic for more information about this property.

About Procfiles

Procfiles offer the best way to configure a production server for web apps.

A Procfile enables you to declare required runtime processes, called process types, for your web app. Process managers in a server use the process types to run and manage the workload. In a Procfile, you declare one process type per line and use the following syntax:

```
PROCESS_TYPE: COMMAND
```

- `PROCESS_TYPE` is typically `web` or `worker`. A `web` process handles HTTP traffic, and a `worker` process runs background jobs.
- `COMMAND` is the command line to launch the process.

For example, the following Procfile starts the launch script created by the build process for a Java app:

```
web: build/install/MY-PROJECT-NAME/bin/MY-PROJECT-NAME
```

Specify a Web Server

Follow these steps to specify a web server using a Procfile. For more information on configuring a web server for Rails apps, see the Configure a Ruby Web Server section.

1. Create a blank file with a command line for a `web` process type.

2. Save it as a file named `Procfile` with no extension in the root directory of your app.

3. Push your app.

Configure a Ruby Web Server

Elastic Runtime uses the default standard Ruby web server library WEBrick for Ruby and Ruby on Rails apps. However, Elastic Runtime can support a more robust production web server, such as Phusion Passenger, Puma, Thin, or Unicorn.

To instruct Elastic Runtime to use a web server other than WEBrick, perform the following steps:

1. Add the gem for the web server to your Gemfile.

2. In the `config` directory of your app, create a new configuration file or modify an existing file. Refer to your web server documentation for how to configure this file. The following example uses the Puma web server:

```
# config/puma.rb
threads 8,32
workers 3
```
on_worker_boot do
  # things workers do
end

3. In the root directory of your app, create a Procfile and add a command line for a `web` process type that points to your web server. Refer to your web server documentation for how to configure the specific command for a process type.

The following example shows a command that starts a Puma web server and specifies the app runtime environment, TCP port, and paths to the server state information and configuration files:

```bash
web: bundle exec puma -e $RAILS_ENV -p 1234 -S ~/puma -C config/puma.rb
```
Developing Buildpacks

Buildpacks enable you to packaging frameworks and/or runtime support for your application. Cloud Foundry provides with system buildpacks out-of-the-box and provides an interface for customizing existing buildpacks and developing new ones.

About Customizing and Creating Buildpacks

If your application uses a language or framework that the Cloud Foundry system buildpacks do not support, do one of the following:

- Use a Cloud Foundry Community Buildpack.
- Use a Heroku Third-Party Buildpack.
- Customize an existing buildpack or create your own custom buildpack. A common development practice for custom buildpacks is to fork existing buildpacks and sync subsequent patches from upstream. For information on customizing an existing buildpack or creating your own, see the following:
  - Custom Buildpacks
  - Packaging Dependencies for Offline Buildpacks

About Maintaining Buildpacks

After you have modified an existing buildpack or created your own, it is necessary to maintain it. Refer to the following when maintaining your own buildpacks:

- Merging from Upstream Buildpacks
- Upgrading Dependency Versions

Note: To configure a production server for your web app, see the Configuring a Production Server topic.
Custom Buildpacks

Page last updated:

Buildpack Scripts

A buildpack repository contains three main scripts, situated in a folder named `bin`.

`bin/detect`

The `detect` script determines whether or not to apply the buildpack to an application. The script is called with one argument, the `build` directory for the application. The `build` directory contains the application files uploaded when a user performs a `cf push`.

The `detect` script returns an exit code of `0` if the buildpack is compatible with the application. In the case of system buildpacks, the script also prints the buildpack name, version, and other helpful information to `STDOUT`.

The following is an example `detect` script written in Ruby that checks for a Ruby application based on the existence of a `Gemfile`:

```ruby
#!/usr/bin/env ruby

gemfile_path = File.join ARGV[0], "Gemfile"

if File.exist?(gemfile_path)
  puts "Ruby"
  exit 0
else
  exit 1
end
```

Optionally, the buildpack detect script can output additional details decided by the buildpack developer. These additional details include buildpack versioning information and a detailed list of configured frameworks and their associated versions.

The following is an example of the detailed information returned by the Java buildpack:

```
java-buildpack=v3.0-https://github.com/cloudfoundry/java-buildpack.git#3bd15e1 open-jdk-jre=1.8.0_45 spring-auto-configuration=1.7.0 RELEASE tomcat-access-logging-support=2.4.0_RELEASE tomcat-instance=8.0.21 ...
```

`bin/compile`

The `compile` script builds a droplet by packaging the application’s dependencies, assuring that the application has all the necessary components needed to run.

The script is run with two arguments: the `build` directory for the application and the `cache` directory, which is a location the buildpack can use to store assets during the build process. During the execution of the `compile` script, all output sent to `STDOUT` is relayed through the `cf CLI` to the user.

The following is an example of a simple `compile` script:
#!/usr/bin/env	ruby

STDOUT.sync = true

build_path = ARGV[0]
cache_path = ARGV[1]

install_ruby

private
def install_ruby
  puts "Installing Ruby"
  # !!!! build tasks go here !!!!
  # download ruby
  # install ruby
end

bin/release

The release script provides feedback metadata to Cloud Foundry indicating how the application should be executed. The script is run with one argument, the build directory. The script must generate a YAML file in the following format:

```
default_process_types:
  web: start_command.filetype
```

*default_process_types* indicates the type of application being run and the command used to start it. At this time, only *web* type of applications are supported.

**Note:** To define environment variables for your buildpack, add a bash script to the .profile.d directory in the root folder of your application.

The following example shows what a Rack application’s release script might return:

```
default_process_types:
  web: bundle exec rackup config.ru -p $PORT
```

**Note:** The web command runs as `bash -c COMMAND` when Cloud Foundry starts your application. Refer to the command attribute section for more information about custom start commands.

### Droplet Filesystem

The buildpack staging process extracts the droplet into the /home/vcap directory inside the instance container, and creates the following filesystem tree:

```
app/
  logs/
  tmp/
  staging_info.yml
```

The app directory contains BUILD_DIR contents, and staging_info.yml contains the staging metadata saved in the droplet.

### Packaging Custom Buildpacks

Cloud Foundry buildpacks work with limited or no Internet connectivity. A Cloud Foundry operator can use the buildpack packager to give the same flexibility to custom buildpacks, enabling them to work in partially or completely disconnected environments.
Using the Buildpack Packager

1. Create a manifest.yml in your buildpack.

2. Run the packager in cached mode:

   ```
   $ buildpack-packager --cached
   ```

   The packager will add (almost) everything in your buildpack directory into a zip file. It will exclude anything marked for exclusion in your manifest.

   In cached mode, the packager will download and add dependencies as described in the manifest.

   The packager has the following option flags:

   - `--force-download`: By default, the packager stores the dependencies that it downloads while building a cached buildpack in a local cache at `~/.buildpack-packager`. Storing dependencies enables the packager to avoid re-downloading them when repackaging similar buildpacks. Running `buildpack-packager --cached` with the `--force-download` option forces the packager to download dependencies from the S3 host and ignore the local cache. When packaging an uncached buildpack, `--force-download` does nothing.

   - `--use-custom-manifest`: To include a different manifest file in your packaged buildpack, you can call the packager with the `--use-custom-manifest PATH/TO/MANIFEST.YML` option. The packager generates a buildpack with the specified manifest. If you are building a cached buildpack, the packager vendors dependencies from the specified manifest as well.

   For more information, see the documentation at the `buildpack-packager Github repo`

Using and Sharing the Packaged Buildpack

After you have packaged your buildpack using `buildpack-packager` you can use the resulting `.zip` file locally, or share it with others by uploading it to any network location that is accessible to the CLI. Users can then specify the buildpack with the `-b` option when they push apps. See Deploying Apps with a Custom Buildpack for details.

You can also use the `cf create-buildpack` command to upload the buildpack into your Cloud Foundry deployment, making it accessible without the `-b` flag:

```
$ cf create-buildpack BUILDPACK PATH POSITION [--enable|--disable]
```

You can find more documentation in the `Managing Buildpacks` topic.

Specifying Default Versions

As of `buildpack-packager version 2.3.0`, you can specify the default version for a dependency by adding a `default_versions` object to the `manifest.yml` file. The `default_versions` object has two properties, `name` and `version`. For example:

```
default_versions:
  - name: go
    version: 1.6.3
  - name: other-dependency
    version: 1.1.1
```

To specify a default version:

1. Add the `default_version` object to your manifest, following the rules below. You can find a complete example manifest in the Cloud Foundry `go-buildpack` repo.

2. Run the `default_version_for` script from the `compile-extensions` repo, passing the path of your `manifest.yml` and the dependency name as arguments. The following command uses the example manifest from step 1:

```
$ ./compile-extensions/bin/default_version_for manifest.yml go 1.6.3
```

Rules for Specifying a Default Version

The `buildpack-packager` script validates this object according to the following rules:
You can create at most one entry under `default_versions` for a single dependency. The following example causes `buildpack-packager` to fail with an error because the manifest specifies two default versions for the same `go` dependency.

```yaml
Incorrect: will fail to package
default_versions:
- name: go
  version: 1.6.3
- name: go
  version: 1.3.1
```

If you specify a `default_version` for a dependency, you must also list that dependency and version under the `dependencies` section of the manifest. The following example causes `buildpack-packager` to fail with an error because the manifest specifies `version: 1.6.3` for the `go` dependency, but lists `version: 1.5.4` under `dependencies`.

```yaml
Incorrect: will fail to package
default_versions:
dependencies:
- name: go
  version: 1.6.3
  url: https://storage.googleapis.com/golang/go1.5.4.linux-amd64.tar.gz
  md5: 27b1c469797292064c65c995fe3f0386
  cf_stacks:
  - cflinuxfs2
```

### Deploying Apps with a Custom Buildpack

Once a custom buildpack has been created and pushed to a public git repository, the git URL can be passed via the cf CLI when pushing an application.

For example, for a buildpack that has been pushed to Github:

```bash
$ cf push my-new-app -b git://github.com/johndoe/my-buildpack.git
```

Alternatively, you can use a private git repository, with https and username/password authentication, as follows:

```bash
$ cf push my-new-app -b https://username:password@github.com/johndoe/my-buildpack.git
```

By default, Cloud Foundry uses the default branch of the buildpack's git repository. You can specify a different branch using the git url as shown in the following example:

```bash
$ cf push my-new-app -b https://github.com/johndoe/my-buildpack.git#my-branch-name
```

Additionally, you can use tags or shas in a git repository, as follows:

```bash
$ cf push my-new-app -b https://github.com/johndoe/my-buildpack#v1.4.2
$ cf push my-new-app -b https://github.com/johndoe/my-buildpack#42951e29f8d22732fe836d541015120d8af93
```

The application will then be deployed to Cloud Foundry, and the buildpack will be cloned from the repository and applied to the application.

**Note:** If a buildpack is specified using `cf push -b` the `detect` step will be skipped and as a result, no buildpack `detect` scripts will be run.

### Disabling Custom Buildpacks

Operators can choose to disable custom buildpacks. For more information, see [Disabling Custom Buildpacks](#).

**Note:** A common development practice for custom buildpacks is to fork existing buildpacks and sync subsequent patches from upstream. To merge upstream patches to your custom buildpack, use the approach that Github recommends for [syncing a fork](#).
Packaging Dependencies for Offline Buildpacks

Page last updated:

This topic describes the dependency storage options available to developers creating custom offline buildpacks.

Package dependencies in the buildpack

The simplest way to package dependencies in a custom buildpack is to keep the dependencies in your buildpack source. However, this is strongly discouraged. Keeping the dependencies in your source consumes unnecessary space.

To avoid keeping the dependencies in source control, load the dependencies into your buildpack and provide a script for the operator to create a zipfile of the buildpack.

For example, the operator might complete the following process:

```
# Clones your buildpack
$ git clone http://YOUR-GITHUB-REPO.example.com/repo
$ cd SomeBuildPackName

# Creates a zipfile using your script
$ ./SomeScriptName
----> downloading-dependencies... done
----> creating zipfile: ZippedBuildPackName.zip

# Adds the buildpack zipfile to the Cloud Foundry instance
$ cf create-buildpack SomeBuildPackName ZippedBuildPackName.zip 1
```

Pros

- Least complicated process for operators
- Least complicated maintenance process for buildpack developers

Cons

- Cloud Foundry admin buildpack uploads are limited to 1 GB, so the dependencies might not fit
- Security and functional patches to dependencies require updating the buildpack

Package selected dependencies in the buildpack

This is a variant of the package dependencies in the buildpack method described above. In this variation, the administrator edits a configuration file such as `dependencies.yml` to include a limited subset of the buildpack dependencies, then packages and uploads the buildpack.

*Note:* This approach is strongly discouraged. Please see the Cons section below for more information.

The administrator completes the following steps:
# Clones your buildpack
$ git clone http://YOUR-GITHUB-REPO.example.com/repo
$ cd

# Selects dependencies
$ vi dependencies.yml  # Or copy in a preferred config

# Builds a package using your script
$ ./package
  --> downloading-dependencies... done
  --> creating zipfile: cobol_buildpack.zip

# Adds the buildpack to the Cloud Foundry instance
$ cf create-buildpack cobol-buildpack cobol_buildpack.zip 1

# Pushes an app using your buildpack
$ cd ~/my_app
$ cf push my-cobol-webapp -b cobol-buildpack

--->
deploying app
--->
downloading dependencies:
https://OUR-INTERNAL-SITE.example.com/dependency/repo/dep1.tgz.... done
https://OUR-INTERNAL-SITE.example.com/dependency/repo/dep2.tgz.... WARNING: dependency not found!

Pros

- Possible to avoid the Cloud Foundry admin buildpack upload size limit in one of two ways:
  - If the administrator chooses a limited subset of dependencies
  - If the administrator maintains different packages for different dependency sets

Cons

- More complex for buildpack maintainers
- Security updates to dependencies require updating the buildpack
- Proliferation of buildpacks that require maintenance:
  - For each configuration, there is an update required for each security patch
  - Culling orphan configurations may be difficult or impossible
  - Administrators need to track configurations and merge them with updates to the buildpack
  - May result in with a different config for each app

Rely on a local mirror

In this method, the administrator provides a compatible file store of dependencies. When running the buildpack, the administrator specifies the location of the file store. The buildpack should handle missing dependencies gracefully.

The administrator completes the following process:

# Clones your buildpack
$ git clone http://YOUR-GITHUB-REPO.example.com/repo
$ cd

# Builds a package using your script
$ ./package https:///dependency/repo
  --> creating zipfile: cobol_buildpack.zip

# Adds the buildpack to the Cloud Foundry instance
$ cf create-buildpack cobol-buildpack cobol_buildpack.zip 1

# Pushes an app using your buildpack
$ cd ~/my_app
$ cf push my-cobol-webapp -b cobol-buildpack
  --> deploying app
  --> downloading dependencies:
  https://OUR-INTERNAL-SITE.example.com/dependency/repo/dep1.tgz.... done
  https://OUR-INTERNAL-SITE.example.com/dependency/repo/dep2.tgz.... WARNING: dependency not found!
Avoids the Cloud Foundry admin buildpack upload size limit
Leaves the administrator completely in control of providing dependencies
Security and functional patches for dependencies can be maintained separately on the mirror given the following conditions:

- The buildpack is designed to use newer semantically versioned dependencies
- Buildpack behavior does not change with the newer functional changes

Cons

- The administrator needs to set up and maintain a mirror
- The additional config option presents a maintenance burden
Merging from Upstream Buildpacks

Page last updated:

This topic describes how to maintain your forked buildpack by merging it with the upstream buildpack. This allows you to keep your fork updated with changes from the original buildpack, providing patches, updates, and new features.

The following procedure assumes that you are maintaining a custom buildpack that was forked from a Cloud Foundry system buildpack. However, you can use the same procedure to update a buildpack forked from any upstream buildpack.

To sync your forked buildpack with an upstream Cloud Foundry buildpack:

1. Navigate to your forked repo on GitHub and click Compare in the upper right to display the Comparing changes page. This page shows the unmerged commits between your forked buildpack and the upstream buildpack.

2. Inspect the unmerged commits and confirm that you want to merge them all.

3. In a terminal window, navigate to the forked repo and set the upstream remote as the Cloud Foundry buildpack repo.
   
   ```bash
   cd ~/workspace/ruby-buildpack
   git remote add upstream git@github.com:cloudfoundry/ruby-buildpack.git
   ```

4. Pull down the remote upstream changes.
   
   ```bash
   git fetch upstream
   ```

5. Merge the upstream changes into the intended branch. You may need to resolve merge conflicts. This example shows merging the `master` branch of the upstream buildpack into the `master` branch of the forked buildpack.
   
   ```bash
   git checkout master
   git merge upstream/master
   ```

   **Note:** When merging upstream buildpacks, do not use `git rebase`. This approach is not sustainable because you confront the same merge conflicts repeatedly.

6. Run the buildpack test suite to ensure that the upstream changes do not break anything.
   
   ```bash
   BUNDLE_GEMFILE=cf.Gemfile buildpack-build
   ```

7. Push the updated branch.
   
   ```bash
   git push
   ```

Your forked buildpack is now synced with the upstream Cloud Foundry buildpack.

For more information about syncing forks, see the Github topic Syncing a Fork.
Upgrading Dependency Versions

Page last updated:

This topic describes how to upgrade a dependency version in a custom buildpack. These procedures enable Cloud Foundry (CF) operators to maintain custom buildpacks that contain dependencies outside of the dependencies in the CF system buildpacks.

Cloud Foundry Buildpacks Team Process

The CF buildpacks team uses the following tools to update dependencies:

- A Concourse deployment of the buildpacks-ci pipelines
- Pivotal Tracker for workflow management

Note: The procedures in this topic refer to the tools used by the CF buildpacks team. However, the procedures do not require the specific tools mentioned above. You can use any CI and workflow management tool to update dependencies in custom buildpacks.

When the New Releases job in the notifications pipeline detects a new version of a tracked dependency in a buildpack, it creates a Tracker story about building and including the new version of the dependency in the buildpack manifests. It also posts a message as the dependency-notifier to the #buildpacks channel in the Cloud Foundry Slack.

Building the Binaries

For all dependencies, you must build the binary from source or acquire the binary as a tarball from a trusted source. For most dependencies, the CF buildpacks team builds the binaries from source.

Note: The steps below assume you are using a Concourse deployment of the buildpacks-ci pipelines and Pivotal Tracker.

To build the binary for a dependency, perform the following steps:

1. Change into the buildpacks-ci directory and check that there are no uncommitted changes.

   ```shell
   $ cd ~/workspace/buildpacks-ci
   $ git status
   ```

2. Check out the binary-builds branch. This is an orphan branch of buildpacks-ci that the CF buildpacks team uses as a separate resource on Concourse to trigger the binary building process.

   ```shell
   $ git checkout binary-builds
   ```

3. Pull the branch to make sure it is up to date.

   ```shell
   $ git pull -r
   ```

4. Locate the YAML file for the buildpack you want to build a binary for. The directory contains YAML files for all the packages and dependencies tracked by the CF buildpacks team. Each YAML file correlates to the build queue for one dependency or package and the naming format is DEPENDENCY-NAME.yml. For example, the YAML file tracking the build queue for Ruby is called ruby-builds.yml and contains the following contents:

   ```yaml
   ---
   ruby: []
   ```

5. Different buildpacks use different signatures for verification. Determine which signature your buildpack requires by consulting the list below and follow the instructions to locate the SHA256, MD5, or GPG signature for the binary:

   - For the SHA256 of a file, run `sha256sum -a 256 FILE`.
   - For the MD5 of a file, run `md5 FILE`.
   - For the GPG signature (for Nginx), see the Nginx Downloads page.

6. Add the version and verification for the new binary to the YAML file as attributes of an element under the dependency name. For example, to build the Ruby 2.3.0 binary verified with SHA256, add the following:
You can enqueue builds for multiple versions at once. For example, to build both the Ruby 2.3.0 binary and the Ruby 2.3.1 binary, add the following:

```ruby
- version: 2.3.0
  sha256: ba5ba60e5f1aa21b4e9ec9b0d55f9db57286cb546aacc4b5a28c7f1f59467c507
- version: 2.3.1
  sha256: b87c738cb2032bf4920fef8e3864d50e8364dc5cfb3ae9d99d8d523c07c0236945c5797dcd
```

Stage your changes for commit:

```
git add .
```

Commit your changes using the Tracker story number.

```
git commit -m "YOUR-COMMIT-MESSAGE[#STORY_NUMBER]"
```

Push your changes to the remote origin.

```
git push
```

Pushing your changes triggers the binary building process, which you can monitor at the binary-builder pipeline of your own Concourse deployment. When the build completes, it adds a link to the Concourse build run to the Tracker story for the new release.

**Note:** Binary builds are executed by the CF Binary Builder and the binary-builder pipeline.

### Updating Buildpack Manifests

After you build the binary for a dependency that you can access and download from a URL, follow these instructions to add the dependency version to the buildpack manifest.

**Note:** The steps below assume you are using a Concourse deployment of the buildpacks-ci pipelines and Pivotal Tracker.

1. Change into the directory of the buildpack for which you want to update dependencies and check out the `develop` branch.

   ```
   cd ~/workspace/ruby-buildpack
   git checkout develop
   ```

2. Open the `manifest.yml` for the buildpack and remove or add dependencies.

   ```
   dependencies:
   - name: ruby
     version: 2.3.0
     md5: 5353420300b11ebeb11949824b5624b
     url: https://pivotal-buildpacks.s3.amazonaws.com/concourse-binaries/ruby/ruby-2.3.0-linux-x64.tgz
     cf_stacks:
     - cflinuxfs2
   ```

   - Follow the current structure of the manifest. For example, if the manifest includes the two most recent patch versions for each minor version of the language, do the same, such as both `ruby-2.1.9` and `ruby-2.1.8`.
   - Paste in the `url` and the `md5` from the `build-BINARY-NAME` job that ran in the Concourse binary-builder pipeline.

   **Note:** In the PHP buildpack, you may see a `modules` line for each PHP dependency in the manifest. Do not include this in your new PHP dependency entry. This will be added to the manifest by the `ensure-manifest-has-modules` Concourse job in the `php-buildpack` when you
3. Replace any other mentions of the old version number in the buildpack repo with the new version number. The CF buildpack team uses Ag for text searching.

```bash
$ ag OLD-VERSION
```

4. Run the following command to package and upload the buildpack, setup the org and space for tests in the specified CF deployment, and run the CF buildpack tests.

```bash
$ BUNDLE_GEMFILE=cf.Gemfile buildpack-build
```

If the command fails, you may need to fix or change the tests, fixtures, or other parts of the buildpack.

5. Once the test suite completely passes, push your changes:

```bash
$ git add .
$ git commit -m "YOUR-MESSAGE[#TRACKER-STORY-ID]"
$ git push
```

6. Watch the LANGUAGE-buildpack pipeline in Concourse. Once the test suite builds pass for the buildpack (the specs-lts-develop and specs-edge-develop job), you can mark the Tracker story for the new Dependency release as delivered. Paste links for those successful test suite builds in the Tracker story.

---

**Buildpacks**

The following list contains information about the buildpacks maintained by the CF buildpacks team.

**Go Buildpack**

- **Go**:
  - Built from: a tarred binary ([GO-VERSION.linux-amd64.tar.gz](https://golang.org/dl)) provided by Google on the Go [Downloads](https://golang.org/dl) page
  - Verified with: the MD5 of the tarred binary
  - Example usage: Using the Google Tarred Binary for Go 1.6.2

**Godep**:

- Built from: a source code `.tar.gz` file from the Godep Github releases page
- Verified with: the SHA256 of the source example

**Nodejs Buildpack**

- **Node**:
  - Verified with: the SHA256 of the `node-vVERSION.tar.gz` file listed on [https://nodejs.org/dist/vVERSION/SHASUMS256.txt](https://nodejs.org/dist/vVERSION/SHASUMS256.txt) For example, for Node version 4.4.6, the CF buildpacks team verifies with the SHA256 for `node-v4.4.6.tar.gz` on its SHASUMS256 page.
  - Example: Enqueuing binary builds for Node 4.4.5 and 6.2.0

**Python Buildpack**

- **Python**:
  - Verified with: the MD5 of the `Gzipped source tarball`, listed on: [https://www.python.org/downloads/release/python-VERSION/](https://www.python.org/downloads/release/python-VERSION/) where `VERSION` has no periods. For example, for Python version `2.7.12`, use the MD5 for the **Gzipped source tarball** on its Downloads page.
Java Buildpack

OpenJDK:

- Built from: the tarred OpenJDK files managed by the CF Java Buildpack team.
- Verified with: the MD5 of the tarred OpenJDK files.

Ruby Buildpack

JRuby:

- Verified with: the MDS of the source .tar.gz file from the JRuby Downloads page.
- Example: Enqueuing binary build for JRuby 9.1.2.0

Ruby:

- Verified with: the SHA256 of the source from the Ruby Downloads page.
- Example: Enqueuing binary builds for Ruby 2.2.5 and 2.3.1

Bundler:

- Verified with: the SHA256 of the .gem file from Rubygems.
- Example: Enqueuing binary build for Bundler 1.12.5

PHP Buildpack

PHP:

- Verified with: the SHA256 of the .tar.gz file from the PHP Downloads page.
- For PHP5 versions, the CF buildpacks team enqueues builds in the php-builds.yml file in the binary-builds branch. For PHP7 versions, the CF buildpacks team enqueues builds in the php7-builds.yml file in the binary-builds branch.
- Example: Enqueuing binary builds for PHP 5.5.37, 5.6.23, and 7.0.8

Nginx:

- Verified with: the gpg-rsa-key-id and gpg-signature of the version. The gpg-rsa-key-id is the same for each version/build, but the gpg-signature will be different. This information is located on the Nginx Downloads page.
- Example: Enqueuing binary build for Nginx 1.11.0

HTTPD:

- Verified with: the MDS of the .tar.gz file from the HTTPD Downloads page.
- Example: Enqueuing binary build for HTTPD 2.4.20

Composer:

- Verified with: the SHA256 of the composer.phar file from the Composer Downloads page.
- For Composer, there is no build process as the composer.phar file is the binary. In the manual process, connect to the pivotal-buildpacks S3 bucket using the correct AWS credentials. Create a new directory with the name of the composer version (ex. 1.0.2) and put the appropriate composer.phar file into that directory. For Composer v1.0.2, connect and create the php-binaries/trusty/composer/1.0.2 directory. Then place the composer.phar file into that directory so the binary is available at php-binaries/trusty/composer/1.0.2/composer.phar.
- Example: Automated enqueuing of binary build for Composer 1.1.2

Note: The buildpacks-ci binary-builder pipeline automates the process of detecting, uploading, and updating Composer in the manifest.
Staticfile Buildpack

Nginx:

- Verified with: the `gpg-rsa-key-id` and `gpg-signature` of the version. The `gpg-rsa-key-id` is the same for each version/build, but the `gpg-signature` will be different. This information is located on the Nginx Downloads page.
- Example: Enqueueing binary build for Nginx 1.11.0

Binary Buildpack

The Binary buildpack does not have any dependencies.
Services

Page last updated:

The documentation in this section is intended for developers and operators interested in creating Managed Services for Cloud Foundry. Managed Services are defined as having been integrated with Cloud Foundry via APIs, and enable end users to provision reserved resources and credentials on demand. For documentation targeted at end users, such as how to provision services and integrate them with applications, see Services Overview.

To develop Managed Services for Cloud Foundry, you'll need a Cloud Foundry instance to test your service broker with as you are developing it. You must have admin access to your CF instance to manage service brokers and the services marketplace catalog. For local development, we recommend using BOSH Lite to deploy your own local instance of Cloud Foundry.

Table of Contents

- Overview
- Service Broker API
- Managing Service Brokers
- Access Control
- Catalog Metadata
- Dashboard Single Sign-On
- Example Service Brokers
- Binding Credentials
- Application Log Streaming
- Route Services
- Manage Application Requests with Route Services
- Supporting Multiple Cloud Foundry Instances
Overview

Architecture & Terminology

Services are integrated with Cloud Foundry by implementing a documented API for which the cloud controller is the client; we call this the Service Broker API. This should not be confused with the cloud controller API, often used to refer to the version of Cloud Foundry itself; when one refers to “Cloud Foundry v2” they are referring to the version of the cloud controller API. The services API is versioned independently of the cloud controller API.

Service Broker is the term we use to refer to a component of the service which implements the service broker API. This component was formerly referred to as a Service Gateway, however as traffic between applications and services does not flow through the broker we found the term gateway caused confusion. Although gateway still appears in old code, we use the term broker in conversation, in new code, and in documentation.

Service brokers advertise a catalog of service offerings and service plans, as well as interpreting calls for provision (create), bind, unbind, and deprovision (delete). What a broker does with each call can vary between services; in general, 'provision' reserves resources on a service and 'bind' delivers information to an application necessary for accessing the resource. We call the reserved resource a Service Instance. What a service instance represents can vary by service; it could be a single database on a multi-tenant server, a dedicated cluster, or even just an account on a web application.

Implementation & Deployment

How a service is implemented is up to the service provider/developer. Cloud Foundry only requires that the service provider implement the service broker API. A broker can be implemented as a separate application, or by adding the required http endpoints to an existing service.

Because Cloud Foundry only requires that a service implements the broker API in order to be available to Cloud Foundry end users, many deployment models are possible. The following are examples of valid deployment models.

- Entire service packaged and deployed by BOSH alongside Cloud Foundry
- Broker packaged and deployed by BOSH alongside Cloud Foundry, rest of the service deployed and maintained by other means
- Broker (and optionally service) pushed as an application to Cloud Foundry user space
- Entire service, including broker, deployed and maintained outside of Cloud Foundry by other means
Service Broker API v2.8

Changes

Change Policy

- Existing endpoints and fields will not be removed or renamed.
- New optional endpoints, or new HTTP methods for existing endpoints, may be added to enable support for new features.
- New fields may be added to existing request/response messages. These fields must be optional and should be ignored by clients and servers that do not understand them.

Changes Since v2.7

1. Brokers may now include a requires: ["route_forwarding"] on their catalog endpoint.

2. On bind, the Cloud Controller will now send a new top-level key, bind_resource, under which the required parameters of the binding are found. This would include, for example, app_guid for an app binding and route for a route binding. For backwards compatibility, app_guid will remain a top-level key in addition to being included in the bind_resource.

3. Adds support for a route_service_url key in the binding response.

Dependencies

v2.8 of the services API has been supported since:

- Final build 226 of cf-release
- v2.44.0 of the Cloud Controller API
- CLI v6.14.0

API Overview

The Cloud Foundry services API defines the contract between the Cloud Controller and the service broker. The broker is expected to implement several HTTP (or HTTPS) endpoints underneath a URI prefix. One or more services can be provided by a single broker, and load balancing enables horizontal scalability of redundant brokers. Multiple Cloud Foundry instances can be supported by a single broker using different URL prefixes and credentials.
Requests from the Cloud Controller to the broker contain a header that defines the version number of the Broker API that Cloud Controller will use. This header will be useful in future minor revisions of the API to allow brokers to reject requests from Cloud Controllers that they do not understand. While minor API revisions will always be additive, it is possible that brokers will come to depend on a feature that was added after 2.0, so they may use this header to reject the request. Error messages from the broker in this situation should inform the operator of what the required and actual version numbers are so that an operator can go upgrade Cloud Controller and resolve the issue. A broker should respond with a 412 Precondition Failed message when rejecting a request.

The version numbers are in the format MAJOR.MINOR, using semantic versioning such that 2.9 comes before 2.10. An example of this header as of publication time is:

```
X-Broker-Api-Version: 2.8
```

Cloud Controller (final release v145+) authenticates with the Broker using HTTP basic authentication (the Authorization header) on every request and will reject any broker registrations that do not contain a username and password. The broker is responsible for checking the username and password and returning a 401 Unauthorized message if credentials are invalid. Cloud Controller supports connecting to a broker using SSL if additional security is desired.

The first endpoint that a broker must implement is the service catalog. Cloud Controller will initially fetch this endpoint from all brokers and make adjustments to the user-facing service catalog stored in the Cloud Controller database. If the catalog fails to initially load or validate, Cloud Controller will not allow the operator to add the new broker and will give a meaningful error message. Cloud Controller will also update the catalog whenever a broker is updated, so you can use update-service-broker with no changes to force a catalog refresh.
When Cloud Controller fetches a catalog from a broker, it will compare the broker’s id for services and plans with the unique_id values for services and plans in the Cloud Controller database. If a service or plan in the broker catalog has an id that is not present amongst the unique_id values in the database, a new record will be added to the database. If services or plans in the database are found with unique_id s that match the broker catalog’s id, Cloud Controller will update the records to match the broker’s catalog.

If the database has plans which are not found in the broker catalog, and there are no associated service instances, Cloud Controller will remove these plans from the database. Cloud Controller will then delete services that do not have associated plans from the database. If the database has plans which are not found in the broker catalog, and there are provisioned instances, the plan will be marked “inactive” and will no longer be visible in the marketplace catalog or be provisionable.

**Request**

**Route**

```
GET /v2/catalog
```

**cURL**

```
curl -H "X-Broker-API-Version: 2.8" http://username:password@broker-url/v2/catalog
```

**Response**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>The expected response body is below.</td>
</tr>
</tbody>
</table>

**Body**

CLI and web clients have different needs with regard to service and plan names. A CLI-friendly string is all lowercase, with no spaces. Keep it short – imagine a user having to type it as an argument for a longer command. A web-friendly display name is camel-cased with spaces and punctuation supported.

<table>
<thead>
<tr>
<th>Response field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>services*</td>
<td>array-of-objects</td>
<td>Schema of service objects defined below:</td>
</tr>
<tr>
<td>id*</td>
<td>string</td>
<td>An identifier used to correlate this service in future requests to the catalog. This must be unique within Cloud Foundry, using a GUID is recommended.</td>
</tr>
<tr>
<td>name*</td>
<td>string</td>
<td>The CLI-friendly name of the service that will appear in the catalog. All lowercase, no spaces.</td>
</tr>
<tr>
<td>description*</td>
<td>string</td>
<td>A short description of the service that will appear in the catalog.</td>
</tr>
<tr>
<td>bindable*</td>
<td>boolean</td>
<td>Whether the service can be bound to applications.</td>
</tr>
<tr>
<td>tags</td>
<td>array-of-strings</td>
<td>Tags provide a flexible mechanism to expose a classification, attribute, or base technology of a service, enabling equivalent services to be swapped out without changes to dependent logic in applications, buildpacks, or other services. E.g. mysql, relational, redis, key-value, caching, messaging, amqp.</td>
</tr>
<tr>
<td>metadata</td>
<td>object</td>
<td>A list of metadata for a service offering. For more information, see Service Metadata.</td>
</tr>
<tr>
<td>requires</td>
<td>array-of-strings</td>
<td>A list of permissions that the user would have to give the service, if they provision it. The only permissions currently supported are syslog_drain and route_forwarding; for more info see Application Log Streaming and Route Services.</td>
</tr>
<tr>
<td>plan_updateable</td>
<td>boolean</td>
<td>Whether the service supports upgrade/downgrade for some plans. Please note that the misspelling of the attribute plan_updatable to plan_updateable was done by mistake. We have opted to keep that misspelling instead of fixing it and thus breaking backward compatibility.</td>
</tr>
<tr>
<td>plans*</td>
<td>array-of-objects</td>
<td>A list of plans for this service, schema defined below:</td>
</tr>
</tbody>
</table>
Adding a Broker to Cloud Foundry

Once you’ve implemented the first endpoint GET /v2/catalog above, you’ll want to register the broker with CF to make your services and plans available to end users.

Asynchronous Operations

Previously, Cloud Foundry only supported synchronous integration with service brokers. Brokers must return a valid response within 60 seconds and if the response is 200 CREATED, users expect a service instance to be usable. This limits the services brokers can offer to those that can be provisioned in 60 seconds; brokers could return a success prematurely, but this leaves users wondering why their service instance is not usable and when it will be.

With support for Asynchronous Operations, brokers still must respond within 60 seconds but may now return a 202 ACCEPTED, indicating that the requested operation has been accepted but is not complete. This triggers Cloud Foundry to poll a new endpoint /v2/service_instances/:guid/last_operation until the broker indicates that the requested operation has succeeded or failed. During the intervening time, end users are able to discover the state of the requested operation using Cloud Foundry API clients such as the CLI.

For an operation to be executed asynchronously, all three components (CF API client, CF, and broker) must support the feature. The parameter accept_incomplete=true must be passed in a request by the CF API client, triggering CF to include the same parameter in a request to the broker. The broker can then choose to execute the request synchronously or asynchronously.

If the broker executes the request asynchronously, the response must use the status code 202 ACCEPTED; the response body should be the same as if the broker were serving the request synchronously.

Note: Asynchronous Operations are currently supported only for provision, update, and deprovision. Bind and unbind will be added once the
If the `acceptsIncomplete=true` parameter is not included, and the broker cannot fulfill the request synchronously (guaranteeing that the operation is complete on response), then the broker should reject the request with the status code 422 UNPROCESSABLE ENTITY and the following body:

```
{
  "error": "AsyncRequired",
  "description": "This service plan requires client support for asynchronous service operations."
}
```

To execute a request synchronously, the broker need only return the usual status codes; 201 CREATED for create, and 200 OK for update and delete.

### Sequence Diagram

#### Blocking Operations

The Cloud Controller ensures that service brokers do not receive requests for an instance while an asynchronous operation is in progress. For example, if a broker is in the process of provisioning an instance asynchronously, the Cloud Controller will not allow any update, bind, unbind, or deprovision requests to be made through the platform. A user who attempts to perform one of these actions while an operation is already in progress will get an HTTP 400 with error message “Another operation for this service instance is in progress.”

### When to use Asynchronous Service Operations

Service brokers should respond to all Cloud Controller requests within 60 seconds. Brokers that can guarantee completion of the requested operation with the response may return the synchronous response (e.g. 201 CREATED for a provision request). Brokers that cannot guarantee completion of the operation with the response should implement support for asynchronous provisioning. Support for synchronous or asynchronous responses may vary by service offering, even by service plan.
Polling Last Operation (async only)

When a broker returns status code 202 ACCEPTED for provision, update, or deprovision, Cloud Foundry will begin to poll the /v2/service_instances/:guid/last_operation endpoint to obtain the state of the last requested operation. The broker response must contain the field state and an optional field description.

Valid values for state are in progress, succeeded, and failed. Cloud Foundry will poll the last_operation endpoint as long as the broker returns "state": "in progress". Returning "state": "succeeded" or "state": "failed" will cause Cloud Foundry to cease polling. The value provided for description will be passed through to the CF API client and can be used to provide additional detail for users about the state of the operation.

Request

Route

GET /v2/service_instances/:instance_id/last_operation
cURL

$ curl http://username:password@broker-url/v2/service_instances/:instance_id/last_operation

Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>The expected response body is below.</td>
</tr>
<tr>
<td>410 GONE</td>
<td>Appropriate only for asynchronous delete requests. Cloud Foundry will consider this response a success and remove the resource from its database. The expected response body is { }. Returning this while Cloud Foundry is polling for create or update operations will be interpreted as an invalid response and Cloud Foundry will continue polling.</td>
</tr>
</tbody>
</table>

Responses with any other status code will be interpreted as an error or invalid response; Cloud Foundry will continue polling until the broker returns a valid response or the maximum polling duration is reached. Brokers may use the description field to expose user-facing error messages about the operation state; for more info see Broker Errors.

Body

All response bodies must be a valid JSON Object ({}). This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For success responses, the following fields are valid.

<table>
<thead>
<tr>
<th>Response field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state*</td>
<td>string</td>
<td>Valid values are in progress, succeeded, and failed. While &quot;state&quot;: &quot;in progress&quot;, Cloud Foundry will continue polling. A response with &quot;state&quot;: &quot;succeeded&quot; or &quot;state&quot;: &quot;failed&quot; will cause Cloud Foundry to cease polling.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Optional field. A user-facing message displayed to the Cloud Foundry API client. Can be used to tell the user details about the status of the operation.</td>
</tr>
</tbody>
</table>

{  "state": "in progress",  "description": "Creating service (10% complete)"}
Polling Interval

When a broker responds asynchronously to a request from Cloud Foundry containing the `accepts_incomplete=true` parameter, Cloud Foundry will poll the broker for the operation state at a configured interval. The Cloud Foundry operator can configure this interval in the BOSH deployment manifest using the property `properties.cc.broker_client_default_async_poll_interval_seconds` (defaults to 60 seconds). The maximum supported polling interval is 86400 seconds (24 hours).

Maximum Polling Duration

When a broker responds asynchronously to a request from Cloud Foundry containing the `accepts_incomplete=true` parameter, Cloud Foundry will poll the broker for the operation state until the broker response includes `"state":"succeeded"` or `"state":"failed"`, or until a maximum polling duration is reached. If the max polling duration is reached, Cloud Foundry will cease polling and the operation state will be considered `failed`. The Cloud Foundry operator can configure this max polling duration in the BOSH deployment manifest using the property `properties.cc.broker_client_max_async_poll_duration_minutes` (defaults to 10080 minutes or 1 week).

Additional Resources

- An example broker that implements this feature can be found at [Example Service Brokers](#).
- A demo video of the CLI user experience using the above broker can be found [here](#).

Provisioning

When the broker receives a provision request from Cloud Controller, it should synchronously take whatever action is necessary to create a new service resource for the developer. The result of provisioning varies by service type, although there are a few common actions that work for many services. For a MySQL service, provisioning could result in:

- An empty dedicated `mysqld` process running on its own VM.
- An empty dedicated `mysqld` process running in a lightweight container on a shared VM.
- An empty dedicated `mysqld` process running on a shared VM.
- An empty dedicated database, on an existing shared running `mysqld`.
- A database with business schema already there.
- A copy of a full database, for example a QA database that is a copy of the production database.

For non-data services, provisioning could just mean getting an account on an existing system.

Request

**Route**

```
PUT /v2/service_instances/:instance_id
```

**Note:** The `instance_id` of a service instance is provided by the Cloud Controller. This ID will be used for future requests (bind and deprovision), so the broker must use it to correlate the resource it creates.

**Body**

<table>
<thead>
<tr>
<th>Request field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>organization_guid</code>*</td>
<td>string</td>
<td>The Cloud Controller GUID of the organization under which the service is to be provisioned. Although most brokers will not use this field, it could be helpful in determining data placement or applying custom business rules.</td>
</tr>
<tr>
<td><code>plan_id</code>*</td>
<td>string</td>
<td>The ID of the plan within the above service (from the catalog endpoint) that the user would like provisioned. Because plans have identifiers unique to a broker, this is enough information to determine what to provision.</td>
</tr>
<tr>
<td><code>service_id</code>*</td>
<td>string</td>
<td>The ID of the service within the catalog above. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
</tbody>
</table>
space_guid

Similar to organization_guid, but for the space.

parameters

JSON object

Cloud Foundry API clients can provide a JSON object of configuration parameters with their request and this value will be passed through to the service broker. Brokers are responsible for validation.

accepts_incomplete

boolean

A value of true indicates that both the Cloud Controller and the requesting client support asynchronous provisioning. If this parameter is not included in the request, and the broker can only provision an instance of the requested plan asynchronously, the broker should reject the request with a 422 as described below.

```json
{
  "organization_guid": "org-guid-here",
  "plan_id": "plan-guid-here",
  "service_id": "service-guid-here",
  "space_guid": "space-guid-here",
  "parameters": {
    "parameter1": 1,
    "parameter2": "value"
  }
}
```

cURL

```
curl http://username:password@broker-url/v2/service_instances/:instance_id -d '{
  "organization_guid": "org-guid-here",
  "plan_id": "plan-guid-here",
  "service_id": "service-guid-here",
  "space_guid": "space-guide-here",
  "parameters": {
    "parameter1": 1,
    "parameter2": "value"
  }
}' -X PUT -H "X-Broker-API-Version: 2.8" -H "Content-Type: application/json"
```

In this case, `instance_id` refers to the service instance id generated by Cloud Controller.

Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201 Created</td>
<td>Service instance has been created. The expected response body is below.</td>
</tr>
<tr>
<td>200 OK</td>
<td>May be returned if the service instance already exists and the requested parameters are identical to the existing service instance. The expected response body is below.</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>Service instance creation is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>Should be returned if the requested service instance already exists. The expected response body is <code>{}</code>.</td>
</tr>
</tbody>
</table>
| 422 Unprocessable Entity | Should be returned if the broker only supports asynchronous provisioning for the requested plan and the request did not include `?accepts_incomplete=true`. The expected response body is:

```json
{"error": "AsyncRequired", "description": "This service plan requires client support for asynchronous service operations."}
```

Responses with any other status code will be interpreted as a failure. Brokers can include a user-facing message in the `description` field; for details see Broker Errors.

Body

All response bodies must be a valid JSON Object (`{}`). This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For success responses, the following fields are supported. Others will be ignored. For error responses, see Broker Errors.

<table>
<thead>
<tr>
<th>Response field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| The URL of a web-based management user interface for the service instance; we refer to this as a service dashboard. The URL should contain enough information for the dashboard to identify the resource being accessed (“9189kdfsk0vfniku” in the

© Copyright Pivotal Software Inc, 2013-2018
Updating a Service Instance

Brokers that implement this endpoint can enable users to modify attributes of an existing service instance. The first attribute Cloud Foundry supports users modifying is the service plan. This effectively enables users to upgrade or downgrade their service instance to other plans. To see how users make these requests, see Managing Services.

To enable this functionality, a broker declares support for each service by including `plan_updateable: true` in its `catalog endpoint`. If this optional field is not included, Cloud Foundry will return a meaningful error to users for any plan change request, and will not make an API call to the broker. If this field is included and configured as true, Cloud Foundry will make API calls to the broker for all plan change requests, and it is up to the broker to validate whether a particular permutation of plan change is supported. Not all permutations of plan changes are expected to be supported. For example, a service may support upgrading from plan “shared small” to “shared large” but not to plan “dedicated”. If a particular plan change is not supported, the broker should return a meaningful error message in response.

Request

Route

PATCH `/v2/service_instances/:instance_id`

*Note:* `instance_id` is the global unique ID of a previously-provisioned service instance.

Body

<table>
<thead>
<tr>
<th>Request Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id*</td>
<td>string</td>
<td>The ID of the service within the catalog above. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>plan_id</td>
<td>string</td>
<td>ID of the new plan from the catalog.</td>
</tr>
<tr>
<td>parameters</td>
<td>JSON</td>
<td>Cloud Foundry API clients can provide a JSON object of configuration parameters with their request and this value will be passed through to the service broker. Brokers are responsible for validation.</td>
</tr>
<tr>
<td>previous_values</td>
<td>object</td>
<td>Information about the instance prior to the update.</td>
</tr>
<tr>
<td>previous_values.plan_id</td>
<td>string</td>
<td>ID of the plan prior to the update.</td>
</tr>
<tr>
<td>previous_values.service_id</td>
<td>string</td>
<td>ID of the service for the instance.</td>
</tr>
<tr>
<td>previous_values.organization_id</td>
<td>string</td>
<td>ID of the organization containing the instance.</td>
</tr>
<tr>
<td>previous_values.space_id</td>
<td>string</td>
<td>ID of the space containing the instance.</td>
</tr>
<tr>
<td>accepts_incomplete</td>
<td>boolean</td>
<td>A value of true indicates that both the Cloud Controller and the requesting client support asynchronous update. If this parameter is not included in the request, and the broker can only update an instance of the requested plan asynchronously, the broker should reject the request with a 422 as described below.</td>
</tr>
</tbody>
</table>
cURL

```bash
$ curl http://username:password@broker-url/v2/service_instances/:instance_id -d '{
  "service_id": "service-guid-here",
  "plan_id": "plan-guid-here",
  "parameters": {
    "parameter1": 1,
    "parameter2": "value"
  },
  "previous_values": {
    "plan_id": "old-plan-guid-here",
    "service_id": "service-guid-here",
    "organization_id": "org-guid-here",
    "space_id": "space-guid-here"
  }
}' -X PATCH -H "X-Broker-API-Version: 2.8" -H "Content-Type: application/json"
```

Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>New plan is effective. The expected response body is <code>{}</code>.</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>Service instance update is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.</td>
</tr>
<tr>
<td>422 Unprocessable entity</td>
<td>May be returned if the particular plan change requested is not supported or if the request cannot currently be fulfilled due to the state of the instance (e.g., instance utilization is over the quota of the requested plan). Broker should include a user-facing message in the body; for details see Broker Errors. Additionally, a 422 can also be returned if the broker only supports asynchronous update for the requested plan and the request did not include <code>?accepts_incomplete=true</code>. The expected response body is:</td>
</tr>
</tbody>
</table>

```json
{ "error": "AsyncRequired", "description": "This service plan requires client support for asynchronous service operations."
}
```

Responses with any other status code will be interpreted as a failure. Brokers can include a user-facing message in the `description` field; for details see Broker Errors.

Body

All response bodies must be a valid JSON Object `{ }`. This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For a success response, the expected response body is `{ }`.

Binding

**Note:** Not all services must be bindable — some deliver value just from being provisioned. Brokers that offer services that are bindable should declare them as such using `bindable: true` in the Catalog. Brokers that do not offer any bindable services do not need to implement the endpoint for bind requests.
Types of Binding

Credentials

Credentials are a set of information used by an application or a user to utilize the service instance. If `bindable: true` is declared for a service in the catalog endpoint, users may request generation of credentials either by binding the service instance to an application or by creating a service key. When a service instance is bound to an app, Cloud Foundry will send the app id with the request. When a service key is created, the app id is not included. If the broker supports generation of credentials it should return `credentials` in the response. Credentials should be unique whenever possible, so access can be revoked for one application or user without affecting another. For more information on credentials, see Binding Credentials.

Application Log Streaming

In response to a bind request for an application (app_id included), a broker may also enable streaming of application logs from Cloud Foundry to a consuming service instance by returning `syslog_drain_url`. For details, see Application Log Streaming.

Route Services

If a broker has declared `"requires": ["route_forwarding"]` for a service in the Catalog endpoint, Cloud Foundry will permit a user to bind a service to a route. When bound to a route, the route itself will be sent with the bind request. A route is an address used by clients to reach apps mapped to the route. In response a broker may return a `route_service_url` which Cloud Foundry will use to proxy any request for the route to the service instance at URL specified by `route_service_url`. A broker may declare `"requires": ["route_forwarding"]` but not return `route_service_url`; this enables a broker to dynamically configure a network component already in the request path for the route, requiring no change in the Cloud Foundry router. For more information, see Route Services.

Request

Route

```
PUT /v2/service_instances/:instance_id/service_bindings/:binding_id
```

**Note:** The `:binding_id` of a service binding is provided by the Cloud Controller. `:instance_id` is the ID of a previously-provisioned service instance; `:binding_id` will be used for future unbind requests, so the broker must use it to correlate the resource it creates.

Body

<table>
<thead>
<tr>
<th>Request Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id*</td>
<td>string</td>
<td>ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>plan_id*</td>
<td>string</td>
<td>ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>app_guid</td>
<td>string</td>
<td>GUID of the application that you want to bind your service to. Will be included when users bind applications to service instances.</td>
</tr>
<tr>
<td>bind_resource</td>
<td>JSON object</td>
<td>A JSON object that contains the required fields of the resource being bound. Currently only <code>app_guid</code> for application bindings and <code>route</code> for route bindings are supported.</td>
</tr>
<tr>
<td>parameters</td>
<td>JSON object</td>
<td>Cloud Foundry API clients can provide a JSON object of configuration parameters with their request and this value will be passed through to the service broker. Brokers are responsible for validation.</td>
</tr>
</tbody>
</table>
In this case, `instance_id` refers to the id of an existing service instance in a previous provisioning, while `binding_id` is service binding id generated by Cloud Controller.

### Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201 Created</td>
<td>Binding has been created. The expected response body is below.</td>
</tr>
<tr>
<td>200 OK</td>
<td>May be returned if the binding already exists and the requested parameters are identical to the existing binding. The expected response body is below.</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>Should be returned if the requested binding already exists. The expected response body is <code>{}</code> though the description field can be used to return a user-facing error message, as described in Broker Errors.</td>
</tr>
<tr>
<td>422 Unprocessable Entity</td>
<td>Should be returned if the broker requires that <code>app_guid</code> be included in the request body. The expected response body is: <code>{ &quot;error&quot;: &quot;RequiresApp&quot;, &quot;description&quot;: &quot;This service supports generation of credentials through binding an application only.&quot; }</code></td>
</tr>
</tbody>
</table>

Responses with any other status code will be interpreted as a failure and an unbind request will be sent to the broker to prevent an orphan being created.
on the broker. Brokers can include a user-facing message in the description field; for details see Broker Errors.

Body

All response bodies must be a valid JSON Object (`{}`). This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For success responses, the following fields are supported. Others will be ignored. For error responses, see Broker Errors.

<table>
<thead>
<tr>
<th>Response Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentials</td>
<td>object</td>
<td>A free-form hash of credentials that the bound application can use to access the service. For more information, see Binding Credentials.</td>
</tr>
<tr>
<td>syslog_drain_url</td>
<td>string</td>
<td>A URL to which Cloud Foundry should drain logs for the bound application. requires:syslog_drain must be declared in the catalog endpoint or Cloud Foundry will consider the response invalid. For details, see Application Log Streaming.</td>
</tr>
<tr>
<td>route_service_url</td>
<td>string</td>
<td>A URL to which Cloud Foundry should proxy requests for the bound route. requires:route_forwarding must be declared in the catalog endpoint or Cloud Foundry will consider the response invalid. For details, see Route Services.</td>
</tr>
</tbody>
</table>

```json
{
  "credentials": {
    "uri": "mysql://mysqluser:pass@mysqlhost:3306/dbname",
    "username": "mysqluser",
    "password": "pass",
    "host": "mysqlhost",
    "port": 3306,
    "database": "dbname"
  }
}
```

Unbinding

Note: Brokers that do not provide any bindable services do not need to implement the endpoint for unbind requests.

When a broker receives an unbind request from Cloud Controller, it should delete any resources it created in bind. Usually this means that an application immediately cannot access the resource.

Request

Route

```
DELETE /v2/service_instances/:instance_id/service_bindings/:binding_id
```

The :binding_id in the URL is the identifier of a previously created binding (the same :binding_id passed in the bind request). The request has no body, because DELETE requests generally do not have bodies.

Parameters

The request provides these query string parameters as useful hints for brokers.

<table>
<thead>
<tr>
<th>Query-String Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id*</td>
<td>string</td>
<td>ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>plan_id*</td>
<td>string</td>
<td>ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
</tbody>
</table>

cURL
Response

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>Binding was deleted. The expected response body is <code>{}</code>.</td>
</tr>
<tr>
<td>410 Gone</td>
<td>Should be returned if the binding does not exist. The expected response body is <code>{}</code>.</td>
</tr>
</tbody>
</table>

Responses with any other status code will be interpreted as a failure and the binding will remain in the Cloud Controller database. Brokers can include a user-facing message in the `description` field; for details see Broker Errors.

Body

All response bodies must be a valid JSON Object (`{}`). This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For a success response, the expected response body is `{}`.

Deprovisioning

When a broker receives a deprovision request from Cloud Controller, it should delete any resources it created during the provision. Usually this means that all resources are immediately reclaimed for future provisions.

Request

Route

```
DELETE /v2/service_instances/:instance_id
```

The `:instance_id` in the URL is the identifier of a previously provisioned instance (the same `instance_id` passed in the provision request). The request has no body, because DELETE requests generally do not have bodies.

Parameters

The request provides these query string parameters as useful hints for brokers.

<table>
<thead>
<tr>
<th>Query-String Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id*</td>
<td>string</td>
<td>ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>plan_id*</td>
<td>string</td>
<td>ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.</td>
</tr>
<tr>
<td>accepts_incomplete</td>
<td>boolean</td>
<td>A value of true indicates that both the Cloud Controller and the requesting client support asynchronous deprovisioning. If this parameter is not included in the request, and the broker can only deprovision an instance of the requested plan asynchronously, the broker should reject the request with a 422 as described below.</td>
</tr>
</tbody>
</table>

cURL

```
curl 'http://username:password@broker-url/v2/service_instances/:instance_id/service_bindings/:binding_id?service_id=service-id-here&plan_id=plan-id-here' -X DELETE -H "X-Broker-API-Version: 2.8"
```
<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>Service instance was deleted. The expected response body is <code>{}</code>.</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>Service instance deletion is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.</td>
</tr>
<tr>
<td>410 Gone</td>
<td>Should be returned if the service instance does not exist. The expected response body is <code>{}</code>.</td>
</tr>
</tbody>
</table>
| 422 Unprocessable Entity | Should be returned if the broker only supports asynchronous deprovisioning for the requested plan and the request did not include `?accepts_incomplete=true`. The expected response body is:`
```json
{
  "error": "AsyncRequired",
  "description": "This service plan requires client support for asynchronous service operations."
}
```
, as described below.

Responses with any other status code will be interpreted as a failure and the service instance will remain in the Cloud Controller database. Brokers can include a user-facing message in the `description` field; for details see Broker Errors.

Body

All response bodies must be a valid JSON Object `{}`. This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For a success response, the expected response body is `{}`.

Broker Errors

Response

Broker failures beyond the scope of the well-defined HTTP response codes listed above (like 410 on delete) should return an appropriate HTTP response code (chosen to accurately reflect the nature of the failure) and a body containing a valid JSON Object (not an array).

Body

All response bodies must be a valid JSON Object `{}`. This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For error responses, the following fields are valid. Others will be ignored. If an empty JSON object is returned in the body `{}`, a generic message containing the HTTP response code returned by the broker will be displayed to the requestor.

<table>
<thead>
<tr>
<th>Response Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>string</td>
<td>An error message explaining why the request failed. This message will be displayed to the user who initiated the request.</td>
</tr>
</tbody>
</table>

```
{
  "description": "Something went wrong. Please contact support at http://support.example.com."
}
```

Orphans

The Cloud Controller is the source of truth for service instances and bindings. Service brokers are expected to have successfully provisioned all the instances and bindings Cloud Controller knows about, and none that it doesn’t.

Orphans can result if the broker does not return a response before a request from Cloud Controller times out (typically 60 seconds). For example, if a broker does not return a response to a provision request before Cloud Controller times out, the broker might eventually succeed in provisioning an instance after Cloud Controller considers the request a failure. This results in an orphan instance on the service side.

To mitigate orphan instances and bindings, Cloud Controller will attempt to delete resources it cannot be sure were successfully created, and will keep trying to delete them until the broker responds with a success.

More specifically, when a provision or bind request to the broker fails, Cloud Controller will immediately send a corresponding delete or unbind request. If
If the delete or unbind request fails, Cloud Controller will retry the delete or unbind request ten times with an exponential backoff schedule (over a period of 34 hours).

<table>
<thead>
<tr>
<th>Status code</th>
<th>Result</th>
<th>Orphan mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>200 with malformed response</td>
<td>Failure</td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>201 with malformed response</td>
<td>Failure</td>
<td>Yes</td>
</tr>
<tr>
<td>All other 2xx</td>
<td>Failure</td>
<td>Yes</td>
</tr>
<tr>
<td>408</td>
<td>Failure due to timeout</td>
<td>Yes</td>
</tr>
<tr>
<td>All other 4xx</td>
<td>Broker rejects request</td>
<td></td>
</tr>
<tr>
<td>5xx</td>
<td>Broker error</td>
<td>Yes</td>
</tr>
<tr>
<td>Timeout</td>
<td>Failure</td>
<td>Yes</td>
</tr>
</tbody>
</table>

If the Cloud Controller encounters an internal error provisioning an instance or binding (for example, saving to the database fails), then the Cloud Controller will send a single delete or unbind request to the broker but will not retry.

This orphan mitigation behavior was introduced in cf-release v196.
Managing Service Brokers

Page last updated:

This page assumes you are using cf CLI v6.16 or later.

In order to run many of the commands below, you must be authenticated with Cloud Foundry as an admin user or as a space developer.

Quick Start

Given a service broker that has implemented the Service Broker API, two steps are required to make its services available to end users in all organizations or a limited number of organizations by service plan.

1. **Register a Broker**

2. **Make Plans Public**

As of cf-release 229, CC API 2.47.0, Cloud Foundry supports both standard brokers and space-scoped private brokers. Standard private brokers can offer service plans privately or publish them to specific organizations or to all users. Space-scoped private brokers publish services only to users within the space they are registered to.

Register a Broker

Registering a broker causes Cloud Controller to fetch and validate the catalog from your broker, and save the catalog to the Cloud Controller database. The basic auth username and password which are provided when adding a broker are encrypted in Cloud Controller database, and used by the Cloud Controller to authenticate with the broker when making all API calls. Your service broker should validate the username and password sent in every request; otherwise, anyone could curl your broker to delete service instances.

**Standard Private Brokers**

```
$ cf create-service-broker mybrokername someuser somethingsecure http://mybroker.example.com/
```

**Space-Scoped Private Brokers**

```
$ cf create-service-broker mybrokername someuser somethingsecure http://mybroker.example.com/ --space-scoped
```

Make Plans Public

New service plans from standard brokers are private by default. To make plans available to end users, see Make Plans Public. Instances of a private plan cannot be provisioned until either the plan is made public or is made available to an organization.

New service plans from space-scoped private brokers are automatically published to all users in the broker's space. It is not possible to manage visibility of a space-scoped private broker at the Cloud Foundry instance or organization level.

Multiple Brokers, Services, Plans

Many service brokers may be added to a Cloud Foundry instance, each offering many services and plans. The following constraints should be kept in mind:

- It is not possible to have multiple brokers with the same name
- It is not possible to have multiple brokers with the same base URL
- The service ID and plan IDs of each service advertised by the broker must be unique across Cloud Foundry. GUIDs are recommended for these fields.
See Possible Errors below for error messages and what to do when you see them.

List Service Brokers

```
$ cf service-brokers
Getting service brokers as admin...Cloud Controller
OK

Name    URL
my-service-name http://mybroker.example.com
```

Update a Broker

Updating a broker is how to ingest changes a broker author has made into Cloud Foundry. Similar to adding a broker, update causes Cloud Controller to fetch the catalog from a broker, validate it, and update the Cloud Controller database with any changes found in the catalog.

Update also provides a means to change the basic auth credentials cloud controller uses to authenticate with a broker, as well as the base URL of the broker’s API endpoints.

```
$ cf update-service-broker mybrokername someuser somethingsecure http://mybroker.example.com/
```

Rename a Broker

A service broker can be renamed with the `rename-service-broker` command. This name is used only by the Cloud Foundry operator to identify brokers, and has no relation to configuration of the broker itself.

```
$ cf rename-service-broker mybrokername mynewbrokername
```

Remove a Broker

Removing a service broker will remove all services and plans in the broker’s catalog from the Cloud Foundry Marketplace.

```
$ cf delete-service-broker mybrokername
```

Note: Attempting to remove a service broker will fail if there are service instances for any service plan in its catalog. When planning to shut down or delete a broker, make sure to remove all service instances first. Failure to do so will leave orphaned service instances in the Cloud Foundry database. If a service broker has been shut down without first deleting service instances, you can remove the instances with the CLI; see Purge a Service.

Purge a Service

If a service broker has been shut down or removed without first deleting service instances from Cloud Foundry, you will be unable to remove the service broker or its services and plans from the Marketplace. In development environments, broker authors often destroy their broker deployments and need a way to clean up the Cloud Controller database.

The following command will delete a service offering, all of its plans, as well as all associated service instances and bindings from the Cloud Controller database, without making any API calls to a service broker. For services from v1 brokers, you must provide a provider with `-p PROVIDER`. Once all services for a broker have been purged, the broker can be removed normally.
$ cf purge-service-offering v1-test -p pivotal-software
Warning: This operation assumes that the service broker responsible for this service offering is no longer available, and all service instances have been deleted, leaving orphan records in Cloud Foundry's database. All knowledge of the service will be removed from Cloud Foundry, including service instances and service bindings. No attempt will be made to contact the service broker; running this command without destroying the service broker will cause orphan service instances. After running this command you may want to run either delete-service-auth-token or delete-service-broker to complete the cleanup.

Really purge service offering v1-test from Cloud Foundry? y
OK

Purge a Service Instance

The following command will delete a single service instance, its service bindings and its service keys from the Cloud Controller database, without making any API calls to a service broker. This can be helpful in instances a Service Broker is not conforming to the Service Broker API and not returning a 200 or 410 to requests to delete the service instance.

$ cf purge-service-instance mysql-dev
WARNING: This operation assumes that the service broker responsible for this service instance is no longer available or is not responding with a 200 or 410, and the service instance has been deleted, leaving orphan records in Cloud Foundry's database. All knowledge of this service instance will be removed from Cloud Foundry, including service bindings and service keys.

Really purge service instance mysql-dev from Cloud Foundry? y
Purging service mysql-dev...
OK


Possible Errors

If incorrect basic auth credentials are provided:

Server error, status code: 500, error code: 10001, message: Authentication failed for the service broker API.
Double-check that the username and password are correct:
http://github-broker.a1-app.example.com/v2/catalog

If you receive the following errors, check your broker logs. You may have an internal error.

Server error, status code: 500, error code: 10001, message: The service broker response was not understood

Server error, status code: 500, error code: 10001, message: The service broker API returned an error from http://github-broker.a1-app.example.com/v2/catalog: 404 Not Found


If your broker's catalog of services and plans violates validation of presence, uniqueness, and type, you will receive meaningful errors.

Server error, status code: 502, error code: 270012, message: Service broker catalog is invalid:
Service service-name-1
  service id must be unique
service description is required
service "bindable" field must be a boolean, but has value "true"
Plan plan-name-1
  plan metadata must be a hash, but has value [{"bullet1"="bullet1", "bullet2"}]
Access Control

All new service plans from standard private brokers are private by default. This means that when adding a new broker, or when adding a new plan to an existing broker’s catalog, service plans won’t immediately be available to end users. This lets an admin control which service plans are available to end users, and manage limited service availability.

Space-scoped private brokers are registered to a specific space, and all users within that space can automatically access the broker’s service plans. With space-scoped brokers, service visibility is not managed separately.

Using the CLI

If your CLI and/or deployment of cf-release do not meet the following prerequisites, you can manage access control with cf curl.

Prerequisites

- CLI v6.4.0
- Cloud Controller API v2.9.0 (cf-release v179)
- Admin user access; the following commands can be run only by an admin user

To determine your API version, curl /v2/info and look for api_version.

```
$ cf curl /v2/info
{
  "name": "vcap",
  "build": "2222",
  "support": "https://support.cloudfoundry.com",
  "version": 2,
  "description": "Cloud Foundry sponsored by Pivotal",
  "authorization_endpoint": "https://login.system-domain.example.com",
  "token_endpoint": "https://uaa.system-domain.example.com",
  "api_version": "2.13.0",
  "logging_endpoint": "wss://loggregator.system-domain.example.com:443"
}
```

Display Access to Service Plans

The service-access CLI command enables an admin to see the current access control setting for every service plan in the marketplace, across all service brokers.

```
$ cf service-access
getting service access as admin...
broker: p-rick
  service plan: access orgs
  p-rick: developer limited
broker: p-mysql
  service plan: access orgs
  p-mysql: 100mb-dev all
```

The access column has values all, limited, or none. all means a service plan is available to all users of the Cloud Foundry instance; this is what we mean when we say the plan is “public”. none means the plan is not available to anyone; this is what we mean when we say the plan is “private”. limited means that the service plan is available to users of one or more select organizations. When a plan is limited, organizations that have been granted access are listed.

Flags provide filtering by broker, service, and organization.
Enable Access to Service Plans

Service access is managed at the granularity of service plans, though CLI commands allow an admin to modify all plans of a service at once.

Enabling access to a service plan for organizations allows users of those organizations to see the plan listed in the marketplace (cf marketplace), and if users have the Space Developer role in a targeted space, to provision instances of the plan.

An admin can use the `enable-service-access` command to:

- Enable access to all plans of a service for users of all orgs (access: all)
- Enable access to one plan of a service for users of all orgs (access: all)
- Enable access to all plans of a service for users of a specified organization (access: limited)
- Enable access to one plan of a service for users of a specified organization (access: limited)

Disable Access to Service Plans

An admin can use the `disable-service-access` command to:

- Disable access to all plans of a service for users of all orgs (access: all)
- Disable access to one plan of a service for users of all orgs (access: all)
- Disable access to all plans of a service for users of select orgs (access: limited)
- Disable access to one plan of a service for users of select orgs (access: limited)
$ cf help disable-service-access
NAME:
disable-service-access - Disable access to a service or service plan for one or all orgs.

USAGE:
cf disable-service-access SERVICE [-p PLAN][ -o ORG]

OPTIONS:
-p Disable access to a particular service plan
-o Disable access to a particular organization

Limitations
- You cannot disable access to a service plan for an organization if the plan is currently available to all organizations. You must first disable access for all organizations; then you can enable access for a particular organization.

Using cf curl

The following commands must be run as a system admin user.

Enable Access to Service Plans

Access can be enabled for users of all organizations, or for users of particular organizations. Service plans which are available to all users are said to be “public”. Plans that are available to no organizations, or to particular organizations, are said to be “private”.

Enable access to a plan for all organizations

Once made public, the service plan can be seen by all users in the list of available services. See Managing Services for more information.

To make a service plan public, you need the service plan GUID. To find the service plan GUID, run:

```bash
cf curl /v2/service_plans -X 'GET'
```

This command returns a filtered JSON response listing every service plan. Data about each plan shows in two sections: metadata and entity. The metadata section shows the service plan GUID, while the entity section lists the name of the plan. Note: Because metadata is listed before entity for each service plan, the GUID of a plan is shown six lines above the name.

Example:

```json
$
cf curl /v2/service_plans
...

{ "metadata": { "guid": "1afd5050-664e-4be2-9389-6bf0c967c0c6", "url": "/v2/service_plans/1afd5050-664e-4be2-9389-6bf0c967c0c6", "created_at": "2014-02-12T06:24:04+00:00", "updated_at": "2014-02-12T18:46:52+00:00" },

"entity": { "name": "plan-name-1", "free": true,
"description": "plan-desc-1", "service_guid": "d9011411-1463-477c-b223-82e04996b91f", "extra": "[" bullets": ["bullet1","bullet2"]],
"unique_id": "plan-id-1", "public": false,
"service_url": "/v2/services/d9011411-1463-477c-b223-82e04996b91f",
"service_instances_url": "/v2/service_plans/1afd5050-664e-4be2-9389-6bf0c967c0c6/service_instances"
}
```

In this example, the GUID of plan-name-1 is 1afd5050-664e-4be2-9389-6bf0c967c0c6.

To make a service plan public, run:

```bash
cf curl /v2/service_plans/SERVICE_PLAN_GUID -X 'PUT' -d '{"public":true}'
```
As verification, the "entity" section of the JSON response shows the "public":true key-value pair.

```bash
cf curl /v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111 -X 'PUT' -d '{"public":true}'
```

```json
{
  "metadata": {
    "guid": "1113aa0-124e-4af2-1526-6bfacf61b111",
    "url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111",
    "created_at": "2014-02-12T06:24:04+00:00",
    "updated_at": "2014-02-12T20:55:10+00:00"
  },
  "entity": {
    "name": "plan-name-1",
    "free": true,
    "description": "plan-desc-1",
    "service_guid": "d9011411-1463-477c-8223-82e04996b91f",
    "extra": 
      {
        "[bullet1]: "bullet1",
        "[bullet2]: "bullet2"
      }
    "unique_id": "plan-id-1",
    "public": true,
    "service_url": "/v2/services/d9011411-1463-477c-8223-82e04996b91f",
    "service_instances_url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111/service_instances"
  }
}
```

Enable access to a private plan for a particular organization

Users have access to private plans that have been enabled for an organization only when targeting a space of that organization. See Managing Services for more information.

To make a service plan available to users of a specific organization, you need the GUID of both the organization and the service plan. To get the GUID of the service plan, run the same command described above for enabling access to a plan for all organizations:

```bash
cf curl -X 'GET' /v2/service_plans
```

To find the organization GUIDs, run:

```bash
cf curl /v2/organizations?q=name:YOUR-ORG-NAME
```

The `metadata` section shows the organization GUID, while the `entity` section lists the name of the organization. Note: Because `metadata` is listed before `entity` for each organization, the GUID of an organization is shown six lines above the name.

Example:

```bash
cf curl /v2/organizations?q=name:my-org
```

```json
{
  "metadata": {
    "guid": "c54bf317-d791-4d12-89f0-b56d0936cfdc",
    "url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc",
    "created_at": "2013-05-06T16:34:56+00:00",
    "updated_at": "2013-09-25T18:44:35+00:00"
  },
  "entity": {
    "name": "my-org",
    "billing_enabled": true,
    "quota_definition_guid": "52e5413c-869f-453a-a887-7972ecb85c6a",
    "status": "active",
    "quota_definition_url": "/v2/quotas/52e5413c-869f-453a-a887-7972ecb85c6a",
    "spaces_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/spaces",
    "domain_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/dominions",
    "private_domains_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/private_domains",
    "users_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/users",
    "managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/managers",
    "billing_managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/billing_managers",
    "auditors_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/auditors",
    "app_events_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/app_events"
  }
}
```

In this example, the GUID of my-org is c54bf317-d791-4d12-89f0-b56d0936cfdc.

To make a private plan available to a specific organization, run:
Members of my-org can now see the plan-name-1 service plan in the list of available services when a space of my-org is targeted.

Note: The guid field in the metadata section of this JSON response is the id of the “service plan visibility”, and can be used to revoke access to the plan for the organization as described below.

Disable Access to Service Plans

Disable access to a plan for all organizations

To make a service plan private, follow the instructions above for **Enable Access**, but replace "public":true with "public":false.

Note: organizations that have explicitly been granted access will retain access once a plan is private. To be sure access is removed for all organizations, access must be explicitly revoked for organizations to which access has been explicitly granted. For details see below.

Example making plan-name-1 private:

```
$ cf curl /v2/service_plans/1113aa0-124c-4af2-1526-6bf6c61b111-X 'PUT' -d '{ "public":false}'
```

```
{  "metadata": {  "guid": "1113aa0-124c-4af2-1526-6bf6c61b111",  "url": "/v2/service_plans/1113aa0-124c-4af2-1526-6bf6c61b111",  "created_at": "2014-02-12T06:24:04+00:00",  "updated_at": "2014-02-12T20:55:10+00:00" },  "entity": {  "name": "plan-name-1",  "free": true,  "description": "plan-desc-1",  "service_guid": "d9011411-1463-477c-b223-82e04996b91f",  "extra": { "bullets": ["bullet1","bullet2"] },  "unique_id": "plan-id-1",  "public": false,  "service_url": "/v2/services/d9011411-1463-477c-b223-82e04996b91f",  "service_instances_url": "/v2/service_plans/1113aa0-124c-4af2-1526-6bf6c61b111/service_instances"  }}
```

Disable access to a private plan for a particular organization

To revoke access to a service plan for a particular organization, run:

```
cf curl /v2/service_plan_visibilities/SERVICE_PLAN_VISIBILITIES_GUID-X 'DELETE'
```

Example:
Catalog Metadata

Page last updated:

The Services Marketplace is defined as the aggregate catalog of services and plans exposed to end users of a Cloud Foundry instance. Marketplace services may come from one or many service brokers. The Marketplace is exposed to end users by cloud controller clients (web, CLI, IDEs, etc), and the Cloud Foundry community is welcome to develop their own clients. All clients are not expected to have the same requirements for information to expose about services and plans. This document discusses user-facing metadata for services and plans, and how the broker API enables broker authors to provide metadata required by different cloud controller clients.

As described in the Service Broker API, the only required user-facing fields are `label` and `description` for services, and `name` and `description` for service plans. Rather than attempt to anticipate all potential fields that clients will want, or add endless fields to the API spec over time, the broker API provides a mechanism for brokers to advertise any fields a client requires. This mechanism is the `metadata` field.

The contents of the `metadata` field are not validated by cloud controller but may be by cloud controller clients. Not all clients will make use of the value of `metadata`, and not all brokers have to provide it. If a broker does advertise the `metadata` field, client developers can choose to display some or all fields available.

Note: In the v1 broker API, the `metadata` field was called `extra`.

Community-Driven Standards

This page provides a place to publish the metadata fields required by popular cloud controller clients. Client authors can add their metadata requirements to this document, so that broker authors can see what metadata they should advertise in their catalogs.

Before adding new fields, consider whether an existing one will suffice.

Note: “CLI strings” are all lowercase, no spaces. Keep it short; imagine someone having to type it as an argument for a longer CLI command.

### Services Metadata Fields

<table>
<thead>
<tr>
<th>Broker API Field</th>
<th>Type</th>
<th>Description</th>
<th>CC API Field</th>
<th>Pivotal CLI</th>
<th>Pivotal Apps Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>CLI string</td>
<td>A short name for the service to be displayed in a catalog.</td>
<td>label</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A short 1-line description for the service, usually a single sentence or phrase.</td>
<td>description</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>metadata.displayName</td>
<td>string</td>
<td>The name of the service to be displayed in graphical clients</td>
<td>extra.displayName</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.imageUrl</td>
<td>string</td>
<td>The URL to an image</td>
<td>extra.imageUrl</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.longDescription</td>
<td>string</td>
<td>Long description</td>
<td>extra.longDescription</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.providerDisplayName</td>
<td>string</td>
<td>The name of the upstream entity providing the actual service</td>
<td>extra.providerDisplayName</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.documentationUrl</td>
<td>string</td>
<td>Link to documentation page for service</td>
<td>extra.documentationUrl</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.supportUrl</td>
<td>string</td>
<td>Link to support for the service</td>
<td>extra.supportUrl</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### Plan Metadata Fields

<table>
<thead>
<tr>
<th>Broker API Field</th>
<th>Type</th>
<th>Description</th>
<th>CC API Field</th>
<th>Pivotal CLI</th>
<th>Pivotal Apps Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>CLI string</td>
<td>A short name for the service plan to be displayed in a catalog.</td>
<td>name</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description of the service plan to be displayed in a catalog.</td>
<td>description</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example Broker Response Body

The example below contains a catalog of one service, having one service plan. Of course, a broker can offering a catalog of many services, each having many plans.

<table>
<thead>
<tr>
<th>Broker API Field</th>
<th>Type of</th>
<th>Description</th>
<th>CC API Field</th>
<th>Pivotal CLI</th>
<th>Pivotal Apps Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata.bullets</td>
<td>strings</td>
<td>Features of this plan, to be displayed in a bulleted-list</td>
<td>extra.bullets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>metadata.costs</td>
<td>cost object</td>
<td>An array-of-objects that describes the costs of a service, in what currency, and the unit of measure. If there are multiple costs, all of them could be billed to the user (such as a monthly + usage costs at once). Each object must provide the following keys: <code>{ amount: { usd: float }, unit: string }</code> This indicates the cost in USD of the service plan, and how frequently the cost is occurred, such as “MONTHLY” or “per 1000 messages”.</td>
<td>extra.costs</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>metadata.displayName</td>
<td>string</td>
<td>Name of the plan to be display in graphical clients.</td>
<td>extra.displayName</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Example Cloud Controller Response Body
null
Dashboard Single Sign-On

Page last updated:

Introduction

Single sign-on (SSO) enables Cloud Foundry users to authenticate with third-party service dashboards using their Cloud Foundry credentials. Service dashboards are web interfaces which enable users to interact with some or all of the features the service offers. SSO provides a streamlined experience for users, limiting repeated logins and multiple accounts across their managed services. The user’s credentials are never directly transmitted to the service since the OAuth2 protocol handles authentication.

Dashboard SSO was introduced in cf-release v169 so this or a newer version is required to support the feature.

Enabling the feature in Cloud Foundry

To enable the SSO feature, the Cloud Controller requires a UAA client with sufficient permissions to create and delete clients for the service brokers that request them. This client can be configured by including the following snippet in the cf-release manifest:

```json
properties:
  uaa:
    clients:
      cc-service-dashboards:
        secret: cc-broker-secret
        scope: openid,cloud_controller_service_permissions.read
        authorities: clients.read,clients.write,clients.admin
        authorized-grant-types: authorization_code,client_credentials
```

When this client is not present in the cf-release manifest, Cloud Controller cannot manage UAA clients and an operator will receive a warning when creating or updating service brokers that advertise the `dashboard_client` properties discussed below.

Service Broker Responsibilities

Registering the Dashboard Client

1. A service broker must include the `dashboard_client` field in the JSON response from its catalog endpoint for each service implementing this feature. A valid response would appear as follows:

```json
{
  "services": [
    {
      "id": "44b26033-1f54-4087-b7bc-da9652c2a539",
      "dashboard_client": {
        "id": "p-mysql-client",
        "secret": "p-mysql-secret",
        "redirect_uri": "http://p-mysql.example.com/manage/auth"
      }
    }
  ]
}
```

The `dashboard_client` field is a hash containing three fields:
- **id** is the unique identifier for the OAuth2 client that will be created for your service dashboard on the token server (UAA), and will be used by your dashboard to authenticate with the token server (UAA).
- **secret** is the shared secret your dashboard will use to authenticate with the token server (UAA).
- **redirect_uri** is used by the token server as an additional security precaution. UAA will not provide a token if the callback URL declared by the service dashboard doesn’t match the domain name in `redirect_uri`. The token server matches on the domain name, so any paths will also match; e.g. a service dashboard requesting a token and declaring a callback URL of `http://p-mysql.example.com/manage/auth` would be approved if `redirect_uri` for its client is `http://p-mysql.example.com/`. © Copyright Pivotal Software Inc, 2013-2018
2. When a service broker which advertises the dashboard_client property for any of its services is added or updated, Cloud Controller will create or
update UAA clients as necessary. This client will be used by the service dashboard to authenticate users.

Dashboard URL
A service broker should return a URL for the dashboard_url field in response to a provision request. Cloud Controller clients should expose this URL to
users. dashboard_url can be found in the response from Cloud Controller to create a service instance, enumerate service instances, space summary, and
other endpoints.
Users can then navigate to the service dashboard at the URL provided by dashboard_url , initiating the OAuth2 login flow.

Service Dashboard Responsibilities
OAuth2 Flow
When a user navigates to the URL from dashboard_url , the service dashboard should initiate the OAuth2 login flow. A summary of the flow can be found in
section 1.2 of the OAuth2 RFC . OAuth2 expects the presence of an Authorization Endpoint  and a Token Endpoint . In Cloud Foundry, these
endpoints are provided by the UAA. Clients can discover the location of UAA from Cloud Controller’s info endpoint; in the response the location can be
found in the token_endpoint field.
$ curl api.example.com/info
{"name":"vcap","build":"2222","support":"http://support.example.com","version
":2,"description":"Cloud Foundry sponsored by Pivotal","authorization_endpoint":
"https://login.example.com","token_endpoint":"https://uaa.example.com",
"allow_debug":true}



To enable service dashboards to support SSO for service instances created from different Cloud Foundry instances, the /v2/info url is sent to
service brokers in the `X-Api-Info-Location` header of every API call. A service dashboard should be able to discover this URL from the broker, and
enabling the dashboard to contact the appropriate UAA for a particular service instance.

A service dashboard should implement the OAuth2 Authorization Code Grant type ( UAA docs

,

RFC docs

).

1. When a user visits the service dashboard at the value of dashboard_url , the dashboard should redirect the user’s browser to the Authorization
Endpoint and include its client_id , a redirect_uri (callback URL with domain matching the value of dashboard_client.redirect_uri ), and list of requested
scopes.
Scopes are permissions included in the token a dashboard client will receive from UAA, and which Cloud Controller uses to enforce access. A client
should request the minimum scopes it requires. The minimum scopes required for this workflow are cloud_controller_service_permissions.read and
openid . For an explanation of the scopes available to dashboard clients, see On Scopes.
2. UAA authenticates the user by redirecting the user to the Login Server, where the user then approves or denies the scopes requested by the service
dashboard. The user is presented with human readable descriptions for permissions representing each scope. After authentication, the user’s
browser is redirected back to the Authorization endpoint on UAA with an authentication cookie for the UAA.
3. Assuming the user grants access, UAA redirects the user’s browser back to the value of redirect_uri the dashboard provided in its request to the
Authorization Endpoint. The Location header in the response includes an authorization code.
HTTP/1.1 302 Found
Location: https://p-mysql.example.com/manage/auth?code=F45jH

4. The dashboard UI should then request an access token from the Token Endpoint by including the authorization code received in the previous step.
When making the request the dashboard must authenticate with UAA by passing the client id and secret in a basic auth header. UAA will verify that
the client id matches the client it issued the code to. The dashboard should also include the redirect_uri used to obtain the authorization code for
verification.
5. UAA authenticates the dashboard client, validates the authorization code, and ensures that the redirect URI received matches the URI used to
redirect the client when the authorization code was issues. If valid, UAA responds back with an access token and a refresh token.

Checking User Permissions
© Copyright Pivotal Software Inc, 2013-2018

716

1.7


UAA is responsible for authenticating a user and providing the service with an access token with the requested permissions. However, after the user has been logged in, it is the responsibility of the service dashboard to verify that the user making the request to manage an instance currently has access to that service instance.

The service can accomplish this with a GET to the `/v2/service_instances/:guid/permissions` endpoint on the Cloud Controller. The request must include a token for an authenticated user and the service instance guid. The token is the same one obtained from the UAA in response to a request to the Token Endpoint, described above.

Example Request:

```
curl -H 'Content-Type: application/json' -H 'Authorization: bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoid'
http://api.cloudfoundry.com/v2/service_instances/44b26033-1f54-4087-b7bc-da9652ca539/permissions
```

Response:

```
{
  "manage": true
}
```

The response will indicate to the service whether this user is allowed to manage the given instance. A `true` value for the `manage` key indicates sufficient permissions; `false` would indicate insufficient permissions. Since administrators may change the permissions of users, the service should check this endpoint whenever a user uses the SSO flow to access the service’s UI.

On Scopes
Scopes let you specify exactly what type of access you need. Scopes limit access for OAuth tokens. They do not grant any additional permission beyond that which the user already has.

Minimum Scopes

The following two scopes are necessary to implement the integration. Most dashboard shouldn’t need more permissions than these scopes enabled.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>openid</code></td>
<td>Allows access to basic data about the user, such as email addresses</td>
</tr>
<tr>
<td>cloud_controller_service_permissions.read</td>
<td>Allows access to the CC endpoint that specifies whether the user can manage a given service instance</td>
</tr>
</tbody>
</table>

Additional Scopes

Dashboards with extended capabilities may need to request these additional scopes:

<table>
<thead>
<tr>
<th>Scope</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud_controller.read</td>
<td>Allows read access to all resources the user is authorized to read</td>
</tr>
<tr>
<td>cloud_controller.write</td>
<td>Allows write access to all resources the user is authorized to update / create / delete</td>
</tr>
</tbody>
</table>

Reference Implementation

The MySQL Service Broker is an example of a broker that also implements a SSO dashboard. The login flow is implemented using the OmniAuth library and a custom UAA OAuth Strategy. See this [OmniAuth wiki page](https://github.com/omniauth/omniauth/wiki) for instructions on how to create your own strategy.

The UAA OAuth strategy is used to first get an authorization code, as documented in this section of the UAA documentation. The user is redirected back to the service (as specified by the `callback_path` option or the default `auth/cloudfoundry/callback` path) with the authorization code. Before the application/action is dispatched, the OmniAuth strategy uses the authorization code to get a token and uses the token to request information from UAA to fill the `omniauth.auth` environment variable. When OmniAuth returns control to the application, the `omniauth.auth` environment variable hash will be filled with the token and user information obtained from UAA as seen in the Auth Controller.
Restrictions

- UAA clients are scoped to services. There must be a `dashboard_client` entry for each service that uses SSO integration.
- Each `dashboard_client_id` must be unique across the CloudFoundry deployment.

Resources

- OAuth2
- Example broker with SSO implementation
- Cloud Controller API Docs
- User Account and Authentication (UAA) Service APIs
Example Service Brokers

Page last updated:

The following example service broker applications have been developed - these are a great starting point if you are developing your own service broker.

Ruby

- GitHub repo service - this is designed to be an easy-to-read example of a service broker, with complete documentation, and comes with a demo app that uses the service. The broker can be deployed as an application to any Cloud Foundry instance or hosted elsewhere. The service broker uses GitHub as the service back end.

- MySQL database service - this broker and its accompanying MySQL server are designed to be deployed together as a BOSH release. BOSH is used to deploy or upgrade the release, monitors the health of running components, and restarts or recreates unhealthy VMs. The broker code alone can be found here.

Java

- Spring Cloud - Cloud Foundry Service Broker - This implements the REST contract for service brokers and the artifacts are published to the spring maven repo. This greatly simplifies development: include a single dependency in Gradle, implement interfaces, and configure. A sample implementation has been provided for MongoDB.

- MySQL Java Broker - a Java port of the Ruby-based MySQL broker above.

Go

- Asynchronous Service Broker for AWS EC2 - This broker implements support for the experimental Asynchronous Service Operations, and calls AWS APIs to provision EC2 VMs.
Binding Credentials

Page last updated:

A bindable service returns credentials that an application can consume in response to the `cf bind` API call. Cloud Foundry writes these credentials to the `VCAP_SERVICES` environment variable. In some cases, buildpacks write a subset of these credentials to other environment variables that frameworks might need.

Choose from the following list of credential fields if possible, though you can provide additional fields as needed. Refer to the `Using Bound Services` section of the `Managing Service Instances with the CLI` topic for information on how these credentials are consumed.

<table>
<thead>
<tr>
<th>CREDENTIALS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>Connection string of the form <code>DB-TYPE://USERNAME:PASSWORD@HOSTNAME:PORT/NAME</code>, where <code>DB-TYPE</code> is a type of database such as mysql, postgres, mongodb, or amqp.</td>
</tr>
<tr>
<td>hostname</td>
<td>FQDN of the server host</td>
</tr>
<tr>
<td>port</td>
<td>Port of the server host</td>
</tr>
<tr>
<td>name</td>
<td>Name of the service instance</td>
</tr>
<tr>
<td>vhost</td>
<td>Name of the messaging server virtual host - a replacement for a <code>name</code> specific to AMQP providers</td>
</tr>
<tr>
<td>username</td>
<td>Server user</td>
</tr>
<tr>
<td>password</td>
<td>Server password</td>
</tr>
</tbody>
</table>

The following is an example output of `ENV[VCAP_SERVICES]`.

```json
VCAP_SERVICES=
{
  cleardb: {
    name: "cleardb-1",
    label: "cleardb",
    plan: "spark",
    credentials: {
      name: "ad_c6f4446532610ab",
      hostname: "us-cdbr-east-03.cleardb.com",
      port: "3306",
      username: "b5d435f40dd2b2",
      password: "ebfc00ac",
      uri: "mysql://b5d435f40dd2b2:ebfc00ac@us-cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab",
      jdbcUrl: "jdbc:mysql://b5d435f40dd2b2:ebfc00ac@us-cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab"
    }
  }
  cloudamqp: {
    name: "cloudamqp-6",
    label: "cloudamqp",
    plan: "lemur",
    credentials: {
      uri: "amqp://ksvyjmiv:IwN6dCdZmeQD4O0ZPKpu1YOaLsh8woFlemur.cloudamqp.com/ksvyjmiv"
    }
  }
  cloudamqp: {
    name: "cloudamqp-9dbc6",
    label: "cloudamqp",
    plan: "lemur",
    credentials: {
      uri: "amqp://vhuklnxa:91NFxp9JaAd7ts98vQ1dXW3M0jyMyV9lemur.cloudamqp.com/vhuklnxa"
    }
  }
  rediscloud: {
    name: "rediscloud-1",
    label: "rediscloud",
    plan: "20eb",
    credentials: {
      port: "6379",
    }
  }
}
```

**Note:** If you provide a service that supports a connection string, provide the `uri` key for buildpacks and application libraries to use.

**Note:** Depending on the types of databases you are using, each database might return different credentials.
host: "pub-redis-6379.us-east-1-2.3.ec2.redislabs.com",
password: "!M5zd3qfNl9nUyya"
}
},


Application Log Streaming

By binding an application to an instance of an applicable service, Cloud Foundry will stream logs for the bound application to the service instance.

- Logs for all apps bound to a log-consuming service instance will be streamed to that instance
- Logs for an app bound to multiple log-consuming service instances will be streamed to all instances

To enable this functionality, a service broker must implement the following:

1. In the catalog endpoint, the broker must include `requires: syslog_drain`. This minor security measure validates that a service returning a `syslog_drain_url` in response to the bind operation has also declared that it expects log streaming. If the broker does not include `requires: syslog_drain`, and the bind request returns a value for `syslog_drain_url`, Cloud Foundry will return an error for the bind operation.

2. In response to a bind request, the broker should return a value for `syslog_drain_url`. The syslog URL has a scheme of syslog, syslog-tls, or https and can include a port number. For example:
   
   ```
   "syslog_drain_url": "syslog://logs.example.com:1234"
   ```

How does it work?

1. Service broker returns a value for `syslog_drain_url` in response to bind
2. Loggregator periodically polls CC `/v2/syslog_drain_urls` for updates
3. Upon discovering a new `syslog_drain_url`, Loggregator identifies the associated app
4. Loggregator streams app logs for that app to the locations specified by the service instances’ `syslog_drain_url`

Users can manually configure app logs to be streamed to a location of their choice using User-provided Service Instances. For details, see Using Third-Party Log Management Services.
Route Services

This documentation is intended for service authors who are interested in offering a service to a Cloud Foundry services marketplace. Developers interested in consuming these services can read the Manage Application Requests with Route Services topic.

Introduction

Cloud Foundry application developers may wish to apply transformation or processing to requests before they reach an application. Common examples of use cases are authentication, rate limiting, and caching services. Route Services are a new kind of Marketplace Service that developers can use to apply various transformations to application requests by binding an application’s route to a service instance. Through integrations with service brokers and optionally with the Cloud Foundry routing tier, providers can offer these services to developers with a familiar automated, self-service, and on-demand user experience.

Architecture

Cloud Foundry supports three models for Route Services: fully-brokered services; static, brokered services; and user-provided services. In each case, you configure a route service to process traffic addressed to an app.

Fully-Brokered Service

In this model, the CF router receives all traffic to apps in the deployment before any processing by the route service. Developers can bind a route service to any app, and if an app is bound to a route service, the CF router sends its traffic to the service. After the route service processes requests, it sends them back to the load balancer in front of the CF router. The second time through, the CF router recognizes that the route service has already handled them, and forwards them directly to app instances.

![Route Service Diagram]

The route service can run inside or outside of CF, so long as it fulfills the Service Instance Responsibilities to integrate it with the CF router. A service broker publishes the route service to the CF marketplace, making it available to developers. Developers can then create an instance of the service and bind it to their apps with the following commands:

```
cf create-service BROKER_SERVICE_PLAN SERVICE_INSTANCE
```
```
cf bind-route-service YOUR_APP_DOMAIN SERVICE_INSTANCE [--hostname HOSTNAME]
```

Developers configure the service either through the service provider’s web interface or by passing arbitrary parameters to their `cf create-service` call, through the `-c` flag.

Advantages:

- Developers can use a Service Broker to dynamically configure how the route service processes traffic to specific applications.
- Adding route services requires no manual infrastructure configuration.

Disadvantages:

- Traffic to apps that do not use the service makes fewer network hops; requests for those apps do not pass through the route service.
Traffic to apps that use the route service makes additional network hops, as compared to the static model.

Static, Brokered Service

In this model, an operator installs a static routing service, which might be a piece of hardware, in front of the Load Balancer. The routing service runs outside of Cloud Foundry and receives traffic to all apps running in the CF deployment. The service provider creates a service broker to publish the service to the CF marketplace. As with a fully-brokered service, a developer can use the service by instantiating it with `cf create-service` and binding it to an app with `cf bind-route-service`.

In this model, you configure route services on an app-by-app basis. When you bind a service to an app, the service broker directs the routing service to process that app's traffic rather than pass the requests through unchanged.

Advantages:

- Developers can use a Service Broker to dynamically configure how the route service processes traffic to specific applications.
- Traffic to apps that use the route service takes fewer network hops.

Disadvantages:

- Adding route services requires manual infrastructure configuration.
- Unnecessary network hops for traffic to apps that do not use the route service; requests for all apps hosted by the deployment pass through the route service component.

User-Provided Service

If a route service is not listed in the CF marketplace by a broker, a developer can still bind it to their app as a User-Provided service. The service can run anywhere, either inside or outside of CF, but it must fulfill the integration requirements described in Service Instance Responsibilities. The service also needs to be reachable by an outbound connection from the CF Router.

This model is identical to the fully-brokered service model, except without the broker. Developers configure the service manually, outside of Cloud Foundry. They can then create a user-provided service instance and bind it to their application with the following commands, supplying the URL of their route service:
Advantages:

- Adding route services requires no manual infrastructure configuration.
- Traffic to apps that do not use the service makes fewer network hops; requests for those apps do not pass through the route service.

Disadvantages:

- Developers must manually provision/configure route services out of the context of Cloud Foundry; no service broker automates these operations.
- Traffic to apps that use the route service makes additional network hops, as compared to the static model.

Architecture Comparison

The models above require the broker and service instance responsibilities below, as summarized in the following table:

<table>
<thead>
<tr>
<th>Route Services Architecture</th>
<th>Fulfills CF Service Instance Responsibilities</th>
<th>Fulfills CF Broker Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-Brokered</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static Brokered</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User-Provided</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Enabling Route Services in Pivotal Cloud Foundry

You configure Route Services for your deployment in the Elastic Runtime tile, under Settings > Networking. Depending on your infrastructure, refer to the Elastic Runtime configuration topics for Amazon Web Services, OpenStack, or vSphere and vCloud.

Service Instance Responsibilities

The following applies only when a broker returns `route_service_url` in the bind response.

How It Works

Binding a service instance to a route will associate the `route_service_url` with the route in the Cloud Foundry router. All requests for the route will be proxied to the URL specified by `route_service_url`.

Once a route service completes its function, it is expected to forward the request to the route the original request was sent to. The Cloud Foundry router will include a header that provides the address of the route, as well as two headers that are used by the route itself to validate the request sent by the route service.

Headers

The `X-CF-Forwarded-Url` header contains the URL of the application route. The route service should forward the request to this URL.

The route service should not strip off the `X-CF-Proxy-Signature` and `X-CF-Proxy-Metadata`, as the GoRouter relies on these headers to validate that the request.

SSL Certificates

When Cloud Foundry is deployed in a development environment, certificates hosted by the load balancer will be self-signed (not signed by a trusted certificate authority). When the route service has finished processing an inbound request, and makes a call to the value of `X-CF-Forwarded-Url`, be prepared to accept the self-signed certificate when integrating with a non-production deployment of Cloud Foundry.
Timeouts

Route services must forward the request to the application route within the number of seconds configured by the `router.route_service_timeout` property (default 60 seconds).

In addition, all requests must respond in the number of seconds configured by the `request_timeout_in_seconds` property (default 900 seconds).

Timeouts are configurable for the router using the cf-release BOSH deployment manifest. For more information, see the spec.

Broker Responsibilities

Catalog Endpoint

Brokers must include `requires: ["route_forwarding"]` for a service in the catalog endpoint. If this is not present, Cloud Foundry will not permit users to bind an instance of the service to a route.

Binding Endpoint

When users bind a route to a service instance, Cloud Foundry will send a `bind request` to the broker, including the route address with `bind_resource.route`. A route is an address used by clients to reach apps mapped to the route. The broker may return `route_service_url`, containing a URL where Cloud Foundry should proxy requests for the route. This URL must have a `https` scheme, otherwise the Cloud Controller will reject the binding. `route_service_url` is optional; not returning this field enables a broker to dynamically configure a network component already in the request path for the route, requiring no change in the Cloud Foundry router.

Example Route Services

- **Logging Route Service**: This route service can be pushed as an app to Cloud Foundry. It fulfills the service instance responsibilities above and logs requests received and sent. It can be used to see the route service integration in action by tailing its logs.
- **Rate Limiting Route Service**: This example route service is a simple Cloud Foundry app that provides rate limiting to control the rate of traffic to an application.
- **Spring Boot Example**: Logs requests received and sent; written in Spring Boot

Tutorial

The following instructions show how to use the **Logging Route Service** described in Example Route Services to verify that when a route service is bound to a route, requests for that route are proxied to the route service.

A video of this tutorial is available on Youtube.

Requires CLI version 6.16 or above.

1. Push the **Logging Route Service** as an app.

   ```bash
   $ cf push logger
   ```

2. Create a user-provided service instance, and include the route of the **Logging Route Service** you pushed as `route_service_url`. Be sure to use https for the scheme.

   ```bash
   $ cf create-user-provided-service mylogger -r https://logger.cf.example.com
   ```

3. Push a sample app like **Spring Music**. By default this will create a route `spring-music.cf.example.com`.

   ```bash
   $ cf push spring-music
   ```

4. Bind the user-provided service instance to the route of your sample app. The **bind-route-service** command takes a route and a service instance; the route is specified in the following example by domain `cf.example.com` and hostname `spring-music`.

   ```bash
   $ cf bind-route-service mylogger spring-music.cf.example.com
   ```
5. Tail the logs for your route service.

$ cf logs logger

6. Send a request to the sample app and see in the route service logs that the request is forwarded to it.

$ curl spring-music.cf.example.com
Manage Application Requests with Route Services

This topic describes how to bind a service instance to a route for the purpose of adding preprocessing to application requests.

Route services are a class of marketplace services that perform filtering or content transformation on application requests and responses. This helps to remove the burden on developers who would otherwise have to implement these functions themselves. Popular use cases for route services include rate limiting, authorization, and caching. A route service may reject requests or after some transformation pass the request on to applications.

To use route services, developers must first create a service instance, choosing from compatible Marketplace services. For more information, see the Managing Services and User-provided Service Instances topics. Developers then bind this service instance to a route, and all requests for the route will be preprocessed by the service instance. While some services may support instances being bound to both routes and apps, these operations have different effects. For application requests and responses to be routed through a route service, the service instance must be bound to the route.

A video demonstrating use of a sample route service can be found on YouTube.

Bind a Route to a Service Instance

Binding a route to a service instance can be accomplished using the cf bind-route-service command:

```
$ cf bind-route-service shared-domain.example.com my-route-service --hostname my-app

Binding route my-app.shared-domain.example.com to service instance my-route-service in org my-org / space my-space as developer...
OK
```

Note: When binding a service instance to a route, Cloud Foundry may proxy requests for the route to the service instance, or configure a network component already in the request path.

Some services support additional configuration parameters with the bind request. These parameters are passed in a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file. For a list of supported configuration parameters, see documentation for the particular service offering.

```
$ cf bind-route-service shared-domain.example.com my-route-service --hostname my-app -- "{"rate_limit_threshold_rps":10000}" 

Binding service my-db to app rails-sample in org console / space development as user@example.com...
OK
```

```
$ cf bind-route-service shared-domain.example.com my-route-service --hostname my-app -c /tmp/config.json

Binding route my-app.shared-domain.example.com to service instance my-route-service in org my-org / space my-space as developer...
OK
```

Unbind a Route from a Service Instance

Unbinding a route to a service instance from an application can be accomplished using the cf unbind-route-service command:

```
$ cf unbind-route-service shared-domain.example.com my-route-service --hostname my-app

Unbinding may leave apps mapped to route myapp.superman.cf-app.com vulnerable; e.g. if service instance myspringlogger provides authentication. Do you want to proceed? [y/n] y

Unbinding route my-app.shared-domain.example.com from service instance my-route-service in org my-org / space my-space as developer...
OK
```
Supporting Multiple Cloud Foundry Instances

Page last updated:

It is possible to register a service broker with multiple Cloud Foundry instances. It may be necessary for the broker to know which Cloud Foundry instance is making a given request. For example, when using Dashboard Single Sign-On, the broker is expected to interact with the authorization and token endpoints for a given Cloud Foundry instance.

There are two strategies that can be used to discover which Cloud Foundry instance is making a given request.

Routing & Authentication

The broker can use unique credentials and/or a unique url for each Cloud Foundry instance. When registering the broker, different Cloud Foundry instances can be configured to use different base urls that include a unique id. For example:

- On Cloud Foundry instance 1, the service broker is registered with the url `broker.example.com/123`
- On Cloud Foundry instance 2, the service broker is registered with the url `broker.example.com/456`

X-Api-Info-Location Header

All calls to the broker from Cloud Foundry include an `X-Api-Info-Location` header containing the `/v2/info` url for that instance. The `/v2/info` endpoint will return further information, including the location of that Cloud Foundry instance’s UAA.

Support for this header was introduced in cf-release v212.
Logging and Metrics

Loggregator is the next generation system for aggregating and streaming logs and metrics from all of the user apps and system components in Elastic Runtime.

Table of Contents

- Overview of the Loggregator System
- Loggregator Guide for Cloud Foundry Operators
- Application Logging in Cloud Foundry
- Security Event Logging for Cloud Controller and UAA
- Cloud Foundry Component Metrics
- Deploying a Nozzle to the Loggregator Firehose
- Cloud Foundry Data Sources
- Installing the Loggregator Plugin for cf CLI
- The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment
  - Using SSL with a Self-Signed Certificate in JMX Bridge
  - Deploying JMX Bridge
  - Using JMX Bridge
Overview of the Loggregator System

Loggregator is the next generation system for aggregating and streaming logs and metrics from all of the user apps and system components in an Elastic Runtime deployment.

Using Loggregator

The main use cases are as follows:

- App developers can tail their application logs or dump the recent logs from the CF CLI, or stream these to a third party log archive and analysis service.
- Operators and administrators can access the Loggregator Firehose, the combined stream of logs from all apps, plus metrics data from CF components.
- Operators can deploy ‘nozzles’ to the Firehose. A nozzle is a component that listens to the Firehose for specified events and metrics and streams this data to external services.

Loggregator Components

Source

Sources are logging agents that run on the Cloud Foundry components.

Metron

Metron agents are co-located with sources. They collect logs and forward them to the Doppler servers.

Doppler

Dopplers gather logs from the Metron agents, store them in temporary buffers, and forward them to the Traffic Controller or to third party syslog drains.
Traffic Controller

Handles client requests for logs. Gathers and collates messages from all Doppler servers, and provides external API and message translation (as needed for legacy APIs). Exposes the Firehose.

Firehose

The Firehose is a websocket endpoint which streams all the event data coming from an Elastic Runtime deployment. The data stream includes logs, HTTP events and container metrics from all applications, and metrics from all Elastic Runtime system components. Logs from system components such as Cloud Controller are not included in the firehose and are typically accessed via rsyslog configuration.

Because the data coming from the Firehose may contain sensitive information, such as customer information in the application logs, the Firehose is only accessible by users who have the right permissions.

The Traffic Controller serves the Firehose over websocket at the /firehose endpoint. The events coming out of the Firehose are formatted as protobuf messages conforming to the dropsonde protocol.

The address of the traffic controller can be discovered by hitting the info endpoint on the API and getting the value of the doppler_logging_endpoint.

Example output for a BOSH Lite CF environment:

```
$ cf curl /v2/info | jq .doppler_logging_endpoint
wss://doppler.192.0.2.34.xip.io:443
```

Nozzles

Nozzles are programs which consume data from the Loggregator Firehose. Nozzles can be configured to select, buffer, and transform data, and forward it to other applications and services. For example:

- The [Datadog nozzle](https://docs.pivotal.io) publishes metrics coming from the Firehose to Datadog.
- The [Syslog nozzle](https://docs.pivotal.io) filters out log messages coming from the Firehose and sends it to a syslog server.

See our [Nozzle Tutorial](https://docs.pivotal.io).
This topic contains information for Cloud Foundry deployments operators about how to configure the Loggregator system to avoid data loss with high volumes of logging and metrics data.

Scaling Loggregator

When the volume of log and metric data generated by Elastic Runtime components exceeds the storage buffer capacity of the Dopplers that collect it, data can be lost. Configuring System Logging in Elastic Runtime explains how to scale the Loggregator system to keep up with high stream volume and minimize data loss.

Scaling Nozzles

You can scale nozzles using the subscription ID, specified when the nozzle connects to the Firehose. If you use the same subscription ID on each nozzle instance, the Firehose evenly distributes events across all instances of the nozzle. For example, if you have two nozzles with the same subscription ID, the Firehose sends half of the events to one nozzle and half to the other. Similarly, if you have three nozzles with the same subscription ID, the Firehose sends each instance one-third of the event traffic.

Stateless nozzles should handle scaling gracefully. If a nozzle buffers or caches the data, the nozzle author must test the results of scaling the number of nozzle instances up or down.

Slow Nozzle Alerts

The Traffic Controller alerts nozzles if they consume events too slowly. If a nozzle falls behind, Loggregator alerts the nozzle in two ways:

- **TruncatingBuffer** alerts: If the nozzle consumes messages more slowly than they are produced, the Loggregator system may drop messages. In this case, Loggregator sends the log message, `TB: Output channel too full. Dropped (n) messages`, where “n” is the number of dropped messages. Loggregator also emits a `CounterEvent` with the name `TruncatingBuffer.DroppedMessages`. The nozzle receives both messages from the Firehose, alerting the operator to the performance issue.

- **PolicyViolation** error: The Traffic Controller periodically sends `ping` control messages over the Firehose WebSocket connection. If a client does not respond to a `ping` with a `pong` message within 30 seconds, the Traffic Controller closes the WebSocket connection with the WebSocket error code `ClosePolicyViolation (1008)`. The nozzle should intercept this WebSocket close error, alerting the operator to the performance issue.

An operator can scale the number of nozzles in response to these alerts to minimize the loss of data.

Forwarding Logs to an External Service

You can configure Elastic Runtime to forward log data from components and apps to an external aggregator service instead of routing it to the Loggregator Firehose. Configuring System Logging in Elastic Runtime explains how to enable log forwarding by specifying the aggregator address, port, and protocol.

Using Log Management Services explains how to bind applications to the external service and configure it to receive logs from Elastic Runtime.

Log Message Size Constraints

The Diego cell emits application logs as UDP messages to the Metron. Diego breaks up log messages greater than approximately 60KiB into multiple envelopes to mitigate this constraint.
Application Logging in Cloud Foundry

Loggregator, the Cloud Foundry component responsible for logging, provides a stream of log output from your app and from Cloud Foundry system components that interact with your app during updates and execution.

By default, Loggregator streams logs to your terminal. If you want to persist more than the limited amount of logging information that Loggregator can buffer, you can drain logs to a third-party log management service. See Third-Party Log Management Services.

Cloud Foundry gathers and stores logs in a best-effort manner. If a client is unable to consume log lines quickly enough, the Loggregator buffer may need to overwrite some lines before the client has consumed them. A syslog drain or a CLI tail can usually keep up with the flow of app logs.

Contents of a Log Line

Every log line contains four fields:

1. Timestamp
2. Log type (origin code)
3. Channel: either STDOUT or STDERR
4. Message

Loggregator assigns the timestamp when it receives log data. The log data is opaque to Loggregator, which simply puts it in the message field of the log line. Apps or system components sending log data to Loggregator may include their own timestamps, which then appear in the message field.

Origin codes distinguish the different log types. Origin codes from system components have three letters. The app origin code is APP followed by slash and a digit that indicates the app instance.

Many frameworks write to an app log that is separate from STDOUT and STDERR. This is not supported by Loggregator. Your app must write to STDOUT or STDERR for its logs to be included in the Loggregator stream. Check the buildpack your app uses to determine whether it automatically ensures that your app correctly writes logs to STDOUT and STDERR only. Some buildpacks do this, and some do not.

Log Types and Their Messages

Different types of logs have different message formats, as shown in the examples below. The digit appended to the code indicates the instance index: 0 is the first instance, 1 is the second, and so on.

API

Users make API calls to request changes in app state. Cloud Controller, the Cloud Foundry component responsible for the API, logs the actions that Cloud Controller takes in response.

For example:

```
2016-06-14T14:10:05.36-0700 [API/0] OUT Updated app with guid cdb6e608-8b73-48c1-b7d2-2602d63f033 ("name":"spring-music","instances":1,"memory":512,"environment_json":"PRIVATE DATA HIDDEN")
```

STG

The Diego cell or the Droplet Execution Agent emits STG logs when staging or restaging an app. These actions implement the desired state requested by the user. After the droplet has been uploaded, STG messages end and CELL or DEA messages begin. For STG, the instance index is almost always 0.

For example:

```
2016-06-14T14:10:27.91-0700 [STG/0] OUT Staging...
```
RTR

The Router emits RTR logs when it routes HTTP requests to the app. Router messages include the app name followed by a Router timestamp and then selections from the HTTP request.

For example:

```
2016-06-14T10:51:32.51-0700 [RTR/1]  OUT www.example.com - [14/06/2016:17:51:32.459 +0000] "GET /user/ HTTP/1.1" 200 0 "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/5...
```

LGR

Loggregator emits LGR to indicate problems with the logging process. Examples include “can’t reach syslog drain url” and “dropped log messages due to high rate.”

APP

Every app emits logs according to choices by the developer.

For example:

```
2016-06-14T14:10:15.18-0700 [APP/0]  OUT Exit status 0
```

SSH

The Diego cell emits SSH logs when a user accesses an application container through SSH by using the `cf ssh` command.

For example:

```
2016-06-14T14:16:11.49-0700 [SSH/0]  OUT Successful remote access by 192.0.2.33:7856
```

CELL

The Diego cell emits CELL logs when it starts or stops the app. These actions implement the desired state requested by the user. The Diego cell also emits messages when an app crashes.

For example:

```
2016-06-14T13:44:38.14-0700 [CELL/0]  OUT Successfully created container
```

DEA

The Droplet Execution Agent emits DEA logs beginning when it starts or stops the app. These actions implement the desired state requested by the user. The DEA also emits messages when an app crashes.

For example:

```
2014-02-13T11:44:52.07-0800 [DEA]  OUT Starting app instance (index 1) with guid e51ca6590e1778:46:7-9d86-5b7e801e9c28
```

Writing to the Log from Your App

Your app must write logs to `STDERR` or `STDOUT`. Both are typically buffered, and you should flush the buffer before delivering the message to Loggregator.
Alternatively, you can write log messages to `STDERR` or `STDOUT` synchronously. This approach is mainly used for debugging because it may affect app performance.

### Viewing Logs in the Command Line Interface

You view logs in the CLI using the `cf logs` command. You can tail, dump, or filter log output.

#### Tailing Logs

```
cf logs APP_NAME
```

streams Loggregator output to the terminal.

For example:

```
$ cf logs spring-music
Connected, tailing logs for app spring-music in org example / space development as admin@example.com...

2016-06-14T15:16:12.70-0700 [RTR/4] OUT www.example.com - [14/06/2016:22:16:12.582 +0000] "GET / HTTP/1.1" 200 0 103455 "*" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2225.11 Safari/537.36"

2016-06-14T15:16:20.06-0700 [RTR/4] OUT www.example.com - [14/06/2016:22:16:20.034 +0000] "GET /test HTTP/1.1" 200 0 0 "http://www.example.com" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2225.11 Safari/537.36"

2016-06-14T15:16:22.44-0700 [RTR/4] OUT www.example.com - [14/06/2016:22:17:22.415 +0000] "GET /test HTTP/1.1" 200 0 0 "http://www.example.com/test5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2225.11 Safari/537.36"

...```

Use `Ctrl-C` (^C) to exit the real-time stream.

#### Dumping Logs

```
cf logs APP_NAME --recent
```

displays all the lines in the Loggregator buffer.

#### Filtering Logs

To view some subset of log output, use `cf logs` in conjunction with filtering commands of your choice. In the example below, `grep -v` excludes all Router logs:

```
$ cf logs spring-music --recent | grep -v RTR
2016-06-14T14:10:05.36-0700 [API/0] OUT Updated app with guid cdabc604-0b73-47e1-a7d5-24af2c63f723 ("name":"spring-music","instances":1,"memory":512,"environment_json":"PRIVATE DATA HIDDEN")
2016-06-14T14:10:14.52-0700 [APP/0] OUT - Gracefully stopping, waiting for requests to finish
2016-06-14T14:10:14.52-0700 [CELL/0] OUT Exit status 0
2016-06-14T14:10:14.54-0700 [APP/0] OUT - Goodbye!
2016-06-14T14:10:14.56-0700 [CELL/0] OUT Creating container

...```
Security Event Logging for Cloud Controller and UAA

This topic describes how to enable and interpret security event logging for the Cloud Controller and the User Account and Authentication (UAA) server. Operators can use these logs to retrieve information about a subset of requests to the Cloud Controller and the UAA server for the purposes of security or compliance.

Cloud Controller Logging

The Cloud Controller logs security events to syslog. You must configure a syslog drain to forward your system logs to a log management service.

See the Configuring System Logging in Elastic Runtime topic for more information.

Format for Log Entries

Cloud Controller logs security events in the Common Event Format (CEF). CEF specifies the following format for log entries:

```
CEF:Version|Device Vendor|Device Product|Device Version|Signature ID|Name|Severity|Extension
```

Entries in the Cloud Controller log use the following format:

```
CEF:CEF_VERSION|cloud_foundry|cloud_controller_ng|CC_API_VERSION|SIGNATURE_ID|NAME|SEVERITY|requestMethod=REQUEST_METHOD|cs1Label=userAuthenticationMechanism\tcsl=AUTH_MECHANISM\ncs1Label=vcapRequestId\tcsp=VCAP_REQUEST_ID\nrequest=httpStatusCode\tcs2=RESULT\nrequest=httpStatusCode\tcs3Label=httpStatusCode\nrequest=httpStatusCode\tcs3=RESULT\nrequest=httpStatusCode\tcs4Label=httpStatusCode\nrequest=httpStatusCode\tcs4=RESULT\nrequest=httpStatusCode\tcs5Label=xForwardedFor\nrequest=httpStatusCode\tcs5=X_FORWARDED_FOR_HEADER
```

Refer to the following list for a description of the properties above:

- **CEF_VERSION**: The version of CEF used in the logs.
- **CC_API_VERSION**: The current Cloud Controller API version.
- **SIGNATURE_ID**: The method and path of the request. For example, `/v2/app:GUID`.
- **NAME**: The same as **SIGNATURE_ID**.
- **SEVERITY**: An integer that reflects the importance of the event.
- **REQUEST_METHOD**: The number of milliseconds since the Unix epoch.
- **USERNAME**: The name of the user who originated the request.
- **USER_GUID**: The GUID of the user who originated the request.
- **AUTH_MECHANISM**: The user authentication mechanism. This can be `oauth-access-token`, `basic-auth`, or `no-auth`.
- **VCAP_REQUEST_ID**: The VCAP request ID of the request.
- **REQUEST**: The request path and parameters. For example, `/v2/Info?MY-PARAM=VALUE`.
- **REQUEST_METHOD**: The method of the request. For example, `GET`.
- **RESULT**: The meaning of the HTTP status code of the response. For example, `success`.
- **HTTP_STATUS_CODE**: The HTTP status code of the response. For example, `200`.
- **SOURCE_ADDRESS**: The IP address of the client who originated the request.
- **DESTINATION_ADDRESS**: The IP address of the Cloud Controller VM.
- **X_FORWARDED_FOR_HEADER**: The contents of the X-Forwarded-For header of the request. This is empty if the header is not present.

Example Log Entries

The following list provides several example requests with the corresponding Cloud Controller log entries.
• An anonymous GET request:

CEF:0|cloud_foundry|cloud_controller_ng|2.54.0|GET /v2/info
requestMethod=GET
requestURI=/v2/info
requestAuthMethod=GET
requestIssuer=127.0.0.1
data=192.0.2.1
userAuthenticationMechanism cs1=no-auth
csvcapRequestID cs2=91987189-810-33d-691-5d10b9b99
user cs3=success
httpStatusCode cs4=200
xForwardedFor cs5=198.51.100.1

• A GET request with basic authentication:

CEF:0|cloud_foundry|cloud_controller_ng|2.54.0|GET /v2/syslog_drain_urls
requestMethod=GET
requestURI=/v2/syslog_drain_urls
requestAuthMethod=GET
requestIssuer=127.0.0.1
data=192.0.2.1
userAuthenticationMechanism cs1=basic-auth
vcapRequestID cs2=79187189-890-859-52b-99090c5e
user cs3=success
httpStatusCode cs4=200
xForwardedFor cs5=198.51.100.1

• A GET request with OAuth access token authentication:

CEF:0|cloud_foundry|cloud_controller_ng|2.54.0|GET /v2/apps
requestMethod=GET
requestURI=/v2/apps
requestAuthMethod=GET
requestIssuer=127.0.0.1
data=192.0.2.1
userAuthenticationMechanism cs1=oauth-access-token
vcapRequestID cs2=49f21579-9eb-4bdf-4e9-75f7c75c
user cs3=success
httpStatusCode cs4=200
xForwardedFor cs5=198.51.100.1

• A GET request that results in a 404 error:

CEF:0|cloud_foundry|cloud_controller_ng|2.54.0|GET /v2/apps/7f310103-39aa-4a8b-b92a-9af092a2-fabf
requestMethod=GET
requestURI=/v2/apps/7f310103-39aa-4a8b-b92a-9af092a2-fabf
userAuthenticationMechanism cs1=oauth-access-token
vcapRequestID cs2=49f21579-9eb-4bdf-4e9-75f7c75c
user cs3=success
httpStatusCode cs4=404
xForwardedFor cs5=198.51.100.1

• A POST request that results in a 403 error:

CEF:0|cloud_foundry|cloud_controller_ng|2.54.0|POST /v2/apps
requestMethod=POST
requestURI=/v2/apps
userAuthenticationMechanism cs1=oauth-access-token
vcapRequestID cs2=b0c03111-9999-499b-b29f-d59cfe7cb7ea
user cs3=success
httpStatusCode cs4=403
xForwardedFor cs5=198.51.100.1

UAA Logging

UAA logs security events to a file located at /var/vcap/sys/log/uaa/uaa.log on the UAA VM. Because these logs are automatically rotated, you must configure a syslog drain to forward your system logs to a log management service.

See the Configuring System Logging in Elastic Runtime topic for more information.

Log Events

UAA logs identify the following categories of events:

• Authorization and Password Events
• Scim Administration Events
Example Log Entries

The following sections provide several example requests with the corresponding UAA log entries.

Successful User Authentication

Audit: TokenIssuedEvent ("["openid","scim.read","uaa.user", "cloud_controller.read","password.write","cloud_controller.write","scim.write"]"); principal=a42026d6-5533-1884-eef2-838abcd03e3, origin=[client=cf, user=bob], identityZoneId=[uaa]

This entry records a TokenIssuedEvent.

UAA issued a token associated with the scopes "openid", "scim.read", "uaa.user", "cloud_controller.read", "password.write", "cloud_controller.write", "scim.write" to the user bob.

Failed User Authentication

Audit: UserAuthenticationFailure (bob@example.com);
principal=61965469-c821-46b7-825f-630e12a51d6c, origin=[remoteAddress=198.51.100.1, clientId=cf], identityZoneId=[uaa]

This entry records a UserAuthenticationFailure.

The user bob@example.com originating at 198.51.100.1 failed to authenticate.

Successful User Creation

Audit: UserCreatedEvent ("["user_id=61965469-c821-46b7-825f-630e12a51d6c","username=bob@example.com"]"); principal=91220262-d901-44c0-825f-633133b53d6c, origin=[client=cf, user=admin, details=(198.51.100.1, tokenType=bearertokenValue=<<TOKEN>>, sub=20i03423-dd8c-33e1-938d-e9999e30f500, iss=https://uaa.example.com/oauth/token), identityZoneId=[uaa]

This entry records a UserCreatedEvent.

The admin user originating at 198.51.100.1 created a user named bob@example.com.

Successful User Deletion

Audit: UserDeletedEvent ("["user_id=61965469-c821-46b7-825f-630e12a51d6c","username=bob@example.com"]"); principal=61965469-c821-46b7-825f-630e12a51d6c, origin=[client=admin, details=(remoteAddress=198.51.100.1, tokenType=bearertokenValue=<<TOKEN>>, sub=admin, iss=https://uaa.example.com/oauth/token), identityZoneId=[uaa]

This entry records a UserDeletedEvent.

The admin user originating at 198.51.100.1 deleted a user named bob@example.com.
Cloud Foundry Component Metrics

This topic lists and describes the metrics available for Pivotal Cloud Foundry (PCF) system components. These metrics are streamed from the Loggregator Firehose.

Cloud Controller

Default Origin Name: cc

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failed_job_count.&lt;VM_NAME&gt;-&lt;VM_INDEX&gt;</td>
<td>Number of failed jobs in the &lt;VM_NAME&gt;-&lt;VM_INDEX&gt; queue. This is the number of delayed jobs where the <code>failed_at</code> column is populated with the time of the most recently failed attempt at the job. The failed job count is not specific to the jobs run by the Cloud Controller worker. By default, Cloud Controller deletes failed jobs after 31 days. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>failed_job_count.cc-generic</td>
<td>Number of failed jobs in the cc-generic queue. By default, Cloud Controller deletes failed jobs after 31 days. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>failed_job_count.total</td>
<td>Number of failed jobs in all queues. By default, Cloud Controller deletes failed jobs after 31 days. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>http_status.1XX</td>
<td>Number of HTTP response status codes of type 1xx (informational). This resets when the Cloud Controller process is restarted and is incremented at the end of each request cycle.</td>
</tr>
<tr>
<td>http_status.2XX</td>
<td>Number of HTTP response status codes of type 2xx (success). This resets when the Cloud Controller process is restarted and is incremented at the end of each request cycle. Emitted for each Cloud Controller request.</td>
</tr>
<tr>
<td>http_status.3XX</td>
<td>Number of HTTP response status codes of type 3xx (redirection). This resets when the Cloud Controller process is restarted and is incremented at the end of each request cycle. Emitted for each Cloud Controller request.</td>
</tr>
<tr>
<td>http_status.4XX</td>
<td>Number of HTTP response status codes of type 4xx (client error). This resets when the Cloud Controller process is restarted and is incremented at the end of each request cycle. Emitted for each Cloud Controller request.</td>
</tr>
<tr>
<td>http_status.5XX</td>
<td>Number of HTTP response status codes of type 5xx (server error). This resets when the Cloud Controller process is restarted and is incremented at the end of each request cycle.</td>
</tr>
<tr>
<td>job_queue_length.cc-&lt;VM_NAME&gt;-&lt;VM_INDEX&gt;</td>
<td>Number of background jobs in the &lt;VM_NAME&gt;-&lt;VM_INDEX&gt; queue that have yet to run for the first time. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>job_queue_length.cc-generic</td>
<td>Number of background jobs in the cc-generic queue that have yet to run for the first time. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>job_queue_length.total</td>
<td>Total number of background jobs in the queues that have yet to run for the first time. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.all</td>
<td>Total number of log messages, sum of messages of all severity levels. The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.debug</td>
<td>Number of log messages of severity “debug.” The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.debug1</td>
<td>Not used.</td>
</tr>
<tr>
<td>log_count.debug2</td>
<td>Number of log messages of severity “debug2.” The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.error</td>
<td>Number of log messages of severity “error.” Error is the most severe level. It is used for failures and during error handling. Most errors can be found under this log level, eg. failed unbinding a service, failed to cancel a task, Diego app crashed error, staging completion errors, staging errors, and resource not found. The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.fatal</td>
<td>Number of log messages of severity “fatal.” The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.info</td>
<td>Number of log messages of severity “info.” Examples of info messages are droplet created, copying package, uploading package, access denied due to insufficient scope, job logging, blobstore actions, staging requests, and app running requests.</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018
### Metric Name

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_count.info</td>
<td>Webstore actions, staging requests, and app running requests. The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.actions</td>
<td>Number of log messages of severity &quot;info.&quot; The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.blobs</td>
<td>Number of log messages of severity &quot;blob.&quot; The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.staging</td>
<td>Number of log messages of severity &quot;staging.&quot; The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.app_running</td>
<td>Number of log messages of severity &quot;app running.&quot; The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.warn</td>
<td>Number of log messages of severity &quot;warn.&quot; Warn is also used for failures and during error handling, eg. diagnostics written to file, failed to capture diagnostics, app rollback failed, service broker already deleted, and UAA token problems. The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>log_count.off</td>
<td>Number of log messages of severity &quot;off.&quot; The count resets when the Cloud Controller process is restarted. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>requests.completed</td>
<td>Number of requests that have been processed. Emitted for each Cloud Controller request.</td>
</tr>
<tr>
<td>requests.outstanding</td>
<td>Number of request that are currently being processed. Emitted for each Cloud Controller request.</td>
</tr>
<tr>
<td>tasks_running.count</td>
<td>Number of currently running tasks. Emitted every 30 seconds per VM. This metric is only seen in version 3 of the Cloud Foundry API.</td>
</tr>
<tr>
<td>tasks_running.memory_in_mb</td>
<td>Memory being consumed by all currently running tasks. Emitted every 30 seconds per VM. This metric is only seen in version 3 of the Cloud Foundry API.</td>
</tr>
<tr>
<td>thread_info.event_machine.connection_count</td>
<td>Number of open connections to event machine. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>thread_info.event_machine.resultqueue.num_waiting</td>
<td>Number of scheduled tasks in the result. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>thread_info.event_machine.resultqueue.size</td>
<td>Number of unscheduled tasks in the result. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>thread_info.event_machine.threadqueue.num_waiting</td>
<td>Number of scheduled tasks in the threadqueue. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>thread_info.event_machine.threadqueue.size</td>
<td>Number of unscheduled tasks in the threadqueue. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>thread_info.thread_count</td>
<td>Total number of threads that are either runnable or stopped. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>total_users</td>
<td>Total number of users ever created, including inactive users. Emitted every 10 minutes per VM.</td>
</tr>
<tr>
<td>vcap_sinatra.recent_errors</td>
<td>50 most recent errors. DEPRECATED</td>
</tr>
<tr>
<td>vitals.cpu</td>
<td>Percentage of CPU used by the Cloud Controller process. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.cpu_load_avg</td>
<td>System CPU load averaged over the last 1 minute according to the OS. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.mem_bytes</td>
<td>The RSS bytes (resident set size) or real memory of the Cloud Controller process. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.mem_free_bytes</td>
<td>Total memory available according to the OS. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.mem_used_bytes</td>
<td>Total memory used (active + wired) according to the OS. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.num_cores</td>
<td>The number of CPUs of a host machine. Emitted every 30 seconds per VM.</td>
</tr>
<tr>
<td>vitals.uptime</td>
<td>The uptime of the Cloud Controller process in seconds. Emitted every 30 seconds per VM.</td>
</tr>
</tbody>
</table>

### Diego

Diego metrics have the following origin names:

- auctioneer
- bbs
- cc_uploader
- file_server
- garden_linux
- nsync_bulker
- nsync_listener
- rep
- route_emitter
- ssh_proxy
- stager

© Copyright Pivotal Software Inc, 2013-2018
Default Origin Name: auctioneer

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuctioneerFetchStatesDuration</td>
<td>Time in nanoseconds that the auctioneer took to fetch state from all the cells when running its auction. Emitted every 30 seconds during each auction.</td>
</tr>
<tr>
<td>AuctioneerLRPAuctionsFailed</td>
<td>Cumulative number of LRP instances that the auctioneer failed to place on Diego cells. Emitted every 30 seconds during each auction.</td>
</tr>
<tr>
<td>AuctioneerLRPAuctionsStarted</td>
<td>Cumulative number of LRP instances that the auctioneer successfully placed on Diego cells. Emitted every 30 seconds during each auction.</td>
</tr>
<tr>
<td>AuctioneerTaskAuctionsFailed</td>
<td>Cumulative number of Tasks that the auctioneer failed to place on Diego cells. Emitted every 30 seconds during each auction.</td>
</tr>
<tr>
<td>AuctioneerTaskAuctionsStarted</td>
<td>Cumulative number of Tasks that the auctioneer successfully placed on Diego cells. Emitted every 30 seconds during each auction.</td>
</tr>
<tr>
<td>LockHeld.v1-locks-auctioneer_lock</td>
<td>Whether an auctioneer holds the auctioneer lock: 1 means the lock is held, and 0 means the lock was lost. Emitted every 30 seconds by the active auctioneer.</td>
</tr>
<tr>
<td>LockHeldDuration.v1-locks-auctioneer_lock</td>
<td>Time in nanoseconds that the active auctioneer has held the auctioneer lock. Emitted every 30 seconds by the active auctioneer.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

Default Origin Name: bbs

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBSMasterElected</td>
<td>Emitted once when the BBS is elected as master.</td>
</tr>
<tr>
<td>ConvergenceLRPDuration</td>
<td>Time in nanoseconds that the BBS took to run its LRP convergence pass. Emitted every 30 seconds when LRP convergence runs.</td>
</tr>
<tr>
<td>ConvergenceLRPPreProcessingActualLRPsDeleted</td>
<td>Cumulative number of times the BBS has detected and deleted a malformed ActualLRP in its LRP convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceLRPPreProcessingMalformedRunInfos</td>
<td>Cumulative number of times the BBS has detected a malformed DesiredLRP RunInfo in its LRP convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceLRPPreProcessingMalformedSchedulingInfos</td>
<td>Cumulative number of times the BBS has detected a malformed DesiredLRP SchedulingInfo in its LRP convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceLRPRuns</td>
<td>Cumulative number of times BBS has run its LRP convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceTaskDuration</td>
<td>Time in nanoseconds that the BBS took to run its Task convergence pass. Emitted every 30 seconds when Task convergence runs.</td>
</tr>
<tr>
<td>ConvergenceTaskRuns</td>
<td>Cumulative number of times the BBS has run its Task convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceTasksKicked</td>
<td>Cumulative number of times the BBS has updated a Task during its Task convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ConvergenceTasksPruned</td>
<td>Cumulative number of times the BBS has deleted a malformed Task during its Task convergence pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CrashedActualLRPs</td>
<td>Total number of LRP instances that have crashed. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CrashingDesiredLRPs</td>
<td>Total number of DesiredLRPs that have at least one crashed instance. Emitted every 30 seconds.</td>
</tr>
<tr>
<td></td>
<td>Whether the 'cf-apps' domain is up-to-date, so that CF apps from CC have been</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Domain.cf-apps</td>
<td>Synchronized with DesiredLRPs for Diego to run. 1 means the domain is up-to-date, no data means it is not. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>Domain.cf-tasks</td>
<td>Whether the 'cf-tasks' domain is up-to-date, so that CF tasks from CC have been synchronized with tasks for Diego to run. 1 means the domain is up-to-date, no data means it is not. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDLeader</td>
<td>Index of the leader node in the etcd cluster. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDRaftTerm</td>
<td>Raft term of the etcd cluster. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDReceivedBandwidthRate</td>
<td>Number of bytes per second received by the follower etcd node. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDReceivedRequestRate</td>
<td>Number of requests per second received by the follower etcd node. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDSentBandwidthRate</td>
<td>Number of bytes per second sent by the leader etcd node. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDSentRequestRate</td>
<td>Number of requests per second sent by the leader etcd node. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ETCDWatchers</td>
<td>Number of watches set against the etcd cluster. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LockHeld.v1-locks-bbs_lock</td>
<td>Whether a BBS holds the BBS lock: 1 means the lock is held, and 0 means the lock was lost. Emitted every 30 seconds by the active BBS server.</td>
</tr>
<tr>
<td>LockHeldDuration.v1-locks-bbs_lock</td>
<td>Time in nanoseconds that the active BBS has held the BBS lock. Emitted every 30 seconds by the active BBS server.</td>
</tr>
<tr>
<td>LRPsClaimed</td>
<td>Total number of LRP instances that have been claimed by some cell. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LRPsDesired</td>
<td>Total number of LRP instances desired across all LRPs. Emitted periodically.</td>
</tr>
<tr>
<td>LRPsExtra</td>
<td>Total number of LRP instances that are no longer desired but still have a BBS record. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LRPsMissing</td>
<td>Total number of LRP instances that are desired but have no record in the BBS. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LRPsRunning</td>
<td>Total number of LRP instances that are running on cells. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LRPsUnclaimed</td>
<td>Total number of LRP instances that have not yet been claimed by a cell. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>MetricsReportingDuration</td>
<td>Time in nanoseconds that the BBS took to emit metrics about etcd. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>MigrationDuration</td>
<td>Time in nanoseconds that the BBS took to run migrations against its persistence store. Emitted each time a BBS becomes the active master.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>RequestCount</td>
<td>Cumulative number of requests the BBS has handled through its API. Emitted for each BBS request.</td>
</tr>
<tr>
<td>RequestLatency</td>
<td>Time in nanoseconds that the BBS took to handle requests to its API endpoints. Emitted when the BBS API handles requests.</td>
</tr>
<tr>
<td>TasksCompleted</td>
<td>Total number of Tasks that have completed. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>TasksPending</td>
<td>Total number of Tasks that have not yet been placed on a cell. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>TasksResolving</td>
<td>Total number of Tasks locked for deletion. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>TasksRunning</td>
<td>Total number of Tasks running on cells. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>

Default Origin Name: cc_uploader
<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>MemoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>MemoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>MemoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>MetricsReporting</td>
<td>How long it took to emit the BackingStores, DepotDirs, and LoopDevices metrics. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>DesiredLRPSyncDuration</td>
<td>Time in nanoseconds that the nsync-bulker took to synchronize CF apps and Diego DesiredLRPs. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>LockHeld.v1-locks-nsync_bulker_lock</td>
<td>Whether an nsync-bulker holds the nsync-bulker lock: 1 means the lock is held, and 0 means the lock was lost. Emitted every 30 seconds by the active nsync-bulker.</td>
</tr>
<tr>
<td>LockHeldDuration.v1-locks-nsync_bulker_lock</td>
<td>Time in nanoseconds that the active nsync-bulker has held the convergence lock. Emitted every 30 seconds by the active nsync-bulker.</td>
</tr>
<tr>
<td>LRPCsDesired</td>
<td>Cumulative number of LRPs desired through the nsync API. Emitted on each request desiring a new LRP, every 30 seconds.</td>
</tr>
<tr>
<td>MemoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>MemoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>MemoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>NsyncInvalidDesiredLRPsFound</td>
<td>Number of invalid DesiredLRPs found during nsync-bulkers periodic synchronization. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

Default Origin Name: nsync_listener

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

Default Origin Name: rep

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapacityRemainingContainers</td>
<td>Remaining number of containers this cell can host. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CapacityRemainingDisk</td>
<td>Remaining amount in MiB of disk available for this cell to allocate to containers. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CapacityRemainingMemory</td>
<td>Remaining amount in MiB of memory available for this cell to allocate to containers. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CapacityTotalContainers</td>
<td>Total number of containers this cell can host. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CapacityTotalDisk</td>
<td>Total amount in MiB of disk available for this cell to allocate to containers. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CapacityTotalMemory</td>
<td>Total amount in MiB of memory available for this cell to allocate to containers. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CM</td>
<td>Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ContainerCount</td>
<td>Number of containers hosted on the cell. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>GardenContainerCreationDuration</td>
<td>Time in nanoseconds that the rep Garden backend took to create a container. Emitted after every successful container creation.</td>
</tr>
<tr>
<td>LogMessage</td>
<td>Emitted every 30 seconds.</td>
</tr>
<tr>
<td>logSenderTotalMessagesRead</td>
<td>Count of application log messages sent by Diego Executor. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>RepBulkSyncDuration</td>
<td>Time in nanoseconds that the cell rep took to synchronize the ActualLRPs it has claimed with its actual garden containers. Emitted every 30 seconds by each rep.</td>
</tr>
<tr>
<td>UnhealthyCell</td>
<td>Whether the cell has failed to pass its health check against the garden backend. 0 signifies healthy, and 1 signifies unhealthy. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>

Default Origin Name: route_emitter

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsulDownMode</td>
<td>Whether a route-emitter is operating normally: 0 if the route-emitter is healthy, and 1 when the consul servers are either down or in a bad state.</td>
</tr>
<tr>
<td>LockHeld.v1-locks-route_emitter_lock</td>
<td>Whether a route-emitter holds the route-emitter lock: 1 means the lock is held, and 0 means the lock was lost. Emitted every 30 seconds by the active route-emitter.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>route_emitter_lock</td>
<td>Time in nanoseconds that the active route-emitter has held the route-emitter lock. Emitted every 30 seconds. by the active route-emitter.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>MessagesEmitted</td>
<td>The cumulative number of registration messages that this process has sent. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>RouteEmitterSyncDuration</td>
<td>Time in nanoseconds that the active route-emitter took to perform its synchronization pass. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RoutesRegistered</td>
<td>Cumulative number of route registrations emitted from the route-emitter as it reacts to changes to LRPs. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RoutesSynced</td>
<td>Cumulative number of route registrations emitted from the route-emitter during its periodic route-table synchronization. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RoutesTotal</td>
<td>Number of routes in the route-emitter’s routing table. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RoutesUnregistered</td>
<td>Cumulative number of route unregistrations emitted from the route-emitter as it reacts to changes to LRPs. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>

Default Origin Name: ssh_proxy

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

Default Origin Name: stager

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
<tr>
<td>StagingRequestFailedDuration</td>
<td>Time in nanoseconds that the failed staging task took to run. Emitted each time a staging task fails.</td>
</tr>
<tr>
<td>StagingRequestsFailed</td>
<td>Cumulative number of failed staging tasks handled by each stager. Emitted every time a staging task fails.</td>
</tr>
<tr>
<td>StagingRequestsSucceeded</td>
<td>Cumulative number of successful staging tasks handled by each stager. Emitted every time a staging task completes successfully.</td>
</tr>
<tr>
<td>StagingRequestSucceededDuration</td>
<td>Time in nanoseconds that the successful staging task took to run. Emitted each time a staging task completes successfully.</td>
</tr>
<tr>
<td>StagingStartRequestsReceived</td>
<td>Cumulative number of requests to start a staging task. Emitted by a stager each time it handles a request.</td>
</tr>
</tbody>
</table>
### Default Origin Name: tps_listener

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

### Default Origin Name: tps_watcher

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LockHeld.v1-locks-tps_watcher_lock</td>
<td>Whether a tps-watcher holds the tps-watcher lock: 1 means the lock is held, and 0 means the lock was lost. Emitted every 30 seconds by the active tps-watcher.</td>
</tr>
<tr>
<td>LockHeldDuration.v1-locks-tps_watcher_lock</td>
<td>Time in nanoseconds that the active tps-watcher has held the convergence lock. Emitted every 30 seconds by the active tps-watcher.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration in nanoseconds of the last garbage collector pause.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the process.</td>
</tr>
</tbody>
</table>

### Default Origin Name: DopplerServer

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropsondeListener.currentBufferCount</td>
<td>DEPRECATED</td>
</tr>
<tr>
<td>dropsondeListener.receivedByteCount</td>
<td>DEPRECATED in favor of DopplerServer.udpListener.receivedByteCount.</td>
</tr>
<tr>
<td>dropsondeListener.receivedMessageCount</td>
<td>DEPRECATED in favor of DopplerServer.udpListener.receivedMessageCount.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.containerMetricReceived</td>
<td>Lifetime number of ContainerMetric messages unmarshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.counterEventReceived</td>
<td>Lifetime number of CounterEvent messages unmarshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.errorReceived</td>
<td>Lifetime number of Error messages unmarshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.heartbeatReceived</td>
<td>DEPRECATED</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.httpStartStopReceived</td>
<td>Lifetime number of HttpStartStop messages unmarshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.logMessageTotal</td>
<td>Lifetime number of LogMessage messages unmarshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.unmarshalErrors</td>
<td>Lifetime number of errors when unmarshalling messages.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.valueMetricReceived</td>
<td>Lifetime number of ValueMetric messages unmarshalled.</td>
</tr>
<tr>
<td>httpServer.receivedMessages</td>
<td>Number of messages received by Doppler’s internal MessageRouter. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>LinuxFileDescriptor</td>
<td>Number of file handles for the Doppler’s process.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
</tbody>
</table>
### Metric Name | Description
--- | ---
memoryStats.numBytesAllocatedStack | Instantaneous count of bytes used by the stack allocator.
memoryStats.numFrees | Lifetime number of memory deallocations.
memoryStats.numMallocs | Lifetime number of memory allocations.
messageRouter.numberOfContainerMetricSinks | Instantaneous number of container metric sinks known to the SinkManager. Emitted every 5 seconds.
messageRouter.numberOfDumpSinks | Instantaneous number of dump sinks known to the SinkManager. Emitted every 5 seconds.
messageRouter.numberOfFirehoseSinks | Instantaneous number of firehose sinks known to the SinkManager. Emitted every 5 seconds.
messageRouter.numberOfSyslogSinks | Instantaneous number of syslog sinks known to the SinkManager.
messageRouter.numberOfWebSocketSinks | Instantaneous number of WebSocket sinks known to the SinkManager. Emitted every 5 seconds.
messageRouter.totalDroppedMessages | Lifetime number of messages dropped inside Doppler for various reasons (downstream consumer can't keep up internal object wasn't ready for message, etc.).
sentMessagesFirehose.<SUBSCRIPTION_ID> | Number of sent messages through the firehose per subscription id. Emitted every 5 seconds.
udpListener.receivedByteCount | Lifetime number of bytes received by Doppler's UDP Listener.
udpListener.receivedMessageCount | Lifetime number of messages received by Doppler's UDP Listener.
udpListener.receivedErrorCount | Lifetime number of errors encountered by Doppler's UDP Listener while reading from the connection.
tcpListener.receivedByteCount | Lifetime number of bytes received by Doppler's TCP Listener. Emitted every 5 seconds.
tcpListener.receivedMessageCount | Lifetime number of messages received by Doppler's TCP Listener. Emitted every 5 seconds.
tcpListener.receivedErrorCount | Lifetime number of errors encountered by Doppler's TCP Listener while handshaking, decoding or reading from the connection.
tlsListener.receivedByteCount | Lifetime number of bytes received by Doppler's TLS Listener. Emitted every 5 seconds.
tlsListener.receivedMessageCount | Lifetime number of messages received by Doppler's TLS Listener. Emitted every 5 seconds.
tlsListener.receivedErrorCount | Lifetime number of errors encountered by Doppler's TLS Listener while handshaking, decoding or reading from the connection.
TruncatingBuffer.DroppedMessages | Number of messages intentionally dropped by Doppler from the sink for the specific sink. This counter event will correspond with log messages “Log message output is too high.” Emitted every 5 seconds.
TruncatingBuffer.totalDroppedMessages | Lifetime total number of messages intentionally dropped by Doppler from all of its sinks due to back pressure. Emitted every 5 seconds.
listeners.totalReceivedMessageCount | Total number of messages received across all of Doppler's listeners (UDP, TCP, TLS).
numCpus | Number of CPUs on the machine.
umGoRoutines | Instantaneous number of active goroutines in the Doppler process.
signatureVerifier.invalidSignatureErrors | Lifetime number of messages received with an invalid signature.
signatureVerifier.missingSignatureErrors | Lifetime number of messages received that are too small to contain a signature.
signatureVerifier.validSignatures | Lifetime number of messages received with valid signatures.
Uptime | Uptime for the Doppler's process.

### Etcld

Visit [etcd stats API](#)  

**Default Origin Name:** etcd

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompareAndDeleteFail</td>
<td>CompareAndDeleteFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CompareAndDeleteSuccess</td>
<td>CompareAndDeleteSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CompareAndSwapFail</td>
<td>CompareAndSwapFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CompareAndSwapSuccess</td>
<td>CompareAndSwapSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>CreateFail</td>
<td>CreateFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateSuccess</td>
<td>DeleteSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>DeleteFail</td>
<td>DeleteFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>DeleteSuccess</td>
<td>DeleteSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>EtcdIndex</td>
<td>X-Etcd-Index value from the /stats/store endpoint. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ExpireCount</td>
<td>ExpireCount operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>Followers</td>
<td>Number of etcd followers. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>GetsFail</td>
<td>GetsFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>GetsSuccess</td>
<td>GetsSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>IsLeader</td>
<td>1 if the current server is the leader, 0 if it is a follower. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>Latency</td>
<td>Current latency in milliseconds from leader to a specific follower. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RaftIndex</td>
<td>X-Raft-Index value from the /stats/store endpoint. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>RaftTerm</td>
<td>X-Raft-Term value from the /stats/store endpoint. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ReceivedAppendRequests</td>
<td>Number of append requests this node has processed. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ReceivingBandwidthRate</td>
<td>Number of bytes per second this node is receiving (follower only). Emitted every 30 seconds.</td>
</tr>
<tr>
<td>ReceivingRequestRate</td>
<td>Number of requests per second this node is receiving (follower only). Emitted every 30 seconds.</td>
</tr>
<tr>
<td>SendingBandwidthRate</td>
<td>Number of bytes per second this node is sending (leader only). This value is undefined on single member clusters. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>SendingRequestRate</td>
<td>Number of requests per second this node is sending (leader only). This value is undefined on single member clusters. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>SentAppendRequests</td>
<td>Number of requests that this node has sent. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>SetsFail</td>
<td>SetsFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>SetsSuccess</td>
<td>SetsSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>UpdateFail</td>
<td>UpdateFail operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>UpdateSuccess</td>
<td>UpdateSuccess operation count. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>Watchers</td>
<td>Watchers operation count. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>

---

**Metron Agent**

**Default Origin Name:** MetronAgent

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageAggregator.counterEventReceived</td>
<td>Lifetime number of CounterEvents aggregated in Metron.</td>
</tr>
<tr>
<td>MessageBuffer.droppedMessageCount</td>
<td>Lifetime number of intentionally dropped messages from Metron's batch writer buffer. Batch writing is performed over TCP/TLS only.</td>
</tr>
<tr>
<td>DopplerForwarder.sentMessages</td>
<td>Lifetime number of messages sent to Doppler regardless of protocol. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>dropsondeAgentListener.currentBufferCount</td>
<td>Instantaneous number of Dropsonde messages read by UDP socket but not yet unmarshalled.</td>
</tr>
<tr>
<td>dropsondeAgentListener.receivedByteCount</td>
<td>Lifetime number of bytes of Dropsonde messages read by UDP socket.</td>
</tr>
<tr>
<td>dropsondeAgentListener.receivedMessageCount</td>
<td>Lifetime number of Dropsonde messages read by UDP socket.</td>
</tr>
<tr>
<td>dropsondeMarshaller.containerMetricMarshalled</td>
<td>Lifetime number of ContainerMetric messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.counterEventMarshalled</td>
<td>Lifetime number of CounterEvent messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.errorMarshaled</td>
<td>Lifetime number of Error messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.heartbeatMarshalled</td>
<td>Lifetime number of Heartbeat messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.httpStartStopMarshalled</td>
<td>Lifetime number of HttpStartStop messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.logMessageMarshalled</td>
<td>Lifetime number of LogMessage messages marshalled.</td>
</tr>
<tr>
<td>dropsondeMarshaller.marshalErrors</td>
<td>Lifetime number of errors when marshalling messages.</td>
</tr>
<tr>
<td>dropsondeMarshaller.valueMetricMarshaled</td>
<td>Lifetime number of ValueMetric messages marshalled.</td>
</tr>
<tr>
<td>dropsondeUnmarshaller.containerMetricReceived</td>
<td>Lifetime number of ContainerMetric messages unmarshalled.</td>
</tr>
</tbody>
</table>
Routing

Routing Release metrics have following origin names:

- gorouter
- routing_api
- tcp_emitter
- tcp_emitter

Default Origin Name: gorouter

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numCPUs</td>
<td>Number of CPUs on the machine. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process.</td>
</tr>
<tr>
<td>tcp.sendErrorCount</td>
<td>Lifetime number of errors if writing to Doppler over TCP fails.</td>
</tr>
<tr>
<td>tcp.sentByteCount</td>
<td>Lifetime number of sent bytes to Doppler over TCP.</td>
</tr>
<tr>
<td>tcp.sentMessageCount</td>
<td>Lifetime number of sent messages to Doppler over TCP.</td>
</tr>
<tr>
<td>tls.sendErrorCount</td>
<td>Lifetime number of errors if writing to Doppler over TLS fails.</td>
</tr>
<tr>
<td>tls.sentBytesToBytes</td>
<td>Lifetime number of sent bytes to Doppler over TLS. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>tls.sentMessageCount</td>
<td>Lifetime number of sent messages to Doppler over TLS. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>udp.sendErrorCount</td>
<td>Lifetime number of errors if writing to Doppler over UDP fails.</td>
</tr>
<tr>
<td>udp.sentBytesToBytes</td>
<td>Lifetime number of sent bytes to Doppler over UDP.</td>
</tr>
<tr>
<td>udp.sentMessageCount</td>
<td>Lifetime number of sent messages to Doppler over UDP.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>latency</td>
<td>Time in milliseconds that the Gorouter took to handle requests to its application endpoints. Emitted per router request.</td>
</tr>
<tr>
<td>latency.(component)</td>
<td>Time in milliseconds that the Gorouter took to handle requests from each component to its endpoints. Emitted per router request.</td>
</tr>
<tr>
<td>registry_message.(component)</td>
<td>Lifetime number of route register messages received for each component. Emitted per route-register message.</td>
</tr>
<tr>
<td>rejected_requests</td>
<td>Lifetime number of bad requests received on gorouter. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>requests.(component)</td>
<td>Lifetime number of requests received for each component. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses</td>
<td>Lifetime number of HTTP responses. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses.2xx</td>
<td>Lifetime number of 2xx HTTP responses. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses.3xx</td>
<td>Lifetime number of 3xx HTTP response. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses.4xx</td>
<td>Lifetime number of 4xx HTTP response. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses.5xx</td>
<td>Lifetime number of 5xx HTTP response. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>responses.xxx</td>
<td>Lifetime number of other(non-(2xx-5xx)) HTTP response. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>routed_app_requests</td>
<td>The collector sums up requests for all dea-{index} components for its output metrics. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>total_requests</td>
<td>Lifetime number of requests received. Emitted every 5 seconds.</td>
</tr>
<tr>
<td>ms_since_last_registry_update</td>
<td>Time in millisecond since the last route register has been been received. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>total_routes</td>
<td>Lifetime number of routes registered. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>uptime</td>
<td>Uptime for router. Emitted every second.</td>
</tr>
</tbody>
</table>

Default Origin Name: routing_api

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>key_refresh_events</td>
<td>Total number of events when fresh token was fetched from UAA. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>total_http_routes</td>
<td>Number of HTTP routes in the routing table. Emitted every 30 seconds, or when there is a new HTTP route added. Interval for emitting this metric can be configured with manifest property metrics_reporting_interval.</td>
</tr>
<tr>
<td>total_http_subscriptions</td>
<td>Number of HTTP routes subscriptions. Emitted every 30 seconds. Interval for emitting this metric can be configured with manifest property metrics_reporting_interval.</td>
</tr>
<tr>
<td>total_tcp_routes</td>
<td>Number of TCP routes in the routing table. Emitted every 30 seconds, or when there is a new TCP route added. Interval for emitting this metric can be configured with manifest property metrics_reporting_interval.</td>
</tr>
<tr>
<td>total_tcp_subscriptions</td>
<td>Number of TCP routes subscriptions. Emitted every 30 seconds. Interval for emitting this metric can be configured with manifest property metrics_reporting_interval.</td>
</tr>
<tr>
<td>total_token_errors</td>
<td>Total number of UAA token errors. Emitted every 30 seconds. Interval for emitting this metric can be configured with manifest property metrics_reporting_interval.</td>
</tr>
</tbody>
</table>

Default Origin Name: tcp_emitter

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process. Emitted every 10 seconds.</td>
</tr>
</tbody>
</table>

Default Origin Name: router_configurer (bosh job tcp_router)

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>{session_id}.ConnectionTime</td>
<td>Average connection time to backend in current session. Emitted every 60 seconds per session ID. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
<tr>
<td>{session_id}.CurrentSessions</td>
<td>Total number of current sessions. Emitted every 60 seconds per session ID. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
<tr>
<td>AverageConnectTimeMs</td>
<td>Average backend response time (in ms). Emitted every 60 seconds. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
<tr>
<td>AverageQueueTimeMs</td>
<td>Average time spent in queue (in ms). Emitted every 60 seconds. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
<tr>
<td>TotalBackendConnectionErrors</td>
<td>Total number of backend connection errors. Emitted every 60 seconds. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
<tr>
<td>TotalCurrentUserQueuedRequests</td>
<td>Total number of requests unassigned in queue. Emitted every 60 seconds. Interval value for this metric can be configured with manifest property router_configurer.tcp_stats_collection_interval.</td>
</tr>
</tbody>
</table>

Syslog Drain Binder

Default Origin Name: syslog_drain_binder

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process. Emitted every 10 seconds.</td>
</tr>
<tr>
<td>pollCount</td>
<td>Number of times the syslog drain binder has polled the cloud controller for syslog drain bindings. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>totalDrains</td>
<td>Number of syslog drains returned by cloud controller. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>
### Traffic Controller

Default Origin Name: LoggregatorTrafficController

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dopplerProxy.containermetricsLatency</td>
<td>Duration for serving container metrics via the containermetrics endpoint (milliseconds). Emitted every 30 seconds.</td>
</tr>
<tr>
<td>dopplerProxy.recentlogsLatency</td>
<td>Duration for serving recent logs via the recentLogs endpoint (milliseconds). Emitted every 30 seconds.</td>
</tr>
<tr>
<td>memoryStats.lastGCPauseTimeNS</td>
<td>Duration of the last Garbage Collector pause in nanoseconds.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocated</td>
<td>Instantaneous count of bytes allocated and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedHeap</td>
<td>Instantaneous count of bytes allocated on the main heap and still in use.</td>
</tr>
<tr>
<td>memoryStats.numBytesAllocatedStack</td>
<td>Instantaneous count of bytes used by the stack allocator.</td>
</tr>
<tr>
<td>memoryStats.numFrees</td>
<td>Lifetime number of memory deallocations.</td>
</tr>
<tr>
<td>memoryStats.numMallocs</td>
<td>Lifetime number of memory allocations.</td>
</tr>
<tr>
<td>numCPUS</td>
<td>Number of CPUs on the machine.</td>
</tr>
<tr>
<td>numGoRoutines</td>
<td>Instantaneous number of active goroutines in the Doppler process.</td>
</tr>
<tr>
<td>Uptime</td>
<td>Uptime for the Traffic Controller's process.</td>
</tr>
<tr>
<td>LinuxFileDescriptor</td>
<td>Number of file handles for the TrafficController's process.</td>
</tr>
</tbody>
</table>

### User Account and Authentication (UAA)

Default Origin Name: uaa

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_service.client_authentication_count</td>
<td>Number of successful client authentication attempts since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.client_authentication_failure_count</td>
<td>Number of failed client authentication attempts since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.principal_authentication_failure_count</td>
<td>Number of failed non-user authentication attempts since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.principal_not_found_count</td>
<td>Number of times non-user was not found since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.user_authentication_count</td>
<td>Number of successful authentications by the user since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.user_authentication_failure_count</td>
<td>Number of failed user authentication attempts since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.user_not_found_count</td>
<td>Number of times the user was not found since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.user_password_changes</td>
<td>Number of successful password changes by the user since the last startup. Emitted every 30 seconds.</td>
</tr>
<tr>
<td>audit_service.user_password_failures</td>
<td>Number of failed password changes by the user since the last startup. Emitted every 30 seconds.</td>
</tr>
</tbody>
</table>
Deploying a Nozzle to the Loggregator Firehose

This topic describes deploying a "nozzle" application to the Cloud Foundry (CF) Loggregator Firehose. The Cloud Foundry Loggregator created an example nozzle application for use with this tutorial.

The procedure described below deploys this example nozzle to the Firehose of a Cloud Foundry installation deployed locally with BOSH Lite.

Prerequisites

- BOSH CLI installed locally.
- Spiff installed locally and added to your shell's load path. See [Spiff on GitHub](#).
- BOSH Lite deployed locally using VirtualBox. See [BOSH Lite on GitHub](#).
- A working Cloud Foundry deployment, including Loggregator, deployed with your local BOSH Lite. This serves as our source of data. See [Deploying Cloud Foundry using BOSH Lite](#), or use the `provision_cf` script included in the BOSH Lite release.

**Note**: Deploying Cloud Foundry can take up to several hours, depending on your internet bandwidth, even when using the automated `provision_cf` script.

Step 1: Download Cloud Foundry BOSH Manifest

1. Run `bosh deployments` to identify the name of the current BOSH deployment:

```
$ bosh deployments
+-----------------+--------------+-------------------------------------------------+
| Name             | Release(s)   | Stemcell(s)                                     |
|------------------|--------------+-------------------------------------------------+
| cf-example       | cf-mysql/10  | bosh-vsphere-esxi-ubuntu-trusty-go_agent/2690.3 |
|                  | cf/183.2     |                                                 |
+-----------------+--------------+-------------------------------------------------+
```

2. Run `bosh download manifest DEPLOYMENT-NAME LOCAL-SAVE-NAME` to download and save the current BOSH deployment manifest. You need this manifest to locate information about your databases. Replace `DEPLOYMENT-NAME` with the name of the current BOSH deployment. For this procedure, use `cf.yml` as the `LOCAL-SAVE-NAME`.

```
$ bosh download manifest cf-example cf.yml
Deployment manifest saved to `cf.yml`
```

Step 2: Add UAA client

You must authorize the example nozzle as a UAA client for your CF deployment. To do this, add an entry for the example nozzle as `client` for `uaa` under the `properties` key in your CF deployment manifest. You must enter the example nozzle object in the correct location in the manifest, and with the correct indentation, as described below.

Deployment manifests are YAML files. Visit [YAML](#) to learn about YAML syntax.

1. Open the deployment manifest in a text editor.
2. Locate the left-aligned `properties` key.
3. Under the `properties` key, locate `uaa` at the next level of indentation.
4. Under the `uaa` key, locate the `clients` key at the next level of indentation.
5. Enter properties for the `example-nozzle` at the next level of indentation, exactly as shown below. The `...` in the text below indicate other properties that may populate the manifest at each level in the hierarchy.
properties:
  
  uaa:
  
  clients:
    
    example-nozzle:
      access-token-validity: 1209600
      authorized-grant-types: authorization_code,client_credentials,refresh_token
      override: true
      secret: example-nozzle
      scope: openid,oauth.approvals,doppler.firehose
      authorities: oauth.login,doppler.firehose

6. Save the deployment manifest file.

Step 3: Redeploy Cloud Foundry

1. Use the `bosh deployment` command to set the edited manifest file for your deployment.

   $ bosh deployment cf.yml
   Deployment set to '/Users/example_user/workspace/bosh-lite/cf.yml'

2. Deploy your Cloud Foundry with BOSH.

   $ bosh deploy
   Acting as user 'admin' on deployment 'cf-warden' on 'Bosh Lite Director' 
   Getting deployment properties from director...
   Detecting deployment changes
   ----------------------------
   Releases
   No changes
   Compilation
   No changes
   Update
   No changes
   Resource pools
   No changes
   Disk pools
   No changes
   Networks
   No changes
   Jobs
   No changes
   Properties
   uaa
   clients
   example-nozzle
     + access-token-validity: 1209600
     + authorized-grant-types: authorization_code,client_credentials,refresh_token
     + override: true
     + secret: example-nozzle
     + scope: openid,oauth.approvals,doppler.firehose
     + authorities: oauth.login,doppler.firehose
   Meta
   No changes

   Please review all changes carefully

   Deploying
   ---------
   Are you sure you want to deploy? (type 'yes' to continue):yes

Step 4: Clone Example Release
The Cloud Foundry Loggregator team created an example nozzle application for use with this tutorial.

1. Run `git clone` to clone the main release repository from GitHub.

   ```bash
   git clone git@github.com:cloudfoundry-incubator/example-nozzle-release.git
   Cloning into 'example-nozzle-release'...
   ```

2. Run `git submodule update --init --recursive` to update all of the included submodules.

   ```bash
   git submodule update --init --recursive
   Submodule 'src/github.com/cloudfoundry-incubator/example-nozzle' (git@github.com:cloudfoundry-incubator/example-nozzle.git) registered for path 'src/github.com/cloudfoundry-incubator/example-nozzle'
   Submodule 'src/github.com/cloudfoundry-incubator/uaago' (git@github.com:cloudfoundry-incubator/uaago.git) registered for path 'src/github.com/cloudfoundry-incubator/uaago'
   Cloning into 'src/github.com/cloudfoundry-incubator/example-nozzle'...
   ```

3. Navigate to the `example-release` directory.

   ```bash
cd example-nozzle-release
   ```

Step 5: Prepare Nozzle Manifest

Complete the following steps to prepare the nozzle deployment manifest:

1. In the `example-nozzle-release` directory, navigate to the `templates` directory.

   ```bash
cd templates
   ```

   Within this directory, examine the two YAML files. `bosh-lite-stub.yml` contains the values used to populate the missing information in `template.yml`. By combining these two files, we create a deployment manifest for our nozzle.

2. Create a `tmp` directory for the compiled manifest.

3. Use `Spiff` to compile a deployment manifest from the template and stub, and save this manifest.

   ```bash
   spiff merge templates/template.yml templates/bosh-lite-stub.yml > tmp/manifest_bosh_lite.yml
   ```

4. Run `bosh status --uuid` to obtain your BOSH director UUID.

   ```bash
   bosh status --uuid
   ```

5. In the compiled nozzle deployment manifest, locate the `director_uuid` property. Replace `PLACEHOLDER-DIRECTOR-UUID` with your BOSH director UUID.

   ```yaml
   compilation:
   cloud_properties:
     name: default
   network: example-nozzle-net
   reuse_compilation_vms: true
   workers: 1
   director_uuid: PLACEHOLDER-DIRECTOR-UUID  # replace this
   ```

   *Note*: If you do not want to see the complete deployment procedure, run the following command to automatically prepare the manifest:

   ```bash
   scripts/make_manifest_spiff_bosh_lite
   ```

Step 6: Set Nozzle Deployment Manifest

Use the `bosh deployment` command to set the deployment manifest for the nozzle.
Step 7: Create Nozzle BOSH Release

Use the `bosh create release --name RELEASE-NAME` command to create a BOSH release. Replace RELEASE-NAME with `example-nozzle` to match the UAA client that you created in the CF deployment manifest.

```
bosh create release --name example-nozzle
Syzing blobs...
```
Step 10: View Nozzle Output

The example nozzle outputs all of the data originating coming from the Firehose to its log files. To view this data, SSH into the example-nozzle VM and examine the logs.

1. Run `bosh ssh` to access the nozzle VM at the IP configured in the nozzle’s manifest template stub. 

```bash
$ bosh ssh example-nozzle
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.19.0-25-generic x86_64)
Documentation: https://help.ubuntu.com
Last login: Wed Sep 23 21:20:50 2015 from 192.0.2.1
```

2. Use the `cat` command to output the `stdout` log file.

```bash
$ cat /var/log/example-nozzle/example-nozzle.stdout.log
```

---

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.19.0-25-generic x86_64)

Documentation: https://help.ubuntu.com
Last login: Wed Sep 23 21:20:50 2015 from 192.0.2.1
```

---

```
streaming Firehose (will only succeed if you have admin credentials)
```

---

```
$ cat /var/log/example-nozzle/example-nozzle.stdout.log
  - Streaming Firehose (will only succeed if you have admin credentials)

origin:"DopplerServer" eventType:ValueMetric timestamp:1443046217739507477 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"203.0.113.15" peerType:Server method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"203.0.113.15:49577" userAgent:"Go 1.1 package http" statusCode:200 contentLength:108

peerType:Client method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"" userAgent:"" statusCode:200 contentLength:108

peerType:Client method:POST uri:"uaa.service.cf.internal:8080/oauth/token" remoteAddress:"" userAgent:"" statusCode:200 contentLength:-1

origin:"MetronAgent" eventType:CounterEvent timestamp:144304621890193878 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"203.0.113.146" peerType:Server method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"203.0.113.15:49577" userAgent:"Go 1.1 package http" statusCode:200 contentLength:108

peerType:Client method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"" userAgent:"" statusCode:200 contentLength:108

peerType:Client method:POST uri:"uaa.service.cf.internal:8080/oauth/token" remoteAddress:"" userAgent:"" statusCode:200 contentLength:-1

origin:"MetronAgent" eventType:CounterEvent timestamp:144304621890252169 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"203.0.113.146" peerType:Server method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"203.0.113.15:49577" userAgent:"Go 1.1 package http" statusCode:200 contentLength:108

peerType:Client method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"" userAgent:"" statusCode:200 contentLength:108

peerType:Client method:POST uri:"uaa.service.cf.internal:8080/oauth/token" remoteAddress:"" userAgent:"" statusCode:200 contentLength:-1

origin:"MetronAgent" eventType:CounterEvent timestamp:144304621890294255 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"203.0.113.146" peerType:Server method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"203.0.113.15:49577" userAgent:"Go 1.1 package http" statusCode:200 contentLength:108

peerType:Client method:GET uri:"routing-api.service.cf.internal:3000/v1/routes" remoteAddress:"" userAgent:"" statusCode:200 contentLength:108

peerType:Client method:POST uri:"uaa.service.cf.internal:8080/oauth/token" remoteAddress:"" userAgent:"" statusCode:200 contentLength:-1
```

---

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.19.0-25-generic x86_64)
```

---

```
$bosh ssh example-nozzle
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.19.0-25-generic x86_64)
```

---

```
last login: Wed Sep 23 21:20:50 2015 from 192.0.2.1
```
Cloud Foundry Data Sources

Page last updated:

Currently, Cloud Foundry logs and metrics come from several sources:

- Loggregator is the next generation logging and metrics system for Cloud Foundry. It aggregates metrics from applications and CF system components and streams these out to the CF cli or to third party log management services.

- The Collector is Cloud Foundry's original metric aggregation system. It gathers metrics from all Cloud Foundry system components by querying their `/healthz` and `/varz` endpoints, and then publishes this data to external systems such as Datadog, AWS CloudWatch and OpenTSDB.

  **Note:** The Collector will eventually be deprecated in favor of the Loggregator system.

- The BOSH Health Monitor continually listens for one 'heartbeat' per minute from each deployed VM. These heartbeats contain status updates and lifecycle events. Health Monitor can be extended by plugins to forward heartbeat data to other CF components or third party services.

- Logs from CF components can also be forwarded directly to your own server, bypassing loggregator. See Loggregator for Operators for more information.

Currently, Cloud Foundry supports all of these metrics pipelines. Data from each of these sources can be streamed to a variety of services including the following:

- JMX Bridge
- Datadog
- AWS CloudWatch

See Using Log Management Services for more information about draining logs from Elastic Runtime.
Installing the Loggregator Firehose Plugin for cf CLI

Page last updated:

The Loggregator Firehose plugin for the Cloud Foundry Command Line Interface (cf CLI) allows Cloud Foundry (CF) administrators access to the output of the Loggregator Firehose, which includes logs and metrics from all CF components.

See Using cf CLI Plugins for more information about using plugins with the cf CLI.

Prerequisites

- Administrator access to the Cloud Foundry deployment that you want to monitor
- Cloud Foundry Command Line Interface (cf CLI) 6.12.2 or later

Refer to the Installing the cf CLI topic for information about downloading, installing, and uninstalling the cf CLI.

Install the Plugin

1. Run `cf add-plugin-repo REPO_NAME URL` to add the Cloud Foundry Community plugin repository to your cf CLI plugins.

```bash
$ cf add-plugin-repo CF-Community https://plugins.cloudfoundry.org
```

2. Run `cf install-plugin PLUGIN-NAME -r PLUGIN-REPO` to install the Firehose plugin from the CF Community plugin repository.

```bash
$ cf install-plugin "Firehose Plugin" -r CF-Community
```

View the Firehose

Run `cf nozzle --debug` to view the streaming output of the Firehose, which includes logging events and metrics from CF system components. For more information about logging and metrics in CF, see Overview of the Loggregator System.

```bash
$ cf nozzle --debug
```

Note: You must be logged in as a Cloud Foundry administrator to access the Firehose.

Uninstall the Plugin

Run `cf plugins` to see a list of installed plugins.

```bash
$ cf plugins
Listing Installed Plugins...
OK
Plugin Name   Version  Command Name  Command Help
FirehosePlugin 0.6.0  nozzle       Command to print out messages from the firehose
```

Run `cf uninstall-plugin PLUGIN-NAME` to uninstall the plugin.

```bash
$ cf uninstall-plugin FirehosePlugin
```
Pivotal Cloud Foundry Release Notes and Known Issues

Release Notes

- Pivotal Elastic Runtime Release Notes
- Pivotal Operations Manager Release Notes

Known Issues

- Pivotal Elastic Runtime Known Issues
- Pivotal Operations Manager Known Issues
Pivotal Elastic Runtime v1.7.0.0 Release Notes

About Updating to Elastic Runtime v1.7.30 or Later

If you are currently on Elastic Runtime v1.7.27 or earlier, you should update your Elastic Runtime. Later versions include a major stemcell upgrade to 3233.x. This stemcell uses the Linux kernel v4.4 instead of the v3.19. Ubuntu is no longer providing CVE patches for Linux kernel v3.19.

Before updating Elastic Runtime, ensure that you have updated to Ops Manager v1.7.15 or later.

Resolve Diego BBS Issue when Upgrading from v1.7 to v1.8

The Problem

Note: Fresh installations (not upgrades) of PCF ER 1.8.10 or higher should not be affected by this problem. However, you can still follow these procedures on subsequent deployments to improve the likelihood of a successful upgrade. Future versions of the ERT will be moving off of etcd with the hopes of greatly increasing platform stability during an upgrade.

When upgrading from PCF ERT v1.7.x to v1.8.0-1.8.8, deployments that included multiple instances of the Diego BBS VM would fail to upgrade.

The Impact

Inconsistencies between Diego datasets can result in app downtime or apps not successfully returning during a deploy.

Already Downloaded PCF ERT v1.8.0-1.8.8

If you haven’t attempted an upgrade, download PCF ERT v1.8.10 or higher and proceed with the instructions below.

How to Upgrade Safely to PCF ERT v1.8.10+ and/or Repair a Broken Cluster

This is applicable to all upgrade paths between v1.7.x to v1.8.10+. Upgrades from v1.8.x to v1.8.10+ should not require these steps if your Diego BBS cluster is already healthy. After you’ve successfully upgraded past v1.8.10, these steps should no longer be necessary. You can still follow these procedures on subsequent deployments to improve the likelihood of a successful upgrade. Future versions of the ERT will be moving off of etcd with the hopes of greatly increasing platform stability during an upgrade.

- For single node Diego BBS instance configuration: No action required.
- For multi node Diego BBS instance configuration: Pivotal recommends scaling down to one Diego BBS (see the section below for instructions) before attempting an upgrade. You can do this before or during the upgrade deployment. After the successful upgrade deployment, scale back up to an HA configuration.

Scale Diego BBS to One Instance

In some cases you will need to scale down your Diego BBS cluster to one instance (please see above for a specific details around your upgrade path). To scale down to one instance:

1. Navigate to the Settings tab in the Pivotal Elastic Runtime Tile.
2. Select Resource Config.
3. Find the Diego BBS job.
4. Set Instances to Automatic: 1.
5. Save the Resource Config.
6. Navigate to the Installation Dashboard and click Apply Changes.

7. Upgrade the PCF Elastic Runtime Tile to a version greater than or equal to 1.8.10.


Releases

1.7.70


<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3363.26</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2</td>
<td>1.126.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.69


<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3363.26</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2</td>
<td>1.126.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
</tbody>
</table>
### 1.7.68

- Bumps UAA to v3.6.13.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3363.25</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2</td>
<td>1.126.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

### 1.7.67

- Bumps stemcell to v3363.25.
- Bumps cflinuxfs2 rootfs to v1.126.0.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3363.25</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2</td>
<td>1.126.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
1.7.66

- Bumps UAA to v3.6.12.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.26</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.65

- Bumps stemcell to v3312.26.
- Bumps cf-mysql-release to v26.13.0.
- Patches Cloud Controller to increase the application healthcheck timeout to 10 minutes.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.26</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.13.0</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.64

- Bumps uaa to v3.6.11.
<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.24</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11.0</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.63
- Bumps stemcell to v3312.24.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.24</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11.0</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.62

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.23</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.61

- Bumps uaa-release to v13.12.
- Bumps HAProxy to v1.5.19 and PCRE to v8.40 to patch some security vulnerabilities.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.23</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11.0</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.60

- Bumps stemcell to version 3312.23.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.23</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11.1</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>Component</td>
<td>Version</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.59

- Bumps rootfs to v1.60.0 with stack 1.111.0 for low/medium security fixes
- Bumps UAA to v13.11 to patch session fixation bug

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.22</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.60.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.58

- Bumps the stemcell version to 3312.22.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3312.22</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.56.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
## 1.7.57

- Configures the CAPI debug servers to bind to the loopback device instead of the open interface.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.21</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.56.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

## 1.7.56

- Bumps the rootfs to v1.56 which contains stack version 1.107.0.
- Fixes a bug in the internal MySQL configuration that prevented notifications from being sent when the cluster went into a dataloss-prevention state.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.21</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.56.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
### 1.7.55

- Bumps the stemcell to version 3263.21.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.21</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

### 1.7.54

- Bumps cf-mysql-release to v26.11 to ensure Golang components compile against Golang version 1.7.
- Bumps mysql-backup-release to v1.32.0 to ensure Golang components compile against Golang version 1.7.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.20</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.11</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.32.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
1.7.53

- Secures the router debug servers by ensuring they bind to the loopback device.
- Patches the router to prevent extra ?s from being appended to requests that already contain one.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.20</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.52

- Resolves issue with migration Ops Manager properties from versions of the 1.6 ERT.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.20</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.51

Note: This release was pulled because of an issue discovered with property migrations. A new 1.7.52 release with a patch is forthcoming.
Bumps the stemcell to version 3263.20.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.20</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.50

- Allows external databases to be configured with unique user accounts.
- Bumps UAA to v3.6.6 to redact SAML & OAuth Keys from the ID Zone API.
- Allows a “Max Inflight Container Starts” configuration to be set. This setting will limit the total number of containers that are allowed to be starting at any one time. The default setting is to limit the number of inflight starting containers to 200.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.17</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.5</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.46.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>34</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.7.49

- Patches the route-registrar component to prevent logging credentials.
- Adds configurable audit logging for the Internal MySQL database. Configuration options can be found on the Internal MySQL page.
1.7.48

- Patches the autoscaling release to remove the git dependency that had unpatched vulnerabilities.
- Patches the notifications errand to remove logging that included UAA OAuth tokens.
- Bumps the rootfs to 1.97.0 to cover some low and medium vulnerabilities.

1.7.47

- Improves Diego Cell resiliency when Consul servers become unavailable.
### 1.7.46

- Bumps the stemcell version to 3263.17 to patch a number of low vulnerabilities.
- Patches a bug in the gorouter that caused the router to erroneously include port numbers when setting the `X-CF-Forwarded-Url` header.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.17</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.3</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.43.0</td>
</tr>
<tr>
<td>consul</td>
<td>101*</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0*</td>
</tr>
<tr>
<td>etcd</td>
<td>49*</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>32</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

### 1.7.45

- Bumps the garden-runc release to version 1.1.1 to address CVE-2016-9962. For more details, please see [pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.15</td>
</tr>
<tr>
<td>cf</td>
<td>235*</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.3</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td></td>
</tr>
<tr>
<td>consul</td>
<td></td>
</tr>
<tr>
<td>diego</td>
<td></td>
</tr>
<tr>
<td>etcd</td>
<td></td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.1.1</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>32</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.
### 1.7.44

- Limits the total number of containers that can be in the starting state at any given time to 200.

### 1.7.43

**Note:** The version of AppsManager included in this release will require that Operators upgrade their ruby buildpack to at least version 1.6.18 before performing this upgrade. This buildpack version removes support for Ruby 2.2.3, in addition to making changes to the included bundler, node, and jRuby dependencies. Please see [ruby-buildpack release notes](https://github.com/joyent/node-ruby-buildpack#release-notes) for more details.

- Requires user input for the Internal MySQL Monitor Recipient Email Address field. This field previously defaulted to an invalid email address. Users that wish to use the internal MySQL cluster and deploy the monitor should remember to set a valid email recipient.
1.7.42

- Bumps the stemcell to 3263.15 to address a memory usage issue in rsyslog.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.15</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.3</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.43.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>32</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.14</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.41

- Bumps the stemcell to v3263.14 and the rootfs to v1.94.0 to address a vulnerability in API (USN-3156-1).
- Bumps the Golang buildpack to v1.7.16 to bring in support for Golang v1.7.
- Patches a vulnerability in the Notifications service that allowed unprivileged users to impersonate other users with unauthenticated tokens.
- Patches Cloud Controller to remove logging of database credentials.
- Patches Cloud Controller to remove old functionality that would delete resources at their previous droplet path.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3263.14</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36.3</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.43.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
</tbody>
</table>
## 1.7.40

- Bumps the stemcell to 3263.12.
- Patches Cloud Controller to redact logging of service broker provided credentials when binding application services.
- Patches a vulnerability in the Autoscaling service that allowed unprivileged users to impersonate other users with unauthenticated tokens.
- Patches an issue with the Diego Cell executor cache that could result in failure to stage applications.

### Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>32</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>26</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.13</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

## 1.7.39

- Bumps the stemcell to 3233.6 to address the Linux kernel vulnerability described in USN-3151-2.
- Bumps the rootsfs to version 1.91.0.
- Patches Cloud Controller to prevent disabled buildpacks from being used pushing an application.
- Patches Cloud Controller to redact logging of application environment variables.
- Patches Cloud Controller to prevent SpaceAuditors from being able to stage applications.
- Bumps UAA to 3.6.5, resolving a Tomcat CVE (CVE-2015-6816), and reduces the allowed cipher suites used when communicating to UAA. The accepted ciphers have been restricted to TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.
- Corrects a misconfiguration in the CloudController that prevented applications larger than 1MB from being uploaded.

### Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.6</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
</tbody>
</table>
### 1.7.38
- Resolves an issue with the Diego Cell download cache that could result in unexpected behavior.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

### 1.7.37
- Reduces the allowed cipher suites used when communicating to components of Diego. The accepted ciphers have been restricted to `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` and `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>
1.7.36

- Introduces a patch to prevent loss of application routes when the consul cluster becomes unreachable or loses quorum. Previously, the loss of the consul cluster would result in applications becoming unavailable as their routes were pruned from the routing table. Now, the route emitter will continue to update the router even while consul is down. Operators can monitor the `ConsulDownMode` metric for indication that the installation has detected this issue.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.35

- Bumps UAA to address a Tomcat server vulnerability that could allow for remote code execution.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>
1.7.34

- Bumps the version of PCRE used in both HAProxy and Nginx to address multiple CVEs. Please see https://pivotal.io/security for more information.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.33

- Patches the Diego container runtime to improve the performance of the application of security group rules during application startup.
- Patches the NFS mounter in the internal filestore to resolve an issue that prevented large blob uploads from succeeding. This only affected installations using the internal blobstore. External blobstores like S3 were not impacted.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.32

- Bumps garden-runc to v1.0.2. See the release notes for more information.
- Bumps the stack rootfs to patch the USN-3096-1 and USN-3088-1 vulnerabilities.
- UAA has been patched to correctly return the Basic WWW-Authenticate header if client credentials are not provided in the request to /oauth/token endpoint.
- Exposes the “Signature Algorithm” property on the “Authentication and Enterprise SSO” page. This will allow operators to specify what kind of signature they use for SAML signed requests.

- Exposes the “Request Max Buffer Size” property for HAProxy deployments on the “Networking” page. This property determines the maximum request header payload that HAProxy will accept.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.35.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.31

- Patches Cloud Controller to obfuscate custom buildpack properties that may contain credentials.
- Patches the internal MySQL logging mechanism to prevent accidental credential leaks.
- Bumps garden-runc to v1.0.0.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.3</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.8</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>1.0.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>22</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.30

- Fixes [CVE-2016-5195](https://pivotal.io/security) for more information see [https://pivotal.io/security](https://pivotal.io/security).
## 1.7.29

- Fixed an inconsistency between the hostname and the contents of /etc/hosts that would cause some java apps to crash.
- Ability to configure `drain_wait` and `load_balancer_healthy_threshold` on the router. Should be helpful to customers with F5s.
- UAA now allows for a custom regular expression to be set for Proxy IP's.
- Cloud Controller received a patch to prevent monit from hanging when the NFS server becomes unavailable.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.2</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>0.9.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

### 1.7.28

Note: This release caused an inconsistency between the hostname and the contents of /etc/hosts that would cause some java apps to crash. It has been fixed in the next patch release.

- Bumps the required stemcell version to 3233.2. This stemcell is based on the 4.4 Linux kernel.
Advantages of Garden-runC over Garden-Linux are:

- AppArmor is configured and enforced by default and out-of-the-box for all unprivileged containers.
- Seccomp whitelisting restricts the set of system calls a container can access, greatly reducing the surface area for break-out exploits. This is set up out-of-the-box; you don’t need to do anything.
- The new Garden code base is simpler and more modular, allowing pluggable networking and pluggable rootfs management. It enables container-to-container networking and the new “grootfs” OCI-compliant rootfs downloder.
- Garden-runC uses the same low-level container execution code as docker/k8s for running containers, so that your container images run the same in PCF as elsewhere.

Lastly Garden-runC has been successfully running 100% of the production traffic on Pivotal Web Services (PWS) for around a month. And, it has been tested at lower load for some months before that. However, as always, make sure to test in a staging environment before deploying to production.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3233.2</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>49</td>
</tr>
<tr>
<td>garden-runc</td>
<td>0.9.2</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.27

- Includes a fix for the Critical Vulnerability issue CVE-2016-6655. See https://pivotal.io/security for more information.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.21</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cf-linuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>6</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>
1.7.26

- Fixes an issue where you cannot successfully deploy Consul servers with public IP addresses, in case your network that you have deployed Elastic Runtime to is not an RFC 1918 private network.
- Allows the operator to define supported currencies for the service marketplace. This new property can be configured in the "Apps Manager" configuration form.
- Fixes an issue with running Windows apps in Diego.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.21</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>101</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.11</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.25

- Fixes a bug where the apps-usage service could lose database connectivity while the internal MySQL proxies were being deployed. This only affects deployments using internal mysql as a database.
- Changes the route for the switchboard UI for the internal mysql proxies. Previously, those dashboards were located at proxy-X-p-mysql.internal.SYSTEM_DOMAIN. Now they can be found at proxy-X-p-mysql-ert.SYSTEM_DOMAIN. This should remove the need to include an extra SAN entry in the TLS certificates for the platform.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.21</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>96</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>
1.7.24

- Fixes an issue with upgrades from Elastic Runtime 1.7.20, by updating UAA to version 3.6.1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.21</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>96</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.23

Note: Do not upgrade to this version if you are starting with 1.7.20. You must upgrade to 1.7.24 or greater instead.

- Patches USN-3087-1, USN-3087-2, CVE-2016-6662, USN-3040-1, and USN-2953-1. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.21</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.6</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.33.0</td>
</tr>
<tr>
<td>consul</td>
<td>96</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.22

Note: Do not upgrade to this version if you are starting with 1.7.20. You must upgrade to 1.7.24 or greater instead.

- Re-orders the MySQL Monitor job deployment to deploy after the UAA job, so that the required UAA client is ready before the monitor deploys.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
</table>

© Copyright Pivotal Software Inc, 2013-2018
### 1.7.21

Note: Do not upgrade to this version if you are starting with 1.7.20. You must upgrade to 1.7.24 or greater instead.

- Patches CVE-2016-6651. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.4</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>consul</td>
<td>96</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

### 1.7.20

- Adds a manifest change for Diego that enables successful upgrades to a future 1.8.0 version of Elastic Runtime, once that is released.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.4</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>Component</td>
<td>Version</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.19

- Bumps the PHP buildpack to v4.3.18. The buildpack was exposing the `.profile` file which could contain environment variables and credentials. The PHP buildpack prior to v4.3.18 did not actually allow for execution of the `.profile` file, so it is unlikely that many applications were using it.

### 1.7.18

We’ve discovered a rare condition where a PCF deployment running an internal MySQL cluster can experience a fault in replication that may result in some data loss. When this occurs, previous releases do not log the root cause of the bug. In order to best address this issue, version 1.7.18 contains significant additional telemetry and several defensive features which will account for the failure condition and prevent data loss.

#### Introducing the Replication Canary

We’ve included a new long-running monitor, the **Replication Canary**. The Replication Canary continually monitors the MySQL cluster, watching for instances in which cross-cluster replication has failed. It is enabled by default, and requires an e-mail address in the Advanced Options configuration pane.

In the event that replication has failed, the Canary performs two actions:

- E-mail the Operator: Part of the Replication Canary’s configuration is an e-mail address, which can be directed to any Operator e-mail address, or an escalation system similar to PagerDuty.
- Deny Access: When replication has failed, the Replication Canary will automatically disable user and applications’ ability to access the cluster via the Proxies.
You must set the Monitoring job to 1 in the Resource Config pane, or the Replication Canary will not be enabled, regardless of configuration.

You must also confirm that the Elastic Runtime tile is properly configured to send e-mail. These settings are necessary for any standard Cloud Foundry configuration.

- Ensure that the Notifications errand has been enabled.
- Ensure that SMTP Config has been properly configured.

If either of these are not set, configure and Apply Changes before deploying version 1.7.18.

Introducing the Interruptor

The MySQL nodes have new logic that, when enabled, will prevent a node from re-joining a cluster under certain conditions. This is a second level of protection against the possibility of data loss.

New feature: Backup Improvements

- In the Internal MySQL configuration pane, there’s now an option to take backups from all MySQL nodes. This feature protects your users from data loss in the case that some nodes have different data than the others.
- Additionally, there is now the option to provide either an S3-compatible bucket or SCP endpoint for automated backups.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>26.4</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1.25.0</td>
</tr>
<tr>
<td>mysql-monitoring</td>
<td>5</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>14</td>
</tr>
</tbody>
</table>

1.7.17

- Removes the maximum memory restriction for the NATS VM. This artificial limit prevented some IaaSes from provisioning VMs that could support the NATS job.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.342.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>
1.7.16

- Patches CVE-2016-5388 and USN-3024-1. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Adds an endpoint for Apps Usage Service to help determine if this app has corrupted data. This update will extend the app event data kept from Cloud Controller to 160 days instead of 30 days.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.10</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.15

- Patches USN-3064-1, USN-3048-1, USN-3060-1, USN-3061-1, and USN-3065-1. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Introduces a patch which addresses application routing downtime during upgrades whenever IPSec has been applied to the PCF environment. Additional info can be found in this Knowledge Base article.
- Fixes an issue where the legacy Apps Manager and Apps Usage system applications were not always completely deleted from the environment to make way for the latest versions of these apps.
- Updates the size of the push-errand VM for Autoscale, to increase the chances of the errand’s success.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.17</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.26.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.7</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.14

- Introduces a feature which allows you to use AWS’s Server-Side Encryption (SSE) feature for your S3 buckets if you use S3 buckets as Cloud Controller’s file store.
- Updates the version of Golang used by system apps in Elastic Runtime to a newer version, since Golang 1.5 will soon be end of life.
- Patches Apps Manager to address an issue where app events were not appearing in the older version of Apps Manager (the Ruby version).
• Patches Autoscale to address an issue where autoscaling was not properly scaling apps automatically and sending notifications.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.12</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>36</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.19.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.6</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.13

• Addresses an issue where metron sometimes panics from a concurrency fault while aggregating messages when the load is higher.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.12</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>34</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.17.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.4</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.12

• Adds all link-local addresses, in the range of 169.254.0.0/16, to be excluded from access in the default application security group of Elastic Runtime. This range includes the metadata endpoints of most IaaS providers, such as AWS. This security group does not take effect for upgrades, only new installs of Elastic Runtime. If you would like to update your pre-existing deployment, please refer to the Application Security Groups topic of our documentation.

• Introduces a patch to Cloud Controller’s API that will enable use of the upcoming Spring Cloud Data Flow’s Task support on Cloud Foundry. More about this release will be announced soon.

• Updates Consul to v96 in preparation for future upgrades to Elastic Runtime 1.8 (the release of which is upcoming), which will require all components to adopt a new format for the bosh manifest.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.12</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
</tbody>
</table>
1.7.11

Note: The CloudFormation script for PCF setup on AWS was updated at the same time as this release to switch over to using a NAT EC2 instance instead of a NAT gateway, as there is a known issue with AWS NAT gateways handling hairpin routing requests.

- Patches CVE-2016-5006, USN-3012-1, USN-3010-1, and CVE-2016-4450. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Introduces a fix for GoRouter to address an issue where failover of requests to unresponsive app instances was not working properly whenever the request has a body. See [tracker story](https://pivotal.io/security) for details.
- Increases the default instance size of the UAA instance to handle larger logs.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.12</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>34</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.17.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.4</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018
### 1.7.9
- Patches USN-3020-1. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Addresses an issue with Loggregator not properly setting timeouts on TCP connections to Cloud Controller and UAA.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>24</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>17</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.4</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

### 1.7.8
- Patches CVE-2016-0928 and CVE-2016-0926. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Addresses an issue with the Apps Manager push errand sometimes failing.
- Addresses an issue with rare occurrences of application containers being assigned duplicate IP addresses.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.12</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.15.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629.4</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018

793
1.7.7

- Patches CVE-2016-4468. Additional info can be found at https://pivotal.io/security.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.8</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.15.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.6

- Patches USN-3001-1. Additional info can be found at https://pivotal.io/security.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.8</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.15.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.338.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>629</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

1.7.5

- Patches USN-2985-1, USN-2985-2, USN-2981-1, USN-2970-1, USN-2966-1, USN-2994-1, USN-2987-1, USN-2990-1, USN-2983-1, and USN-2961-1. Additional info can be found at https://pivotal.io/security.

- This also addresses an issue with rare occurrences of application containers being assigned duplicate IP addresses.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.6</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.15.0</td>
</tr>
</tbody>
</table>
### 1.7.4

- This only patches the CF CLI version used to push the Elastic Runtime system apps, like Apps Manager, to be more reliable. You do not need to upgrade to this version if you’re already at Elastic Runtime 1.7.3.

### 1.7.3

- As part of this release, there is a checkbox in the Elastic Runtime tile configuration that asks for every operator/administrator of the deployment to acknowledge that they understand how to implement application security groups successfully to secure their deployments. More info about this topic can be found here, in the [Application Security Groups](#) topic of our documentation.

- This also introduces a few bug fixes, including a patch to Diego to prevent rare occurrences of the route-emitter locking up, another patch to Diego to prevent Diego BBS contention issues, added AWS regions to the Region field dropdown in the S3 option for file storage to enable selection of other regions using a V4 S3 endpoint, a patch for UAA to fix a bug introduced by Elastic Runtime 1.7.2 that broke using external LDAP over SSL, and a patch to the Elastic Runtime tile user-interface for custom branding to perform better when entering base-64 encoded images for your company logos.
## 1.7.2

- Patches USN-2977-1 and CVE-2016-3084. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security). Also increases the maximum length you can have for CF task commands and ensures greater reliability for etcd.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.4</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.3.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>48</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.337.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>621</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

## 1.7.1

- Introduces an option to configure security event tracking in the Cloud Controller logs. Also adds statsd metrics from Cloud Controller.
- Also patches USN-2959-1, USN-2957-1, USN-2949-1, USN-2943-1, and USN-2935-2. Additional info can be found at [https://pivotal.io/security](https://pivotal.io/security).
- If you are using Single Sign-On 1.1.0, it is recommended that you update to Single Sign-On 1.1.1. Additional info can be found at [http://docs.pivotal.io/p-identity/release-notes.html](http://docs.pivotal.io/p-identity/release-notes.html).

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3232.2</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>1.3.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>45</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.337.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
<tr>
<td>push-apps-manager-release</td>
<td>621</td>
</tr>
<tr>
<td>service-backup</td>
<td>1</td>
</tr>
</tbody>
</table>

### Version 1.7.0 New Features

#### Elastic Runtime

© Copyright Pivotal Software Inc, 2013-2018 796 1.7
Diego completely replaces DEAs

Before you upgrade to Pivotal Cloud Foundry 1.7, you must migrate all apps that are currently running on DEA architecture to run on Diego architecture. Pivotal does not support DEA architecture in Pivotal Cloud Foundry 1.7. You may need to scale up your VMs before migrating applications. More information can be found here.

If you have any issues migrating your applications to Diego, please contact Support.

Elastic Runtime Tile UI Changes

The tile has been re-designed in layout to improve the configuration workflow for operators. Several configuration fields have been moved to different sections of the tile. Also, more advanced logic has been implemented for some sections to make configuring complicated networking use cases easier for operators.

Compiled Releases

Installs and upgrades of Elastic Runtime are now much faster than before, as the installation now makes use of compiled binaries instead of source code.

Automated Backups

Operators using Pivotal Elastic Runtime can enable automatic backup on S3-compatible blobstores. This feature can be enabled for ERT running AWS, OpenStack, or vSphere.

Please see this topic for more information.

Docker Private Registries

Please see this topic for information on Docker Trusted and Private registries.

Unused Docker Image Cleanup

In the Application Containers configuration section of Elastic Runtime, you can configure how aggressively unused Docker image files are removed from disk on Diego Cell VMs. The advantage to cleaning up less frequently is that subsequent push/scale events for Docker Images that have previously been run on the platform may be faster, since the image does not need to be downloaded from the Docker registry again. The “Threshold of Disk-Used (MB)” value refers to the total size of all Docker Images (whether in use or not) on an individual Cell VM.

Route Services

Route Services are a new kind of Marketplace Service that developers can use to apply various transformations to application requests by binding an application’s route to a service instance. Through integrations with service brokers and optionally with the Cloud Foundry routing tier, providers can offer these services to developers with a familiar automated, self-service, and on-demand user experience.

Please see this topic for more information.

SpaceScoped Private Brokers

Space scoped private brokers allow a space developer to create and manage a service broker and scope it to a single space in order to allow faster iteration on service brokers without requiring full admin privileges.

Please see the managing service brokers topic for more information. This feature can be managed with the space_scoped_private_broker_creation cloud controller feature flag.

Experimental Feature: Disk and Memory Overcommit Settings
If your apps do not use the full allocation of disk space and memory set in the Resource Config tab, you may want to use this feature. These fields control the amount to overcommit disk and memory resources to assign more memory and disk space to each Diego Cell VM.

Please see this topic for more information.

Experimental Feature: Diego Tasks

Although you can use the V3 Tasks API to run asynchronous tasks, Pivotal reserves the right to change this API or truncate the data about past task runs. For more information about running one-off tasks in Cloud Foundry, see the Cloud Foundry V3 API Docs.

AWS CloudFormation Script Update

You can now specify whether you would like to enable HTTP traffic to port 80 of your ELB if you are setting up an environment in AWS to deploy PCF.

App Manager White Labeling

Pivotal Cloud Foundry operators can now visually brand Apps Manager by changing certain text, colors, and images of the interface. Developers will see the customized interface when logging in, creating an account, resetting a password, or using Apps Manager.

Please see this topic for more information.

Identity (aka UAA Server)

UAA PW Policy Config in ERT

Operators can now specify some password policies in the UAA section of Elastic Runtime, including password length, specific character type requirements, expiration, and maximum password attempts allowed.

Please see this topic for more information.

UAA SAML features

Please see this topic for more information.

UAA Token Lifetime Settings

Please see this topic for more information.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell</td>
<td>3215</td>
</tr>
<tr>
<td>cf</td>
<td>235</td>
</tr>
<tr>
<td>cf-autoscaling</td>
<td>28</td>
</tr>
<tr>
<td>cf-mysql</td>
<td>25.2</td>
</tr>
<tr>
<td>cflinuxfs2-rootfs</td>
<td>0.2.0</td>
</tr>
<tr>
<td>diego</td>
<td>0.1467.0</td>
</tr>
<tr>
<td>etcd</td>
<td>45</td>
</tr>
<tr>
<td>garden-linux</td>
<td>0.337.0</td>
</tr>
<tr>
<td>mysql-backup</td>
<td>1</td>
</tr>
<tr>
<td>notifications</td>
<td>19</td>
</tr>
<tr>
<td>notifications-ui</td>
<td>10</td>
</tr>
</tbody>
</table>
In memory...

We would like to dedicate this work in memory of our dear friend and colleague Dave Liebreich, who passed away on February 22, 2016. Thank you for everything, Dave, you are missed every day by your family here at Pivotal!
How to Upgrade
Upgrading Pivotal Cloud Foundry is documented here.

About Updating to Ops Manager v1.7.15 or Later
If you are currently on Ops Manager v1.7.14 or earlier, you should update your Ops Manager. Later versions include a major stemcell upgrade to 3233.x. This stemcell uses the Linux kernel v4.4 instead of the v3.19. Ubuntu is no longer providing CVE patches for Linux kernel v3.19.

1.7.32 Patch
- Bumps UAA to 13.17

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3363.26</td>
</tr>
<tr>
<td>BOSH-init: v0.0.103</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.17</td>
</tr>
</tbody>
</table>

1.7.31 Patch
- Patches Ubuntu Security Notice USN-3334-1. Additional information can be found at https://pivotal.io/security.
- Bumps stemcell to 3363.26

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3363.26</td>
</tr>
<tr>
<td>BOSH-init: v0.0.103</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.16</td>
</tr>
</tbody>
</table>

1.7.30 Patch
- Fixes a bug where the redirect URI for the BOSH UAA was incorrectly set
- Bumps Nokogiri dependency to 1.7.2 to patch CVE-2017-5029. Additional information can be found at https://pivotal.io/security.
- Bumps BOSH-init to v0.0.103
- Bumps stemcell to 3363.25

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3363.25</td>
</tr>
<tr>
<td>BOSH-init: v0.0.103</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.17</td>
</tr>
</tbody>
</table>
1.7.28 Patch

- Bumps stemcell to 3263.24

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3263.24</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.13</td>
</tr>
</tbody>
</table>

1.7.27 Patch

- Bumps UAA to 13.13

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3263.23</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.13</td>
</tr>
</tbody>
</table>

1.7.26 Patch


<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3263.23</td>
</tr>
<tr>
<td>BOSH Director: 255.14</td>
</tr>
<tr>
<td>vSphere CPI: 37</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.12</td>
</tr>
</tbody>
</table>

1.7.25 Patch

- Patches USN-3249-2. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Fixed a bug where Ops Manager was not setting the correct SAML Service Provider certificate in the BOSH Director UAA.

<table>
<thead>
<tr>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell: 3263.22</td>
</tr>
</tbody>
</table>
1.7.24 Patch
- Patches USN-3220-2. Additional information can be found at https://pivotal.io/security.

1.7.23 Patch
- Patches USN-3208-2. Additional information can be found at https://pivotal.io/security.

1.7.22 Patch
- Patches USN-3161-2
- Patches USN-3169-2
- Patches USN-3172-1
- Bumped the default memory of vSphere and OpenStack appliances to 8GB

1.7.21 Patch
- Move ahead to the 3263.x stemcell line.
- Patches rsyslog memory leak.
- Fixed issue (introduced in 1.7.19) where sometimes multiple UAA processes would start and operators would be stuck at the "Waiting for
authentication to start" screen.

- When deploying the vSphere OVA, fixed an issue (introduced in 1.7.19) where multiple DNS servers can again have spaces between them.
- Operators no longer have to unlock the appliance directly after import.
- Fixed issue where the BOSH health monitor could not send emails if authentication was required for SMTP.
- Add encryption for AWS ephemeral disks as well, when the operator has enabled EBS encryption.

### Versions

<table>
<thead>
<tr>
<th>Stemcell: 3263.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOSH Director: 255.12</td>
</tr>
<tr>
<td>vSphere CPI: 26</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.9</td>
</tr>
</tbody>
</table>

#### 1.7.20 Patch

- Patches USN-3156-1.
- Fixes an upgrade issue where jobs pre-1.7 products could be accidentally treated as non-singleton. In a very specific circumstance, this could have caused data loss.

### Versions

<table>
<thead>
<tr>
<th>Stemcell: 3233.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOSH Director: 255.12</td>
</tr>
<tr>
<td>vSphere CPI: 26</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
<tr>
<td>UAA release: 13.9</td>
</tr>
</tbody>
</table>

#### 1.7.19 Patch

- Patches USN-3151-2. Additional information can be found at https://pivotal.io/security.
- vSphere customers can now specify a custom hostname in the OVA template for the Ops Manager VM
- Ops Manager now exposes a Director flag, `keep_unreachable_vms`, that prevents the Director from deleting VMs that are unreachable
- Ops Manager now uses stronger cipher suites for SSL termination, which now requires TLS1.2 when communicating with it or the UAA attached to it.

### Versions

<table>
<thead>
<tr>
<th>Stemcell: 3233.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOSH Director: 255.12</td>
</tr>
<tr>
<td>vSphere CPI: 26</td>
</tr>
<tr>
<td>AWS CPI: 57</td>
</tr>
<tr>
<td>OpenStack CPI: 23</td>
</tr>
</tbody>
</table>

#### 1.7.18 Patch

- Patches CVE-2016-6657. Additional information can be found at https://pivotal.io/security.
- Fixed a bug that was preventing UAA clients from using the Ops Manager API.

### Versions

<table>
<thead>
<tr>
<th>Stemcell: 3233.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOSH Director: 255.12</td>
</tr>
</tbody>
</table>

© Copyright Pivotal Software Inc, 2013-2018
### 1.7.17 Patch
- Patches USN-3106-2. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Versions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell:</td>
<td>3233.3</td>
</tr>
<tr>
<td>BOSH Director:</td>
<td>255.12</td>
</tr>
<tr>
<td>vSphere CPI:</td>
<td>26</td>
</tr>
<tr>
<td>AWS CPI:</td>
<td>57</td>
</tr>
<tr>
<td>OpenStack CPI:</td>
<td>23</td>
</tr>
</tbody>
</table>

### 1.7.16 Patch
- Patches USN-3099-2. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).
- Ops Manager can now be deployed on AWS M4 machine types.

<table>
<thead>
<tr>
<th>Versions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell:</td>
<td>3233.2</td>
</tr>
<tr>
<td>BOSH Director:</td>
<td>255.12</td>
</tr>
<tr>
<td>vSphere CPI:</td>
<td>26</td>
</tr>
<tr>
<td>AWS CPI:</td>
<td>57</td>
</tr>
<tr>
<td>OpenStack CPI:</td>
<td>23</td>
</tr>
</tbody>
</table>

### 1.7.15 Patch
- Bumped Linux kernel to v4.4
- Patches CVE-2016-6655. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Versions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell:</td>
<td>3233.1</td>
</tr>
<tr>
<td>vSphere CPI:</td>
<td>26</td>
</tr>
<tr>
<td>AWS CPI:</td>
<td>57</td>
</tr>
<tr>
<td>OpenStack CPI:</td>
<td>23</td>
</tr>
</tbody>
</table>

### 1.7.14 Patch
- Patches USN-3087-2. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).

<table>
<thead>
<tr>
<th>Versions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemcell:</td>
<td>3232.21</td>
</tr>
<tr>
<td>vSphere CPI:</td>
<td>26</td>
</tr>
<tr>
<td>AWS CPI:</td>
<td>57</td>
</tr>
<tr>
<td>OpenStack CPI:</td>
<td>23</td>
</tr>
</tbody>
</table>

### 1.7.13 Patch
- Patches CVE-2016-6651. Additional information can be found at [https://pivotal.io/security](https://pivotal.io/security).

© Copyright Pivotal Software Inc, 2013-2018
1.7.12 Patch

- Customers can now opt in to provision a UAA admin client while configuring their Ops Manager with SAML. This client can be used to run scripts against the BOSH Director.
- Fixed a bug with the logout workflow
- Customers can now enter a custom value for instance count in the Resource Config Form

1.7.11 Patch

- Patches USN-3064-1. Additional information can be found at https://pivotal.io/security.
- Bumped UAA to version 11.4

1.7.10 Patch

- Introduced a new feature that lets operators specify a custom host name for the BOSH Director.
- Introduced a new feature that lets operators add a Load Balancer to the Ops Manager Director Job
- For more information on how to set up a load balancer in front of the Director, please refer to the KB article.
- Includes new BOSH Networking release with the correct RPF filter necessary for dual-homing scenarios
- Fixed API endpoint to log out all active sessions

1.7.9 Patch

- Patches USN-3020-1. Additional information can be found at https://pivotal.io/security.
- Fixed an issue that was preventing customers from using OpenStack keystone versions greater than 3.0
1.7.8 Patch

- Patches CVEs including CVE-2016-0897 and CVE-2016-4468. Additional information can be found at https://pivotal.io/security.
- Bumped UAA version to 3.3.0.2
- Customers using vSphere or vCloud Air are now required to enter a default admin password in their OVA or TAR template. Failure to do so will result in the VM to fail to boot. Additional information can be found at http://docs.pivotal.io/pivotalcf/1-7/customizing/vsphere.html
- Customers using the dual-homed feature in Ops Manager v1.6 can now import their installations into v1.7.8 and continue to use the dual-homed BOSH Director
- Customers can now dual-home their BOSH Director starting with new deployments of v1.7.8
- For more information on how to use the dual-homing feature, please refer to the KB article.

1.7.7 Patch

- Patches USN-3001-1. Additional information can be found at https://pivotal.io/security.

1.7.6 Patch

- Patches USN-2016-4435, USN-2985-2, USN-2985-1, USN-2981-1, and USN-2966-1. Additional information can be found at https://pivotal.io/security.
- Fixed an issue in OpenStack that prevented users from assigning multiple AZs in a subnet.

1.7.5 Patch

- Bumps BOSH-Init to v.0.92 to disable HTTP connection reuse to avoid issues observed only on OpenStack environments

1.7.4 Patch

- Fixes an issue that caused Ops Manager deploys to fail due to a BOSH-init bug
- Fixes an issue that caused JavaScript migrations to fail for large tiles
- Bumps UAA to 3.3.0

1.7.3 Patch

- Patches USN 2977-1. Additional info can be found at https://pivotal.io/security.

1.7.2 Patch

- Fixes a bug that caused initial User Authorization and Authentication (UAA) setup to stall in upgrade cases
- Upgraded stemcell version to 3232.3

1.7.1 Patch

- Patches USN-2959-1, USN-2957-1, USN-2949-1, USN-2943-1, and USN-2935-2. Additional info can be found at https://pivotal.io/security.

Version 1.7.0 New Features

Security

All Ops Manager assets are now being protected by UAA User Authorization and Authentication. Customers upgrading from 1.6 will have their Admin user migrated into PCF UAA while retaining their existing password as the passphrase. Ops Manager 1.7 has support for multiple user accounts when
using internal as well as remote SAML identity management. In order to avoid consistency issues, Ops Manager restricts the number of concurrent users in the system by allowing only one user to log in at a time.

Users now have the capability to add custom certificates to all the VMs managed in a BOSH deployment. Please see below for more information about custom certificates:

- Amazon Web Services
- OpenStack
- vSphere
- vCloud

Operators can now see a detailed change log that describes all actions performed by all authenticated users.

Several improvements to stemcell management have been added to Ops Manager 1.7.0. Operators can only upgrade stemcells (i.e. stemcells cannot be downgraded). For AWS installations, only the Light HVM versions can be used. Additionally, tiles installed in Ops Manager will not need to be manually upgraded for security patches to a minor version of the stemcell.

Networks and Availability

For Ops Manager 1.7.0, networks are now a logical collection of subnets. Users can create more than one network, and add one or many subnets to those networks. For Availability Zones (AZs), users can add more than one AZ per subnet. They can also assign products to both networks and AZs. Jobs in a deployment will be balanced across subnets and Availability Zones. Please see the following topics for more information on configuring subnets and AZs.

- Amazon Web Services
- OpenStack
- vSphere
- vCloud

Resource Configuration

There is a new workflow for configuring resources in Ops Manager. This workflow includes a standard way of configuring instance sizes across all infrastructures. Operators also have the ability to opt in or opt out of default sizes set by Pivotal product tiles. For more information, please see:

- Amazon Web Services
- OpenStack
- vSphere

API Endpoints

As of Ops Manager 1.7, all API endpoints are protected by PCF UAA User Authentication and Authorization. Ops Manager has these new endpoints:

- List installed products
- List VM credentials by product
- Unlock Ops Manager with passphrase
- CRUD disk types and vm types
- List static IPs
- Manifest for staged products
- Manifest for deployed products
- Token expiration time can be configured
- Diagnostic information to attach to support tickets
- Get a BOSH manifest for bosh-init deployment
AWS

For AWS installations, EBS (Elastic Block Store) is not supported on all instances. For more information on EBS encryption, please see Configuring Amazon EBS Encryption. Amazon Machine Images (AMIs) can now be built in Frankfurt and Seoul. Additionally, AWS S3 version 4 is now supported.

Several changes have been made for CloudFormation on AWS. The script now considers RDS (Relational Database Service) as optional and supports an HA NAT instance and multiple AZs. For more information, please see the Deploying the CloudFormation Template for PCF on AWS topic.

Security groups that are assigned to EC2 instances are now referenced by ID. Please see below for more information:

- Amazon Web Services
- OpenStack

vSphere

- Datastores can be split by Ephemeral and Persistent disk types
- Unlike other IaaS providers, subnets on vSphere can span availability zones
- vCenter IP field now supports IP or host

OpenStack

For OpenStack Deployments, Ops Manager 1.7.0 now supports connection options and Keystone version 3.

Tile Authors

Multiple improvements have been included in Ops Manager 1.7.0 for Tile Authors. PCF product tiles now must be upgraded using ECMA (JavaScript) migration syntax. Imports into 1.7 are limited to ECMA script migrations only.

For forms, invalid user input can be rejected using regular expressions and form fields support placeholder text. Markdown can be used above a form.

Ops Manager 1.7.0 includes several changes to how you configure products and jobs:

- The `instance_definition` property for jobs replaces `instance_definitions`. `instance_definition` is required, and contains a hash specifying the number of default instances for a job, along with max, min, odd, and the ability to decrease sizing after deploy constraints.
- A job’s instance count can be toggled to 0 based on a selector value.
- The `product template` must include `minimum_version_for_upgrade`.
- Products with pre 1.7 metadata are limited to a single subnet.
- Users must specify whether their product gets persistent disk or not: the field is required.
- `First-network-deprecated` has been removed as a key/value pair.
- Tile authors can limit a product’s jobs to a single AZ.
- Accessor information is available for trusted certificates.

Operational Improvements

In addition to the major features, there are a number of smaller improvements that will enhance the operator’s experience with installing and maintaining Pivotal Cloud Foundry.

- Operators can select and deselect errands before applying changes
- Support for compiled releases
- Users can disable ICMP checks (default off on AWS)
- Checkbox exists to run `bosh deploy --recreate` (recreates all VMs)
- Support for monitoring and Pager Duty™ BOSH plugins
- UAAC gem added to Ops Manager VM
- Automatic cleanup of `/tmp` directory for 24-hour-old files
Bug Fixes

- Some products could not be uploaded via API
- Wrong tile version shown in error messages
- Passwords aggressively scrubbed from logs resulting in non-secret obfuscation
- Job logs cleaned up by file cleaner
- Hints in selectors fixed
- Users had to specify resource pools on vSphere
- Installations were slower by writing logs to database
- When DNS is unreachable 500 error
- Importing installation errors with resource pools result in 500 error
- Partial selection of errands results in all errands checked
- Powering off VM can result in empty installation.yml under certain conditions
- Many API docs errata fixes
- Broken EULA links
- Selector checkboxes were not persisting
- Exports with empty releases causes 500 error
Pivotal Elastic Runtime v1.7 Known Issues

Issue with upgrade path between PCF Elastic Runtime v1.6.44 and v1.7.27

We have detected an issue with the upgrade path from Elastic Runtime version v1.6.44 to v1.7.27. This issue only affect this specific upgrade path. When upgrading your Elastic Runtime with these specific versions there is a version clash between the release versions of etcd used in each of these releases. If you are currently on v1.6.44 and want to upgrade to the v1.7 series, there will be a new release soon. If you are currently on v1.7.27, you will be able to upgrade to future releases. No immediate action is required.

Issues with Application Uptime in PCF Elastic Runtime v1.7.0 - v1.7.15

Issue: If you kill the NATS server without stopping it cleanly or if you upgrade/stop/restart NATS in an environment where the IPSec add-on is applied, then you may experience a period of application downtime. More information can be found in this Knowledge Base article.

Recommended action

Contact Pivotal Support for remediation of application downtime if you have IPSec versions older than v1.5.37 and an Elastic Runtime older than v1.7.15.

Issues with Autoscaling in PCF Elastic Runtime v1.7.10 - 1.7.13

Issue: PCF Elastic Runtime versions v1.7.0 to v1.7.13 contain a version of PCF Autoscale app (a service used to scale your app instances automatically) that periodically crashes once a scaling event has taken place.

This will be fixed soon in a future patch release of Elastic Runtime.

Recommended action

If you are planning to install these affected versions of Elastic Runtime and you use the autoscaling service, before you deploy the upgrade, disable the following: The errand that pushes an updated autoscale app (Deploy CF Autoscaling App) The errand to register the service broker (Register Autoscaling Service Broker)

Disabling these errands will prevent the new malfunctioning autoscale app from being deployed, and will allow you to continue using the old autoscale app as before.

Please note

If you have already upgraded to PCF Elastic Runtime versions v1.7.10 to v1.7.13 and you did not disable these errands, your updated version of the PCF Autoscale app will intermittently crash until a patch has been released. If you are experiencing this issue, you will need to manually scale your apps until the fix has been released.

If you need further assistance, please contact Pivotal Support.

New Issues

- There is an issue on these versions where the Consul server deployment may fail if you are deploying to a private network that does not use RFC 1918 private IP addresses.
- Elastic Runtime v1.7.20 has a version of UAA which does not upgrade successfully to any versions of Elastic Runtime that is not at least v1.7.24.
- Autoscaling does not work in PCF Elastic Runtime versions v1.7.10 to v1.7.13.
- The Single Sign-On service tile operates in lockstep with PCF Elastic Runtime. If you are a customer upgrading from PCF v1.6 to PCF v1.7 and you are using SSO v1.0.x, you must upgrade to the SSO v1.1.0 service tile while you are upgrading the Elastic Runtime Tile to v1.7, in the same “Update” step.
- LDAP SSL with self-signed certificates is not functional in the PCF v1.7.2 release. Upgrading to this release will cause UAA to not start up. This issue will
be addressed in the next patch release (v1.7.3) of PCF.

- The Push Notifications service tile operates in lockstep with PCF Elastic Runtime. If you are a customer upgrading from PCF v1.6 to PCF v1.7 and you are using Push Notifications v1.4.4 or earlier versions, you must upgrade to the Push Notifications v1.4.5 tile when you upgrade the Elastic Runtime Tile to v1.7.
- The Apps Manager console URL has changed from `console.YourSystemDomain` to `apps.YourSystemDomain` with no redirect. If you experience any issues because of this change, please contact Support.
- Apps Manager’s “Tools” page links to the incorrect version of the cf CLI. To get the latest version, download it directly from the CLI releases page.
- When you upgrade Ops Manager from v1.6 to v1.7 on vSphere or vCloud, you might notice that several of your Elastic Runtime VMs will automatically resize to have more CPU, memory, and/or disk. This is because the new Ops Manager defines specific instance types instead of custom sizings, and each instance will adopt an instance type that is the closest match to the previous custom size. For instance, you might see the Diego Cells will increase their CPU to 4 and their disk to 130 GB per Cell if you kept the default size of Cell instances from Elastic Runtime v1.6.x. If you disagree with the instance size automatically selected, you may select a different type anytime.
- No metrics from any jobs in Elastic Runtime v1.7.0 will be emitted to the firehose, so none of these metrics will appear in the JMX bridge. This was fixed in v1.7.1.
- Java apps can take advantage of the custom certificate authority feature of PCF v1.7, but it requires using a more recent Java buildpack version than the one included with Elastic Runtime v1.7.0. The buildpack version is 3.7, and it requires that you also enable the Container Certificate Trust Store feature. The feature is turned off by default, because it can impact staging performance of Java apps, adding an extra 45 seconds or so.
- The cross-container traffic checkbox is removed from Elastic Runtime, because since the migration from the DEA architecture to Diego, this feature setting is now controlled by your CF application security groups.

Existing Issues

- Do not put an apostrophe in database passwords for Elastic Runtime. The notifications and autoscale system apps are unable to use a password with an apostrophe.
- Diego Cells may sometimes time out on app pushes when any given Diego Cell receives its first app push with no buildpack specified for that app. This is because the Cell is caching all of the buildpacks for subsequent app pushes, and this may take longer than what the app staging allows.
- The CLI command for viewing application files, `cf files`, does not work with applications on Diego.
- To SSH into an application container on Diego, your CF user must have the Space Developer role attached to it for the application space.
- The Smoke Tests may occasionally fail at the logging test suite. The Smoke Tests errant is fine to re-run in case of any failures.
- .NET support on Windows cells does not support the same level of security and isolation as seen on Linux cells. At this time, it is only recommended for running “trusted” apps.
- At this time, the container accounts on Windows cells must have permissions to log on locally, which may not be the default.
- Application file names on Windows cells are limited to maximum length of 100 characters. As a workaround, make sure that all filenames in an application directory are shorter than 100 characters before pushing.
- On the Security Config page of Elastic Runtime, after you supply your own cipher sets and click Save, do not erase the HAProxy and Router Cipher fields. The ciphers will not return to their default values when deleted, and will instead be interpreted as an empty cipher set.
- When selecting between internal and external System Database and/or File Storage config, if saved values for external systems fail verification (e.g. a host is not reachable from Ops Manager), the values will persist if you then select ‘Internal Databases’ or ‘Internal File Store’. To resolve this issue, return to your Ops Manager Installation Dashboard and click Revert, located in the upper right corner of the page.
Pivotal Cloud Foundry Ops Manager v1.7.0 Known Issues

New Issues

Some of these issues may be fixed in subsequent patch releases to v1.7. Consult the Ops Manager v1.7 Release Notes for more information.

- Ops Manager v1.7.0 limits the integration with products like CA Single Sign-On, in that a single BOSH Director and Ops Manager can be registered. This limitation will be addressed in v1.7.1.
- Ops Manager uses new instance sizes on vSphere that may result in higher capacity VMs being deployed.
- The initial setup of Ops Manager should be done using a FQDN via DNS as the UAA will use the address for redirects. If the initial setup of Ops Manager is performed by accessing Ops Manager through an IP address or URL that you do not control, that address must not change.
- Ops Manager director is now limited to one network to avoid asymmetric routing. Ops Manager will assign the director to the deployment network unless the user limits the director to one network before upgrading. BOSH can be multi-homed manually. Please file a support ticket if you need help.

Existing Issues

- If EBS encryption is initially disabled, and then later enabled, the change will not apply to the Director's persistent disk.
- If you are using OpenStack Juno, make sure you have applied this patch: https://bugs.launchpad.net/horizon/+bug/1394051

On-Demand Services Require Dedicated Service Networks

If you use any service tile that offers both on-demand and not on-demand modes of operation, clicking Apply Changes in Ops Manager fails if you did not define a dedicated service network for the tile.

To work around this issue, use one of the following methods:

- Create a services network on your IaaS for each affected service tile
- Create a dummy network in Ops Manager, reserve a block of IP ranges, and disable smoke tests for the on-demand service

For more information, see the corresponding Knowledge Base article.