



PRODUCT DOCUMENTATION

Pivotal Cloud Foundry[®]

Version 1.6

© 2018 Pivotal Software, Inc.

Table of Contents

Table of Contents	2
Pivotal Cloud Foundry Documentation	6
Installing Pivotal Cloud Foundry	7
Prerequisites to Deploying Operations Manager and Elastic Runtime	8
Preparing Your Firewall for Deploying Pivotal Cloud Foundry	11
Installing Pivotal Cloud Foundry on AWS	13
Installing Pivotal Cloud Foundry on OpenStack	14
Installing Pivotal Cloud Foundry on vSphere and vCloud Air	16
Using Operations Manager	20
Identifying the API Endpoint for your Elastic Runtime Instance	22
Understanding the Ops Manager Interface	23
Adding and Deleting Products	25
Using Your Own Load Balancer	27
Configuring SSH Access for PCF	31
Starting and Stopping Pivotal Cloud Foundry Virtual Machines	33
Creating New Elastic Runtime User Accounts	36
Logging into the Apps Manager	37
Controlling Apps Manager User Activity with Environment Variables	38
Configuring Your App Autoscaling Instance	40
Managing Scheduled Scaling in the App Autoscaling Service	44
Adding Existing LDAP Users to a Pivotal Cloud Foundry Deployment	48
Backing Up Pivotal Cloud Foundry	51
Restoring Pivotal Cloud Foundry from Backup	58
Upgrading Operations Manager	65
Upgrading Elastic Runtime and Other Pivotal Cloud Foundry Products	69
Monitoring Virtual Machines in Pivotal Cloud Foundry	72
Deploying Pivotal Ops Metrics	74
Using SSL with a Self-Signed Certificate in JMX Bridge	80
Using Pivotal Ops Metrics	83
Troubleshooting and Uninstalling Pivotal Ops Metrics	85
Pivotal Cloud Foundry Troubleshooting Guide	87
Troubleshooting Ops Manager for VMware vSphere	94
Recovering MySQL from Elastic Runtime Downtime	97
Advanced Troubleshooting with the BOSH CLI	101
Pivotal Cloud Foundry Security Overview and Policy	107
Cloud Foundry Concepts	109
Cloud Foundry Subsystems Overview	110
How Applications Are Staged	112
Zero Downtime Deployment and Scaling in CF	113
Orgs, Spaces, Roles, and Permissions	117
Understanding Cloud Foundry Security	120
Stacks	125
Using Docker in Cloud Foundry	127
Glossary	129
Understanding Application Security Groups	130
Cloud Foundry Components	137
Cloud Controller	140
Droplet Execution Agent	145

Messaging (NATS)	148
(Go)Router	150
User Account and Authentication (UAA) Server	152
Warden	158
Diego Architecture	164
Differences Between DEA and Diego Architectures	169
Understanding Application SSH	172
How the Diego Auction Allocates Jobs	173
Operator's Guide	176
Isolating a Pivotal Cloud Foundry Deployment with a Private Network	177
Rotating Elastic Runtime MySQL Credentials	180
Understanding the Elastic Runtime Network Architecture	182
Load Balancer	182
Router	182
Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments	183
Changing the Quota Plan of an Organization with cf CLI	185
Restricting App Access to Internal PCF Components	188
Identifying Elastic Runtime Jobs Using vCenter	192
Understanding the Effects of Single Components on a Pivotal Cloud Foundry Upgrade	194
Configuring Single Sign-On	196
Configuring LDAP	199
Switching Application Domains	202
Scaling Instances in Elastic Runtime	205
Monitoring App and Service Instance Usage	209
Using Diego in Pivotal Cloud Foundry	213
Deploying Diego for Windows	215
Troubleshooting Diego for Windows	223
The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment	227
Providing a Certificate for your SSL Termination Point	234
Administering and Operating Cloud Foundry	236
Managing Custom Buildpacks	237
Adding a Custom Stack	239
Creating and Managing Users with the cf CLI	241
Creating and Managing Users with the UAA CLI (UAAC)	243
Getting Started with the Notifications Service	247
Feature Flags	249
Migrating Apps to Diego	250
Using the Apps Manager	256
Getting Started with the Apps Manager	257
Understanding Apps Manager Permissions	258
Managing Spaces Using the Apps Manager	259
Managing User Accounts in Spaces Using the Apps Manager	261
Managing User Permissions Using the Apps Manager	263
cf Command Line Interface (CLI)	265
Installing the cf Command Line Interface	266
Getting Started with the cf CLI	269
Using the cf CLI with an HTTP Proxy Server	274
Using the cf CLI with a Self-Signed Certificate	277
Using cf CLI Plugins	278
Developing cf CLI Plugins	280

About Starting Applications	281
Developer Guide	282
Considerations for Designing and Running an Application in the Cloud	283
Understanding Application Deployment	286
Deploy an Application	287
Deploying a Large Application	291
Deploying with Application Manifests	293
Scaling an Application Using cf scale	303
Cloud Foundry Environment Variables	304
Using Blue-Green Deployment to Reduce Downtime and Risk	311
Application Logging in Cloud Foundry	314
Troubleshooting Application Deployment and Health	317
Application SSH Overview	322
Accessing Apps with SSH	324
Accessing Services with SSH	329
Trusted System Certificates	331
Services Overview	332
Managing Service Instances with the CLI	334
Managing Service Keys	337
Delivering Service Credentials to an Application	339
User-Provided Service Instances	341
Using Log Management Services	343
Service-Specific Instructions for Streaming Application Logs	345
Streaming Application Logs to Splunk	350
Configuring Play Framework Service Connections	352
Migrating a Database in Cloud Foundry	353
Buildpacks	355
Buildpack Detection	357
Custom Buildpacks	358
Packaging Dependencies for Offline Buildpacks	362
Supported Binary Dependencies	365
Java Buildpack	366
Getting Started Deploying Grails Apps	367
Getting Started Deploying Ratpack Apps	374
Getting Started Deploying Spring Apps	381
Tips for Java Developers	387
Configure Service Connections for Grails	393
Configure Service Connections for Play Framework	396
Configure Service Connections for Spring	397
Cloud Foundry Eclipse Plugin	405
Cloud Foundry Java Client Library	424
Build Tool Integration	426
Ruby Buildpack	430
Getting Started Deploying Ruby Apps	431
Getting Started Deploying Ruby on Rails Apps	436
Deploy a Sample Ruby on Rails Application	438
Configure a Production Server for Ruby Apps	440
Configure Rake Tasks for Deployed Apps	441
Tips for Ruby Developers	442
Environment Variables Defined by the Ruby Buildpack	448

Configure Service Connections for Ruby	449
Node.js Buildpack	452
Tips for Node.js Applications	453
Environment Variables Defined by the Node Buildpack	456
Configure Service Connections for Node.js	457
Binary Buildpack	459
Go Buildpack	461
PHP Buildpack	464
Composer	465
New Relic	467
PHP Buildpack Configuration	468
Deploying and Developing PHP Apps	470
Tips for PHP Developers	473
Python Buildpack	474
Staticfile Buildpack	475
Configuration	475
Services	477
Overview	478
Service Broker API v2.7	479
Managing Service Brokers	494
Access Control	498
Catalog Metadata	504
Dashboard Single Sign-On	508
Example Service Brokers	512
Binding Credentials	513
Application Log Streaming	515
Route Services	516
Supporting Multiple Cloud Foundry Instances	521
Logging and Metrics	522
Overview of the Loggregator System	523
Using Loggregator	523
Loggregator Components	523
Loggregator Guide for Cloud Foundry Operators	525
Cloud Foundry System Metrics	526
Deploying a Nozzle to the Loggregator Firehose	542
Cloud Foundry Data Sources	547
Installing the Loggregator Firehose Plugin for cf CLI	548
Routing	549
Pivotal Cloud Foundry Release Notes and Known Issues	550
Pivotal Elastic Runtime v1.6.0.0 Release Notes	551
Pivotal Cloud Foundry Ops Manager v1.6 Release Notes	586
Pivotal Cloud Foundry Ops Metrics v1.6.X Release Notes	592
Pivotal Cloud Foundry Metrics Release Notes	594
Pivotal Elastic Runtime v1.6 Known Issues	598
Pivotal Cloud Foundry Ops Manager v1.6.0 Known Issues	600
Pivotal Cloud Foundry Ops Metrics v1.6.X Known Issues	601
Pivotal Cloud Foundry Metrics Release Notes	602

Pivotal Cloud Foundry Documentation

- [Installing Pivotal Cloud Foundry](#)
A quick guide to installing [Pivotal Cloud Foundry](#) (PCF).
- [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.
- [Administering Elastic Runtime](#)
A guide to managing Elastic Runtime at the administrator level.
- [Using the 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.
- [Routing](#)
A guide to using routing in Elastic Runtime.
- [Release Notes](#)
Release notes and known issues for Pivotal Operations Manager, Pivotal Elastic Runtime, and Pivotal MySQL Developer.

Installing Pivotal Cloud Foundry

 **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.

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.

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

If you experience a problem while following the steps below, refer to the [Known Issues](#), or 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](#)

Install Pivotal Cloud Foundry:

- [Installing Pivotal Cloud Foundry on AWS](#)
- [Installing Pivotal Cloud Foundry on OpenStack](#)
- [Installing Pivotal Cloud Foundry on vSphere and vCloud Air](#)

Prerequisites to Deploying Operations Manager and Elastic Runtime

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 explains system requirements for deploying the Pivotal Operations Manager and Elastic Runtime applications.

vSphere/vCenter Requirements

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.
- 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 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. 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:** If you are using the Cisco Nexus 1000v Switch, refer to the [Using the Cisco Nexus 1000v Switch with Ops Manager](#) topic for more information.

 **Note:** When installing Ops Manager on a vSphere environment with multiple ESXi hosts, you must use network-attached or shared storage devices. Local storage devices do not support sharing across multiple ESXi hosts.

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: 120GB
- Memory: 60GB
- Two public IP addresses: One for Elastic Runtime and one for Ops Manager
- vCPU cores: 28
- Overall CPU: 17 GHz
- Virtual infrastructure administrator privileges to enable Elastic Runtime to automatically power VMs on and off

 **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.large instance with 100 GB of allocated storage.
- 2 S3 Buckets
- EC2 Instances:
 - 10 t2.micros
 - 1 r3.xlarge (1 per DEA)
 - 1 c4.xlarge
 - 2 m3.large

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.

General Requirements

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

- The following user privileges:
 - Datastore (Allocate space, Browse datastore, Low-level file operations, Remove file, Update virtual machine files)
 - Folder (All)
 - Network (Assign network)
 - Resource (All)
 - vApp (All)
 - Virtual machine (All)
- (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: 172.16.64.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.

 **Note:** If you have DHCP, refer to the Troubleshooting Guide to avoid issues with your installation.

- 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`

 **Note:** 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.

- The cf command line interface (cf CLI) tool version 6.1.1 or higher.
- One or more NTP servers.

- Capacity for the following virtual machines:

Virtual Machine	Instances	CPU	RAM (MB)	Ephemeral Disk (MB)	Persistent Disk (MB)	Static IP
Message Bus (NATS)	1	1	1024	1024	0	✓
consul	1	1	1024	1024	1024	✓
etcd	1	1	1024	1024	1024	✓
Blob Store (NFS Server)	1	1	1024	2048	10240	✓
Cloud Controller Database	1	1	1024	2048	2048	✓
OAuth2 Server Database (UAA Database)	1	1	1024	2048	8192	✓
Apps Manager Database	1	1	1024	2048	1024	✓
Cloud Controller	1	1	4096	20480	0	
HAProxy	0	1	1024	2048	0	✓
Router	1	1	1024	2048	0	✓
Health Manager	1	1	1024	1024	0	
Clock Global	1	1	1024	2048	0	
Cloud Controller Worker	1	1	1024	2048	0	
Collector	0	1	1024	2048	0	
OAuth2 Server (UAA)	1	1	1024	2048	0	
MySQL Proxy	0	1	1024	4096	0	✓
MySQL Server	1	1	4096	10000	10000	✓
MySQL Backup Service	0	1	4096	20000	0	
Application Execution (DEA)	1	1	4096	10240	0	
Doppler Server	1	1	1024	2048	0	✓
Loggregator Trafficcontroller	1	1	1024	2048	0	✓
Push Apps Manager	1	1	1024	1024	0	
Push App Usage Service	1	1	1024	1024	0	
Run Smoke Tests	1	1	1024	1024	0	
Notifications with UI	1	1	1024	2048	0	
Deploy CF Autoscaling App	1	1	512	1024	0	
Register Autoscaling Service Broker	1	1	512	1024	0	
Destroy Autoscaling Service Broker	1	1	512	1024	0	
Run CF Acceptance Tests	1	1	512	1024	0	
Run CF Acceptance Tests without internet	1	1	1024	1024	0	
Compilation	1	1	1024	20480	0	
TOTALS	27	30	37 GB	102 GB	33 GB	13

[Return to the Installing PCF Guide](#) ↵

Preparing Your Firewall for Deploying Pivotal Cloud Foundry

Page last updated:

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 HAProxy or, if configured, your own load balancer
- **80**: Routes to HAProxy 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 iptables commands to configure a firewall.

 **Note:** `GATEWAY_EXTERNAL_IP` is a placeholder. Replace this value with your `PUBLIC_IP`.

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 want to remove all existing filtering or Network Address Translation (NAT) rules, run the following commands:

```
$ 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 GATEWAY_EXTERNAL_IP=88.198.252.242  
$ 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_INTERNAL_IP  
$ iptables -t nat -A POSTROUTING -d $SHA_PROXY_IP -s $INTERNAL_NETWORK_RANGE \  
-p tcp --dport 443 -j SNAT --to $GATEWAY_INTERNAL_IP
```

- **PREROUTING:**

```
$ iptables -t nat -A PREROUTING -d $GATEWAY_EXTERNAL_IP -p tcp --dport \
  25555 -j DNAT --to $PIVOTALCF_IP
$ iptables -t nat -A PREROUTING -d $GATEWAY_EXTERNAL_IP -p tcp --dport \
  443 -j DNAT --to $HA_PROXY_IP
$ iptables -t nat -A PREROUTING -d $GATEWAY_EXTERNAL_IP -p tcp --dport \
  80 -j DNAT --to $HA_PROXY_IP
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8443 -j DNAT \
  --to $PIVOTALCF_IP:443
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT \
  --to $HA_PROXY_IP:80
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8022 -j DNAT \
  --to $PIVOTALCF_IP:22
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8080 -j DNAT \
  --to $PIVOTALCF_IP:80
```

6. Run the following command to save the iptables:

```
$ service iptables save
```

For more information about administering IP tables with `iptables`, refer to the [iptables documentation](#).

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` 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:

- o `nslookup`
- o `dig`
- o `host`
- o 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.

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.

 **Note:** If you are performing an upgrade from PCF 1.5 to 1.6, please review [Upgrading Ops Manager](#) for critical upgrade information.

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.

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. [Deploying Elastic Runtime on AWS](#)

Installing Pivotal Cloud Foundry on OpenStack

Page last updated:

This guide describes how to install [Pivotal Cloud Foundry](#) (PCF) on OpenStack.

Complete the following procedures to install PCF on OpenStack:

1. [Provisioning the OpenStack Infrastructure](#)
2. [Configuring Ops Manager Director after Deploying PCF on OpenStack](#)
3. [Installing Elastic Runtime after Deploying PCF on OpenStack](#)

Supported Versions

Pivotal's automated testing environments currently run on Mirantis OpenStack 6.1 (Juno). Pivotal Cloud Foundry has also been installed on OpenStack releases and distributions based on Havana, Icehouse, Juno, and Kilo (Keystone v2) from different vendors including Canonical, EMC, 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).
- 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.
- Your OpenStack project must have the following resources before you install Pivotal Cloud Foundry:
 - 70 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 root disk)
 - 32 vCPUs
 - 1 TB of storage
 - Neutron networking with floating IP support
- 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

 **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:

- 70 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 root disk)
- 32 vCPUs
- 1 TB of storage
- Neutron networking with floating IP support

• 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

20GB volumes within 300 seconds.

- Miscellaneous:

- If any 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.
- 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:

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

ID	Name	Memory_MB	Disk	Ephemeral	VCPUs
1	m1.small	2048	20	0	1
2	m1.medium	4096	40	0	2
3	m1.large	8192	80	0	4
4	m1.xlarge	16384	160	0	8

Installing Pivotal Cloud Foundry on vSphere 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 and vCloud Air. It will walk you through how to deploy the installation virtual machine known as Ops Manager, configure your vSphere settings, and install PCF Elastic Runtime.

If you experience a problem while following the steps below, check the [Known Issues](#), or refer to the [Pivotal Cloud Foundry 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](#) (PCF) and the Pivotal product suite.

Note: If you are performing an upgrade from PCF 1.5 to 1.6, see [Upgrading Ops Manager](#) for critical upgrade information.

Step 1: Confirm System Requirements

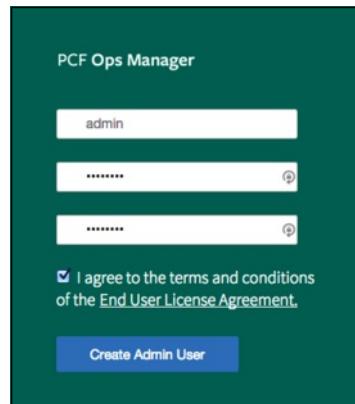
Before you begin your PCF deployment, ensure that your system meets the minimum requirements. The requirements include capacity for the virtual machines necessary for the deployment, a supported version of the cf command line interface tool, and certain user privileges. Refer to the [Prerequisites to Deploying Operations Manager and Elastic Runtime](#) topic for the complete list.

Step 2: Deploy PCF Ops Manager

1. Log in to [Pivotal Network](#) and download the Ops Manager virtual appliance file that matches your infrastructure.
2. Deploy Ops Manager using the procedure that corresponds to the VMware environment that you use:
 - [Deploying Operations Manager to vSphere](#)
 - [Deploying Operations Manager to vCloud Air and vCloud](#)

Step 3: Configure and Install PCF

Goal: Configure and install Ops Manager Director (included), Elastic Runtime, and MySQL for PCF. Skip the import steps for Elastic Runtime or MySQL for PCF if you do not want to import these products.

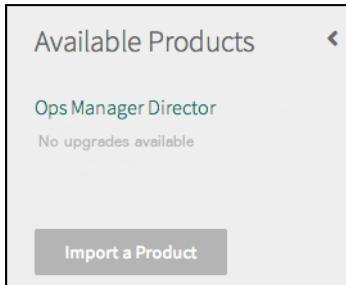


1. Use the public IP address of your Ops Manager instance to launch Ops Manager in a browser.
2. Enter a **Username** and a **Password**.
3. Read the **End User License Agreement**, and select the checkbox to accept.
4. Click **Create Admin User** to create an Ops Manager administrator account.

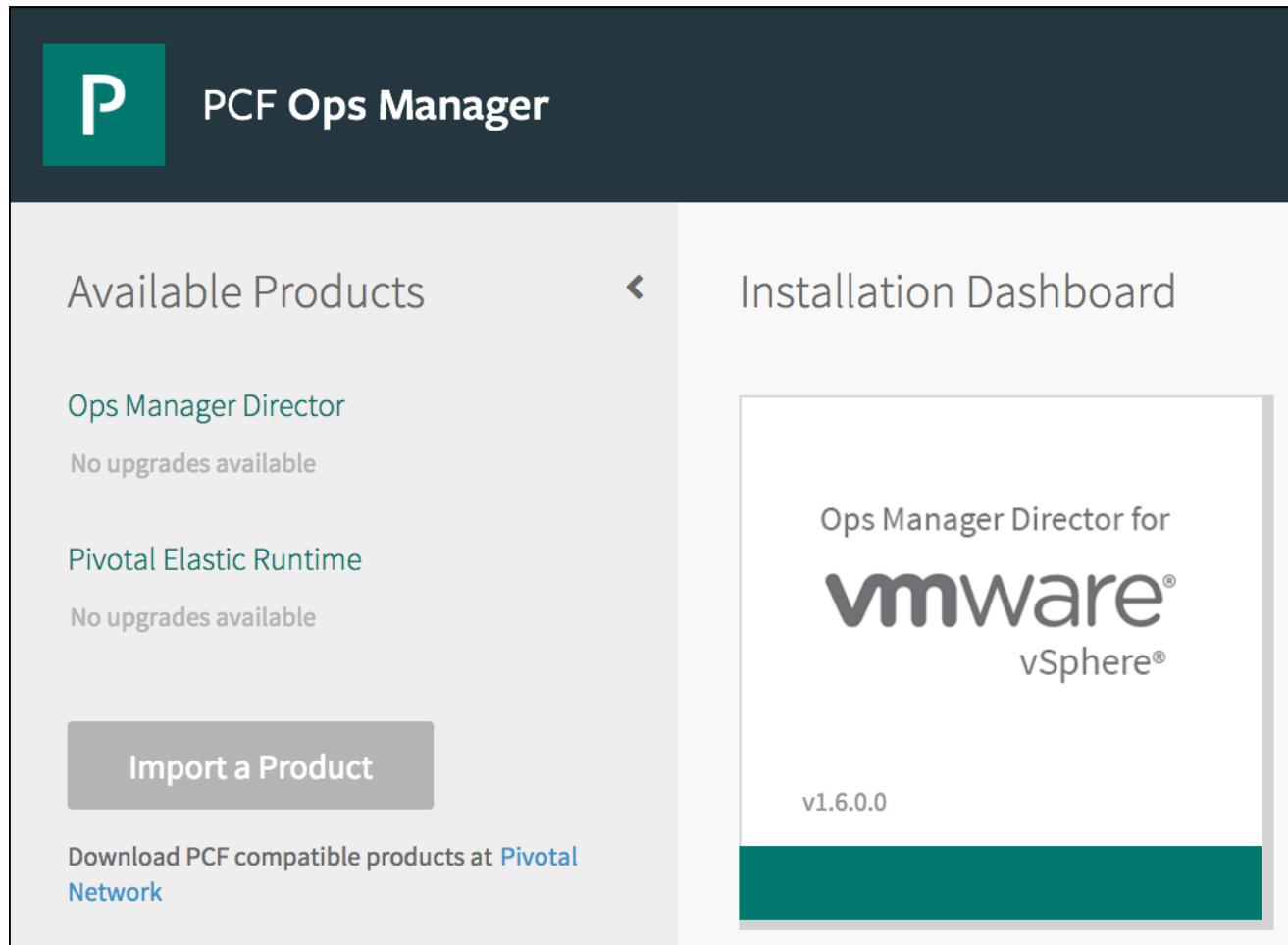
 **Note:** On your first login attempt, an error message that the connection is untrusted appears because you are attempting to connect securely to a website with a self-signed certificate. Add Ops Manager as an exception to bypass this message on subsequent logins.

Import Products

1. Download Elastic Runtime and MySQL for PCF at [Pivotal Network](#).
 - o Elastic Runtime: Click the **Pivotal Cloud Foundry** banner to access the PCF product page. Use the drop-down menu to select an Elastic Runtime release.
 - o MySQL for PCF: Select the **All Products** tab to view available products. Search for “MySQL for PCF” and select a release.
2. From the Available Products view, click **Import a Product**.



3. Select the Elastic Runtime .pivotal file that you downloaded from Pivotal Network and click **Open**. After the import completes, Elastic Runtime appears in the Available Products view.
4. Repeat the previous step for MySQL for PCF.
5. In the Available Products view, hover over Elastic Runtime and click **Add**. Repeat this step for MySQL for PCF.



PCF Ops Manager

Available Products

- Ops Manager Director
No upgrades available
- Pivotal Elastic Runtime
No upgrades available

Import a Product

Download PCF compatible products at [Pivotal Network](#)

Installation Dashboard

Ops Manager Director for
vmware[®]
vSphere[®]

v1.6.0.0

Configure Ops Manager Director

Your Ops Manager download includes a tile for the version of Ops Manager Director that corresponds to your IaaS. Refer to one of the following topics for help configuring your Ops Manager Director product tile:

- [Configuring Ops Manager Director for VMware vSphere ↗](#)
- [Configuring Ops Manager Director for vCloud Air and vCloud ↗](#)

Configure Elastic Runtime

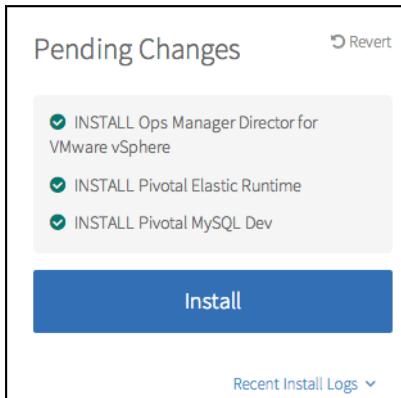
See [Configuring Elastic Runtime for vSphere and vCloud Air ↗](#).

Configure MySQL

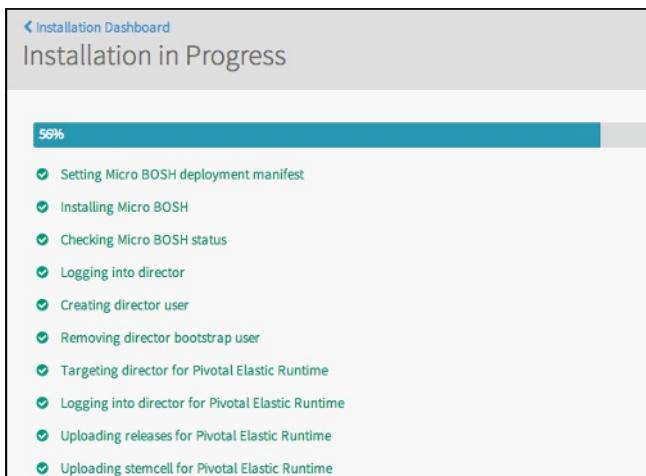
1. Click the **MySQL Dev** tile.
2. Configure the MySQL Service Plan, Lifecycle Errands, and resource sizes. Assign Availability Zones and an Ops Manager network, then click **Save**.
3. Click the **Installation Dashboard** link to return to the Installation Dashboard.

Install Products

1. Click **Install**.



2. Your updated PCF installation begins deploying.



3. When the deployment is finished, a success message appears.

 **Note:** On the recommended hardware infrastructure, deployment should take less than one hour and require no user intervention.

You now have a fully functional installation of PCF and Pivotal MySQL Service.

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.

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 Pivotal 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.

Using Operations Manager

 **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.

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>.

 **Warning:** The Ops Manager API is not officially supported and could change without notice.

- [Identifying the API Endpoint for your Elastic Runtime Instance](#)

Using Operations Manager and Installed Products

- [Understanding the Ops Manager Interface](#)
- [Adding and Deleting Products](#)
- [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](#)
- [Configuring Ops Manager Director for OpenStack](#)
- [Deploying Elastic Runtime on AWS](#)
- [Configuring Elastic Runtime for vSphere and vCloud](#)
- [Installing Elastic Runtime after Deploying Pivotal Cloud Foundry on OpenStack](#)
- [Using Your Own Load Balancer](#)
- [Configuring SSH Access for PCF](#)
- [Starting and Stopping Pivotal Cloud Foundry Virtual Machines](#)
- [Creating New Elastic Runtime 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](#)
- [Adding Existing LDAP Users to a Pivotal Cloud Foundry Deployment](#)
- [Deleting an AWS Installation from the Console](#)

Backing Up and Upgrading

- [Backing Up Pivotal Cloud Foundry](#)
- [Restoring Pivotal Cloud Foundry from Backup](#)

- [Upgrading Operations Manager](#)
- [Creating a Proxy ELB for Diego SSH without CloudFormation ↗](#)
- [Upgrading Elastic Runtime and Other Pivotal Cloud Foundry Products](#)

Monitoring, Logging, and Troubleshooting

- [Monitoring Virtual Machines in Pivotal Cloud Foundry](#)
- [Deploying JMX Bridge](#)
- [Using SSL with a Self-Signed Certificate in JMX Bridge](#)
- [Using JMX Bridge](#)
- [Troubleshooting and Uninstalling JMX Bridge](#)
- [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](#)

Identifying the API Endpoint for your Elastic Runtime Instance

Page last updated:

The API endpoint, or target URL, for your Elastic Runtime instance is the URL of the Cloud Controller in your Elastic Runtime instance. Find your 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.

GETTING STARTED

```
$ cf help
$ cf login -a https://api.your_endpoint.com
API endpoint: https://api.your_endpoint.com
Username> your_username
Password> your_password
Org> your_org
Space> your_space
$ cf push your_app
```

From the Command Line

From a command line, use the `cf api` command to view your API endpoint.

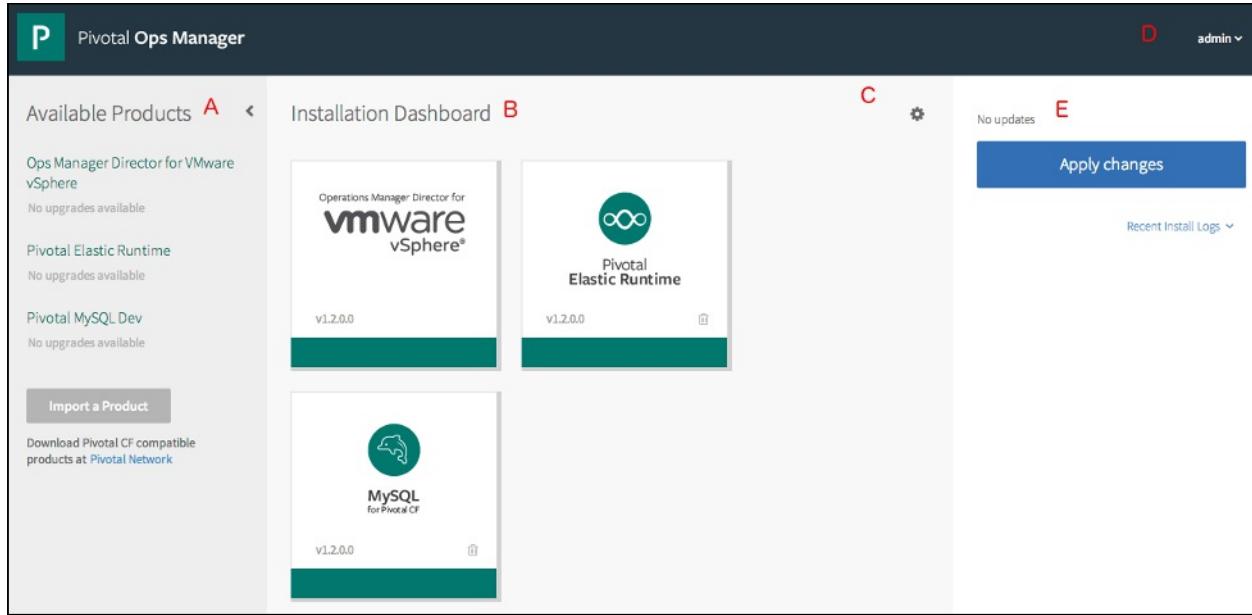
Example:

```
$ cf api
API endpoint: https://api.example.com (API version: 2.2.0)
```

Understanding the Ops Manager Interface

Page last updated:

This topic describes key features of the [Pivotal Cloud Foundry](#) (PCF) Operations Manager interface.



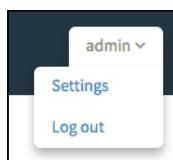
PCF Ops Manager

- A—Available Products:** Displays a list of products you have imported that are ready for installation. Click the **Import a Product** link to add a new product to Ops Manager.
- 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.
 - Import installation settings:** Navigates to the **Import installation settings** view. Select this option to import an existing PCF installation with all of its assets. When you import an installation, the prior installation disappears and is replaced by the newly imported one.
 - Export installation 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. Click on **Settings** to access the password change screen.
- E—Pending Changes view:** Displays queued installations and updates that will install during the next deploy.

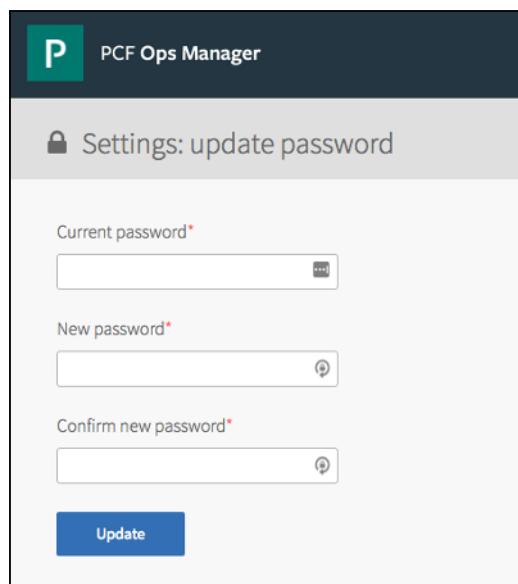
Note: When an update depends on prerequisites, the prerequisites automatically install first.

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 password associated with your account:



The screenshot shows a password update form within the PCF Ops Manager interface. The title bar reads "PCF Ops Manager". The main heading is "Settings: update password". The form contains three text input fields: "Current password*" (with a red asterisk indicating it is required), "New password*" (with a red asterisk), and "Confirm new password*" (with a red asterisk). Each input field has a small "..." icon to its right. Below the input fields is a blue "Update" button. The entire form is set against a light gray background.

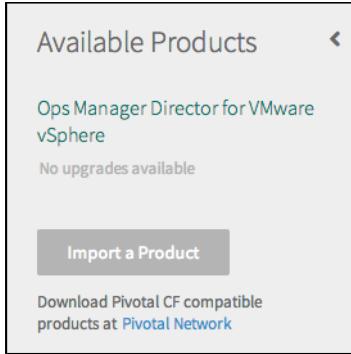
Adding and Deleting Products

Page last updated:

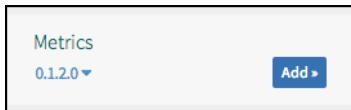
Refer to this topic for help adding and deleting additional products from your [Pivotal Cloud Foundry](#) (PCF) installation, such as [Pivotal RabbitMQ® for PCF](#).

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 Available Products view to expose the **Add** button, then click **Add**.



5. The product tile appears in the Installation Dashboard. If the product requires configuration, the tile appears orange.



If necessary, configure the product.

6. **(Optional)** In the product configuration view, select the **Lifecycle Errands** tab 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](#).

Note: By default, Ops Manager reruns lifecycle errands even if they are not necessary due to settings left from a previous install. Leaving errands checked at all times can cause updates and other processes to take longer. To prevent a lifecycle errand from running, deselect the checkbox for the errand in the **Settings** tab on the product tile before installing the product.

Installation Dashboard

Redis for Pivotal CF

Settings Status Credentials Logs

Assign Networks

Redis

Lifecycle Errands

Resource Config

Lifecycle Errands

Lifecycle errands are scripts that run at designated points during an installation.

Broker Registrar

Post-Install Errands

Save

The **Broker Registrar** checkbox is an example of a lifecycle 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.

7. 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.

Using Your Own Load Balancer

Page last updated:

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

 **Note:** Application logging with [Loggregator](#) requires WebSockets. To use another logging service, see the [Using Third-Party Log Management Services](#) topic.

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 IPs

1. In Pivotal Operations Manager, click the **Elastic Runtime** tile.
2. In the left column, select **IPs and Ports** if not already selected.
3. In the **HAProxy IPs** field, delete any existing IP addresses. This field should be blank.



4. Select **Router IPs**.

5. In the **Router IPs** field, enter the IP address or addresses for PCF that you registered with your load balancer in Step 2. Save this change.



Step 5: Configure Security Settings

1. Click on **Security Config**.

2. Provide an **SSL Termination Certificate** for your SSL Termination Point.

- In a production environment, use a signed **SSL Certificate** from a known certificate authority (CA). Copy and paste the values for **Certificate PEM** and **Private Key PEM** from the signed certificate into the appropriate text fields.
 - In a development or testing environment, you may use a self-signed certificate with the following steps:

 Note: Pivotal does not recommend using a self-signed certificate for production deployments.

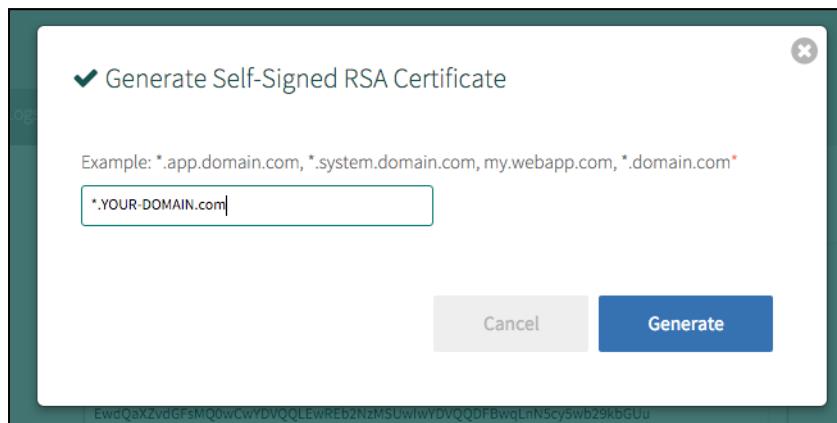
Note: Certificates generated in Elastic Runtime are signed by the Operations Manager Certificate Authority. They are not technically self-signed, but they are referred to as 'Self-Signed Certificates' in the Ops Manager GUI and throughout this documentation.

- Click the **Generate Self-Signed RSA Certificate** link to launch the RSA certificate wizard.
 - Enter your system and app domains in wildcard format. Optionally, also add custom domains in wildcard format. The example below uses `*.YOUR-DOMAIN.com`.

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.

Note: You can generate a single certificate for two domains separated by a comma. For example, `*.apps.YOUR-DOMAIN.com, *.system.YOUR-DOMAIN.com`.



- Click **Generate**.
- Elastic Runtime populates the **SSL Certificate** fields with RSA certificate and private key information.

- Configure **Ignore SSL certificate verification**. Select this option if you are using self-signed certificates or certificates generated from Ops Manager.
- Configure **HAProxy SSL Ciphers** and **Router SSL Ciphers**. Leave these fields blank unless you want to use a specific set of SSL ciphers for HAProxy or the Router.
- Configure **Disable HTTP traffic to HAProxy**. If you select the **Disable HTTP traffic to HAProxy** checkbox, your deployment rejects all port 80 traffic to HAProxy. Additionally, this option sets the secure flag in the `VCAP_ID` cookie that the Router generates.

Note: If you enable this checkbox and your deployment is not using HAProxy, configure your external load balancer to reject port 80 traffic. If you do not do this, traffic to port 80 is forwarded to applications but loses session stickiness.

- Configure **Enable cross container traffic**. Select this checkbox to disable the restriction that prevents containers in the same DEA or Diego Cell from communicating back to the host interface. Pivotal does not recommend selecting this checkbox.
- Configure **Enable TLS on the Router**. Select this setting to enable [SSL termination](#) on the Router.
- Click **Save**.

Step 6: Finalize Changes

- Return to the [Ops Manager Installation Dashboard](#)
- Click **Install**.

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 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 running on Diego:

1. Open the **Pivotal Elastic Runtime** tile in Ops Manager.
2. Under the **Settings** tab, select the **Diego** section.
3. Enable or disable the **Allow SSH access to apps** checkbox.

Pivotal Elastic Runtime

Settings Status Credentials Logs

Diego is a new replacement for DEAs.

Assign Networks

Assign Availability Zones

System Database Config

File Storage Config

IPs and Ports

Security Config

MySQL Proxy Config

Cloud Controller

System Logging

SSO Config

LDAP Config

SMTP Config

Diego

Use Diego by default instead of DEAs

Allow SSH access to apps

Application Containers Subnet Pool *

10.254.0.0/22

Save

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 `10.244.0.34.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 Diego 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's `access_zN` 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.

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.

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`:

```
o → bosh vms
Acting as user 'director' on 'p-bosh-399383d4525762cdc35c'
Deployment 'cf-1ef2da789c0ed8f3567f

Director task 26

Task 26 done

+-----+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+-----+
| clock_global-partition-458f9d7042365ff810e9/0 | running | clock_global-partition-458f9d7042365ff810e9 | 10.0.16.14 |
| cloud_controller-partition-458f9d7042365ff810e9/0 | running | cloud_controller-partition-458f9d7042365ff810e9 | 10.0.16.13 |
| cloud_controller_worker-partition-458f9d7042365ff810e9/0 | running | cloud_controller_worker-partition-458f9d7042365ff810e9 | 10.0.16.15 |
| consul_server-partition-458f9d7042365ff810e9/0 | running | consul_server-partition-458f9d7042365ff810e9 | 10.0.16.6 |
| diego_brain-partition-458f9d7042365ff810e9/0 | running | diego_brain-partition-458f9d7042365ff810e9 | 10.0.16.17 |
| diego_cell-partition-458f9d7042365ff810e9/0 | running | diego_cell-partition-458f9d7042365ff810e9 | 10.0.16.18 |
| diego_cell-partition-458f9d7042365ff810e9/1 | running | diego_cell-partition-458f9d7042365ff810e9 | 10.0.16.19 |
| diego_cell-partition-458f9d7042365ff810e9/2 | running | diego_cell-partition-458f9d7042365ff810e9 | 10.0.16.20 |
| diego_database-partition-458f9d7042365ff810e9/0 | running | diego_database-partition-458f9d7042365ff810e9 | 10.0.16.9 |
| doppler-partition-458f9d7042365ff810e9/0 | running | doppler-partition-458f9d7042365ff810e9 | 10.0.16.21 |
| etcd_server-partition-458f9d7042365ff810e9/0 | running | etcd_server-partition-458f9d7042365ff810e9 | 10.0.16.8 |
| loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0 | running | loggregator_trafficcontroller-partition-458f9d7042365ff810e9 | 10.0.16.22 |
| mysql-partition-458f9d7042365ff810e9/0 | running | mysql-partition-458f9d7042365ff810e9 | 10.0.16.12 |
| mysql_proxy-partition-458f9d7042365ff810e9/0 | running | mysql_proxy-partition-458f9d7042365ff810e9 | 10.0.16.11 |
| nats-partition-458f9d7042365ff810e9/0 | running | nats-partition-458f9d7042365ff810e9 | 10.0.16.7 |
| router-partition-458f9d7042365ff810e9/0 | running | router-partition-458f9d7042365ff810e9 | 10.0.16.10 |
| uaa-partition-458f9d7042365ff810e9/0 | running | uaa-partition-458f9d7042365ff810e9 | 10.0.16.16 |
+-----+-----+-----+-----+
VMs total: 17
```

You can see the full name of each VM in the `Job/index` 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.

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 three Diego Cell VMs is `diego_cell-partition-458f9d7042365ff810e9/2`.

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](#), without 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-458f9d7042365ff810e9`:

```
$ bosh start nats-partition-458f9d7042365ff810e9
Processing deployment manifest
-----
You are about to start nats-partition-458f9d7042365ff810e9/0

Detecting deployment changes
-----
Start nats-partition-458f9d7042365ff810e9/0? (type 'yes' to continue): yes

Performing `start nats-partition-458f9d7042365ff810e9/0'...
...
Started updating job nats-partition-458f9d7042365ff810e9 > nats-partition-458f9d7042365ff810e9/0 (canary). Done (00:00:43)

Task 42 done

nats-partition-458f9d7042365ff810e9/0 has been started
```

Note: To start a specific instance of a VM, include the `/INDEX` 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-458f9d7042365ff810e9/0
```

Start Order	Component	Job/index name prefix (in <code>bosh vms</code> output)
1	NATS	<code>nats-</code>
2	consul	<code>consul_server-</code>
3	etcd	<code>etcd_server-</code>
4	Diego Database	<code>diego_database-</code>
5	NFS Server	<code>nfs_server-</code>
6	Router	<code>router-</code>
7	MySQL Proxy	<code>mysql_proxy-</code>
8	MySQL Server	<code>mysql-</code>
9	Cloud Controller Database	<code>ccdb-</code>
10	UAA Database	<code>uaadb-</code>
11	Apps Manager Database	<code>consoledb-</code>
12	Cloud Controller	<code>cloud_controller-</code>
13	HAProxy	<code>ha_proxy-</code>
14	Health Manager	<code>health_manager-</code>
15	Clock Global	<code>clock_global-</code>
16	Cloud Controller Worker	<code>cloud_controller_worker-</code>
17	Collector	<code>collector-</code>
18	UAA	<code>uaa-</code>
19	Diego Brain	<code>diego_brain-</code>
20	Diego Cell	<code>diego_cell-</code>
21	DEA	<code>dea-</code>
22	Doppler Server	<code>doppler-</code>
23	Loggregator Traffic Controller	<code>loggregator_trafficcontroller-</code>

Stopping PCF Virtual Machines

In the order specified in the [Stop Order table](#) below, run `bosh stop 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](#), without the `/INDEX` 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-458f9d7042365ff810e9`:

```
$ bosh stop loggregator_trafficcontroller-partition-458f9d7042365ff810e9

Processing deployment manifest
-----
You are about to stop loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0

Detecting deployment changes
-----
Stop loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0? (type 'yes' to continue): yes

Performing `stop loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0'...

...
Started updating job loggregator_trafficcontroller-partition-458f9d7042365ff810e9 > loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0 (canary). Done (00:00:37)

loggregator_trafficcontroller-partition-458f9d7042365ff810e9/0 has been stopped
```

Note: To stop a specific instance of a VM, include the `/INDEX` at the end of its full name. In the example here, you could stop only the third Diego Cell instance by running: `$ bosh stop diego_cell-partition-458f9d7042365ff810e9/2`

Stop Order	Component	Job/index name prefix (in <code>bosh vms</code> output)
1	Loggregator Traffic Controller	<code>loggregator_trafficcontroller-</code>
2	Doppler Server	<code>doppler-</code>
3	DEA	<code>dea-</code>
4	Diego Cell	<code>diego_cell-</code>
5	Diego Brain	<code>diego_brain-</code>
6	UAA	<code>uaa-</code>
7	Collector	<code>collector-</code>
8	Cloud Controller Worker	<code>cloud_controller_worker-</code>
9	Clock Global	<code>clock_global-</code>
10	Health Manager	<code>health_manager-</code>
11	HAProxy	<code>ha_proxy-</code>
12	Cloud Controller	<code>cloud_controller-</code>
13	Apps Manager Database	<code>consoledb-</code>
14	UAA Database	<code>uaadb-</code>
15	Cloud Controller Database	<code>ccdb-</code>
16	MySQL Server	<code>mysql-</code>
17	MySQL Proxy	<code>mysql_proxy-</code>
18	Router	<code>router-</code>
19	NFS Server	<code>nfs_server-</code>
20	Diego Database	<code>diego_database-</code>
21	etcd	<code>etcd_server-</code>
22	consul	<code>consul_server-</code>
23	NATS	<code>nats-</code>

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 cf 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 `console.YOUR-SYSTEM-DOMAIN`. Refer to [Elastic Runtime > Cloud Controller](#) 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 PCF Guide](#)

Logging into the Apps Manager

Page last updated:

Complete the following steps to log in to the Apps Manager:

1. Browse to `console.YOUR-SYSTEM-DOMAIN`. In Pivotal Ops Manager, refer to **Pivotal Elastic Runtime > Settings > Cloud Controller** to locate your system domain.
2. Log in to the Apps Manager using the UAA Administrator User credentials. In Pivotal Ops Manager, refer to **Pivotal Elastic Runtime > Credentials** for these credentials.

Controlling Apps Manager User Activity with Environment Variables

Page last updated:

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 Org Manager or Space Manager role from managing existing users and roles within their Orgs or Spaces. This allows admins to centrally manage users and roles.

Set this variable to `true` to allow users with the Org Manager or Space Manager role to manage existing users and roles within their Orgs or Spaces.

 **Note:** The previous `ENABLE_NON_ADMIN_USER_MANAGEMENT` environment variable that controlled many of the above features is 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](#).

```
$ cf api api.YOUR-SYSTEM-DOMAIN
```

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 `apps-manager` 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 apps-manager`. 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.

```
$ cf set-env apps-manager-blue ENABLE_NON_ADMIN_ROLE_MANAGEMENT true
```

4. Reinitialize your live Apps Manager with the new environment variable value.

```
$ 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.

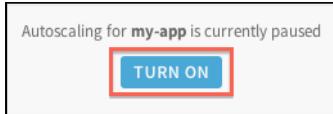
Follow the steps below to configure your App Autoscaling service instance.

1. Log in to the Apps Manager: [How to log in.](#)
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**.

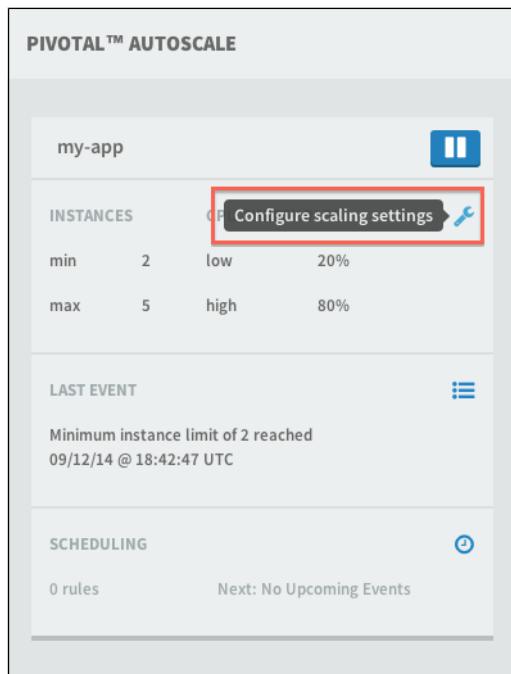
Note: You must specifically have the role of Space Developer to access the **Manage** link for the app autoscaling service. Space Managers, Space Auditors, and all Org roles do not have the permission to make changes to App Autoscaling. See [Managing User Accounts in Spaces](#) [Using the Apps Manager](#) for help managing user roles.

SERVICES		
		Add Service
SERVICE INSTANCE	SERVICE PLAN	BOUND APPS
my-database	MySQL Database	1
my-autoscaler	App Autoscaler Gold	1

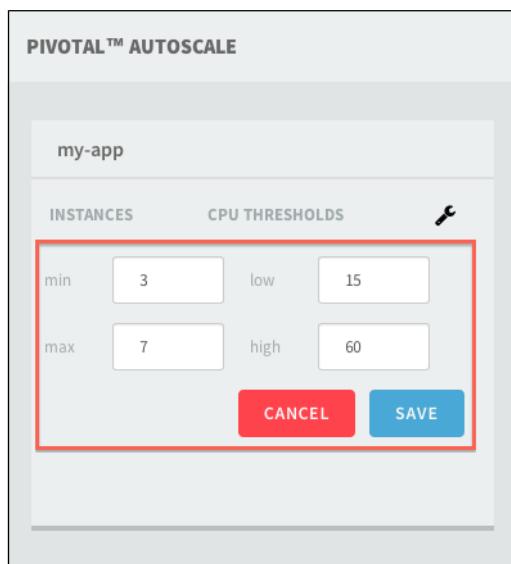
4. By default, new autoscaling instances are paused. If paused, click **Turn On** to enable monitoring of your application and start the autoscaling process.



5. Click the wrench icon on your Autoscaling dashboard.



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.

INSTANCES	CPU THRESHOLDS
min	3 low 15%
max	7 high 60%

LAST EVENT

Minimum instance limit of 2 reached
09/12/14 @ 18:42:47 UTC

SCHEDULING

0 rules Next: No Upcoming Events

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 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](#)

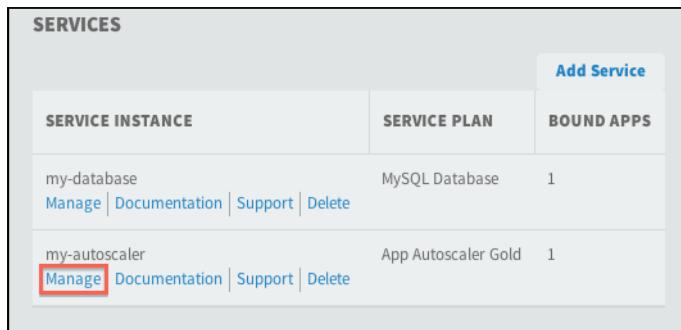
Managing Scheduled Scaling in the App Autoscaling Service

Page last updated:

Follow the steps below to manage your App Autoscaling service instance.

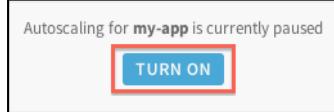
1. Log in to the Apps Manager: [How to log in.](#)
2. Select a space containing an App Autoscaling service instance from the org dashboard or from the left navigation bar.
3. In the Services section of the space dashboard, under your App Autoscaling service instance name, click **Manage**.

Note: To access the **Manage** link for the app autoscaling service, you must have the role of **Space Developer**. See [Managing User Accounts in Spaces Using the Apps Manager](#) for help managing user roles.



SERVICES		
SERVICE INSTANCE	SERVICE PLAN	BOUND APPS
my-database	MySQL Database	1
Manage Documentation Support Delete		
my-autoscaler	App Autoscaler Gold	1
Manage Documentation Support Delete		

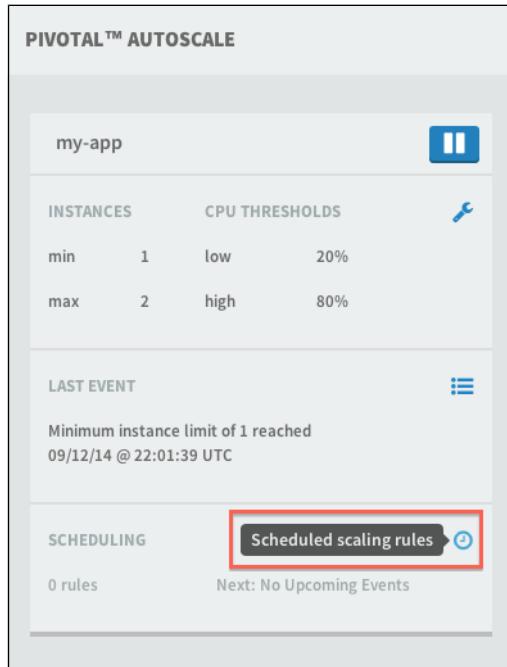
4. By default, new autoscaling instances are paused. If paused, click **Turn On** to enable monitoring of your application and start the autoscaling process.



Autoscaling for **my-app** is currently paused

TURN ON

5. Click the clock icon on your Autoscaling dashboard.



PIVOTAL™ AUTOSCALE

my-app

INSTANCES **CPU THRESHOLDS**

min	1	low	20%
max	2	high	80%

LAST EVENT

Minimum instance limit of 1 reached
09/12/14 @ 22:01:39 UTC

SCHEDULING

Scheduled scaling rules 

0 rules Next: No Upcoming Events

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.

SCHEDULING: MY-APP SERVER TIME: 09/12/14 @ 20:55:25 UTC X

+ New UNSAVED

on at ▲
*All times are UTC

repeats every
 S M T W T F S

min	<input type="text" value="5"/>	low	<input type="text" value="20%"/>
max	<input type="text" value="10"/>	high	<input type="text" value="80%"/>

ADD

7. After saving, the left side of the Scheduling interfaces shows your rule. Click your rule to edit it.

SCHEDULING: MY-APP SERVER TIME: 09/12/14 @ 20:55:25 UTC X

+ New 11/21/2014 / 02:00 X
5 to 10 instances

on at ▲
*All times are UTC

repeats every
 S M T W T F S

min	<input type="text" value="2"/>	low	<input type="text" value="20%"/>
max	<input type="text" value="5"/>	high	<input type="text" value="80%"/>

ADD

8. Edit your existing rule and click **Save** to save your changes.

SCHEDULING: MY-APP SERVER TIME: 09/12/14 @ 20:55:25 UTC X

+ New 11/21/2014 / 02:00 X
5 to 10 instances

on at ▲
All times are UTC

repeats every
 S M T W T F S

min	<input type="text" value="5"/>	low	<input type="text" value="20%"/>
max	<input type="text" value="15"/>	high	<input type="text" value="80%"/>

SAVE

9. In the left pane of the Scheduling interfaces, click the X for a rule to delete it.

SCHEDULING: MY-APP SERVER TIME: 09/12/14 @ 20:55:25 UTC X

+ New

11/21/2014 / 02:00 X

5 to 15 instances

on at *All times are UTC

repeats every S M T W T F S

min	5	low	20%
max	15	high	80%

SAVE

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.

PIVOTAL™ AUTOSCALE

my-app II

INSTANCES	CPU THRESHOLDS		
min	2	low	20%
max	5	high	80%

LAST EVENT #

Minimum instance limit of 2 reached
09/12/14 @ 20:57:55 UTC

SCHEDULING 🕒

1 rules Next: 11/21/14 @ 02:00:00 UTC

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: MY-APP SERVER TIME: 09/12/14 @ 21:27:22 UTC **X**

+ New UNSAVED

11/21/2014 / 02:00 **X**

5 to 10 instances

on 11/28/2014 at 4:00 *All times are UTC

repeats every S M T W T F S

min	10	low	20%
max	20	high	80%

ADD

- 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: MY-APP SERVER TIME: 09/12/14 @ 21:27:22 UTC **X**

+ New UNSAVED

11/21/2014 / 02:00 **X**

5 to 10 instances

Mon, Fri / 04:00 **II X**

10 to 20 instances

on 11/28/2014 at 4:00 *All times are UTC

repeats every S M T W T F S

min	1	low	20%
max	3	high	80%

ADD

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.

Adding Existing LDAP Users to a Pivotal Cloud Foundry Deployment

Page last updated:

This topic describes the procedure for adding existing LDAP users to an LDAP-enabled [Pivotal Cloud Foundry](#) (PCF) deployment.

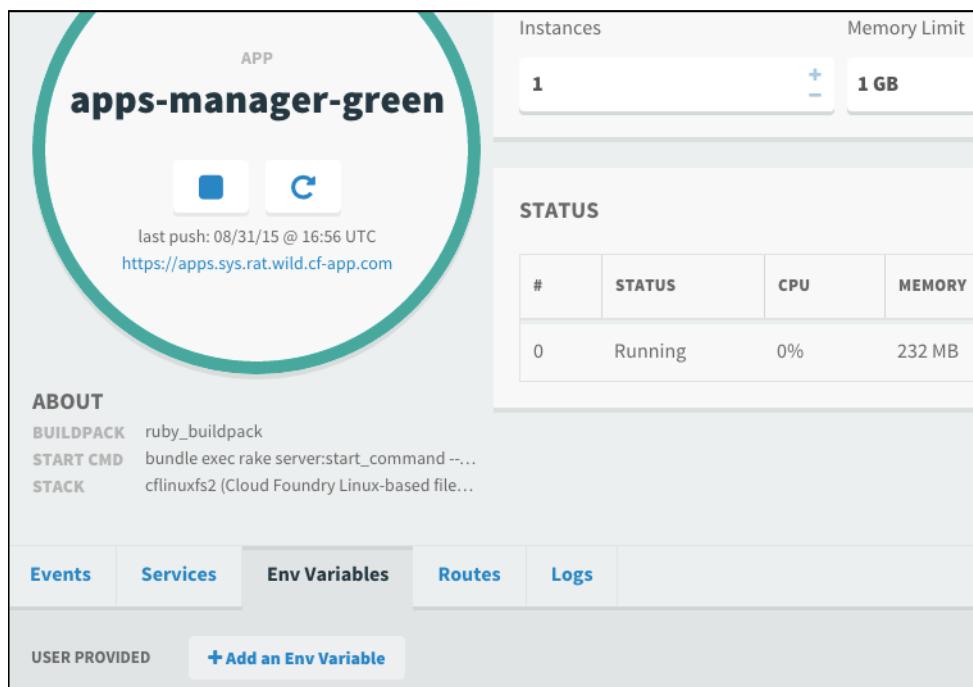
 **Note:** You must have admin access to the PCF Ops Manager Installation Dashboard for your deployment to complete the procedure described here.

Step 1: Restrict User Management to Admin Only

 **Note:** After this change, Org Managers are unable to change space and org roles for users.

Follow this procedure to disable **Invitations**, **Create Account**, and **Reset Password** from the Apps Manager console so that users are not allowed to register new accounts. Pivotal recommends this because invitations create duplicate accounts, which can cause authentication problems that are difficult to resolve.

1. Launch Apps Manager at <https://apps.SYSTEM-DOMAIN>.
2. Login with UAA admin user credentials. To find your UAA admin credentials in the PCF Ops Manager Installation Dashboard, click the **Elastic Runtime** tile, select **Credentials**, and find UAA.
3. Select your **System** org.
4. Select your **apps_manager** space and click the **apps_manager** app.



APP
apps-manager-green
last push: 08/31/15 @ 16:56 UTC
<https://apps.sys.rat.wild.cf-app.com>

Instances		Memory Limit
1	+	1 GB

STATUS

#	STATUS	CPU	MEMORY
0	Running	0%	232 MB

ABOUT

BUILDPACK: ruby_buildpack
START CMD: bundle exec rake server:start_command ~...
STACK: cflinuxfs2 (Cloud Foundry Linux-based file...)

Events **Services** **Env Variables** **Routes** **Logs**

USER PROVIDED [+ Add an Env Variable](#)

5. Select **Environment Variables**.



ENABLE_NON_ADMIN_USER_MANAGEMENT

false

[Cancel](#) **Save**

6. Locate **ENABLE_NON_ADMIN_USER_MANAGEMENT** and set it to **false**.

7. Restart the app to apply the new configuration.

Step 2: Add LDAP Users

 **Note:** Do not create new users in Elastic Runtime via the Cloud Foundry command line interface (CF CLI), by UAAC, or by using invitations in Apps Manager. This will create a user identity in the internal user store separate from the LDAP user identity. Instead, follow the procedure described below.

There are two ways to add existing LDAP users to your PCF deployment:

- In bulk, using the UAA-LDAP 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 LDAP user must log in to Apps Manager or to the CF CLI using their LDAP credentials. Users will not have access to any org or space until these are granted by an Org 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 following APIs to write a connector to manage users and permissions in Cloud Foundry.

Step 1: Create User

1. Create the user in UAA by running the following command. Replace `EXAMPLE-USERNAME` with the username of the LDAP user you wish to add.

```
$ curl "http://localhost/Users" -X POST -H "Content-Type: application/json" \
-d '{
  "userName": "EXAMPLE-USERNAME",
  "emails": [{"value": "EXAMPLE-USERNAME@EXAMPLE-DOMAIN.com"}],
  "origin": "ldap",
  "externalId": "cn=EXAMPLE-USERNAME,ou=Users,dc=test,dc=com"}'
```

2. Use the [Users API](#) to create a User record in the Cloud Controller Database with the existing user's LDAP GUID.

```
$ curl "https://api.YOUR-DOMAIN/v2/users" -d '{
  "guid": "YOUR-USER-LDAP-GUID"
}' -X POST \
-H "Authorization: bearer YOUR-BEARER-TOKEN" \
-H "Host: YOUR-HOST-URL" \
-H "Content-Type: application/x-www-form-urlencoded" \
-H "Cookie: "
```

Step 2: Provide User Access to Orgs

Associate the user with the appropriate orgs in your Elastic Runtime deployment, using the [Organizations API](#).

Step 3: 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](#).

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 Product 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.

Cloud Controller	Vm Credentials	vcap / e38252ff3dad2d28
	Staging Upload Credentials	staging_upload_user / c845fc832d5cef490674
	Bulk Api Credentials	bulk_api / ad87614112cb7cc82c9c
	Db Encryption Credentials	db_encryption / db54fc70546bbf12eb28
	Encrypt Key	

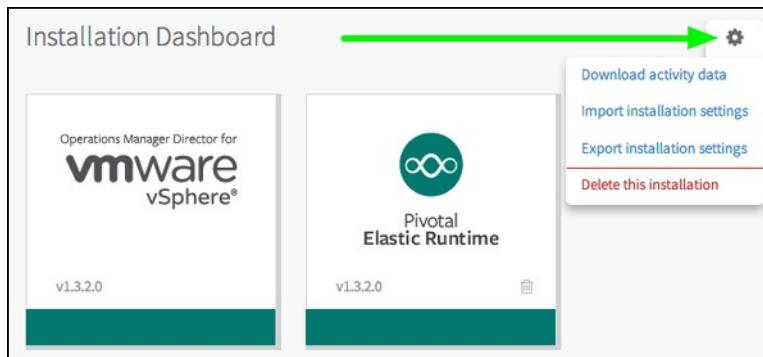
Export Installation Settings

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 Product 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.



Note: For versions of Ops Manager 1.3 or older, the process of archiving files for export may exceed the timeout limit of 600 seconds and result in a [500](#) error. To resolve this issue, you can manually increase the timeout value, or assign additional resources to the Ops Manager VM to improve performance. For more information, see the [Pivotal Support Knowledge Base](#).

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 Ops Manager Director. You access the BOSH Director using this IP address.

JOB	INDEX	IPS	CLOUD
Ops Manager Director	0	10.0.0.3	VM 9 8 d

1. Click **Credentials** and record the Director credentials.

JOB	NAME	CREDENTIALS
Ops Manager Director	VM credentials	vcap / [REDACTED]
	BOSH agent credentials	vcap / [REDACTED]
	Director credentials	director / [REDACTED]
	MANIFEST credentials	manifest / [REDACTED]

1. From the command line, run `bosh target` to log into the BOSH Director using the IP address and credentials that you recorded:

```
$ bosh target 10.0.0.3
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 |
|          | cf/183.2   |          |
+-----+-----+-----+
```

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.

You must back up each of the following:

- Cloud Controller Database
- UAA Database
- Apps Manager Database
- NFS Server
- Pivotal MySQL Server

Note: If you are running PostgreSQL and are on the default internal databases, follow the instructions below. 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.

Note: To follow the backup instructions below, your network must be configured to allow access to the BOSH Director virtual machine (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@10.0.0.10:nfs.tar.gz \ and vcap@10.0.0.03:/nfs.tar.gz`

Stop Cloud Controller

1. From a command line, run `bosh deployment DEPLOYMENT-MANIFEST` to select your deployment. The manifest is located in `/var/tempest/workspaces/default/deployments` if you are using the Ops Manager VM. If you are using a separate jumpbox, locate the manifest from the [Download the BOSH Deployment Manifest](#) procedure.

```
$ bosh deployment cf.yml
Deployment set to '/home/working_directory(cf.yml'
```

2. Run `bosh vms` to view a list of VMs in your selected deployment.

```
$ bosh vms
+-----+-----+-----+-----+
| Job/index | State | Resource Pool | IPs | 
+-----+-----+-----+-----+
| 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. Run `bosh stop SELECTED-VM` for each Cloud Controller VM. You do not need to stop the Cloud Controller worker VMs.

```
$ bosh stop cloud_controller-partition-bd784
Acting as user 'director'...
You are about to stop cloud_controller-partition-bd784/0
Detecting deployment changes
-----
Stop cloud_controller-partition-bd784/0? (type 'yes' to continue)
```

Back Up the Cloud Controller Database

Note: Follow these instructions only if you are using a PostgreSQL database.

1. In the BOSH deployment manifest, `cf.yml`, locate the Cloud Controller database (CCDB) component under the `ccdb` key and record the IP address:

```
ccdb:
  address: 10.85.52.96
  port: 2544
  db_scheme: postgres
```

2. From the Installation Dashboard in Ops Manager, select **Elastic Runtime** and click **Credentials**. Record the Cloud Controller database credentials.

Cloud Controller

Vm Credentials

vcap /

3. SSH into the Cloud Controller database VM as the admin using the IP address and password recorded in the previous steps.

```
$ ssh vcap@10.85.52.96
Password:*****
```

4. Run `find /var/vcap | grep 'bin/pg_dump'` to find the locally installed psql client on the CCDB VM. For example:

```
$ root@10.85.52.96:~# 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 10.85.52.96 -U admin -p 2544 cedb > cedb.sql
```

6. Exit from the Cloud Controller database VM.

7. Run `scp` to copy the exported database to your local machine.

```
$ scp vcap@10.85.52.96:~/cedb.sql .
```

Back Up the UAA Database

 **Note:** Follow these instructions only if you are using a PostgreSQL database.

1. In the BOSH deployment manifest, `cf.yml`, locate the `uaadb` component and record the IP address:

```
uaadb:
  address: 10.85.52.101
  port: 2544
  db_scheme: postgresql
```

2. From the Installation Dashboard in Ops Manager, select **Elastic Runtime** and click **Credentials**. Record the UAA database credentials.

UAA Database

Vm Credentials

vcap /

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.

```
$ root@10.85.52.101:~# 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 10.85.52.101 -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@10.85.52.101:~/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: 10.85.52.104
  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**. Record the Apps Manager database credentials.

Apps Manager Database	Vm Credentials	vcap / dd4dd4dd4d44
-----------------------	----------------	---------------------

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.

```
$ root@10.85.52.104:~# 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 10.85.52.104 -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@10.85.52.104:~/console.sql .
```

Back Up NFS Server

1. In the BOSH deployment manifest, `cf.yml`, locate the `nfs_server` component and record the address:

```

nfs_server:
  address: 10.0.0.10
  network: 10.0.0.0/24
  syslog_aggregator:
    address:
    port:

```

2. From the Installation Dashboard in Ops Manager, select **Elastic Runtime** and click **Credentials**. Record the NFS Server credentials.

NFS Server	Vm Credentials	vcap /
------------	----------------	--------

3. SSH into the NFS server VM and create a TAR file:

```
$ ssh vcap@10.0.0.10 '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: `ssh vcap@10.0.0.10 '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 using an internal MySQL, this will also include the Cloud Controller, UAA, and Apps Manager.

- From the Installation Dashboard in Ops Manager, select **Pivotal Elastic Runtime**. Click **Status** and record the IP address of the first instance of MySQL Server.

JOB	INDEX	IPS	CID
NATS	0	10.0.1.12	i-8db9275e
consul	0	10.0.1.53	i-47e57c94
etcd	0	10.0.1.13	i-8eb9275d
NFS Server	0	10.0.1.14	i-02b927d1
Cloud Controller Database	0	10.0.1.15	i-30b628e3
UAA Database	0	10.0.1.16	i-31b628e2
Apps Manager Database	0	10.0.1.17	i-32b628e1
Cloud Controller	0	10.0.1.18	i-aab52b79
Router	0	10.0.1.19	i-e4b42a37
Health Manager	0	10.0.1.20	i-feb42a2d
Clock Global	0	10.0.1.21	i-aeb52b7d
Cloud Controller Worker	0	10.0.1.22	i-41b42a92
UAA	0	10.0.1.23	i-36b52be5
MySQL Proxy	0	10.0.1.25	i-2c9e07ff
MySQL Server	0	10.0.1.26	i-11b22cc2

- Click **Credentials** and record the Mysql Admin Credentials of MySQL Server.

MySQL Server	Vm Credentials	vcap / 26c36dfff49c638
	Mysql Admin Credentials	root / bf45c700a0c74d54d066

- SSH into the Mysql database VM as the admin using the IP address and password recorded in the previous steps.

- Run the following command to dump the data from the p-mysql database and save it:

```
$ mysqldump -u root -p -h 10.0.1.26 --all-databases > user_databases.sql
```

- Run `scp` to copy the exported database to your local machine.

```
$ scp vcap@10.0.1.26:~/user_databases.sql .
```

Start Cloud Controller

- Run `bosh vms` to view a list of VMs in your selected deployment. The names of the Cloud Controller VMs begin with `cloud_controller`.

```
$ bosh vms
+-----+-----+-----+-----+
| Job/index | State | Resource Pool | IPs | 
+-----+-----+-----+-----+
| cloud_controller-partition-bd784/0 | failing | 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 |
+-----+-----+-----+-----+
```

2. BOSH indicates a stopped VM with the state: `failing`. Run `bosh start SELECTED-VM` for each stopped Cloud Controller VM.

```
$ bosh start cloud_controller-partition-bd784
Processing deployment manifest
-----
Processing deployment manifest
-----
You are about to start cloud_controller-partition-bd784/0
Processing deployment manifest
-----
Detecting deployment changes
-----
Start cloud_controller-partition-bd784/0? (type 'yes' to continue): yes
Performing 'start cloud_controller-partition-bd784/0'...
Director task 1428
Started preparing deployment
Started preparing deployment > Binding deployment. Done (00:00:00)
Started preparing deployment > Binding releases. Done (00:00:00)
Started preparing deployment > Binding existing deployment. Done (00:00:01)
Started preparing deployment > Binding resource pools. Done (00:00:00)
Started preparing deployment > Binding stemcells. Done (00:00:00)
Started preparing deployment > Binding templates. Done (00:00:00)
Started preparing deployment > Binding properties. Done (00:00:00)
Started preparing deployment > Binding unallocated VMs. Done (00:00:00)
Started preparing deployment > Binding instance networks. Done (00:00:00)
Done preparing deployment (00:00:01)
Started preparing package compilation > Finding packages to compile. Done (00:00:01)
Started preparing dns > Binding DNS. Done (00:00:00)
Started preparing configuration > Binding configuration. Done (00:00:13)
Started updating job cloud_controller-partition-bd784/0 > cloud_controller-partition-bd784/0 (canary)^@. Done (00:01:44)
Task 1428 done
Started 2015-02-25 17:54:28 UTC
Finished 2015-02-25 17:56:27 UTC
Duration 0:01:59
cloud_controller-partition-bd784/0 has been started
```

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

Page last updated:

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.

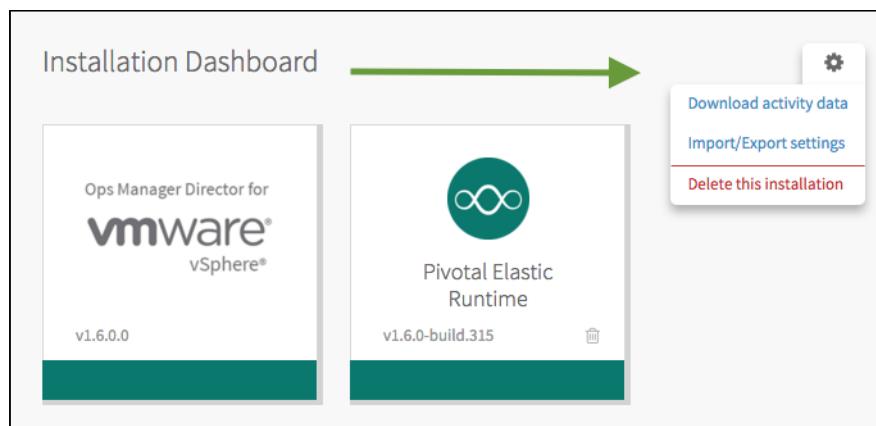
 **Note:** The procedure described in this topic restores a running Elastic Runtime deployment to the state captured by backup files. This procedure does not deploy Elastic Runtime. See the [Installing PCF Guide](#) for information about deploying Elastic Runtime.

Import Installation Settings

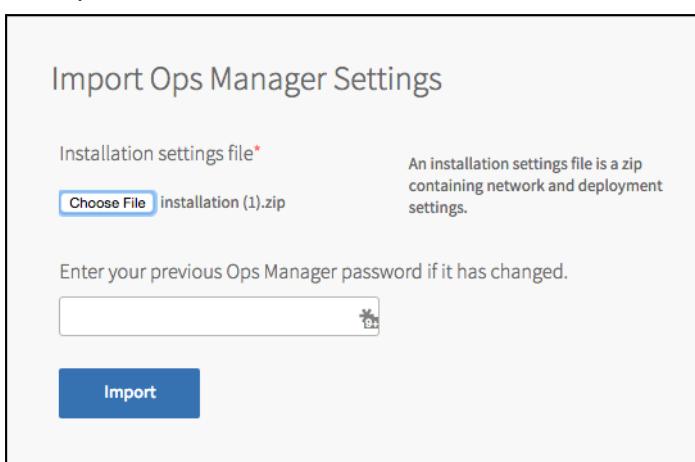
 **Note:** Pivotal recommends that you export your installation settings before importing from a backup. See the [Export Installation Settings](#) section of the *Backing Up Pivotal Cloud Foundry* topic for more information.

Import installation settings imports the settings and assets of an existing PCF installation. Importing an installation overwrites any existing installation.

1. From the Product Installation Dashboard, click the gear icon and select **Import installation settings**.



1. Browse to and select a PCF installation file.
2. If the admin password has changed since the file was exported, enter your previous Ops Manager password before you click **Import**. Enter the original password that was used when the installation was exported. If the admin password has not changed, skip this step.
3. Click **Import**.



The dialog box is titled 'Import Ops Manager Settings'. It has two main sections: 'Installation settings file*' and a description of what a zip file contains. The 'Choose File' button is highlighted with a blue border, and the file 'installation (1).zip' is listed. Below this, there is a password input field with a 'password' icon and a note: 'Enter your previous Ops Manager password if it has changed.' At the bottom is a large blue 'Import' button.

4. Return to the Product Installation Dashboard and click **Apply Changes** to install the imported installation settings.

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. Delete the `bosh-deployments.yml` file. Deleting `bosh-deployments.yml` 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.

```
sudo rm /var/tempest/worksheets/default/deployments/bosh-deployments.yml
```

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 Ops Manager Director. You access the BOSH Director using this IP address.

Job	Index	IPS	Config
Ops Manager Director	0	10.0.0.3	V98d

1. Click **Credentials** and record the Director credentials.

JOB	NAME	CREDENTIALS
Ops Manager Director	VM credentials	vcap / [REDACTED]
	BOSH agent credentials	vcap / [REDACTED]
	Director credentials	director / [REDACTED]
	NATS credentials	nats / [REDACTED]

1. From the command line, run `bosh target` to log into the BOSH Director using the IP address and credentials that you recorded:

```
$ bosh target 10.0.0.3
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 |
|          | cf/183.2 |          |
+-----+-----+-----+
```

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 the servers associated with your PCF installation..

You must restore each of the following:

- Cloud Controller Database
- Apps Manager Database
- UAA Database
- NFS Server
- Pivotal MySQL Server

Note: If you are running PostgreSQL and are on the default internal databases, follow the instructions below. If you are running your databases or filestores externally, disregard instructions for restoring the Cloud Controller, Apps Manager, and UAA Databases.

Stop Cloud Controller

1. From a command line, run `bosh deployment DEPLOYMENT-MANIFEST` to select your deployment. The manifest is located in `/var/tempest/worksheets/default/deployments` if you are using the Ops Manager VM. If you are using a separate jumpbox, locate the manifest from the [Download the BOSH Deployment Manifest](#) procedure.

```
$ bosh deployment cf.yml
Deployment set to '/home/working_directory(cf.yml'
```

2. Run `bosh vms` to view a list of VMs in your selected deployment.

```
$ bosh vms
+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+
| 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. Run `bosh stop SELECTED-VM` for each Cloud Controller VM. You do not need to stop the Cloud Controller worker VMs.

```
$ bosh stop cloud_controller-partition-bd784
Acting as user 'director'...
You are about to stop cloud_controller-partition-bd784/0
Detecting deployment changes
-----
Stop cloud_controller-partition-bd784/0? (type 'yes' to continue)
```

Restore the Cloud Controller Database

 **Note:** Follow these instructions only if you are using a PostgreSQL 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**.

1. Use `scp` to send the Cloud Controller Database backup file to the Cloud Controller Database VM.

```
$ scp cedb.sql vcap@YOUR-CCDB-VM-IP-ADDRESS:~/
```

2. SSH into the Cloud Controller Database VM.

```
$ ssh vcap@YOUR-CCDB-VM-IP
```

3. Log in to the psql client

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 cedb
```

4. Drop the database schema and create a new one to replace it.

```
ccedb=# drop schema public cascade;
ccedb=# create schema public;
```

5. Restore the database from the backup file.

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 cedb < ~/cedb.sql
```

Restore the Apps Manager Database from its backup state

 **Note:** Follow these instructions only if you are using a PostgreSQL database.

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**.

2. Use `scp` to copy the database backup file to the UAA Database VM.

```
$ scp console.sql vcap@YOUR-CONSOLE-DB-VM-IP-ADDRESS:~/
```

3. SSH into the UAA Database VM.

```
$ ssh vcap@YOUR-CONSOLE-DB-VM-IP-ADDRESS
```

4. Log into the psql client.

```
$ /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.

```
console=# drop schema public cascade;
console=# create schema public;
```

6. Restore the database from the backup file.

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 console < ~/console.sql
```

Restore UAA Database

 **Note:** Follow these instructions only if you are using a PostgreSQL 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**.

```
$ 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.

```
$ vcap@10.85.52.101:~# find /var/vcap | grep 'bin/psql'
/var/vcap/data/packages/postgres/5.1/bin/psql
```

4. Log in to the psql client:

```
$ vcap@10.85.52.101:~# /var/vcap/data/packages/postgres/5.1/bin/psql -U vcap -p 2544 uaa
```

5. Run the following commands to drop the tables:

```
drop schema public cascade;
create schema public;
\q
```

6. Exit the UAA Database VM.

```
$ exit
```

Restore the UAA Database from its backup state

 **Note:** Follow these instructions only if you are using a PostgreSQL database.

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**.

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**.

1. Run `ssh YOUR-NFS-VM-IP-ADDRESS` to enter the NFS VM.

```
$ ssh vcap@10.0.0.10
```

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 zxf 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

1. Use `scp` to send the MySQL Database backup file to the MySQL Database VM. 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**.

```
$ scp user_databases.sql vcap@YOUR-MYSQL-VM-IP-ADDRESS:~/
```

2. SSH into the MySQL Database VM.

```
$ ssh vcap@YOUR-MYSQL-VM-IP
```

3. Use the MySQL password and IP address to restore the MySQL database by running the following command.

```
mysql -h YOUR-MYSQL-SERVER-IP -u root -p < ~/.user_databases.sql
```

4. Log in to the MySQL client and flush privileges.

```
$ mysql -u root -p -h
mysql> flush privileges;
```

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`.

```
$ bosh vms
+-----+-----+-----+
| Job/index | State | Resource Pool | IPs | 
+-----+-----+-----+
| cloud_controller-partition-bd784/0 | failing | 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 |
+-----+-----+-----+
```

2. BOSH indicates a stopped VM with the state: `failing`. Run `bosh start SELECTED-VM` for each stopped Cloud Controller VM.

```
$ bosh start cloud_controller-partition-bd784
Processing deployment manifest
-----
Processing deployment manifest
-----
You are about to start cloud_controller-partition-bd784/0
Processing deployment manifest
-----
Detecting deployment changes
-----
Start cloud_controller-partition-bd784/0? (type 'yes' to continue): yes
Performing 'start cloud_controller-partition-bd784/0'...
Director task 1428
Started preparing deployment
Started preparing deployment > Binding deployment. Done (00:00:00)
Started preparing deployment > Binding releases. Done (00:00:00)
Started preparing deployment > Binding existing deployment. Done (00:00:01)
Started preparing deployment > Binding resource pools. Done (00:00:00)
Started preparing deployment > Binding stemcells. Done (00:00:00)
Started preparing deployment > Binding templates. Done (00:00:00)
Started preparing deployment > Binding properties. Done (00:00:00)
Started preparing deployment > Binding unallocated VMs. Done (00:00:00)
Started preparing deployment > Binding instance networks. Done (00:00:00)
Done preparing deployment (00:00:01)
Started preparing package compilation > Finding packages to compile. Done (00:00:01)
Started preparing dns > Binding DNS. Done (00:00:00)
Started preparing configuration > Binding configuration. Done (00:00:13)
Started updating job cloud_controller-partition-bd784/0 > cloud_controller-partition-bd784/0 (canary)^@. Done (00:01:44)
Task 1428 done
Started 2015-02-25 17:54:28 UTC
Finished 2015-02-25 17:56:27 UTC
Duration 00:01:59
cloud_controller-partition-bd784/0 has been started
```

Upgrading Operations Manager

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.

Important: Read the Known Issues section of the [Pivotal Cloud Foundry \(PCF\) Release Notes](#) before getting started.

Complete the following steps to upgrade Pivotal Cloud Foundry Operations Manager (Ops Manager). Select the procedure that corresponds to your upgrading scenario.

Pivotal strongly recommends that you [back up all critical data](#) prior to upgrading.

 **Note:** To upgrade your [Pivotal Cloud Foundry \(PCF\)](#) installation 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 Ops Manager release 1.1 and you are upgrading to 1.3, you must sequentially install 1.2 and 1.3.

 **Note:** If any of your applications were deployed originally to PCF Elastic Runtime 1.3 or earlier, or were specifically configured to run on Canonical's lucid64 stack, you must migrate them to a newer OS stack before upgrading to PCF 1.6. Pivotal recommends using the cf CLI [stack-changer plugin](#) to move your apps from lucid64, which is no longer supported, to cflinuxfs2, a stack based on Ubuntu Trusty 14.04. See the [New cflinuxfs2 Stack](#) support page for instructions on how to perform this migration.

Upgrading to PCF 1.6

This section contains important guidelines that you must follow if you are upgrading an existing PCF 1.5 deployment to PCF 1.6. Failure to follow these instructions may jeopardize your existing deployment data or cause your upgrade to fail.

Before You Upgrade

- In Ops Manager, delete the Diego Beta tile, if installed.
- On the **Resource Config** page of your existing Elastic Runtime 1.5 configuration, scale the number of consul servers down to 1 instance. You can scale the number of instances back up after you have upgraded Elastic Runtime. If you are not running any consul servers as part of your deployment, you can ignore this step.
- If you are upgrading PCF on AWS, create and configure an ELB for enabling Diego SSH connections. See [Creating an SSH Proxy for Diego with an AWS ELB](#) for instructions.

During the Elastic Runtime Upgrade

- If your existing PCF 1.5 deployment uses internal databases, do not modify the selected Postgres/MySQL configuration on the **System Database Config** page. PCF 1.6 allows you to deploy all of your internal databases to MySQL for improved high-availability. However, you should only select the MySQL option after you have migrated all of your existing Postgres data. If you wish to migrate your data in order to access this feature in PCF 1.6, please contact Pivotal support first.
- If your existing PCF 1.5 deployment uses an external S3 filestore, do not modify the pre-populated value used in the bucket name fields on the **File Storage Config** page. PCF 1.6 allows you to specify four different buckets for various cloud-controller artifacts; however, you should continue to use your existing 1.5 bucket names across all four buckets in order to retain existing data.
- If you wish all new applications pushed to your PCF 1.6 deployment to use [Diego](#) by default, select the **Use Diego by default instead of DEAs** option on the [Diego configuration screen](#). If you do not select this option, application developers must explicitly specify to use Diego when pushing their applications. See [Diego Migration](#).

After the Upgrade

- On the **Resource Config** page of Elastic Runtime, scale your consul server instances back up to the desired number.
- Advise your application developers on targeting Diego when pushing their applications. See [Diego Migration](#).

Upgrading with Installed Products

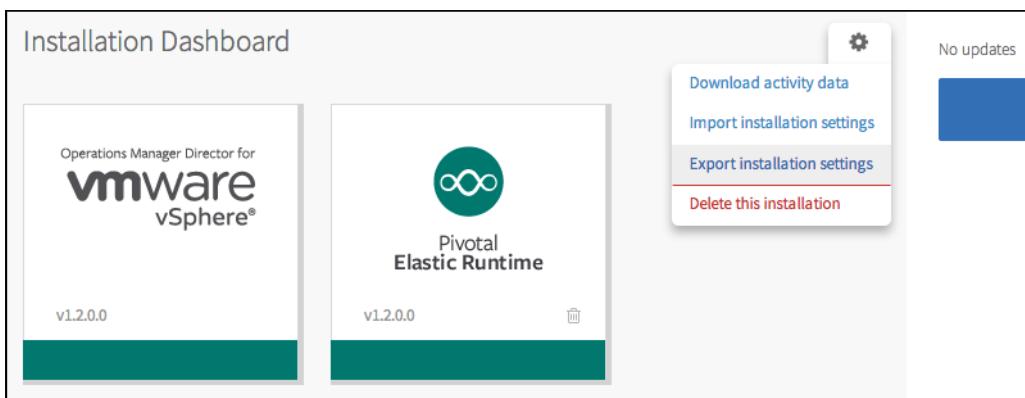
Follow the steps below to keep all installed products when you upgrade Ops Manager.

Note: 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.

1. 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 50 GB, or enough to handle the size of the resources.
2. Ensure that every product tile on the Installation Dashboard is compatible with the new version of Ops Manager. To be compatible, a product must meet the minimum version requirement for that product. If a product does not meet this requirement, you must upgrade the product or remove the tile before upgrading Ops Manager.

For specific compatibility information, refer to the full [Product Version Matrix](#).

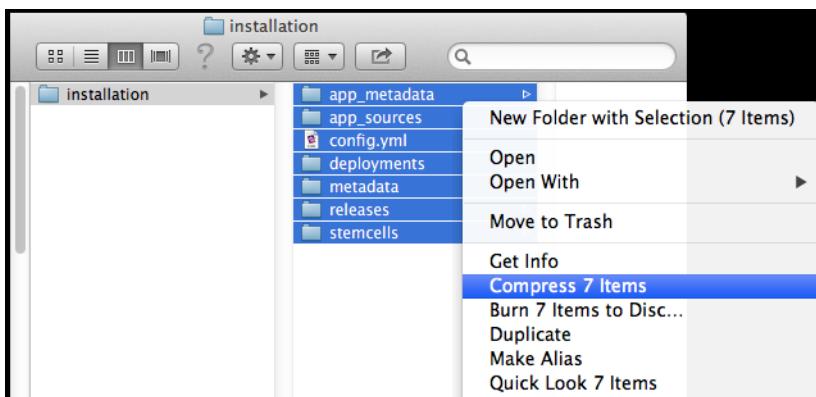
3. From the Product Dashboard, select **Actions > Export installation settings**.



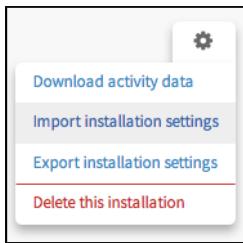
This exports the current PCF installation with all of its assets. When you export an installation, the export contains the base VM images and necessary packages, and references to the installation IP addresses. As a result, an exported file can be very large, as much as 5 GB or more.

- Because of the size of the exported file, exporting can take tens of minutes.
- Some browsers do not provide feedback on the status of the export process, and may appear to hang.
- Some operating systems may automatically unzip the exported installation. If this occurs, create a zip file of the unzipped export.

Note: When creating a zip file of an unzipped export, do not start compressing at the “installation” folder level. Instead, start compressing at the level containing the `config.yml` file:

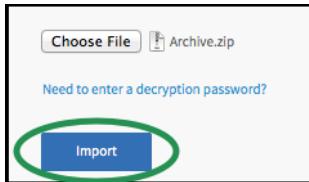


4. Download the latest Ops Manager VM Template from the [Pivotal Network](#) site.
5. Record the IP address of the existing Ops Manager VM.
6. To avoid IP conflicts, power off the existing Ops Manager VM.
7. Deploy the new Ops Manager VM. See [Deploying Operations Manager to vSphere](#) or [Deploying Operations Manager to vCloud Air and vCloud](#).
8. From the Product Dashboard, select **Actions > Import installation settings**.



9. Click **Choose File**, browse to the installation zip file exported in Step 1, and click **Choose**.

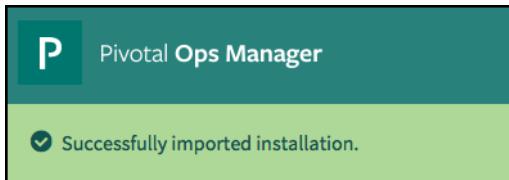
10. If the admin password has changed since the file was exported, you must select **Need to enter a decryption password?** before you click **Import** to prevent an error. Enter the original password that was used when the installation was exported. If the admin password has not changed, you may skip this step.



11. Click **Import**.

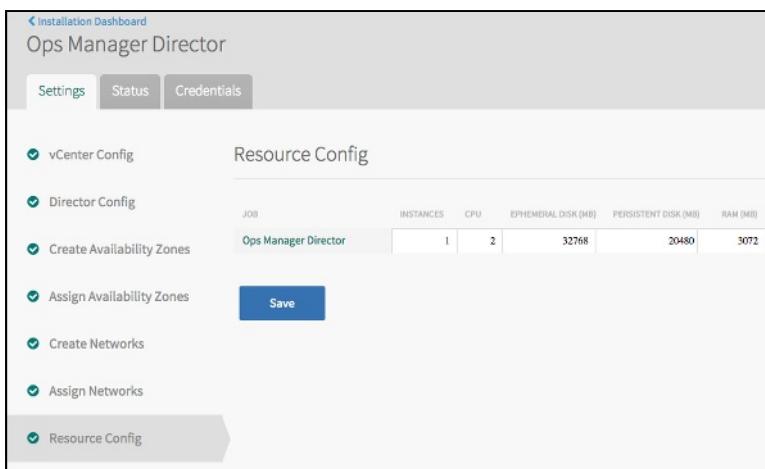
Note: Importing can take tens of minutes. Some browsers do not provide feedback on the status of the import process, and may appear to hang.

12. A “Successfully imported installation” message appears upon completion.



13. Select **Resource Config**.

14. Change the value of the **Ephemeral Disk (MB)** for the Ops Manager Director to **32768**.



15. Click **Save**.

16. Click **Apply Changes**. This immediately imports and applies upgrades to all tiles in a single transaction.

17. Click each service tile, select the **Status** tab, and confirm that all VMs appear and are in good health.

18. Remove the original Ops Manager VM from your IaaS if the new installation functions correctly.

Note: Independently from upgrading Ops Manager, you can upgrade individual products such as Pivotal Cloud Foundry Elastic Runtime, [Pivotal MySQL](#), or [RabbitMQ](#) in your PCF deployment. See [Upgrading Products in a PCF Deployment](#).

Uninstalling and Reinstalling Ops Manager

Follow these steps to upgrade Ops Manager if you have no installed products, or you have no installed products that you want to keep.

Note: If you want to reinstall Ops Manager, and would like to import your information to the new installation, you will need the exported Cloud Controller database and the configuration from the previous installation.

1. Browse to the Ops Manager web interface and select **Delete this installation**.
2. In vSphere, vCloud Air, or vCloud, power off and delete your existing Ops Manager VM.
3. Download the latest Ops Manager VM Template from the [Pivotal Network](#) site.
4. Deploy the new Ops Manager VM. See one of the following topics:
 - [Deploying Operations Manager to vSphere](#)
 - [Deploying Operations Manager to vCloud Air and vCloud](#)

Rolling Back an Upgraded Installation

Note: To roll back an installation, or if upgrading fails, Pivotal recommends that you contact Pivotal Support.

If you want to roll back or troubleshoot a failed installation on your own, consult the table below. The table shows guidelines for compatibility between your exported configuration and available Ops Manager versions.

Current Version	Can import into 1.2	Can import into 1.3	Notes
1.2	✓	✓	After successful import to 1.3, 1.2 export configuration is no longer valid.
1.3		✓	

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.

Elastic Runtime Snapshot

Current Pivotal Cloud Foundry Elastic Runtime Details

Version: 1.6.0

Release Date: 23 October 2015

Software component version: Cloud Foundry 222

Compatible Ops Manager Version(s): 1.6.x

vSphere support? Yes

AWS support? Yes

OpenStack support? Yes

Upgrading Elastic Runtime to the Latest Version

Consider the following compatibility information before upgrading Elastic Runtime for Pivotal Cloud Foundry.

 **Note:** Before you upgrade to Ops Manager 1.5.x, you must first upgrade PCF Elastic Runtime to any version in its 1.3.x minor release. This allows PCF Elastic Runtime upgrades after you install OpsManager 1.5.x.

For more information, refer to the full [Product Compatibility Matrix](#).

Ops Manager Version	Supported Upgrades from Imported Elastic Runtime Installation
1.3.x	<ul style="list-style-type: none">From 1.2.0 to 1.3.0, 1.3.1, 1.3.2, 1.3.3, 1.3.4, or 1.3.5From 1.2.1 to 1.3.0, 1.3.1, 1.3.2, 1.3.3, 1.3.4, or 1.3.5From 1.2.2 to 1.3.0, 1.3.1, 1.3.2, 1.3.3, 1.3.4, or 1.3.5From 1.3.0 to 1.3.1, 1.3.2, 1.3.3, 1.3.4, or 1.3.5From 1.3.1 to 1.3.2, 1.3.3, 1.3.4, or 1.3.5From 1.3.2 to 1.3.3, 1.3.4, or 1.3.5From 1.3.3 to 1.3.4 or 1.3.5From 1.3.4 to 1.3.5
1.4.x	<ul style="list-style-type: none">From 1.3.0 to 1.4.0From 1.3.1 to 1.4.0From 1.3.2 to 1.4.0From 1.3.3 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5From 1.3.4 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5From 1.3.5 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5From 1.4.0 to 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5From 1.4.1 to 1.4.2, 1.4.3, 1.4.4, or 1.4.5From 1.4.2 to 1.4.3, 1.4.4, or 1.4.5From 1.4.3 to 1.4.4 or 1.4.5From 1.4.4 to 1.4.5
	<ul style="list-style-type: none">From 1.3.0 to 1.4.0From 1.3.1 to 1.4.0From 1.3.2 to 1.4.0

1.5.x	<ul style="list-style-type: none"> From 1.3.3 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.3.4 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.3.5 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.4.0 to 1.4.1, 1.4.2, 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.1 to 1.4.2, 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.2 to 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.3 to 1.4.4, 1.4.5, or 1.5.0 From 1.4.4 to 1.4.5 or 1.5.0 From 1.4.5 to 1.5.0
1.6.x	<ul style="list-style-type: none"> From 1.3.0 to 1.4.0 From 1.3.1 to 1.4.0 From 1.3.2 to 1.4.0 From 1.3.3 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.3.4 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.3.5 to 1.4.0, 1.4.1, 1.4.2, 1.4.3, 1.4.4, or 1.4.5 From 1.4.0 to 1.4.1, 1.4.2, 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.1 to 1.4.2, 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.2 to 1.4.3, 1.4.4, 1.4.5, or 1.5.0 From 1.4.3 to 1.4.4, 1.4.5, or 1.5.0 From 1.4.4 to 1.4.5 or 1.5.0 From 1.4.5 to 1.5.0 From 1.5.0 to 1.5.1, 1.5.2, 1.5.3, 1.5.4, 1.5.5, 1.5.6, or 1.6.0 From 1.5.1 to 1.5.2, 1.5.3, 1.5.4, 1.5.5, 1.5.6, or 1.6.0 From 1.5.2 to 1.5.3, 1.5.4, 1.5.5, 1.5.6, or 1.6.0 From 1.5.3 to 1.5.4, 1.5.5, 1.5.6, or 1.6.0 From 1.5.4 to 1.5.5, 1.5.6, or 1.6.0 From 1.5.5 to 1.5.6 or 1.6.0 From 1.5.6 to 1.6.0

Install using the Pivotal Operations Manager

To install Elastic runtime for PCF, follow the procedure for installing Pivotal Cloud Foundry Operations Manager tiles:

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.

Upgrading PCF Products

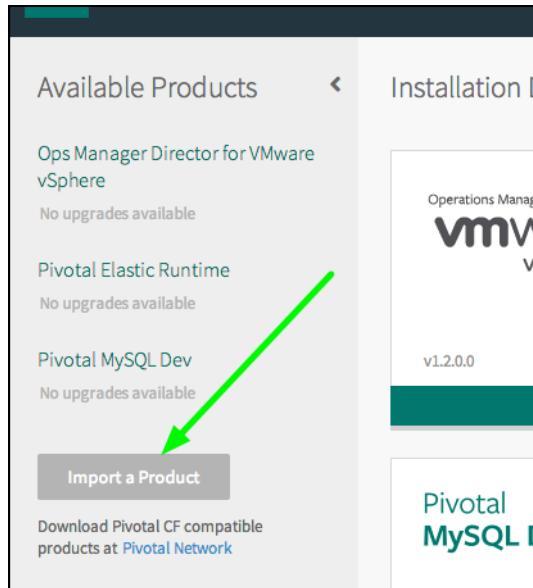
Important: Read the Known Issues section of the [Pivotal Cloud Foundry Release Notes](#) before getting started.

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: 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 1.1 and you are upgrading to 1.3, you must sequentially install 1.2

and 1.3.

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 50 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**.

Monitoring Virtual Machines in 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 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.

Installation Dashboard Pivotal Elastic Runtime																
Settings	Status	Credentials	Logs													
JOB	INDEX	IPS	CID	LOAD AVG15	CPU	MEMORY	SWAP	SYSTEM DISK	EPHEM. DISK	PERS. DISK	LOGS					
HAProxy	0	10.0.0.254	vm-9985a13c-106a-48d1-a3de-d0e0e816c857	0.06%	0.1%	9.6%	0.0%	41%	5%	N/A						
NATS	0	10.0.0.5	vm-dee49615-aea8-4f4f-bf0f-b1060083ddef	0.12%	0.1%	9.7%	0.0%	41%	21%	N/A						
			vm-6d43e59e-													

The columns display the following information:

VM Data Point	Details
Job	Each job represents a component running on one or more VMs that Ops Manager deployed.
Index	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.
IPs	IP address of the job VM.
CID	Uniquely identifies the VM.
Load Avg15	CPU load average over 15 minutes.
CPU	Current CPU usage.
Memory	Current memory usage.
Swap	Swap file percentage.
System Disk	System disk space usage.
Ephem. Disk	Ephemeral disk space usage.
Pers. Disk	Persistent disk space usage.
Logs	Download link for the most recent log files.

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-guestd:** 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.

Deploying Pivotal Ops Metrics

Page last updated:

The Pivotal Ops Metrics tool is a JMX extension for Elastic Runtime. Pivotal Ops Metrics collects and exposes system data from Cloud Foundry components via a JMX endpoint. Use this system data to monitor your installation and assist in troubleshooting.

The Pivotal Ops Metrics tool is composed of the following three virtual machines:

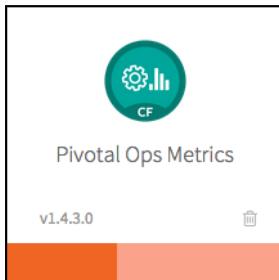
- The JMX Provider
- A VM that governs compilation
- A nozzle for the [Loggregator Firehose](#)

Follow the instructions below to deploy Pivotal Ops Metrics using the [Pivotal Cloud Foundry](#) (PCF) Operations Manager.

Step 1: Install the Ops Metrics Tile

 **Note:** Pivotal strongly suggests that you install [Pivotal Cloud Foundry Elastic Runtime](#) before installing Ops Metrics. If you do not install Elastic Runtime first, the system displays numerous errors if you configure Ops Metrics to pull data from the non-existent Elastic Runtime installation.

1. [Download Ops Metrics](#).
2. Import Ops Metrics into Ops Manager by following the instructions for [Adding and Importing Products](#).
3. On the Installation Dashboard, click the **Pivotal Ops Metrics** tile.

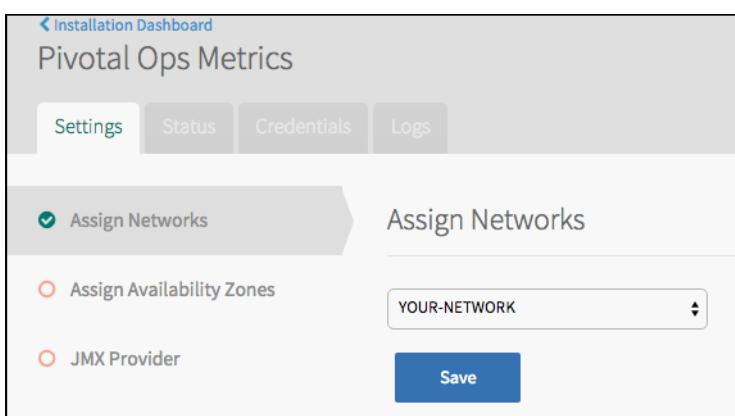


The orange bar on the **Pivotal Ops Metrics** tile indicates that the product requires configuration.

Step 2: Configure Networks

1. Select **Assign Networks**.
2. Use the drop-down menu to select a Network.

 **Note:** Ops Metrics uses the default Assigned Network if you do not select a different network.



Step 3: Configure Availability Zones

1. (vSphere and Amazon Web Services Only) Select **Assign Availability Zones**. These are the Availability Zones that you [create](#) when configuring Ops Manager Director.
- o Select an Availability Zone under **Place singleton jobs**. Ops Manager runs Metrics jobs with a single instance in this Availability Zone.
 - o Select one or more Availability Zones under **Balance other jobs**. Ops Manager balances instances of Metrics jobs with more than one instance across the Availability Zones that you specify.

The screenshot shows the 'Pivotal Ops Metrics' interface with the 'Settings' tab selected. On the left, a sidebar lists several configuration options: 'Assign Networks' (checked), 'Assign Availability Zones' (highlighted in orange), 'JMX Provider' (unchecked), 'OpenTSDB Firehose Nozzle' (unchecked), 'Resource Config' (checked), and 'Stemcell' (checked). The main content area is titled 'Availability Zone Assignments'. It contains two sections: 'Place singleton jobs in' and 'Balance other jobs in'. Under 'Place singleton jobs in', the 'MY-ZONE' radio button is selected. Under 'Balance other jobs in', the 'MY-ZONE' checkbox is checked. A 'Save' button is located at the bottom right of the form.

Step 4: Configure JMX Provider

1. Select **JMX Provider**.
2. Enter a new username and password into the **JMX Provider credentials** username and password fields.
3. Record these credentials. You will use these to connect JMX clients to the JMX Provider.

Step 5: Configure SSL

1. (Optional) Select the **Enable SSL** checkbox. Enabling SSL requires JMX clients to use SSL to connect to the JMX Provider. If SSL is not enabled, JMX clients can connect to the JMX Provider without SSL credentials.

Assign Availability Zones

JMX Provider credentials *

Enable SSL

SSL Certificate

Certificate PEM

Private Key PEM

[Generate Self-Signed RSA Certificate](#)

Save

If you select the **Enable SSL** checkbox, you must also provide an SSL certificate and private key. There are two ways to provide an SSL certificate and private key:

- If you are using a signed certificate, paste an X.509 certificate in the **Certificate PEM** field and a PKCS#1 private key in the **Private Key PEM** field.
- If you want to use SSL but do not want to use a signed certificate, you must perform the following actions:

1. Generate a self-signed certificate on the server.
2. Import the self-signed certificate to a trust store on the client.
3. Start jConsole, or another monitoring tool, with the trust store.

For more information, see [Using SSL with a Self-Signed Certificate](#).

Once you have provided an SSL certificate and private key, click **Save**.

Assign Networks

Credentials to connect to JMX Provider

JMX Provider credentials *

Enable SSL

SSL Certificate

-----BEGIN CERTIFICATE-----
MIIDUTCCAg2AwIBAgIVAP2UDTsMs56HI
UstM4R0yyPMruhuMA0GCSqGSIb3DQE
B
BQUAMDsxCzAJBgNVBAYTAIVTMRAwDgY

-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEzINM7S/rp1rlylKM
aK11yYFyGi/JN9Z6E51SwxKfCapgS
HSof6zERl5ijwtgZ7nHobymp3UnW5q
IxSeKpg5JaQKM+u5/bzw2Z4qAg+FSO

[Generate Self-Signed RSA Certificate](#)

Save

Step 6: Resource Configuration

Note: Do not change the **OpenTSDB Firehose Nozzle** instance count unless you have a running Elastic Runtime installation. If necessary, you can skip this step and perform it at a later time.

To receive metrics data from the PCF Elastic Runtime firehose, including Diego metrics, change the **OpenTSDB Firehose Nozzle** instance count from to .

Step 7: Apply Changes

1. Navigate to the PCF Ops Manager Installation Dashboard.
2. In the Pending Changes view, click **Apply Changes** to install Pivotal Ops Metrics.

The screenshot shows the 'Pending Changes' section of the Installation Dashboard. It displays three components: 'Operations Manager Director for vSphere' (v1.3.0.0), 'Pivotal Elastic Runtime' (v1.3.0.0.alpha.323.9052878), and 'Pivotal Ops Metrics' (v1.3.0.0). The 'Pivotal Ops Metrics' component has a green status bar. On the right, there is a summary of pending changes: 'INSTALL Metrics' and 'UPDATE Pivotal Elastic Runtime'. Below this is a large blue 'Apply changes' button.

3. When complete, a “Changes Applied” message appears.

Step 7: Find the IP of the JMX Provider

1. Click **Return to Product Dashboard**.
2. Click the **Pivotal Ops Metrics** tile and select the **Status** tab.
3. Record the IP address of the JMX Provider.

Note: After installation, your JMX client connects to this IP address at port 44444 using the credentials that you supplied.

The screenshot shows the 'Metrics' section of the Product Dashboard. The 'Status' tab is selected. A table titled 'Jobs on Availability Zone "default"' is displayed, showing the following data:

JOB	INDEX	IPS	CID
JMX Provider	0	10.85.52.127	vm-26195407-c448-4a89

Step 8: Configure the Metrics IP Address

1. Return to the **Installation Dashboard**. Click the **Ops Manager Director** tile and select **Director Config**.
2. In the **Metrics IP Address** field, enter the IP address of the JMX Provider. Click **Save**.

Installation Dashboard

Ops Manager Director for VMware vSphere

Settings Status Credentials Logs

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Director Config

NTP Servers (comma delimited)*
time.apple.com

Metrics IP Address
10.85.52.127

Enable VM Resurrector Plugin

Save

Step 9: (Optional) Allocate a VM for the Collector

Note: As of Elastic Runtime version 1.6, application containers run by default in Diego. If you exclusively use Diego for deploying applications, you do not need to complete this step. The collector only gathers metrics if you use the [Droplet Execution Agent \(DEA\)](#) application container engine.

1. Return to the **Installation Dashboard**. Click the **Pivotal Elastic Runtime** tile and select **Resource Config**.
2. Change the number of **Instances** for the **Collector** job from **0** to **1**. Click **Save**.

Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks

Assign Availability Zones

HAProxy

Router IPs

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Lifecycle Errands

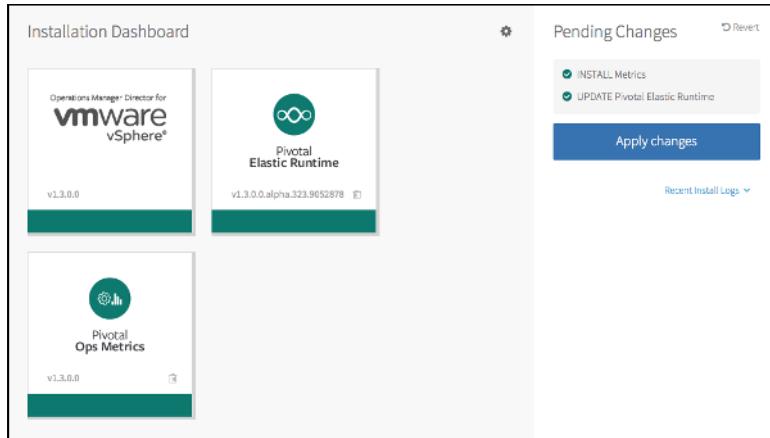
Resource Config

Resource Config

JOB	INSTANCES
HAProxy	1
NATS	1
etcd	1
Health Manager	1
NFS Server	1
Cloud Controller Database	1
Cloud Controller	1
Clock Global	1
Cloud Controller Worker	1
Router	1
Collector	1
UAA Database	1
UAA	1
Login	1
Console Database	1
MySQL Server	1
DEA	1

Step 10: Complete Installation

1. In the Pending Changes view, click **Apply Changes**.



2. When complete, a "Changes Applied" message appears. Click **Return to Product Dashboard**. Pivotal Ops Metrics is now installed and configured.

Once installed and configured, metrics for Cloud Foundry components automatically report to the JMX endpoint. Your JMX client uses the credentials supplied to connect to the IP address of the Pivotal Ops Metrics JMX Provider at port 44444.

Final Reminder

If you installed Ops Metrics before installing Elastic Runtime tile, you can revisit and enable the **OpenTSDB Firehose Nozzle** in [Step 6: Resource Configuration](#) after you install Elastic Runtime.

Using SSL with a Self-Signed Certificate in JMX Bridge

Page last updated:

Secure Socket Layer (SSL) is a standard protocol for establishing an encrypted link between a server and a client. To communicate over SSL, a client needs to trust the SSL certificate of the server.

This topic explains how to use SSL with a self-signed certificate in JMX Bridge (formerly Ops Metrics). This SSL layer secures traffic between JMX Bridge and the user, and is separate from the SSL layer [configured between Elastic Runtime](#) and the rest of the Ops Manager environment.

There are two kinds of SSL certificates: signed and self-signed.

- **Signed:** A Certificate Authority (CA) signs the certificate. A CA is a trusted third party that verifies your identity and certificate request, then sends you a digitally signed certificate for your secure server. Client computers automatically trust signed certificates. Signed certificates are also called *trusted certificates*.
- **Self-signed:** Your own server generates and signs the certificate. Clients do not automatically trust self-signed certificates. To communicate over SSL with a server providing a self-signed certificate, a client must be explicitly configured to trust the certificate.

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.

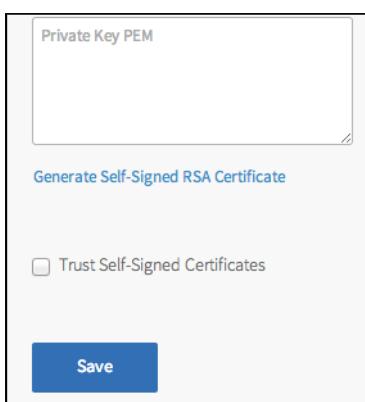
The following procedure configures a JMX user client to trust a self-signed certificate by importing the certificate to its truststore, an internal keystore. To use a trusted certificate signed by a CA, you only need to paste the Certificate and Key into the fields in the Ops Manager JMX Bridge tile, as shown in [Step 1, Option 2](#), below.

Step 1: Supply SSL Certificate

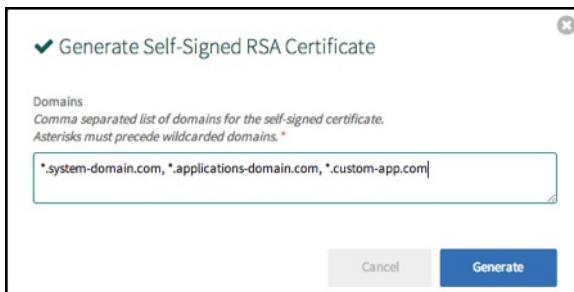
Option 1: Generate Self-Signed Certificate

Follow the steps below to generate a self-signed certificate on your server:

1. In Pivotal Ops Manager, click the **JMX Bridge** tile.
2. Check **Enable SSL**.
3. Click **Generate Self-Signed RSA Certificate** and check the **Trust Self-Signed Certificates** box.



4. Enter your system and application domains in wildcard format. Optionally, also add any custom domains in wildcard format. Click **Generate**.



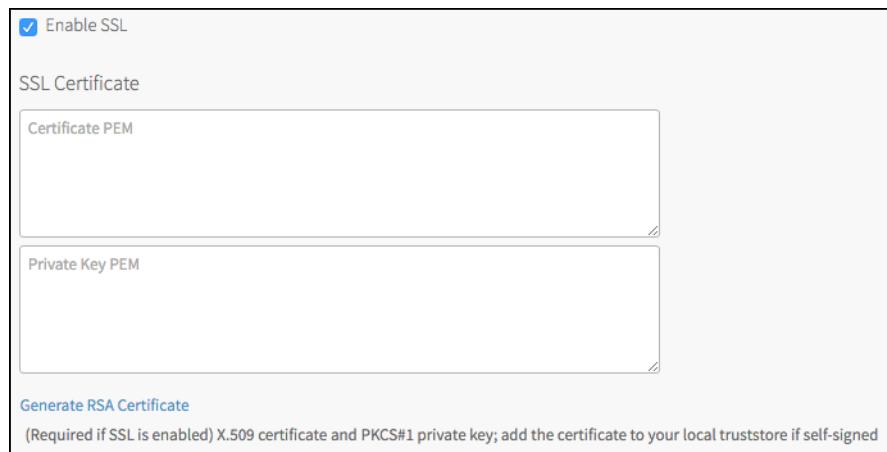
5. Select and copy the certificate.



6. Paste the certificate into a text file and save as a `.cer` file, such as `MY-JMX-BRIDGE.cer`.

Option 2: Use an Existing Self-Signed Certificate

1. In Pivotal Ops Manager, click the **JMX Bridge** tile.
2. Check **Enable SSL**.
3. Paste your certificate and private key into the appropriate boxes. This is your X.509 certificate and PKCS#1 private key.



Step 2: Import the Self-signed Certificate to a Truststore

Follow the steps below to import the self-signed certificate to your client:

1. Copy your certificate file `MY-JMX-BRIDGE.cer` from your server to your client.
2. Navigate to the client directory where you copied the saved certificate.
3. Use `keytool -import` to import the certificate with an alias of `ops-metrics-ssl` to the truststore `localhost.truststore`:

```
$ keytool -import -alias ops-metrics-ssl -file MY-JMX-BRIDGE.cer -keystore localhost.truststore
```

- o If `localhost.truststore` already exists, a password prompt appears. Enter the keystore password that you recorded in a previous step.
- o If `localhost.truststore` does not exist, you must create a password.

4. Verify the details of the imported certificate.

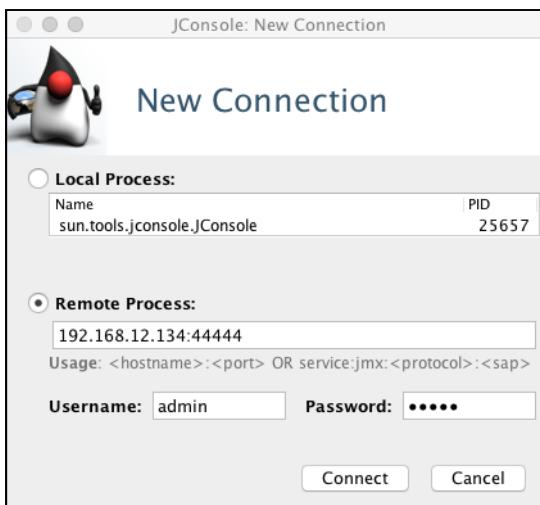
Step 3: Start a Monitoring Tool with the Truststore

Once you import the self-signed certificate to `localhost.truststore` on the client, configure your monitoring tool, such as Jconsole, to use the truststore. You do this from a command line, by starting your monitoring tool with the location and password of the truststore.

1. Pass in the location of `localhost.truststore` to your monitoring tool with the `javax.net.ssl.trustStore` property, and its password with the `javax.net.ssl.trustStorePassword` property. For example, you would invoke jConsole with:

```
$ jconsole -J-Djavax.net.ssl.trustStore=/lib/home/jcert/localhost.truststore -J-Djavax.net.ssl.trustStorePassword=KEYSTORE_PASSWORD
```

2. In the **Remote Process** field, enter the fully qualified hostname of the Maximus server, port number `44444`.



3. To complete the **Username** and **Password** fields, refer to the **Credentials** tab of the JMX Bridge tile in Pivotal Ops Manager. By default, these credentials are `admin` and `admin`.

Your monitoring tool should now communicate with your server through the SSL connection.

Using Pivotal Ops Metrics

Page last updated:

Pivotal Ops Metrics is a Java Management Extensions (JMX) tool for Elastic Runtime. To help you monitor your installation and assist in troubleshooting, Pivotal Ops Metrics collects and exposes system data from Cloud Foundry components via a JMX endpoint.

Cloud Controller Metrics

Pivotal Ops Metrics reports the number of Cloud Controller API requests completed and the requests sent but not completed.

The number of requests sent but not completed represents the pending activity in your system, and can be higher under load. This number will vary over time, and the range it can vary over depends on specifics of your environment such as hardware, OS, processor speeds, load, etc. In any given environment, though, you can establish a typical range of values and maximum for this number.

Use the Cloud Controller metrics to ensure that the Cloud Controller is processing API requests in a timely manner. If the pending activity in your system increases significantly past the typical maximum and stays at an elevated level, Cloud Controller requests may be failing and additional troubleshooting may be necessary.

The following table shows the name of the Cloud Controller metric, what the metric represents, and the metric type (data type).

METRIC NAME	DEFINITION	METRIC TYPE (DATA TYPE)
cc.requests.completed	Number of Cloud Controller API requests completed since this instance of Cloud Controller started	Counter (Integer)
cc.requests.outstanding	Number of Cloud Controller API requests made but not completed since this instance of Cloud Controller started	Counter (Integer)

See the [Cloud Controller](#) topic for more information about the Cloud Controller.

Router Metrics

Pivotal Ops Metrics reports the number of sent requests and the number of completed requests for each Cloud Foundry component.

The difference between these two metrics is the number of requests made to a component but not completed, and represents the pending activity for that component. The number for each component can vary over time, and is typically higher under load. In any given environment, though, you can establish a typical range of values and maximum for this number for each component.

Use these metrics to ensure that the Router is passing requests to other components in a timely manner. If the pending activity for a particular component increase significantly past the typical maximum and stays at an elevated level, additional troubleshooting of that component may be necessary. If the pending activity for most or all components increases significantly and stays at elevated values, troubleshooting of the router may be necessary.

The following table shows the name of the Router metric, what the metric represents, and the metric type (data type).

METRIC NAME	DEFINITION	METRIC TYPE (DATA TYPE)
router.requests [component=c]	Number of requests the router has received for component c since this instance of the router has started c can be Cloud Controller or DEA	Counter (Integer)
router.responses [status=s,component=c]	Number of requests completed by component c since this instance of the router has started c can be Cloud Controller or DEA s is http status family: 2xx, 3xx, 4xx, 5xx, and other	Counter (Integer)

See the [Router](#) topic for more information about the Router.

Droplet Execution Agent Metrics

Pivotal Ops Metrics reports four data values for each Droplet Execution Agent (DEA) instance. If you have multiple DEA instances, the metrics reported are per DEA, and not the total across all of your DEAs.

Use these metrics to ensure that your system has enough disk space and memory to deploy and run your applications. For example, if `available_disk_ratio` or `available_memory_ratio` drop too low, the Cloud Controller will be unable to make reservations for a new application, and attempts to deploy that application will fail.

The following table shows the name of the DEA metric, what the metric represents, and the metric type (data type).

METRIC NAME	DEFINITION	METRIC TYPE (DATA TYPE)
available_disk_ratio	Percentage of disk available for allocation by future applications/staging requests	Gauge (Float, 0-1)
available_memory_ratio	Percentage of memory available for allocation by future applications/staging requests	Gauge (Float, 0-1)
mem_free_bytes	Current amount of memory free on the DEA	Gauge (Integer)
mem_used_bytes	Current amount of memory actually used, not just allocated, on the DEA	Gauge (Integer)

See the [Droplet Execution Agent](#) topic for more information about Droplet Execution Agents.

Virtual Machine Metrics

Pivotal Ops Metrics reports data for each virtual machine (VM) in a deployment. Use these metrics to monitor the health of your Virtual Machines.

The following table shows the name of the Virtual Machine metric, what the metric represents, and the metric type (data type).

METRIC NAME	DEFINITION	METRIC TYPE (DATA TYPE)
system.mem.percent	Percentage of memory used on the VM	Gauge (Float, 0-100)
system.swap.percent	Percentage of swap used on the VM	Gauge (Float, 0-100)
system.disk.ephemeral.percent	Percentage of ephemeral disk used on the VM	Gauge (Float, 0-100)
system.disk.system.percent	Percentage of system disk used on the VM	Gauge (Float, 0-100)
system.cpu.sys	Amount of CPU spent in system processes	Gauge (Float)
system.cpu.user	Amount of CPU spent in user processes	Gauge (Float)
system.cpu.wait	Amount of CPU spent in waiting processes	Gauge (Float)

Troubleshooting and Uninstalling Pivotal Ops Metrics

Page last updated:

The Pivotal Ops Metrics tool is a JMX extension for Elastic Runtime. Pivotal Ops Metrics collects and exposes system data from Cloud Foundry components via a JMX endpoint. Use this system data to monitor your installation and assist in troubleshooting.

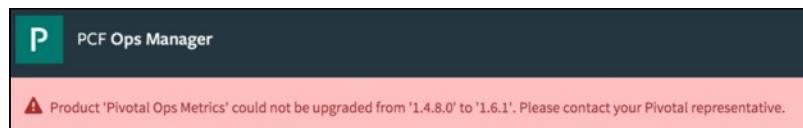
The Pivotal Ops Metrics tool is composed of three virtual machines:

- The JMX Provider
- A VM that governs compilation
- A nozzle for the [Loggregator Firehose](#)

To deploy Pivotal Ops Metrics, see the [Deploying Pivotal Ops Metrics](#) topic.

Resolve Upgrade Error for 1.4 to 1.6

If you see the following error during an upgrade:



1. [Uninstall Ops Metrics 1.4](#)
2. [Install Ops Metrics 1.6](#)

Uninstall Ops Metrics

1. Scale the collector resource from Elastic Runtime to .

Resource Config

JOB	INSTANCES	CPU	RAM (MB)	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)
Consul	1	1	1024	2048	1024
NATS	1	1	1024	2048	0
etcd	1	1	1024	2048	1024
Diego BBS	1	1	1024	2048	1024
NFS Server	1	1	1024	2048	10240
Router	1	1	1024	2048	0
MySQL Proxy	1	1	1024	4096	0
MySQL Server	1	2	8192	30000	10000
Cloud Controller Database (Postgres)	1	1	1024	2048	2048
UAA Database (Postgres)	1	1	1024	2048	8192
Apps Manager Database (Postgres)	1	1	1024	2048	1024
Cloud Controller	1	1	4096	20480	0
HAProxy	1	1	1024	2048	0
Health Manager	1	1	1024	2048	0
Clock Global	1	1	1024	2048	0
Cloud Controller Worker	1	1	1024	2048	0
Collector	0	1	1024	2048	0
UAA	1	1	1024	2048	0
Diego Brain	1	1	2048	4096	1024
Diego Cell	3	2	16384	65536	0
DEA	1	2	16384	32768	0
Doppler Server	1	1	1024	2048	0
Loggregator Trafficcontroller	1	1	1024	2048	0

If you do not scale the collector resource, you get the following error:

The screenshot shows a dark-themed interface for PCF Ops Manager. At the top, there is a green button with a white 'P' icon and the text 'PCF Ops Manager'. Below this, a red warning box contains the text: '⚠️ Pivotal Elastic Runtime requires 'p-metrics' version '~> 1.2' as a dependency.'

2. Proceed with uninstallation. See the [Deleting a Product](#) section of the [Adding and Deleting Products](#) topic for details.

Pivotal Cloud Foundry Troubleshooting Guide

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 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-IP-ADDRESS/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 `microbosh`. In this case, `microbosh` 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

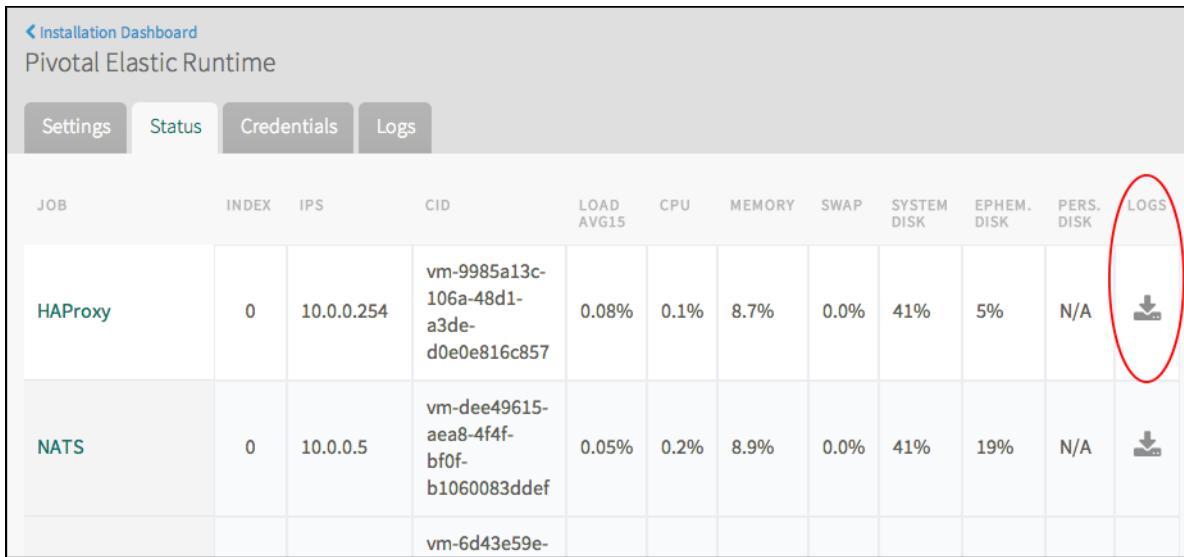
This section contains general tips for locating where a particular problem is called out in the log files. Refer to the later sections for tips regarding specific logs (such as those for Elastic Runtime Components).

- 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.



JOB	INDEX	IPS	CID	LOAD AVG15	CPU	MEMORY	SWAP	SYSTEM DISK	EPHEM. DISK	PERS. DISK	LOGS
HAProxy	0	10.0.0.254	vm-9985a13c-106a-48d1-a3de-d0e0e816c857	0.08%	0.1%	8.7%	0.0%	41%	5%	N/A	
NATS	0	10.0.0.5	vm-dee49615-aea8-4f4f-bf0f-b1060083ddef	0.05%	0.2%	8.9%	0.0%	41%	19%	N/A	

3. Browse to the **Pivotal Elastic Runtime > Logs** tab.



FILENAME	UPDATED AT
Downloaded:	
/var/tempest/.../dea-0-792ba29b2bc8.zip	2014-02-20 18:27:22 UTC
Pending:	
/var/tempest/.../cloud_controller-0-6d1150865104.zip	2014-02-20 18:27:18 UTC

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 DEA log directory contains subdirectories for the `dea_logging_agent`, `dea_next`, `monit`, and `warden` processes. To view the standard error stream for `warden`, download the DEA logs and open `dea.0.job > warden > warden.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:

```
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:

```
cd /var/tempest/workspaces/default/deployments/micro
```

7. You may want to begin by running a tail command on the `current` log:

```
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 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

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:

```
cloud_controller/0: Timed out pinging to f690db09-876c-475e-865f-2cece06aba79 after 600 seconds (00:10:24)
```

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**.
3. Become root.
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”:

```
2013-10-03_14:35:48.58456 #[608] INFO: Message: {"method"=>"ping", "arguments"=>[], "reply\_to"=>"director.f4b7df14-cb8f.19719508-e0dd-4f53-b755-58b6336058ab"},  
2013-10-03_14:35:48.60182 #[608] INFO: reply\_to: director.f4b7df14-cb8f.19719508-e0dd-4f53-b755-58b6336058ab:  
payload: {:value=>"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:

```
2013-10-03\ 14:35:21.83222 #[608] INFO: loaded new infrastructure settings:
{"vm"=>{"name"=>"vm-4d800de4-b0a5-4992-aea6a0386e18e", "id"=>"vm-360"}, 
 "agent\_id"=>"56aea4ef-6aa9-4c39-8019-7024ccfdde4",
 "networks"=>{"default"=>{"ip"=>"192.168.86.19",
 "netmask"=>"255.255.255.0", "cloud\_properties"=>{"name"=>"VMNetwork"}, 
 "default"=>["dns", "gateway"], 
 "dns"=>["192.168.86.2", "192.168.86.17"], "gateway"=>"192.168.86.2",
 "dns\_ record\_name"=>"0.nats.default.cf-d729343071061.microbosh",
 "mac"=>"00:50:56:9b:71:67"}, "disks"=>{"system"=>0, "ephemeral"=>1,
 "persistent"=>{}}, "ntp"=>[], "blobstore"=>{"provider"=>"dav",
 "options"=>{"endpoint"=>"http://192.168.86.17:25250",
 "user"=>"agent", "password"=>"agent"}, 
 "mbus"=>"nats://nats:nats@192.168.86.17:4222",
 "env"=>{"bosh"=>{"password"=>"$6$40ftQ9K4rvvC/8ADZHW0"}}}
```

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@192.168.86.17:4222"
```

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"}  

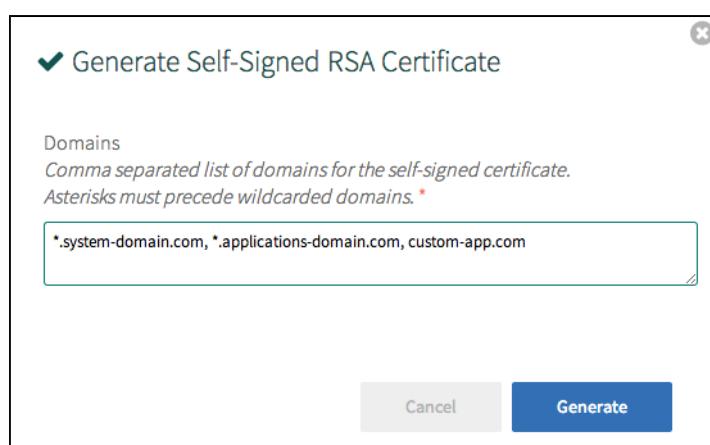
/home/tempest-web/tempest/web/vendor/bundle/ruby/1.9.1/gems/mechanize-2.7.2/lib/mechanize/http/agent.rb:306:in
`fetch': 403 => Net::HTTPForbidden for https://login.api.x.y/oauth/authorizeresponse_type=code&client_id=portal&redirect_uri=https%3...
-- unhandled response (Mechanize::ResponseCodeError)
```

Scenario 2: Your PCF install exits with a `creates/updates/deletes an app (FAILED - 1)` error message with the following stack trace:

```
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 interface IP.
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

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:

```
10.17.108.1 - Ops Manager (User assigned)  
10.17.108.2 - MicroBOSH (Ops Manager assigned)  
10.17.108.3 - Reserved (Ops Manager reserved)
```

To resolve this issue, ensure that the next two subsequent IP addresses from the manually assigned address are unassigned.

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

Page last updated:

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, [redacted].` 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.

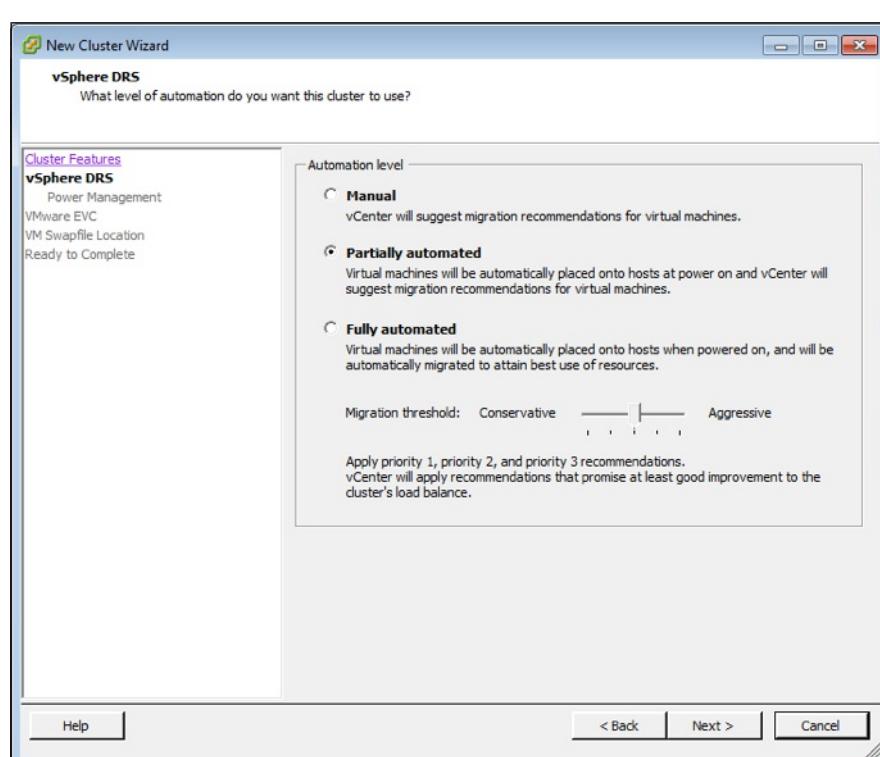
To resolve this issue, launch vCenter, navigate to **Administration > vCenter Server Settings > Statistics**, and reset the vCenter Statistics Interval Duration setting to 5 minutes.

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. To resolve this issue you must log in to the Ops Manager VM and edit `/etc/network/interfaces` as follows:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

# Generated with set_dynamic_ip script
# iface eth0 inet dhcp

# Manually configured
iface eth0 inet static
    address 10.70.128.16
    netmask 255.255.255.0
    network 10.70.128.0
    broadcast 10.70.128.255
    gateway 10.70.128.1

dns-nameservers 10.70.0.3
```

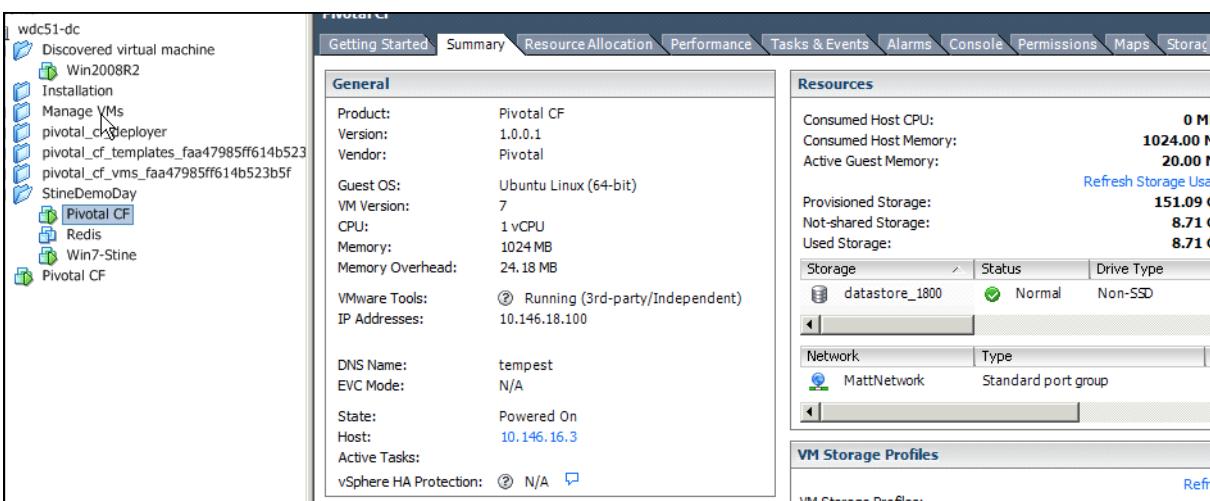
After editing the file run `sudo /etc/init.d/networking restart`. The interfaces files will be overwritten on reboot.

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.



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 ESX 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

Page last updated:

This topic describes the manual procedure for recovering a terminated Elastic Runtime cluster. In the event that all of the MySQL virtual machines (VMs) are terminated, the cluster will not automatically restart. For example, this process is necessary when all Elastic Runtime VMs are removed on AWS. Follow this procedure to recreate the lost VMs before completing the bootstrapping process to recover the cluster.

Prepare MySQL VMs for Bootstrapping

1. Run `bosh deployments` on the Ops Manager Director.

```
$ bosh deployments
Acting as user 'director' on 'p-bosh-30c19bdd43c55c627d70'

+-----+-----+-----+
| Name | Release(s) | Stemcell(s) | Cloud Config |
+-----+-----+-----+
| cf-e82cbf44613594d8a155 | cf-autoscaling/28 | bosh-aws-xen-hvm-ubuntu-trusty-go_agent/3140 | none |
| | cf-mysql/23 | | |
| | cf225 | | |
| | diego/0.1441.0 | | |
| | etcd/18 | | |
| | garden-linux/0.327.0 | | |
| | notifications-ui/10 | | |
| | notifications/19 | | |
| | push-apps-manager-release/397 | | |
+-----+-----+-----+
```

2. Download the manifest.

```
$ bosh download manifest cf-e82cbf44613594d8a155 /tmp/cf.yml
Acting as user 'director' on deployment 'cf-e82cbf44613594d8a155' on 'p-bosh-30c19bdd43c55c627d70'
Deployment manifest saved to '/tmp/cf.yml'
```

3. Set BOSH to use the deployment manifest you downloaded.

```
$ bosh deployment /tmp/cf.yml
```

4. Run `bosh vms` to check for placeholder VMs.

```
$ bosh vms
Acting as user 'director' on 'p-bosh-30c19bdd43c55c627d70'
Deployment 'cf-e82cbf44613594d8a155'

Director task 33

Task 33 done

+-----+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+-----+
| unknown/unknown | unresponsive agent | | |
| unknown/unknown | unresponsive agent | | |
| unknown/unknown | unresponsive agent | | |

```

 **Note:** If you do not see these three `unknown/unknown` items on your list, you might see the `mysql-partition` VMs, which indicates that your VMs were not destroyed. If that is the case, disregard the rest of the instructions in this section, Prepare MySQL VMs for Bootstrapping, and follow the instructions in the [Bootstrap](#) section below.

5. Run the BOSH Cloud Check interactive command `bosh cck` to delete the placeholder VMs. If given the option, select **Delete VM reference**.

```

$ bosh cck
Acting as user 'director' on deployment 'cf-e82cbf44613594d8a155' on 'p-bosh-30c19bdd43c55c627d70'
Performing cloud check...

Director task 34
Started scanning 22 vms
Started scanning 22 vms > Checking VM states. Done (00:00:10)
Started scanning 22 vms > 19 OK, 0 unresponsive, 3 missing, 0 unbound, 0 out of sync. Done (00:00:00)
  Done scanning 22 vms (00:00:10)

Started scanning 10 persistent disks
Started scanning 10 persistent disks > Looking for inactive disks. Done (00:00:02)
Started scanning 10 persistent disks > 10 OK, 0 missing, 0 inactive, 0 mount-info mismatch. Done (00:00:00)
  Done scanning 10 persistent disks (00:00:02)

Task 34 done

Started 2015-11-26 01:42:42 UTC
Finished 2015-11-26 01:42:54 UTC
Duration 00:00:12

Scan is complete, checking if any problems found.

Found 3 problems

Problem 1 of 3: VM with cloud ID 'i-afe2801f' missing.
  1. Skip for now
  2. Recreate VM
  3. Delete VM reference
Please choose a resolution [1 - 3]: 3

Problem 2 of 3: VM with cloud ID 'i-36741a86' missing.
  1. Skip for now
  2. Recreate VM
  3. Delete VM reference
Please choose a resolution [1 - 3]: 3

Problem 3 of 3: VM with cloud ID 'i-ce751b7e' missing.
  1. Skip for now
  2. Recreate VM
  3. Delete VM reference
Please choose a resolution [1 - 3]: 3

Below is the list of resolutions you've provided
Please make sure everything is fine and confirm your changes

  1. VM with cloud ID 'i-afe2801' missing.
    Delete VM reference

  2. VM with cloud ID 'i-36741a86' missing.
    Delete VM reference

  3. VM with cloud ID 'i-ce751b7e' missing.
    Delete VM reference

Apply resolutions? (type 'yes' to continue): yes
Applying resolutions...

Director task 35
Started applying problem resolutions
Started applying problem resolutions > missing_vm 11: Delete VM reference. Done (00:00:00)
Started applying problem resolutions > missing_vm 27: Delete VM reference. Done (00:00:00)
Started applying problem resolutions > missing_vm 26: Delete VM reference. Done (00:00:00)
  Done applying problem resolutions (00:00:00)

Task 35 done

Started 2015-11-26 01:47:08 UTC
Finished 2015-11-26 01:47:08 UTC
Duration 00:00:00
Cloudcheck is finished

```

6. Run `bosh edit deployment` to launch a `vi` editor and modify the deployment.

```
$ bosh edit deployment
```

- Search for the `jobs` section: `jobs`.
- Search for the `mysql-partition`: `name: mysql-partition`.
- Search for the `update` section: `update`.
- Update `max_in_flight` to `3`.

e. Add a line: `canaries: 0` below the `max_in_flight` line.

f. Set `update.serial` to `false`.

7. Run `bosh deploy` to apply these changes.

 **Note:** Review the changes for accuracy. Only type `yes` if the changes listed are correct.

```
Jobs
mysql-partition-f8abb6ac43ccc9fe16d7
  update
    ± max_in_flight:
      - 1
      + 3
    + canaries: 0

Properties
No changes

Please review all changes carefully

Deploying
-----
Are you sure you want to deploy? (type 'yes' to continue):
```

After you type `yes`, BOSH attempts to re-create the lost VMs:

```
Done creating bound missing vms > mysql-partition-f8abb6ac43ccc9fe16d7/0 (00:01:54)
Done creating bound missing vms > mysql-partition-f8abb6ac43ccc9fe16d7/2 (00:01:54)
Done creating bound missing vms > mysql-partition-f8abb6ac43ccc9fe16d7/1 (00:02:04)
```

Then BOSH fails to update the lost VMs. You can ignore the error message.

```
Failed updating job mysql-partition-f8abb6ac43ccc9fe16d7 > mysql-partition-f8abb6ac43ccc9fe16d7/1: 'mysql-partition-f8abb6ac43ccc9fe16d7/1' is not running after update (00:05:57)
Failed updating job mysql-partition-f8abb6ac43ccc9fe16d7 > mysql-partition-f8abb6ac43ccc9fe16d7/0: 'mysql-partition-f8abb6ac43ccc9fe16d7/0' is not running after update (00:05:59)
Failed updating job mysql-partition-f8abb6ac43ccc9fe16d7 > mysql-partition-f8abb6ac43ccc9fe16d7/2: 'mysql-partition-f8abb6ac43ccc9fe16d7/2' is not running after update (00:06:01)
Failed updating job mysql-partition-f8abb6ac43ccc9fe16d7 (00:06:01)
```

```
Error 400007: 'mysql-partition-f8abb6ac43ccc9fe16d7/1' is not running after update
```

8. Run `bosh vms` a second time to validate that you now have the VMs all in the failing state.

```
$ bosh vms
Acting as user 'director' on 'p-bosh-30c19bdd43c55c627d70'
Deployment 'cf-e82cbf44613594d8a155'

Director task 33

Task 33 done
+-----+-----+-----+
| Job/index | State | Resource Pool | IPs | +-----+
+-----+-----+-----+
| mysql-partition-f8abb6ac43ccc9fe16d7/0 | failing | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.12 | +-----+
| mysql-partition-f8abb6ac43ccc9fe16d7/1 | failing | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.60 | +-----+
| mysql-partition-f8abb6ac43ccc9fe16d7/2 | failing | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.61 | +-----+
```

Next, complete the bootstrapping process in the following section.

Bootstrap

 **Note:** Bootstrapping requires you to run commands from the [Ops Manager Director](#). Follow the instructions to use the [BOSH CLI](#) for command-line access.

Bootstrapping identifies the primary node to synchronize all of the nodes in a cluster. You must run all of the steps in this section to ensure successful future deployments and accurate reporting of the status of your jobs.

1. Follow the [Bootstrap](#) process to restart your MySQL cluster.
2. Run `bosh vms` to verify that your MySQL cluster is functioning as expected:

```
$ bosh vms
Acting as user 'director' on 'p-bosh-30c19bdd43c55c627d70'
Deployment `cf-e82cbf44613594d8a155'

Director task 33

Task 33 done
+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+
| mysql-partition-f8abb6ac43ccc9fe16d7/0 | running | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.12 |
| mysql-partition-f8abb6ac43ccc9fe16d7/1 | running | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.60 |
| mysql-partition-f8abb6ac43ccc9fe16d7/2 | running | mysql-partition-f8abb6ac43ccc9fe16d7 | 10.0.16.61 |
```

3. Run `bosh edit deployment` to launch a `vi` editor and reset the deployment manifest.

```
$ bosh edit deployment
```

- Set `update.canaries` to `1`.
- Set `update.max_in_flight` to `1`.
- Set `update.serial` to `true`.

4. Run `bosh deploy` to apply these changes.

Resolve Common Issues

- You may experience an error for too many key identities loaded in your authentication agent:

```
> Received disconnect from 10.0.1.19: 2: Too many authentication failures for bosh_64898ue98
```

If so, you can clear all of them using `ssh-add -D`.

- If you have an issue with not being able to `monit start`, it may be due to a bug in `mariadb_ctrl` that does not collect a defunct process. In that case, call Pivotal support for help. You can also see the [Known Issues](#) topic for more information.

Advanced Troubleshooting with the BOSH CLI

Page last updated:

This page assumes you are running cf CLI v6.

You must log into the BOSH Director and run specific commands using the BOSH Command Line Interface (CLI) to perform advanced troubleshooting.

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](#).

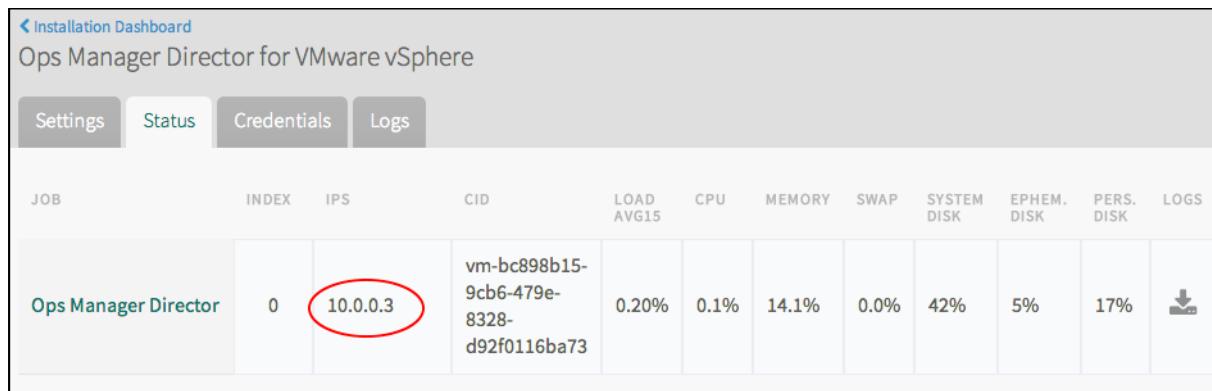
Prepare to Use the BOSH CLI

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. You must log out of the Ops Manager interface to use the BOSH CLI.

1. Open the Ops Manager interface. 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 Ops Manager Director job. This is the IP address of the VM where the BOSH Director runs.



Ops Manager Director for VMware vSphere											
JOB	INDEX	IPS	CID	LOAD AVG15	CPU	MEMORY	SWAP	SYSTEM DISK	EPHEM. DISK	PERS. DISK	LOGS
Ops Manager Director	0	10.0.0.3	vm-bc898b15-9cb6-479e-8328-d92f0116ba73	0.20%	0.1%	14.1%	0.0%	42%	5%	17%	

4. Select the **Credentials** tab.
5. Record the BOSH 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.

SSH into Ops Manager

Use SSH to connect to the Ops Manager web application VM.

To SSH into the Ops Manager VM:

vSphere:

You will 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-IP-ADDRESS`.
2. When prompted, enter the password that you set during the .ova deployment into vCenter:

```
$ ssh ubuntu@10.0.0.6
Password: *****
```

AWS and OpenStack:

1. Locate the Ops Manager public IP 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-MGR-IP
```

Log into the BOSH Director

1. To use the BOSH CLI without installing the BOSH CLI gem, set the `BUNDLE_GEMFILE` environment variable to point to the BOSH Gemfile for the Ops Manager interface. The following command creates an alias that sets `BUNDLE_GEMFILE` and calls `bundle exec bosh`. This alias allows you to run `bosh COMMAND` instead of `bundle exec bosh COMMAND`.

```
$ alias bosh="BUNDLE_GEMFILE=/home/tempest-web/tempest/web/bosh.Gemfile bundle exec bosh"
```

 **Note:** This alias and the manual change to the `BUNDLE_GEMFILE` environment variable last only until the end of the current session.

2. Verify that no BOSH processes are running on the Ops Manager VM. You should not proceed with troubleshooting until all BOSH processes have completed or you have ended them.
3. Run `bosh target OPS-MANAGER-DIRECTOR-IP-ADDRESS` to target your Ops Manager VM using the BOSH CLI.
4. Log in using the BOSH Director credentials:

```
$ bosh target 10.0.0.6
Target set to 'Ops Manager'
Your username: director
Enter password: *****
Logged in as 'director'
```

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:

```
find /var/tempest/workspaces/default/deployments -name cf-*.*yml
```

The table below shows the naming conventions for deployment files.

Product	Deployment Filename Convention
Elastic Runtime	cf-<20-character_random_string>.yml
MySQL Dev	cf_services-<20-character_random_string>.yml
Other	<20-character_random_string>.yml

 **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:

- Replace the `director_uuids` value with the UUID value.
- 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.

```
$ bosh vms
Deployment `cf-66987630724a2c421061'

Director task 99

Task 99 done

+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+
| unknown/unknown | running | push-apps-manager | 192.168.86.13 |
| unknown/unknown | running | smoke-tests | 192.168.86.14 |
| cloud_controller/0 | running | cloud_controller | 192.168.86.23 |
| collector/0 | running | collector | 192.168.86.25 |
| consoledb/0 | running | consoledb | 192.168.86.29 |
| dea/0 | running | dea | 192.168.86.47 |
| health_manager/0 | running | health_manager | 192.168.86.20 |
| loggregator/0 | running | loggregator | 192.168.86.31 |
| loggregator_router/0 | running | loggregator_router | 192.168.86.32 |
| nats/0 | running | nats | 192.168.86.19 |
| nfs_server/0 | running | nfs_server | 192.168.86.21 |
| router/0 | running | router | 192.168.86.16 |
| saml_login/0 | running | saml_login | 192.168.86.28 |
| syslog/0 | running | syslog | 192.168.86.24 |
| uaa/0 | running | uaa | 192.168.86.27 |
| uaadb/0 | running | uaadb | 192.168.86.26 |
+-----+-----+-----+
```

VMs total: 15
Deployment `cf_services-2c3c918a135ab5f91ee1'

Director task 100

Task 100 done

```
+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+
| mysql_gateway/0 | running | mysql_gateway | 192.168.86.52 |
| mysql_node/0 | running | mysql_node | 192.168.86.53 |
+-----+-----+-----+
```

VMs total: 2

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 ssh` 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 Resurrector 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 cloudcheck` 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

```
$ bosh clouddcheck
ccdb/0 (vm-3e37133c-bc33-450e-98b1-f86d5b63502a) is not responding:
- Ignore problem
- Reboot VM
- Recreate VM using last known apply spec
- Delete VM reference (DANGEROUS!)
```

Missing VM

```
$ bosh clouddcheck
VM with cloud ID `vm-3e37133c-bc33-450e-98b1-f86d5b63502a' missing:
- Ignore problem
- Recreate VM using last known apply spec
- Delete VM reference (DANGEROUS!)
```

Unbound Instance VM

```
$ bosh clouddcheck
VM `vm-3e37133c-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

```
$ bosh clouddcheck
VM `vm-3e37133c-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. ha_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. uaadb/0
14. uaa/0
15. login/0
16. consoledb/0
17. dea/0
18. loggregator/0
19. loggregator_traffic_controller/0
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
```

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.
 - b. Adding a new post to <http://www.pivotal.io/security>.

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.

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, a discussion of key concepts, and a glossary of terms. Refer to this guide to learn more about Cloud Foundry fundamentals.

General Concepts

- [Cloud Foundry Subsystems Overview](#)
- [How Applications are Staged](#)
- [Zero Downtime Deployment and Scaling in CF](#)
- [Orgs, Spaces, Roles, and Permissions](#)
- [Understanding Cloud Foundry Security](#)
- [Stacks](#)
- [Using Docker in Cloud Foundry](#)
- [Glossary](#)
- [Understanding Application Security Groups](#)

Architecture

- [Cloud Foundry Components](#)
- [Cloud Controller](#)
- [Droplet Execution Agent](#)
- [Messaging \(NATS\)](#)
- [\(Go\)Router](#)
- [User Account and Authentication \(UAA\) Server](#)
- [Warden](#)

Diego

- [Diego Architecture](#)
- [Differences between DEA and Diego Architectures](#)
- [Understanding Application SSH](#)
- [How the Diego Auction Allocates Jobs](#)

Cloud Foundry Subsystems Overview

Page last updated:

The Industry-Standard Cloud Platform

Cloud platforms let anyone deploy network applications 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.

Not all cloud platforms are created equal. Some have limited language and framework support, lack key application 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 application services, or cloud.

Cloud Foundry is ideal for anyone interested in removing the cost and complexity of configuring infrastructure for their applications. Developers can deploy their applications to Cloud Foundry using their existing tools and with zero modification to their code.

Cloud Foundry Subsystems

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 applications and other processes on the cloud's VMs, balancing demand and managing app lifecycles.
3. The [\(Go\)router](#) routes incoming traffic from the world to the VMs that are running the applications that the traffic demands, usually working with a customer-provided load balancer.

Cloud Foundry designates two types of VMs: the Component VMs that constitute the platform's infrastructure, and the Application VMs that host apps for the outside world. Within CF, the Diego system distributes the hosted app load over all of the Application 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 Application VMs run duplicate instances of the same application. This means that apps must be portable. Cloud Foundry distributes application source code to VMs with everything the VMs need to compile and run the apps locally. This includes the OS [stack](#) that the application 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.

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.

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 applications, custom configurations, and other resources. To store large binary files, such as droplets, CF maintains an internal Blob Store. To store and share temporary information, such as internal component states, CF uses the distributed value-store systems [Consul](#) and [etcd](#).

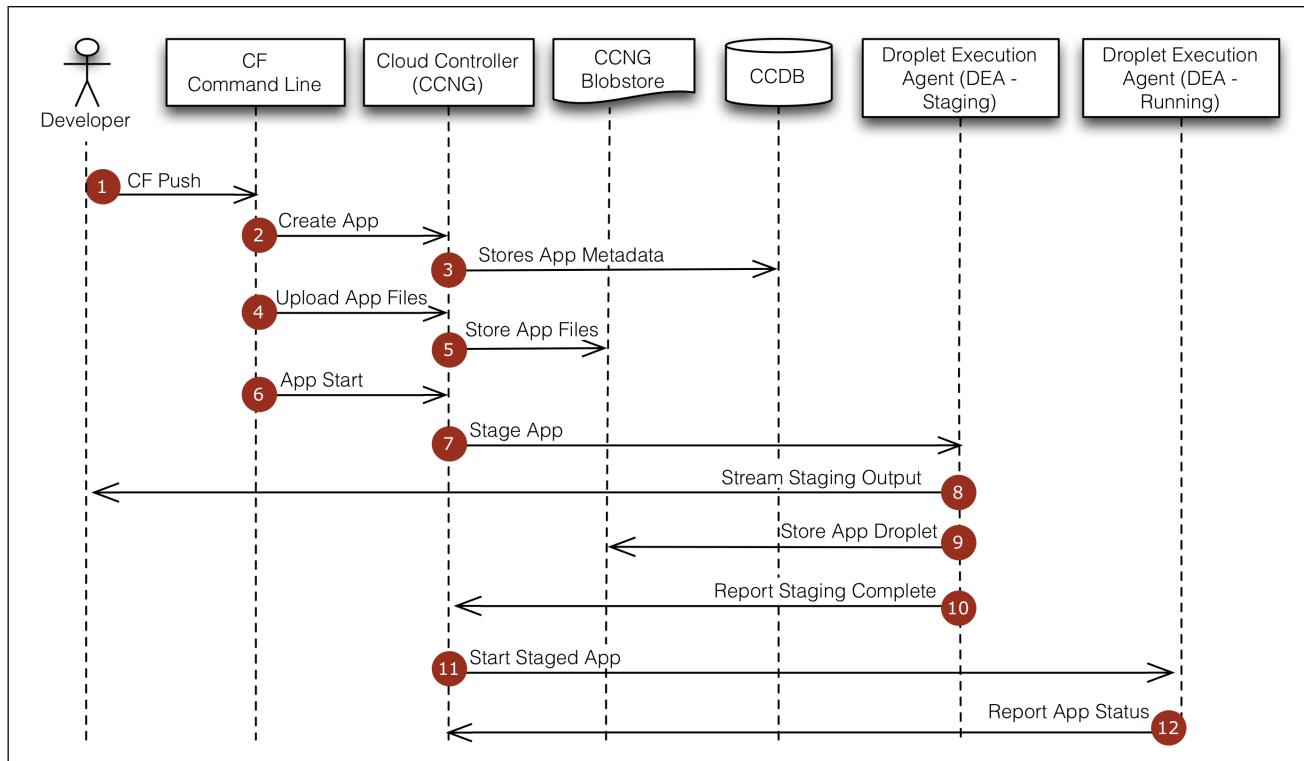
Cloud Foundry components communicate with each other by posting messages internally using http and https protocols, and by sending [NATS](#) messages to each other directly.

As the cloud operates, the Cloud Controller VM, router VM, and all VMs running applications 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 a [nozzles](#).

Typical applications depend on free or metered [services](#) such as databases or third-party APIs. To incorporate these into an application, 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 applications to make calls out to it.

How Applications Are Staged

Page last updated:



Zero Downtime Deployment and Scaling in CF

Page last updated:

To increase the capacity and availability of the Cloud Foundry platform, and to decrease the chances of downtime, you can scale a deployment up using the strategies described below.

This topic also describes the requirements for a zero downtime deployment. A zero downtime deployment ensures that if individual components go down, your deployment continues to run.

For more information regarding zero downtime deployment, see the [Scaling Instances in Elastic Runtime](#) topic.

Zero Downtime Deployment

This section describes the required configurations for achieving a zero downtime deployment.

Application Instances

Deploy at least two instances of every application.

Components

Scale your components as described in the [Scaling Platform Availability](#) section below. Components should be distributed across two or more [availability zones](#) (AZs).

Space

Ensure that you allocate and maintain enough of the following:

- Free space on DEAs so that apps expected to deploy can successfully be staged and run.
- Disk space and memory in your deployment such that if one DEA is down, all instances of apps can be placed on the remaining DEAs.
- Free space to handle one AZ going down if deploying in multiple AZs.

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.

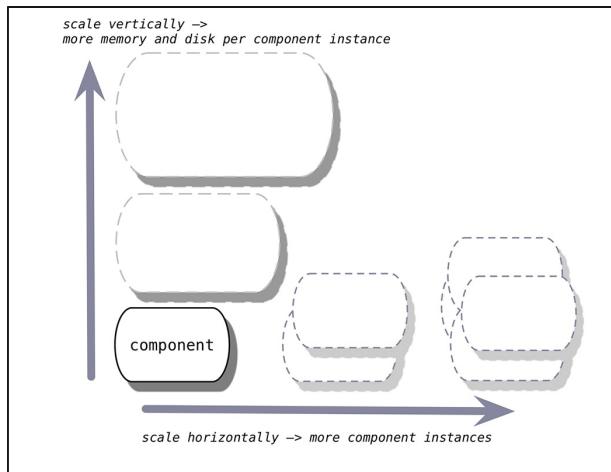
For [Amazon Web Services](#), review the documentation regarding scaling instances. If you are using OpenStack, see the topic regarding [managing projects and users](#). For vSphere, review the [Configuring Ops Manager Director for VMware vSphere](#) topic.

Ops Manager Resurrector

Enable the [Ops Manager Resurrector](#).

Scaling Platform Capacity

You can scale platform capacity vertically by adding memory and disk, or horizontally by adding more VMs running instances of Cloud Foundry components.



Trade-offs and Benefits

The nature of a particular application should determine whether you scale vertically or horizontally.

DEAs:

The optimal sizing and CPU/memory balance depends on the performance characteristics of the apps that will run on the DEA.

- The more DEAs are horizontally scaled, the higher the number of NATS messages the DEAs generate. There are no known limits to the number of DEA nodes in a platform.
- The denser the DEAs (the more vertically scaled they are), the larger the NATS message volume per DEA, as each message includes details of each app instance running on the DEA.
- Larger DEAs also make for larger points of failure: the system takes longer to rebalance 100 app instances than to rebalance 20 app instances.

Router:

Scale the router with the number of incoming requests. In general, this load is much less than the load on DEA nodes.

Health Manager:

The Health Manager works as a failover set, meaning that only one Health Manager is active at a time. For this reason, you only need to scale the Health Manager to deal with instance failures, not increased deployment size.

Cloud Controller:

Scale the Cloud Controller with the number of requests to the API and with the number of apps in the system.

Scaling Platform Availability

To scale the Cloud Foundry platform for high availability, the actions you take fall into three categories.

- For components that support multiple instances, increase the number of instances to achieve redundancy.
- For components that do not support multiple instances, choose a strategy for dealing with events that degrade availability.
- For database services, plan for and configure backup and restore where possible.

Note: Data services may have single points of failure depending on their configuration.

Scalable Processes

You can think of components that support multiple instances as scalable processes. If you are already scaling the number of instances of such components to increase platform capacity, you need to scale further to achieve the redundancy required for high availability.

If you are using DEAs:

Job	Number	Notes
Load Balancer	1	
MySQL Server	≥ 3	Set this to an odd number to support cluster quorum. For more information about cluster quorum, see Cluster Scaling, Node Failure, and Quorum .
MySQL Proxy	≥ 2	
NATS Server	≥ 2	If you lack the network bandwidth, CPU utilization, or other resources to deploy two stable NATS servers, we recommend that you use one NATS server.
HM9000	≥ 2	
Cloud Controller	≥ 2	More Cloud Controllers help with API request volume.
Gorouter	≥ 2	Additional Gorouters help bring more available bandwidth to ingress and egress.
Collector	1	
UAA	≥ 2	
DEA	≥ 3	More DEAs add application capacity.
Consul	≥ 3	This must be set to an odd number.
Doppler Server (formerly Loggregator Server)	≥ 2	Deploying additional Doppler servers splits traffic across them. We recommend you have at least two per Availability Zone.
Loggregator Traffic Controller	≥ 2	Deploying additional Loggregator Traffic Controllers allows you to direct traffic to them in a round-robin manner. We recommend you have at least two per Availability Zone.
etcd	≥ 3	Must be set to an odd number.

If you are using DIEGO:

Job	Number	Notes
Diego Cell	≥ 3	
Diego Brain	≥ 2	One per AZ, or two if only one AZ.
Diego BBS	≥ 3	This must be set to an odd number.
DEA	0	
Consul	≥ 3	This must be set to an odd number.
MySQL Server	≥ 3	This must be set to an odd number.
MySQL Proxy	≥ 2	
NATS Server	≥ 2	If you lack the network bandwidth, CPU utilization, or other resources to deploy two stable NATS servers, we recommend that you use one NATS server.
Cloud Controller	≥ 2	Cloud Controllers help with API request volume.
Router	≥ 2	Additional Gorouters help bring more available bandwidth to ingress and egress.
UAA	≥ 2	
Doppler Server	≥ 2	Deploying additional Doppler servers splits traffic across them. We recommend to have at least two per Availability Zone.
Loggregator TC	≥ 2	Deploying additional Loggregator Traffic Controllers allows you to direct traffic to them in a round-robin manner. We recommend you have at least two per Availability Zone.
etcd	≥ 3	This must be set to an odd number.
Health Manager	0	

Single-node Processes

You can think of components that do not support multiple instances as single-node processes. Since you cannot increase the number of instances of these components, you should choose a different strategy for dealing with events that degrade availability.

First, consider the components whose availability affects the platform as a whole.

HAProxy:

Cloud Foundry deploys with a single instance of HAProxy for use in lab and test environments. Production environments should use your own highly-available load balancing solution.

NATS:

You might run NATS as a single-node process if you lack the resources to deploy two stable NATS servers.

Cloud Foundry continues to run any apps that are already running even when NATS is unavailable for short periods of time. The components publishing messages to and consuming messages from NATS are resilient to NATS failures. As soon as NATS recovers, operations such as health management and router updates resume and the whole Cloud Foundry system recovers.

Because NATS is deployed by BOSH, the BOSH resurrector will recover the VM if it becomes non-responsive.

NFS Server:

For some deployments, an appropriate strategy would be to use your infrastructure's high availability features to immediately recover the VM where the NFS Server runs. In others, it would be preferable to run a scalable and redundant blobstore service. Contact our support if you need help.

Collector:

This component is not in the critical path for any operation.

Compilation:

This component is active only during platform installation and upgrades.

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.

Contact our support if you require replicated databases or any assistance.

Orgs, Spaces, Roles, and Permissions

Page last updated:

Cloud Foundry uses role-based access control (RBAC), with each role granting permissions in either an org or a space.

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.

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 Roles and Permissions

Org Manager

Assign this role to managers or other users who need to administer the account.

An Org Manager can do the following:

- Add and manage users
- View users and edit org roles
- View the org quota
- Create, view, edit, and delete spaces
- Invite and manage users in spaces
- View the status, number of instances, service bindings, and resource use of each application in every space in the org
- Add domains

 **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 Auditor

Assign this role to people who need to view but not edit user information and org quota usage information.

An Org Auditor can do the following:

- View users and org roles
- View the org quota

Billing Manager

Assign this role to people who need to create and manage billing account and payment information.

A Billing Manager can do the following:

- Set the org spending limit
- Create and set payment information
- Read invoices and payment history
- Create and edit the invoice notification email addresses

 **Note:** The Billing Manager role is only relevant for Cloud Foundry environments deployed with a billing engine.

Space Roles and Permissions

Space Manager

Assign this role to managers or other users who need to administer a space.

A Space Manager can do the following:

- Add and manage users in the space
- View the status, number of instances, service bindings, and resource use of each application in the space

Space Developer

Assign this role to application developers or other users who need to manage applications and services in a space.

A Space Developer can do the following:

- Deploy an application
- Start or stop an application
- Rename an application
- Delete an application
- Create, view, edit, and delete services in a space
- Bind or unbind a service to an application
- Rename a space
- View the status, number of instances, service bindings, and resource use of each application in the space
- Change the number of instances, memory allocation, and disk limit of each application in the space
- Associate an internal or external URL with an application

Space Auditor

Assign this role to people who need to view but not edit the space.

A Space Auditor can do the following:

- View the status, number of instances, service bindings, and resource use of each application in the space

Understanding Cloud Foundry Security

Page last updated:

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:

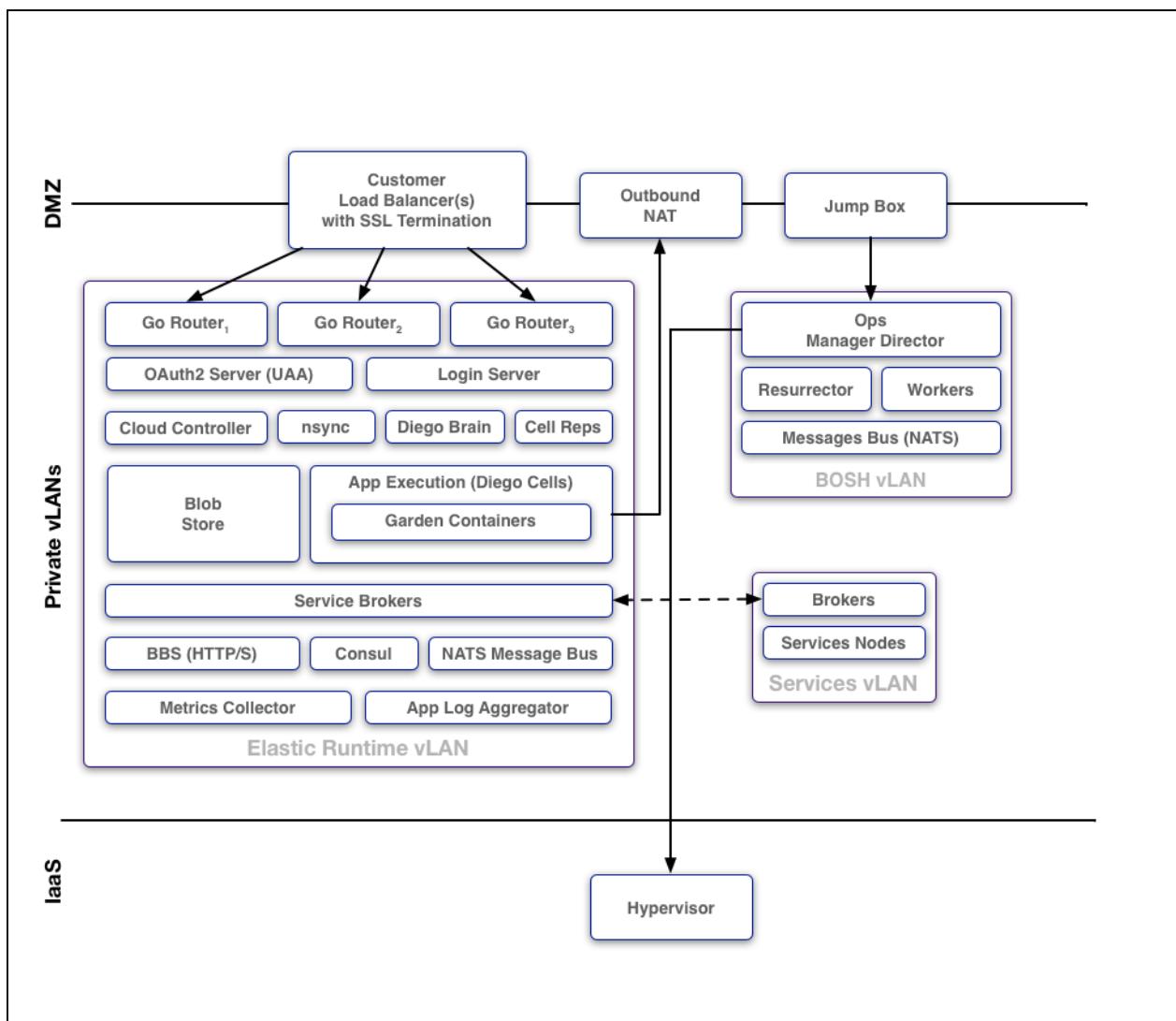
- Implements role-based access controls, applying and enforcing roles and permissions to ensure that users can only view and affect the spaces for which they have been granted access.
- Ensures security of application bits in a multi-tenant environment.
- Prevents possible denial of service attacks through 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 [OAuth2](#) 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 [Understanding Apps Manager Permissions](#) and [Managing User Spaces in Accounts 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.

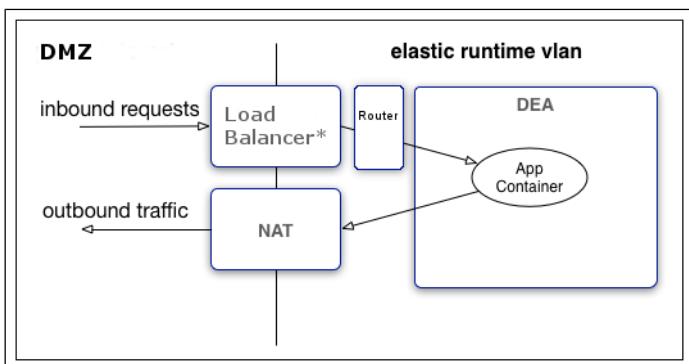
This restrictive operating environment is designed for security and stability. If a spoofing attack bypasses the physical firewall for your deployment, Cloud Foundry [network traffic rules](#) help prevent the attack from accessing application containers.

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.



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, using Linux IPTables. 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.

For Linux containers, this 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 Linux 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](#)). Each container is placed in its own cgroup, which requires the container to use a fair share of CPU compared to the relative CPU share of other containers. This placement also determines the maximum amount of memory the container may use.

 **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 namespacing 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).

 **Note:** The selected third-party broker controls how securely to communicate managed service credentials.

A third-party broker might offer a dashboard client in its catalog. Dashboard clients require a text string defined as a `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 reduce 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.

Stacks

Page last updated:

A stack is a prebuilt root filesystem (rootfs) which works in tandem with a buildpack and is used to support running applications.

 **Note:** You must ensure compatibility for buildpacks on the stacks they are running against.

 **Note:** Docker apps do not utilize stacks.

Available Stacks

- The linux `cflinuxfs2` stack is derived from Ubuntu Trusty 14.04. Refer to the Github stacks page for supported [libraries](#).
- Use the windows cell `windows2012R2` stack for .NET apps.

Building Stacks

The scripts for building the available Cloud Foundry stacks and for building custom stacks reside in the GitHub [Stacks](#) project.

Restaging Applications

Stacks are regularly updated to address [CVEs](#). Apps will pick up those changes as new releases of Elastic Runtime are deployed. However, if your app is statically linking to something provided in the rootfs, it may require your app to be restaged to pick up the new changes.

1. Use the `cf stacks` command to list all 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. Use the `cf push APPNAME -s STACKNAME` to change your stack and restage your application. `-s STACKNAME` specifies a particular stack for your application. For example, specifying `-s windows2012R2` ensures that Windows-based applications are sent to a Windows-based cell. The default stack is `cflinuxfs2`. For example, running this command restages the app `testappmx1`:

```
$ cf push testappmx1 -s cflinuxfs2
Using stack cflinuxfs2...
OK
Creating app testappmx1 in org MY-ORG / space development as developer@example.com...
OK

Creating route testappmx1.cfapps.io...
OK

Binding testappmx1.cfapps.io to testappmx1...
OK

Uploading testappmx1...
Uploading app files from: /Users/rex/workspace/app_files
Uploading 13.4M, 544 files
Done uploading
OK

Starting app testappmx1 in org MY-ORG / space development as developer@example.com...
----> Downloaded app package (21M)
----> Buildpack version 1.3.0
----> Compiling Ruby/Rack
----> Using Ruby version: ruby-2.0.0
----> Installing dependencies using 1.7.12
    Running: bundle install --without development:test --path vendor/bundle --binstubs vendor/bundle/bin -j4 --deployment
    Fetching gem metadata from http://rubygems.org/ ...
    Installing rack-rewrite 1.5.0
    Installing rack 1.5.2
    Using bundler 1.7.12
    Installing ref 1.0.5
    Installing libv8 3.16.14.3
    Installing therubyracer 0.12.1
    Your bundle is complete!
    Gems in the groups development and test were not installed.
    It was installed into ./vendor/bundle
    Bundle completed (16.63s)
    Cleaning up the bundler cache.
##### WARNING:
No Procfile detected, using the default web server (webrick)

----> Uploading droplet (45M)

1 of 1 instances running

App started

OK

App testappmx1 was started using this command bundle exec rackup config.ru -p $PORT

Showing health and status for app testappmx1 in org MY-ORG / space development as developer@example.com...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: testappmx1.cfapps.io
last uploaded: Wed Apr 8 23:40:57 UTC 2015
  state      since      cpu    memory    disk
#0  running  2015-04-08 04:41:54 PM  0.0%  57.3M of 1G  128.8M of 1G
```

For API information, review the Stacks section of the [Cloud Foundry API Documentation](#).

Using Docker in Cloud Foundry

Page last updated:

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:

```
$ cf enable-feature-flag diego_docker
```

To disallow Docker-based applications, a Cloud Controller administrator can run the following command:

```
$ cf disable-feature-flag diego_docker
```

 **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.

Pushing a Docker image with the CF CLI

Follow these instructions to deploy updated or new Docker images using Cloud Foundry.

Ensure that you are running cf CLI 6.13.0 or a later version. See [Installing the cf Command Line Interface](#) for installation instructions.

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
```

 **Note:** See [Docker Support in CF + Diego](#) for information about pushing a docker image with an earlier version of the cf CLI.

Docker Caveats

This section contains known issues and limitations with running Docker images in Cloud Foundry.

- Diego only supports secure registries bearing certificates signed by a common Certificate Authority (CA). Diego does not currently support self-signed certificates.
- Diego currently only supports v2 Docker registries with Docker 1.6+.
- 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.

Glossary

Page last updated:

Term	Definition
API	Application Programming Interface
Availability Zone (AZ)	A functionally independent segment of network infrastructure, often correlated with geographical region, designated to increase availability and fault-tolerance. A cloud operator can select or assign AZs on platforms such as AWS and vSphere.
BOSH	BOSH is an open framework for managing the full development and deployment life cycle of large-scale distributed software applications.
CLI	Command Line Interface
DEA	Droplet Execution Agent. The DEA is the component in Elastic Runtime responsible for staging and hosting applications.
Domains	A domain is a domain name like <code>shared-domain.com</code> . Domains can also be multi-level and contain sub-domains like the “myapp” in <code>myapp.shared-domain.com</code> . Domain objects belong to an org and are not directly bound to apps.
Droplet	An archive within Elastic Runtime that contains the application ready to run on a DEA. A droplet is the result of the application staging process.
Managed Services	Services provided by third-parties, but integrated into Cloud Foundry via APIs so that Cloud Foundry users can provision reserved resources and credentials on demand. Also called Custom Services .
Management	You can manage spaces and orgs with the <code>cf</code> command line interface, the Cloud Controller API, and the Cloud Foundry Eclipse Plugin.
Org	An org is the top-most meta object within the Elastic Runtime infrastructure. Only an account with administrative privileges on a Elastic Runtime instance can manage its orgs.
Routes	A route, based on a domain with an optional host as a prefix, may be associated with one or more applications. For example, <code>myapp</code> is the host and <code>shared-domain.com</code> is the domain when using the route <code>myapp.shared-domain.com</code> . It is possible to have a route that represents <code>shared-domain.com</code> without a host. Routes are children of domains and are directly bound to apps.
Service	A “factory” which produces service instances.
Service Instance	A reserved resource provisioned by a service. The resource provisioned will differ by service; could be a database or an account on a multi-tenant application.
Spaces	An org can contain multiple spaces. You can map a domain to multiple spaces, but you can map a route to only one space.
Staging	The process in Elastic Runtime by which the raw bits of an application are transformed into a droplet that is ready to execute.
UAA	User Account and Authentication Service, which provides the technological basis for Dashboard Single Sign-On available to Cloud Foundry users when accessing pertinent Managed Services.
Warden	The mechanism for containerization on DEAs that ensures applications running on Elastic Runtime have a fair share of computing resources and cannot access either the system code or other applications running on the DEA.

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:

```
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”

- Platform-wide running ASG set, also called “default-running”
- Space-scoped staging ASG set
- Space-scoped running ASG set

The following table indicates the differences between the four sets.

When an ASG is bound to the...	the ASG rules are applied to...
Platform-wide staging ASG set	the staging lifecycle for all apps and tasks.
Platform-wide running ASG set	the running lifecycle for all app and task instances.
Space-scoped staging ASG set	the staging lifecycle for apps and tasks in a particular space.
Space-scoped running ASG set	the running lifecycle for app and task instances in a particular space.

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 bind-security-group` 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:

Attribute	Description	Notes
<code>protocol</code>	<code>tcp</code> , <code>udp</code> , <code>icmp</code> , or <code>all</code>	Required
<code>destination</code>	A single IP address, an IP address range like <code>192.0.2.0-192.0.2.50</code> , or a CIDR block to allow network access to	
<code>ports</code>	A single port, multiple comma-separated ports, or a single range of ports that can receive traffic. Examples: <code>443</code> , <code>80,8080,8081</code> , <code>8080-8081</code>	Required when <code>protocol</code> is <code>tcp</code> or <code>udp</code>
<code>code</code>	ICMP code	Required when <code>protocol</code> is <code>icmp</code>
<code>type</code>	ICMP type	Required when <code>protocol</code> is <code>icmp</code>
<code>log</code>	Set to <code>true</code> 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.	Logging is only supported when the protocol type is <code>tcp</code> .
<code>description</code>	An optional text field for operators managing security group rules	

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](#).

Task	For more information, see
1. Review the existing ASGs. If this is a new deployment, probably these are the Default ASGs alone.	View ASGs
2. Create new ASGs.	Create ASGs
3. Update the existing ASGs.	Update ASGs
4. Bind ASGs to an ASG set.	Bind ASGs
5. If you need to delete ASGs, unbind and delete them.	Unbind ASGs Delete ASGs

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:

Command	Output
<code>cf security-groups</code>	All ASGs
<code>cf staging-security-groups</code>	All ASGs applied to the platform-wide staging ASG set
<code>cf running-security-groups</code>	All ASGs applied to the platform-wide running ASG set
<code>cf security-group SECURITY-GROUP</code>	All rules in the ASG named <code>SECURITY-GROUP</code> , for example, <code>cf security-group dns</code>

 **Note:** You can also view ASGs in Apps Manager under the **Settings** tab of a space or an app.

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.

In the following example, `my-asg` is the name of a security group, and `~/workspace/my-asg.json` is the 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

 **Note:** Binding 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 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](#) 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](#) 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
```

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.

ASG	For access to
<code>dns</code>	DNS, either public or private
<code>public-networks</code>	Public networks, excluding IaaS metadata endpoints
<code>private-networks</code>	Private networks in accordance with RFC-1918
<code>load-balancers</code>	The internal Elastic Runtime load balancer and others
<code>internal-proxies</code>	Internal proxies
<code>internal-databases</code>	Internal databases

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 `public_networks` example below accounts for this recommendation.

Example `public_networks` ASG:

```
[  
 {  
 "destination": "0.0.0.0-9.255.255.255",  
 "protocol": "all"  
 },  
 {  
 "destination": "11.0.0.0-169.253.255.255",  
 "protocol": "all"  
 },  
 {  
 "destination": "169.255.0.0-172.15.255.255",  
 "protocol": "all"  
 },  
 {  
 "destination": "172.32.0.0-192.167.255.255",  
 "protocol": "all"  
 },  
 {  
 "destination": "192.169.0.0-255.255.255.255",  
 "protocol": "all"  
 }  
 ]
```

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.

Example `private_networks` ASG:

```
[  
 {  
 "protocol": "tcp",  
 "destination": "10.0.0.0-10.255.255.255",  
 "ports": "443"  
 },  
 {  
 "protocol": "tcp",  
 "destination": "172.16.0.0-172.31.255.255",  
 "ports": "443"  
 },  
 {  
 "protocol": "tcp",  
 "destination": "192.168.0.0-192.168.255.255",  
 "ports": "443"  
 }  
 ]
```

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.

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.

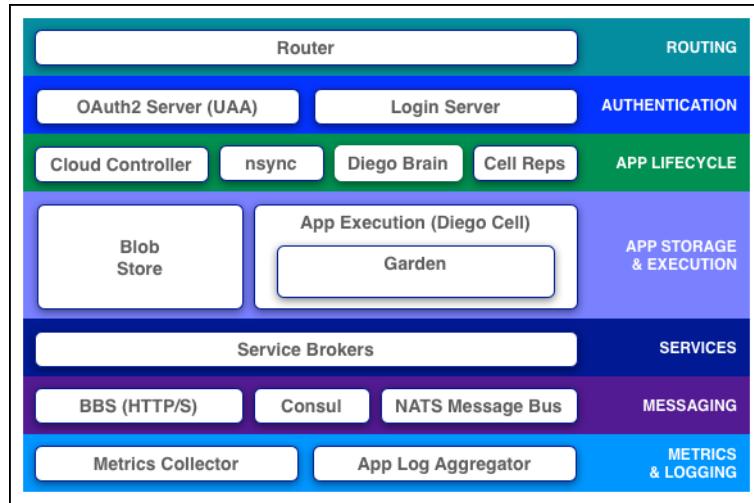
You can download the latest release of the ASG Creator from the Cloud Foundry incubator repository on Github: <https://github.com/cloudfoundry-incubator/asg-creator/releases/latest>

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.



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

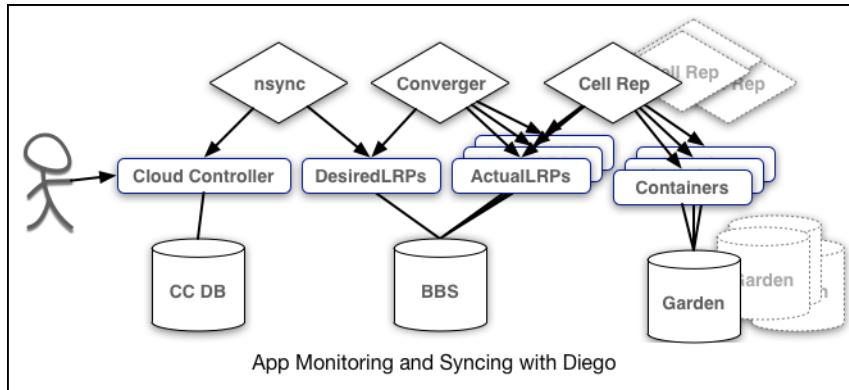
The [Cloud Controller](#) (CC) directs the deployment of applications. When a developer pushes an application to Cloud Foundry, she is targeting the Cloud Controller. The Cloud Controller then directs the Diego Brain through the [CC-Bridge](#) to coordinate individual [Diego cells](#) to stage and run applications.

In [pre-Diego architecture](#), the Cloud Controller's Droplet Execution Agent (DEA) performed these app lifecycle tasks.

The Cloud Controller also maintains records of [orgs](#), [spaces](#), [user roles](#), [services](#), and more.

nsync, Converger, 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, Converger, and Cell Reps use a more distributed approach.



These three components work along a spectrum of representations that extends between the user and the application containers. At one end, the user sets how each application should be scaled. At the other end, instances of that application running on widely-distributed VMs may crash or become unavailable.

The nsync, Converger, and Cell Rep components work together along this spectrum to keep apps running as they should. nsync watches for changes in application scaling instructions from the Cloud Controller and writes them into `DesiredLRP` structures in Diego's internal BBS database. Inside each cell, the Cell Rep watches the Garden for the state and health of the application instances running in the cell's containers, and updates corresponding `ActualLRP` values in the shared BBS as they change locally.

At the center of the process, the Diego Brain's Converger component monitors the `DesiredLRP` and `ActualLRP` values, and launches or kills application instances as appropriate to reconcile `ActualLRP` counts with `DesiredLRP` values.

App Storage and Execution

Blob Store

The blob store is a repository for large binary files, which GitHub cannot easily manage because GitHub is designed for code. Blob store binaries include:

- Application code packages
- Buildpacks
- Droplets

Diego Cell

Each application VM has a Diego Cell that executes application start and stop actions locally, manages the VM's containers, and reports app status and other data to the BBS and [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.

Cloud Controller

Page last updated:

The Cloud Controller provides REST API endpoints for clients to access the system. The Cloud Controller maintains a database with tables for orgs, spaces, services, user roles, and more.

Droplet Execution Agent (DEA) Placement Algorithm

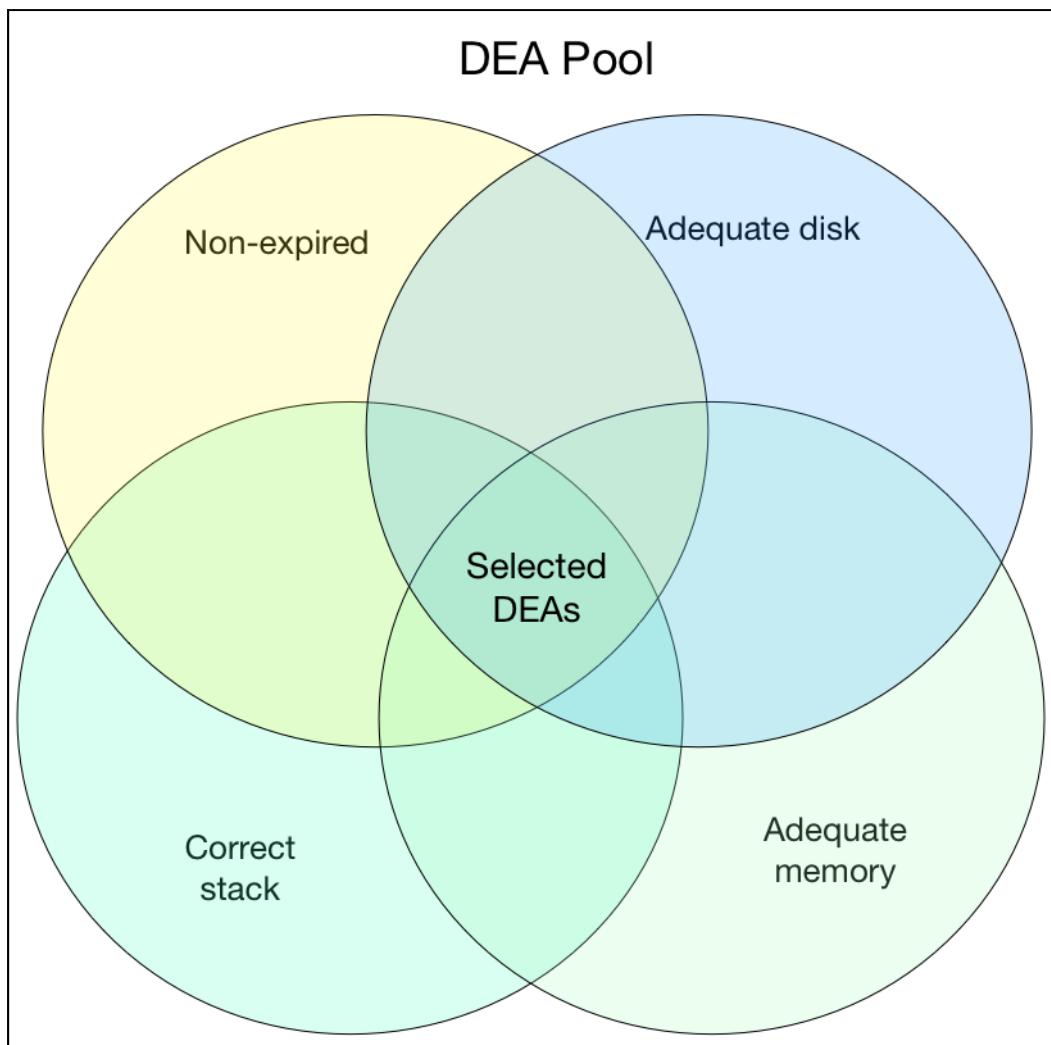
 **Note:** Pivotal Cloud Foundry® (PCF) versions 1.6 and above no longer use the DEA Placement Algorithm. The [Diego Auction](#) replaces its function. This section is published here to help people who are running older versions of PCF.

Whenever Cloud Foundry needs to spin up a new instance of an app, the Cloud Controller is responsible for [selecting a droplet execution agent \(DEA\) to run it](#).

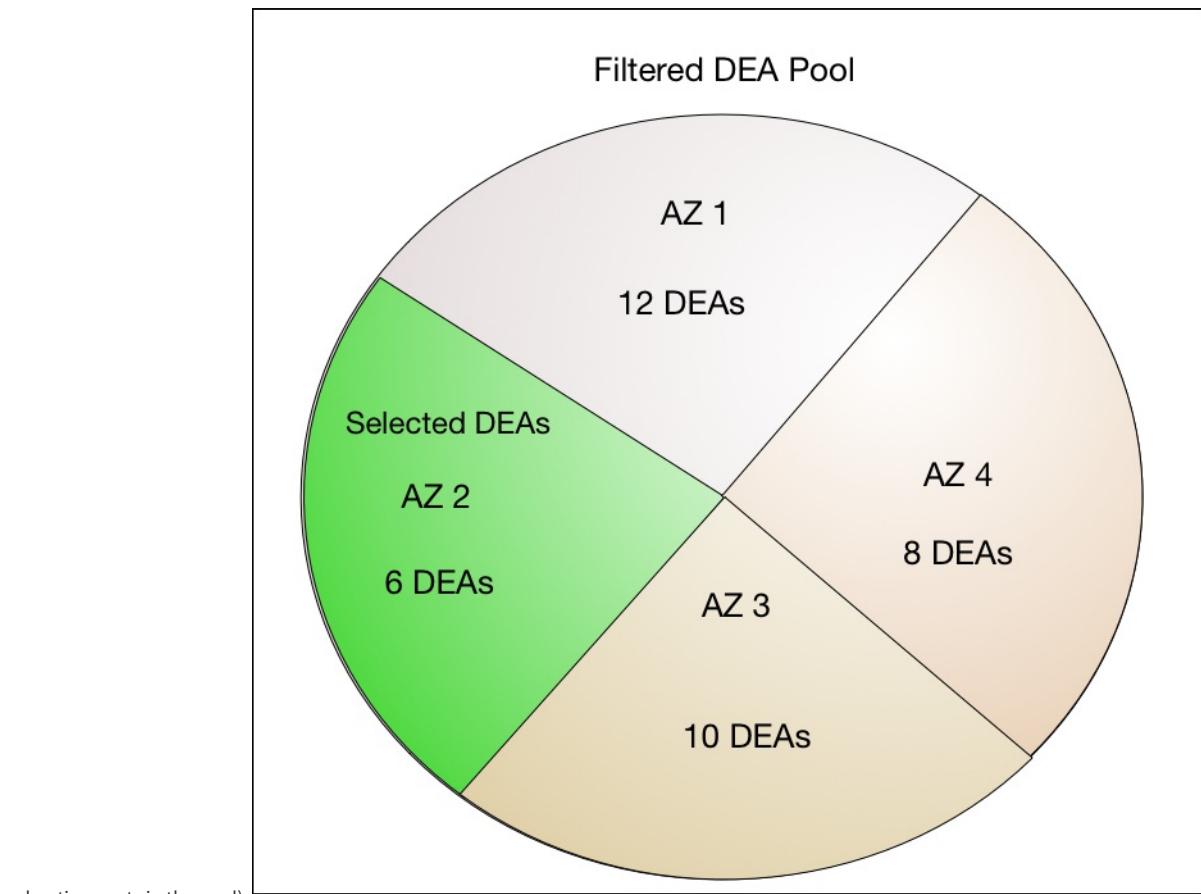
DEAs broadcast their availability to the Cloud Controller through a NATS message called an `advertisement`, which contains a `stats` hash with information about their available memory, available disk, the stack which the DEA runs, and an expiration time.

The Cloud Controller collects these advertisements in a construct called a pool. When the Cloud Controller needs to find a DEA to run an app, it runs through the following steps, using criteria (minimum thresholds for disk, memory, etc.) specific to the app that the chosen DEA will run:

1. It removes the expired DEA advertisements from the pool.
2. It filters the remaining advertisements to include only those:
 - with adequate disk
 - with adequate memory
 - running the required stack (linux or windows)

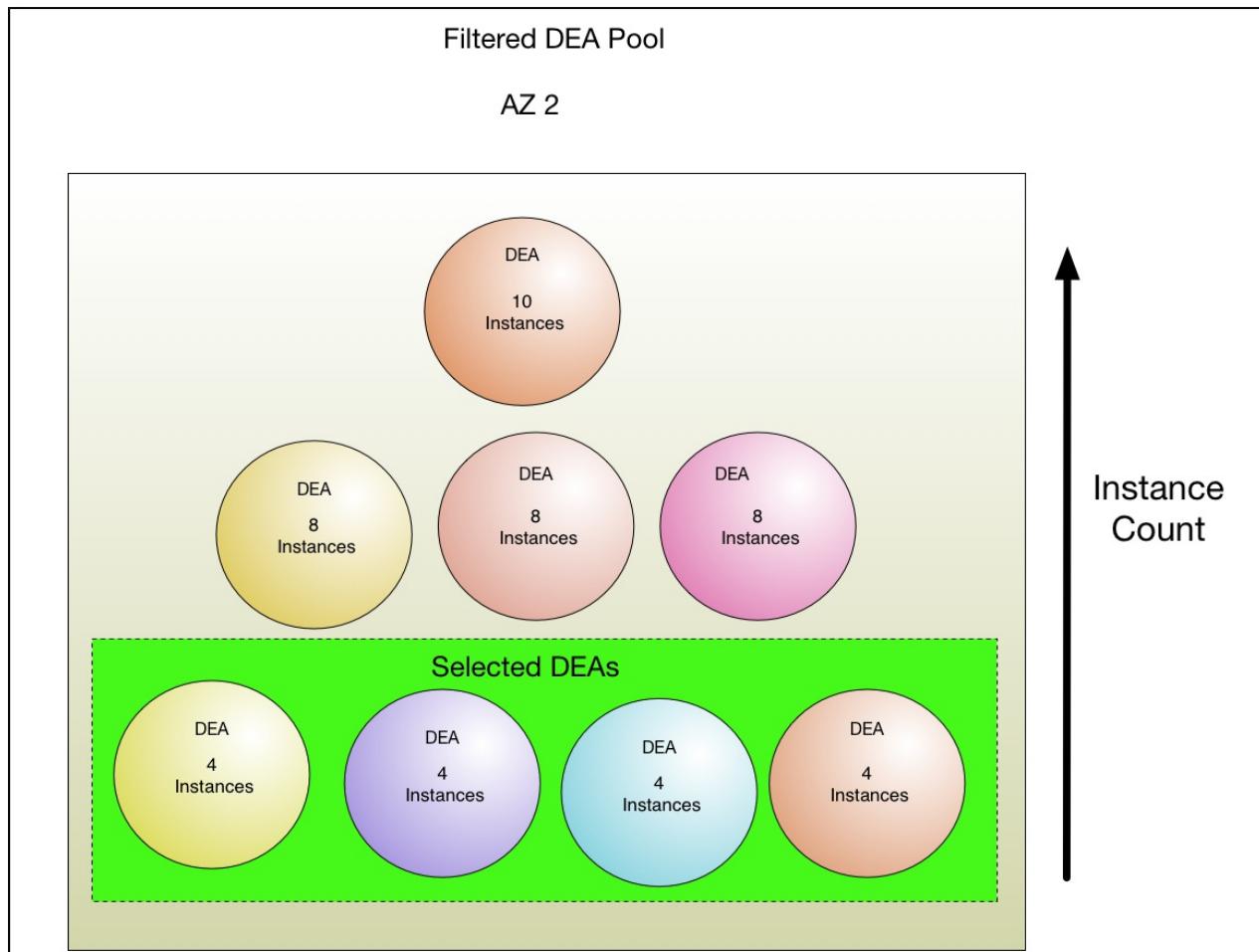


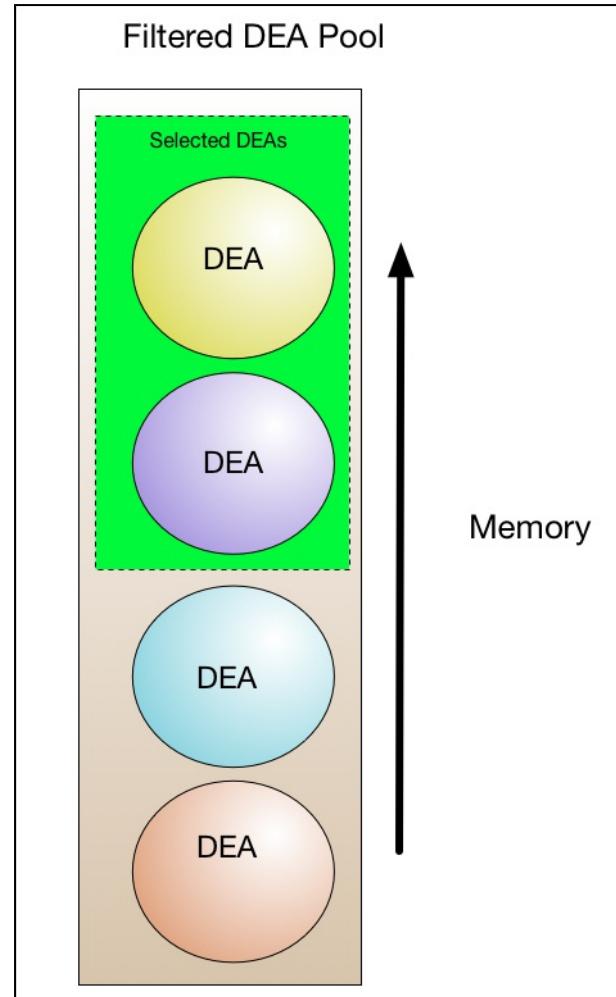
3. It then narrows its search to DEAs running in the availability zone with the fewest running instances (according to the information provided by the



advertisements in the pool).

4. It then narrows its search to the DEAs with the fewest running instances.





5. It then narrows its search to the top half of the DEAs, sorted by memory.

6. It then randomly selects one of the remaining DEAs.

Considerations

It is important to note that the Cloud Controller uses this algorithm to balance new app instances between DEAs when the new app instances are created, but do not balance already-running apps.

For example, suppose a set of apps are running on DEAs in two AZs, and one AZ temporarily goes down. While the second AZ is down, all instances will be placed on the remaining AZ. After the second AZ comes back online, new instances will be allocated to DEAs there, since the algorithm favors DEAs in the zone with the fewest running instances. However, instances running on the first AZ will not be moved to the other AZ, so the imbalance will persist.

An imbalance may also result from a deploy where DEAs have had a change to their source code or stemcell.

It is possible to rebalance the DEAs between AZs in two ways:

- Restarting the app, which may result in a brief down-time.
- Terminating and restarting half of the instances one by one.

Database (CC_DB)

The Cloud Controller database has been tested with Postgres.

Blob Store

The Cloud Controller manages a blob store for the following:

- **Resources:** Files that are uploaded to the Cloud Controller with a unique SHA such that they can be reused without re-uploading the file
- **App Packages:** Unstaged files that represent an application
- **Droplets:** Result of taking an app package, staging it by processing a buildpack, and preparing it to run

The blob store uses the [Fog](#) Ruby gem such that it can use abstractions like Amazon S3 or an NFS-mounted file system for storage.

NATS Messaging

The Cloud Controller interacts with other core components of the Cloud Foundry platform using the NATS message bus. For example, the Cloud Controller uses NATS when performing the following interactions:

- Instructing a DEA to stage an application by processes a buildpack for the app and to prepare it to run
- Instructing a DEA to start or stop an application
- Receiving information from the Health Manager about applications
- Subscribing to Service Gateways that advertise available services
- Instructing Service Gateways to handle provisioning, unprovisioning, binding, and unbinding operations for services

Testing

By default `rspec` runs a test suite with the SQLite in-memory database. Specify a connection string using the `DB_CONNECTION` environment variable to test against Postgres and MySQL. For example:

```
DB_CONNECTION="postgres://postgres@localhost:5432/ccng" rspec
DB_CONNECTION="mysql2://root:password@localhost:3306/ccng" rspec
```

Travis currently runs three build jobs against SQLite, Postgres, and MySQL.

Logs

Cloud Controller manages its logs using [Steno](#). Each log entry includes a `source` field to designate from which code module the entry originates. Some of the possible sources are `cc.app`, `cc.app_stager`, `cc.dea.client` and `cc.healthmanager.client`.

Configuration

The Cloud Controller uses a YAML configuration file. For an example, see `config/cloud_controller.yml`. Some of the keys the Cloud Controller reads from this configuration file include:

- `logging`: A [Steno configuration hash](#)
- `bulk_api` : - Basic auth credentials for the application state bulk API. In Cloud Foundry, the health manager uses this endpoint to retrieve the expected state of every user application.
- `uaa` : URL and credentials for connecting to the [UAA](#), the Cloud Foundry OAuth 2.0 server.

Droplet Execution Agent

Page last updated:

 **Note:** Pivotal Cloud Foundry® (PCF) versions 1.6 and above no longer use Droplet Execution Agents (DEAs). [Diego Cells](#) replace their function. This topic is published here to help people who are running older versions of PCF.

A Droplet Execution Agent (DEA) performs the following key functions:

- **Manage Warden containers:** The DEA stages applications and runs applications in [Warden](#) containers.
- **Stage applications:** When a new application or a new version of an application is pushed to Cloud Foundry, the Cloud Controller selects a DEA from the pool of available DEAs to stage the application. The DEA uses the appropriate buildpack to stage the application. The result of this process is a droplet.
- **Run droplets:** A DEA manages the lifecycle of each application instance running in it, starting and stopping droplets upon request of the [Cloud Controller](#). The DEA monitors the state of a started application instance, and periodically broadcasts application state messages over [NATS](#) for consumption by the [HM9000](#).

Directory Server

When the DEA receives requests for directories and files, it redirects them to the Directory Server URL. The URL is signed by the DEA, and the Directory Server checks the validity of the URL with the DEA before serving it.

Configurable Directory Server behaviors are controlled by keys in the DEA configuration file, `dea.yml`, described below in [DEA Configuration](#).

The Directory Server is written in Go and can be found in the `go/` directory of the DEA source code repository. It is a replacement for the older directory server that was embedded in the DEA itself.

DEA Health Checks

A DEA periodically checks the health of the applications running in it.

If a URL is mapped to an application, the DEA attempts to connect to the port assigned to the application. If the application port is accepting connections, the DEA considers that application state to be `Running`. If there is no URL mapped to the application, the DEA checks the system process table for the application process PID. If the PID exists, the DEA considers that application state to be `Running`.

The DEA also checks for a `AppState` object for the application.

Usage

You can run the `dea` executable at the command line by passing the path to the DEA configuration file:

```
$ bin/dea config/dea.yml
```

DEA Configuration

DEA behavior is configured in the `dea.yml` file.

The following is a partial list of the keys that the DEA reads from the YAML file:

- `logging` : A [Steno configuration](#)
- `nats_uri` : A URI of the form `nats://host:port` that the DEA uses to connect to NATS
- `warden_socket` : The path to a UNIX domain socket that the DEA uses to communicate to a Warden server.

A sample `dea.yml` file follows:

 Note: See https://github.com/cloudfoundry/dea_ng/blob/master/lib/dea/config.rb for optional configuration keys.

```
# See src/lib/dea/config.rb for optional config values.

# Base directory for dea. Application directories, DEA temp files, etc. are all relative to this.
base_dir: /tmp/dea_ng

logging:
  level: debug

resources:
  memory_mb: 2048
  memory_overcommit_factor: 2
  disk_mb: 2048
  disk_overcommit_factor: 2

nats_uri: nats://localhost:4222/

pid_filename: /tmp/dea_ng.pid

warden_socket: /tmp/warden.sock

evacuation_delay_secs: 10

index: 0

staging:
  enabled: true
  platform_config:
    cache: /var/vcap/data/stager/package_cache/ruby
  environment:
    PATH: /usr/local/ruby/bin
    BUILDPACK_CACHE: /var/vcap/packages/buildpack_cache
    memory_limit_mb: 1024
    disk_limit_mb: 2048
    max_staging_duration: 900 # 15 minutes

dea_ruby: /usr/bin/ruby

# For Go-based directory server
directory_server:
  v1_port: 4385
  v2_port: 5678
  file_api_port: 1234
  streaming_timeout: 10
logging:
  level: info

stacks:
  - lucid64

# Hook scripts for droplet start/stop
# hooks:
#   before_start: path/to/script
#   after_start: path/to/script
#   before_stop: path/to/script
#   after_stop: path/to/script
```

Running the DEA Standalone

When you contribute to the DEA, we recommend that you run it as a standalone component. Follow the instructions below for running the DEA as a standalone component on Vagrant 1.1x.

Refer to the [Vagrant documentation](#) for instructions on installing Vagrant.

```
$ git clone http://github.com/cloudfoundry/dea_ng
$ bundle install

# check that your version of vagrant is 1.1 or greater
$ vagrant --version

# create your test VM
$ rake test_vm
```

Failure of the `rake test_vm` step followed by an error similar to “undefined method ‘configure’ for Vagrant” or “found character that cannot start any token while scanning for the next token” can result from using an unsupported version of Vagrant. Ensure you have installed Vagrant version 1.1 or higher.

```
# initialize the test VM
$ vagrant up

# shell into the VM
$ vagrant ssh

# start warden
$ cd /warden/warden
$ bundle install
$ rvmsudo bundle exec rake warden:start[config/test_vm.yml] > /tmp/warden.log &

# start the DEA's dependencies
$ cd /vagrant
$ bundle install
$ git submodule update --init
$ foreman start > /tmp/foreman.log &

# run the dea tests
$ bundle exec rspec
```

Staging

See the [How Applications Are Staged](#) topic for a description of the application staging flow that the DEA uses.

A DEA publishes the following staging messages on NATS:

- `staging.advertise` : Stagers broadcast their capacity/capability.
- `staging.locate` : Stagers respond to any message on this subject with a `staging.advertise` message. CC uses this to bootstrap.
- `staging.UUID.start` : Stagers respond to requests on this subject to stage applications.
- `staging` : Stagers, in a queue group, respond to requests to stage an application. Note: This is an old protocol.

Logs

[Steno](#) handles the DEA logging. DEA logs to STDOUT by default. You can configure the DEA to log to a file, a syslog server, or both. The logging level specifies the verbosity of the logs, for example `warn`, `info`, or `debug`.

Messaging (NATS)

Page last updated:

This information was adapted from the [NATS README](#). NATS is a lightweight publish-subscribe and distributed queueing messaging system written in Ruby.

Install NATS

```
$ gem install nats
# or
$ rake geminstall

$ nats-sub foo &
$ nats-pub foo 'Hello World!'
```

Basic Usage

```
require "nats/client"

NATS.start do

  # Simple Subscriber
  NATS.subscribe('foo') { |msg| puts "Msg received: '#{msg}'" }

  # Simple Publisher
  NATS.publish('foo.bar.baz', 'Hello World!')


  # Unsubscribing
  sid = NATS.subscribe('bar') { |msg| puts "Msg received: '#{msg}'" }
  NATS.unsubscribe(sid)

  # Requests
  NATS.request('help') { |response| puts "Got a response: '#{response}'" }

  # Replies
  NATS.subscribe('help') { |msg, reply| NATS.publish(reply, "I'll help!") }

  # Stop using NATS.stop, exits EM loop if NATS.start started the loop
  NATS.stop

end
```

Wildcard Subscriptions

```
# "*" matches any token, at any level of the subject.
NATS.subscribe('foo.*.baz') { |msg, reply, sub| puts "Msg received on [#{sub}]: '#{msg}'" }
NATS.subscribe('foo.bar.*') { |msg, reply, sub| puts "Msg received on [#{sub}]: '#{msg}'" }
NATS.subscribe('*.*.bar.*') { |msg, reply, sub| puts "Msg received on [#{sub}]: '#{msg}'" }

# ">" matches any length of the tail of a subject and can only be the last token
# E.g. 'foo.>' will match 'foo.bar', 'foo.bar.baz', 'foo.foo.bar.baz.22'
NATS.subscribe('foo.>') { |msg, reply, sub| puts "Msg received on [#{sub}]: '#{msg}'" }
```

Queues Groups

```
# All subscriptions with the same queue name will form a queue group
# Each message will be delivered to only one subscriber per queue group, queueing semantics
# You can have as many queue groups as you want
# Normal subscribers will continue to work as expected.
NATS.subscribe(subject, :queue => 'job.workers') { |msg| puts "Received '#{msg}'" }
```

Advanced Usage

```
# Publish with closure, callback fires when server has processed the message
NATS.publish('foo', 'You done?') { puts 'msg processed!' }

# Timeouts for subscriptions
sid = NATS.subscribe('foo') { received += 1 }
NATS.timeout(sid, TIMEOUT_IN_SECS) { timeout_recd = true }

# Timeout unless a certain number of messages have been received
NATS.timeout(sid, TIMEOUT_IN_SECS, :expected => 2) { timeout_recd = true }

# Auto-unsubscribe after MAX_WANTED messages received
NATS.unsubscribe(sid, MAX_WANTED)

# Multiple connections
NATS.subscribe('test') do |msg|
  puts "received msg"
  NATS.stop
end

# Form second connection to send message on
NATS.connect { NATS.publish('test', 'Hello World!') }
```

(Go)Router

Page last updated:

The router routes traffic coming into Cloud Foundry to the appropriate component, whether it's an operator addressing the [Cloud Controller](#) or an application user accessing an app running on a Diego Cell. The router is implemented in Go. Implementing a custom router in Go gives the router full control over every connection, which makes it easier to support WebSockets and other types of traffic (for example, through HTTP CONNECT). A single process contains all routing logic, removing unnecessary latency.

Refer to the following instructions for help getting started with the Gorouter in a standalone environment.

Setup

```
$ git clone https://github.com/cloudfoundry/gorouter.git
$ cd gorouter
$ git submodule update --init
$ ./bin/go install router/router
$ gem install nats
```

Start

```
# Start NATS server in daemon mode
$ nats-server -d

# Start Gorouter
$ ./bin/router
```

Usage

Gorouter receives route updates via [NATS](#). By default, routes that have not been updated in two minutes are pruned. Therefore, to maintain an active route, ensure that the route is updated at least every two minutes. The format of these route updates is as follows:

```
{
  "host": "127.0.0.1",
  "port": 4567,
  "uris": [
    "my_first_url.vcap.me",
    "my_second_url.vcap.me"
  ],
  "tags": {
    "another_key": "another_value",
    "some_key": "some_value"
  }
}
```

Such a message can be sent to both the `router.register` subject to register URIs, and to the `router.unregister` subject to unregister URIs, respectively.

```
$ nohup ruby -rsinatra -e 'get("/") { "Hello!" }' &
$ nats-pub 'router.register' '{"host":"127.0.0.1","port":4567,
  "uris":["my_first_url.vcap.me","my_second_url.vcap.me"],
  "tags":{"another_key":"another_value","some_key":"some_value"}}'
Published [router.register] : '{"host":"127.0.0.1","port":4567,
  "uris":["my_first_url.vcap.me","my_second_url.vcap.me"],
  "tags":{"another_key":"another_value","some_key":"some_value"}}'
$ curl my_first_url.vcap.me:8080
Hello!
```

Instrumentation

Gorouter provides `/varz` and `/healthz` http endpoints for monitoring.

The `/routes` endpoint returns the entire routing table as JSON. Each route has an associated array of `host:port` entries.

All of the endpoints require http basic authentication, credentials for which you can acquire through NATS. You can explicitly set the `port`, `user` and `password` (`pass` is the config attribute) in the gorouter.yml config file `status` section.

```
status:  
port: 8080  
user: some_user  
pass: some_password
```

Example interaction with `curl`:

```
$ curl -vvv "http://someuser:somepass@127.0.0.1:8080/routes"  
* About to connect() to 127.0.0.1 port 8080 (#0)  
* Trying 127.0.0.1...  
* connected  
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)  
* Server auth using Basic with user 'someuser'  
> GET /routes HTTP/1.1  
> Authorization: Basic c29tZXVzZXI6c29tZXBhc3M=  
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5  
> Host: 127.0.0.1:8080  
> Accept: */*  
>  
< HTTP/1.1 200 OK  
< Content-Type: application/json  
< Date: Mon, 25 Mar 2013 20:31:27 GMT  
< Transfer-Encoding: chunked  
<  
{"0295dd314aaf582f201e655cbd74ade5.cloudfoundry.me":["127.0.0.1:34567"],  
"03e316d6aa375d1dc1153700da5f1798.cloudfoundry.me":["127.0.0.1:34568"]}
```

User Account and Authentication (UAA) Server

Page last updated:

The UAA is the identity management service for Cloud Foundry. Its primary role is as an OAuth2 provider, issuing tokens for client applications to use when they act on behalf of Cloud Foundry users. In collaboration with the login server, it can authenticate users with their Cloud Foundry credentials, and can act as an SSO service using those credentials (or others). It has endpoints for managing user accounts and for registering OAuth2 clients, as well as various other management functions.

Quick Start

Deploy Locally

To start, fetch the UAA from Git, then build and run all of the components that comprise the UAA and the example programs (uaa, samples/api, and samples/app) with a single invocation of gradle:

```
$ git clone git://github.com/cloudfoundry/uaa.git
$ cd uaa
$ ./gradlew run
```

If successful, the three apps will run together on a single instance of Tomcat listening on port 8080, with endpoints `/uaa`, `/app`, and `/api`.

Deploy to Cloud Foundry

You can also build the app and push it to Cloud Foundry:

```
$ ./gradlew :cloudfoundry-identity-uaa:war
$ cf push MYUAA -m 512M -p uaa/build/libs/cloudfoundry-identity-uaa-1.8.0.war --no-start
$ cf set-env MYUAA SPRING_PROFILES_ACTIVE default
$ cf start MYUAA
```

In the steps above, replace the following:

- `MYUAA` with a unique application name
- `1.8.0` with the appropriate version label from your build

Local Command Line Usage

1. Run the UAA server as described above:

```
$ cd uaa
$ ./gradlew run
```

2. Start another terminal. From the project base directory, query the login endpoint about the system:

```
$ curl -H "Accept: application/json" localhost:8080/uaa/login
{
  "timestamp": "2012-03-28T18:25:49+0100",
  "app": {"version": "1.8.3"},
  "commit_id": "cba0958",
  "prompts": {"username": ["text", "Email"],
              "password": ["password", "Password"]}
}
```

3. Log in with the UAA Ruby gem:

```
$ gem install cf-uaac
$ uaac target http://localhost:8080/uaa
$ uaac token get marissa koala
```

If you do not specify username and password, you will be prompted to supply them.

This authenticates and obtains an access token from the server using the OAuth2 implicit grant, similar to the approach intended for a standalone client like the Cloud Foundry Command Line Interface (cf CLI). The token is stored in `~/.uaac.yml`. Open `~/.uaac.yml` to obtain the access token for your `cf` target, or use `--verbose` on the login command line above to see it in the command shell.

4. Log in as a resource server and retrieve the token details:

```
$ uaac target http://localhost:8080/uaa
$ uaac token decode [token-value-from-above]
```

You should see your username and the client id of the original token grant on STDOUT:

```
jwt: e3a7d065-8514-43c2-b1f8-f8ab761791e2
sub: f796d1a2-eca3-4079-a317-f3224f8f7832
scope: scim.userids.password.write openid cloud_controller.write cloud_controller.read
client_id: cf
cid: cf
user_id: f796d1a2-eca3-4079-a317-f3224f8f7832
user_name: marissa
email: marissa@test.org
iat: 1413495264
exp: 1413538464
iss: http://localhost:8080/uaa/oauth/token
aud: scim openid cloud_controller password
```

Remote Command Line Usage

The command line example in the previous section should work against the UAA, although token encoding is unnecessary as you will not have the client secret.

In this case, there is no need to run a local uaa server. Query the external login endpoint about the system:

```
$ curl -H "Accept: application/json" uaa.run.pivotl.io/login
{
  "timestamp": "2014-09-15T18:25:04+0000",
  "app": {"version": "1.8.3"},
  "commit_id": "git-metadata-not-found",
  "prompts": {"username": ["text", "Email"], "password": ["password", "Password"]}
}
```

You can then try logging in with the UAA Ruby gem.

```
$ gem install cf-uaac  # If you have not already installed cf-uaac
$ uaac target https://uaa.run.pivotl.io
$ uaac token get [yourusername] [yourpassword]
```

If you do not specify a username and password, you will be prompted to supply them.

This authenticates and obtains an access token from the server using the OAuth2 implicit grant, which is the same grant that a client like the cf CLI uses.

Integration Tests

Run the integration tests with the following command:

```
$ ./gradlew integrationTest
```

This starts a uaa server running in a local Apache Tomcat instance. By default, the service URL is set to `http://localhost:8080/uaa`.

You can set the environment variable `CLOUD_FOUNDRY_CONFIG_PATH` to a directory containing a `uaa.yml` file in which the various URLs used in the tests

are changed, and the uaa server context root may be set.

Custom YAML Configuration

To modify the runtime parameters you can provide a `uaa.yml`, as shown in the following example:

```
$ cat > /tmp/uaa.yml
uaa:
  host: uaa.appcloud21.dev.mozycloud
  test:
    username: dev@cloudfoundry.org # defaults to vcap_tester@vmware.com
    password: changeme
    email: dev@cloudfoundry.org
```

Then from `uaa/uaa`:

```
$ CLOUD_FOUNDRY_CONFIG_PATH=/tmp ./gradlew test
```

The webapp looks for a YAML file in the following locations (later entries override earlier ones) when it starts up.

```
classpath:uaa.yml
file:${CLOUD_FOUNDRY_CONFIG_PATH}/uaa.yml
file:${UAACONFIGFILE}
${UAACONFIGURL}
```

Test with PostgreSQL or MySQL

The default UAA unit tests (`./gradlew test`) use hsqldb.

To run the unit tests using PostgreSQL:

```
$ echo "spring_profiles: default,postgresql" > src/main/resources/uaa.yml
$ ./gradlew test integrationTest
```

To run the unit tests using MySQL:

```
$ echo "spring_profiles: default,mysql" > src/main/resources/uaa.yml
$ ./gradlew test integrationTest
```

The database configuration for the common and scim modules is located at `common/src/test/resources/(mysql|postgresql).properties` and `scim/src/test/resources/(mysql|postgresql).properties`.

UAA Projects

There are several projects: the main `uaa` server application, and some samples:

- `common` is a module containing a JAR with all the business logic. It is used in the webapps below.
- `uaa` is the UAA server. `uaa` provides an authentication service plus authorized delegation for back-end services and apps by issuing OAuth2 access tokens.
- `api` (sample) is an OAuth2 resource service which returns a mock list of deployed apps. `api` is a service that provides resources that other applications might want to access on behalf of the resource owner.
- `app` (sample) is a user application that uses both of the above. `app` is a webapp that needs single sign-on and access to the `api` service on behalf of users.
- `scim` is the [SCIM](#) user management module used by UAA.

UAA Server

The authentication service is `uaa`. It is a plain Spring MVC webapp. Deploy as normal in Tomcat or your container of choice, or execute `./gradlew run` to run it directly from the `uaa` directory in the source tree. When running with Gradle, it listens on port 8080 and the URL is `http://localhost:8080/uaa`.

The UAA Server supports the APIs defined in the UAA-APIs document. To summarize:

1. The OAuth2 `/authorize` and `/token` endpoints
2. A `/login_info` endpoint to allow querying for required login prompts
3. A `/check_token` endpoint, to allow resource servers to obtain information about an access token submitted by an OAuth2 client.
4. SCIM user provisioning endpoint
5. OpenID connect endpoints to support authentication `/userinfo` and `/check_id` (todo). Implemented roughly enough to get it working (so `/app` authenticates here), but not to meet the spec.

Command line clients can perform authentication by submitting credentials directly to the `/authorize` endpoint. There is an `ImplicitAccessTokenProvider` in Spring Security OAuth that can do the heavy lifting if your client is Java.

By default, `uaa` will launch with a context root `/uaa`.

Configuration

There is a `uaa.yml` file in the application which provides defaults to the placeholders in the Spring XML. Wherever you see `${placeholder.name}` in the XML, there is an opportunity to override it either by providing a System property (`-D` to JVM) with the same name, or a custom `uaa.yml` (as described above).

All passwords and client secrets in the config files are plain text, but they will be inserted into the UAA database encrypted with BCrypt.

User Account Data

The default is to use an in-memory RDBMS user store that is pre-populated with a single test user: `marissa` with password `koala`.

To use Postgresql for user data, activate the Spring profile `postgresql`.

You can configure the active profiles in `uaa.yml` using:

```
spring_profiles: postgresql,default
```

To use PostgreSQL instead of HSQSL:

```
$ echo "spring_profiles: postgresql,default" > src/main/resources/uaa.yml
$ ./gradlew run
```

To bootstrap a microcloud-type environment, you need an admin client; there is a database initializer component that inserts one. If the default profile is active (i.e. not `postgresql`), there is also a cf CLI client so that the gem login works out of the box. You can override the default settings and add additional clients in `uaa.yml`:

```
oauth:
clients:
admin:
  authorized-grant-types: client_credentials
  scope: read,write,password
  authorities: ROLE_CLIENT,ROLE_ADMIN
  id: admin
  secret: adminclientsecret
  resource-ids: clients
```

You can use the admin client to create additional clients. You will need a client with read/write access to the `scim` resource to create user accounts. The integration tests take care of this automatically by inserting client and user accounts as necessary to make the tests work.

Sample Applications

Two sample applications are included with the UAA: `/api` and `/app`.

Run them with `./gradlew run` from the `uaa` root directory. All three apps, `/uaa`, `/api`, and `/app` get deployed simultaneously.

The API Sample Application

An example resource server. It hosts a service that returns a list of mock applications under `/apps`.

The App Sample Application

This is a user interface app (primarily aimed at browsers) that uses OpenId Connect for authentication (i.e. SSO) and OAuth2 for access grants. It authenticates with the Auth service, then accesses resources in the API service. Run it with `./gradlew run` from the `uaa` root directory.

The application can operate in multiple different profiles according to the location (and presence) of the UAA server and the Login application. By default, the application looks for a UAA on `localhost:8080/uaa`, but you can change this by setting an environment variable (or System property) called `UAA_PROFILE`. In the application source code (`samples/app/src/main/resources`), you will find multiple properties files pre-configured with different likely locations for those servers. They are all in the form `application-UAA_PROFILE.properties`. The naming convention adopted is that the `UAA_PROFILE` is `local` for the localhost deployment, `vcap` for a `vcap.me` deployment, and `staging` for a staging deployment. The profile names are double-barreled (e.g. `local-vcap` when the login server is in a different location than the UAA server).

Use Cases

1. See all apps

GET `/app/apps`

Browser is redirected through a series of authentication and access grant steps (which could be slimmed down to implicit steps not requiring user at some point), and then the photos are shown.

2. See the currently logged in user details, a bag of attributes grabbed from the open id provider

GET `/app`

Login Application

A user interface for authentication. The UAA can also authenticate user accounts, but only if it manages them itself, and it only provides a basic UI. You can brand and customize the login app for non-native authentication and for more complicated UI flows, like user registration and password reset.

The login application is itself an OAuth2 endpoint provider, but delegates those features to the UAA server. Therefore, configuration for the login application consists of locating the UAA through its OAuth2 endpoint URLs and registering the login application itself as a client of the UAA. There is a `login.yml` for the UAA locations, such as for a local `vcap` instance:

```
uaa:  
  url: http://uaa.vcap.me  
  token:  
    url: http://uaa.vcap.me/oauth/token  
  login:  
    url: http://uaa.vcap.me/login.do
```

There is also an environment variable (or Java System property), `LOGIN_SECRET`, for the client secret that the app uses when it authenticates itself with the UAA. The login app is registered by default in the UAA only if there are no active Spring profiles. In the UAA, the registration is located in the `oauth-clients.xml` config file:

```
id: login  
secret: loginsecret  
authorized-grant-types: client_credentials  
authorities: ROLE_LOGIN  
resource-ids: oauth
```

Use Cases

1. Authenticate

```
GET /login
```

The sample app presents a form login interface for the backend UAA, and also an OpenID widget so the user can authenticate using Google credentials or others.

2. Approve OAuth2 token grant

```
GET /oauth/authorize?client_id=app&response_type=code...
```

Standard OAuth2 Authorization Endpoint. The UAA handles client credentials and all other features in the back end, and the login application is used to render the UI (see `access_confirmation.jsp`).

3. Obtain access token

```
POST /oauth/token
```

Standard OAuth2 Authorization Endpoint passed through to the UAA.

Warden

Page last updated:

Warden manages isolated, ephemeral, and resource-controlled environments.

The primary goal of Warden is to provide a simple API for managing isolated environments. These isolated environments, or *containers*, can be limited in terms of CPU usage, memory usage, disk usage, and network access. The only currently supported OS is Linux.

 **Note:** Pivotal Cloud Foundry® versions 1.6 and above no longer use Warden. The platform-independent Garden subsystem replaces its function. This topic is published here to help people who are running older versions of PCF.

Components

The [Warden](#) repository in GitHub contains the following components:

- `warden` : Server
- `warden-protocol` : Protocol definition, used by both the server and clients
- `warden-client` : Client (Ruby)
- `em-warden-client` : Client (Ruby's EventMachine)

Getting Started

This guide assumes Ruby 1.9 and Bundler are already available. Ensure that Ruby 1.9 has GNU readline library support through the 'libreadline-dev' package and zlib support through the 'zlib1g-dev' package.

Install Kernel

If you are running Ubuntu 10.04 (Lucid), make sure the backported Natty kernel is installed. After installing, reboot the system before continuing.

```
$ sudo apt-get install -y linux-image-generic-lts-backport-natty
```

Install Dependencies

```
$ sudo apt-get install -y build-essential debootstrap
```

Set up Warden

Run the `setup` routine. The `setup` routine compiles the C code bundled with Warden and sets up the base file system for Linux containers.

```
$ sudo bundle exec rake setup[config/linux.yml]
```

 **Note:** If `sudo` complains that `bundle` cannot be found, use `sudo env PATH=$PATH` to pass your current `PATH` to the `sudo` environment.

The `setup` routine sets up the file system for the containers at the directory path specified under the key `server -> container_rootfs_path` in the config file `config/linux.yml`.

Run Warden

```
$ sudo bundle exec rake warden:start[config/linux.yml]
```

Interact with Warden

```
$ bundle exec bin/warden
```

Linux Implementation

Each application deployed to Cloud Foundry runs within its own self-contained environment, a Linux Warden container. Cloud Foundry operators can configure whether contained applications can or cannot directly interact with other applications or other Cloud Foundry system components.

Applications are typically allowed to invoke other applications in Cloud Foundry by leaving the system and re-entering through the Load Balancer positioned in front of the Cloud Foundry routers. The Warden containers isolate processes, memory, and the file system. Each Warden container also provides a dedicated virtual network interface that establishes IP filtering rules for the app, which are derived from network traffic rules. This restrictive operating environment is designed for security and stability.

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 are not aware of each other's presence. This means that these containers are given their own Process ID (PID), namespace, network namespace, and mount namespace.

Resource control is managed through the use of Linux Control Groups ([cgroups](#)). Every container is placed in its own control group, where it is configured to use a fair share of CPU compared to other containers and their relative CPU share, and the maximum amount of memory it may use.

CPU

Each container is placed in its own cgroup, which requires the container to use a fair share of CPU compared to the relative CPU share of other containers.

CPU shares do not work as direct percentages of total CPU usage. Instead, they provide a relative share of CPU usage in a given time window.

For example, if cgroup A has 10 shares and cgroup B has 30, and if their processes are both requesting lots of CPU usage, the kernel will grant $10/40 = 25\%$ time to A and $30/40 = 75\%$ time to B.

If B is idle, A may receive up to all the CPU. Shares per cgroup range from 2 to 1024, with 1024 the default. Both Diego apps and DEA apps scale the number of allocated shares linearly with the amount of memory, with an app instance requesting 8G of memory getting the upper limit of 1024 shares. Diego also guarantees a minimum of 10 shares per app instance.

This prevents oversubscription by a single application and the starving of CPU to other containers.

Networking

The DEA for each Warden container assigns a dedicated virtual network interface that is one side of a virtual ethernet pair created on the host. The other side of the virtual ethernet pair is only visible on the host from the root namespace. The pair is configured to use IPs in a small and static subnet. Traffic to and from the container is forwarded using NAT. Operators can configure global and container-specific network traffic rules that become Linux iptable rules to filter and log outbound network traffic.

Filesystem

Every container receives a private root filesystem. For Linux containers, this 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 Linux packages and Warden-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.

Difference with LXC

The *Linux Containers* or *LXC* project has goals that are similar to those of Warden: isolation and resource control. They both use the same Linux kernel primitives to achieve their goals. Early versions of Warden used LXC.

The major difference between the two projects is that LXC is explicitly tied to Linux, while you can implement Warden backends for any operating system that implements some way of isolating environments. It is a daemon that manages containers and can be controlled via a simple API rather than a set of tools that are individually executed.

While the Linux backend for Warden was initially implemented with LXC, the current version no longer depends on it. Because Warden relies on a very small subset of the functionality that LXC offers, we decided to create a tool that only implements the functionality we need in under 1k LOC of C code. This tool executes preconfigured hooks at different stages of the container start process, such that required resources can be set up without worrying about concurrency issues. These hooks make the start process more transparent, allowing for easier debugging when parts of this process are not working as expected.

Container Lifecycle

Warden manages the entire lifecycle of containers. The API allows users to create, configure, use, and destroy containers. Additionally, Warden can automatically clean up unused containers when needed.

Create

Every container is identified by its *handle*, which Warden returns upon creating it. The handle is a hexadecimal representation of the IP address that is allocated for the container. Regardless of whether the backend providing the container functionality supports networking, Warden allocates an IP address to identify a container.

Once a container is created and its handle is returned to the caller, it is immediately ready for use. All resources will be allocated, the necessary processes will be started and all firewalling tables will have been updated.

If Warden is configured to clean up containers after activity, it will use the number of connections that have referenced the container as a metric to determine inactivity. If the number of connections referencing the container drops to zero, the container will automatically be destroyed after a preconfigured interval. If in the mean time the container is referenced again, this timer is cancelled.

Use

The container can be used by running arbitrary scripts, copying files in and out, modifying firewall rules and modifying resource limits. Refer to the [Interface](#) section for a complete list of operations.

Destroy

When a container is destroyed, either per user request, or automatically after being idle, Warden first kills all unprivileged processes running inside the container. These processes first receive a `TERM` signal, followed several seconds later by a `KILL` signal if they have not yet exited. When these processes have terminated, Warden sends a `KILL` to the root of the container process tree. Once all resources the container used have been released, its files are removed and it is considered destroyed.

Networking

Interface

Warden uses a line-based JSON protocol to communicate with its clients, which it does over a Unix socket located at `/tmp/warden.sock` by default. Each command invocation is formatted as a JSON array, where the first element is the command name and subsequent elements can be any JSON object. Warden responds to the following commands:

create [CONFIG]

Creates a new container.

Returns the container handle, which is used to identify the container.

The optional `CONFIG` parameter is a hash that specifies configuration options used during container creation. The supported options are:

bind_mounts

If supplied, this specifies a set of paths to be bind mounted inside the container. The value must be an array. The elements in this array specify the bind mounts to execute, and are executed in order. Every element must be of the form:

```
[  
  # Path in the host filesystem  
  "/host/path",  
  
  # Path in the container  
  "/path/in/container",  
  
  # Optional hash with options. The 'mode' key specifies whether the bind  
  # mount should be remounted as 'ro' (read-only) or 'rw' (read-write).  
  {  
    "mode" => "ro|rw"  
  }  
]
```

grace_time

If specified, this setting overrides the default time period during which no client references a container before the container is destroyed. The value can either be the number of seconds as floating point number or integer, or the `null` value to completely disable the grace time.

disk_size_mb

If specified, this setting overrides the default size of the container scratch filesystem. The value is expected to be an integer number.

spawn HANDLE SCRIPT [OPTS]

Run the script `SCRIPT` in the container identified by `HANDLE`.

Returns a job identifier that can be used to reap the job exit status at some point in the future. Also, the connection that issued the command may disconnect and reconnect later, but still be able to reap the job.

The optional `OPTS` parameter is a hash that specifies options modifying the command being run. The supported options are:

privileged

If true, this specifies that the script should be run as root.

link HANDLE JOB_ID

Reap the script identified by `JOB_ID`, running in the container identified by `HANDLE`.

Returns a three-element tuple containing the integer exit status, a string containing its `STDOUT` and a string containing its `STDERR`. These elements may be `null` when they cannot be determined (e.g. the script could not be executed, was killed, etc.).

stream HANDLE JOB_ID

Stream `STDOUT` and `STDERR` of scripts identified by `JOB_ID`, running in the container identified by `HANDLE`.

Returns a two-element tuple containing the type of stream: `STDOUT` or `STDERR` as the first element, and a chunk of the stream as the second element.
Returns an empty tuple when no more data is available in the stream.

limit HANDLE (mem) [VALUE]

Set or get resource limits for the container identified by `HANDLE`.

The following resources can be limited:

- The memory limit is specified in number of bytes. It is enforced using the control group associated with the container. When a container exceeds this limit, the kernel kills one or more of its processes. Additionally, the Warden is notified that an OOM happened. The Warden subsequently tears down the container.

net HANDLE in

Forward a port on the external interface of the host to the container identified by `HANDLE`.

Returns the port number that is mapped to the container. This port number is the same on the inside of the container.

net HANDLE out ADDRESS[/MASK][:PORT]

Allow traffic from the container identified by `HANDLE` to the network address specified by `ADDRESS`. Additionally, the address may be masked to allow a network of addresses, and a port to only allow traffic to a specific port.

Returns `ok`.

copy HANDLE in SRC_PATH DST_PATH

Copy the contents at `SRC_PATH` on the host to `DST_PATH` in the container identified by `HANDLE`.

Returns `ok`.

File permissions and symbolic links are preserved, while hardlinks are materialized. If `SRC_PATH` contains a trailing `/`, only the contents of the directory are copied. Otherwise, the outermost directory, along with its contents, is copied. The unprivileged user will own the files in the container.

copy HANDLE out SRC_PATH DST_PATH [OWNER]

Copy the contents at `SRC_PATH` in the container identified by `HANDLE` to `DST_PATH` on the host.

Returns `ok`.

Its semantics are identical to `copy HANDLE in` except with respect to file ownership. By default, root owns the files on the host. If you supply the `OWNER` argument (in the form of `USER:GROUP`), files on the host will be chowned to this user/group after the copy has completed.

stop HANDLE

Stop processes running inside the container identified by `HANDLE`.

Returns `ok`.

Because this takes down all processes, unfinished scripts will likely terminate without an exit status being available.

destroy HANDLE

Stop processes and destroy all resources associated with the container identified `HANDLE`.

Returns `ok`.

Because everything related to the container is destroyed, artifacts from running an earlier script should be copied out before calling `destroy`.

Configuration

You can configure Warden by passing a configuration file when it starts. Refer to `config/linux.yml` in the repository for an example configuration.

System Prerequisites

Warden runs on Ubuntu 10.04 and higher.

A backported kernel needs to be installed on 10.04. This kernel is available as `linux-image-server-lts-backport-natty` (substitute `generic` for `server` if you are running Warden on a desktop variant of Ubuntu 10.04).

Other dependencies are:

- `build-essential` (for compiling the Warden C bits)
- `debootstrap` (for bootstrapping the base filesystem of the container)
- `quota` (for managing file system quotas)

Run `rake setup` for further Warden bootstrapping.

Hacking

The included tests create and destroy real containers, so they require system prerequisites to be in place. They need to be run as root if the backend to be tested requires it.

See `root/<backend>/README.md` for backend-specific information.

Diego Architecture

Page last updated:

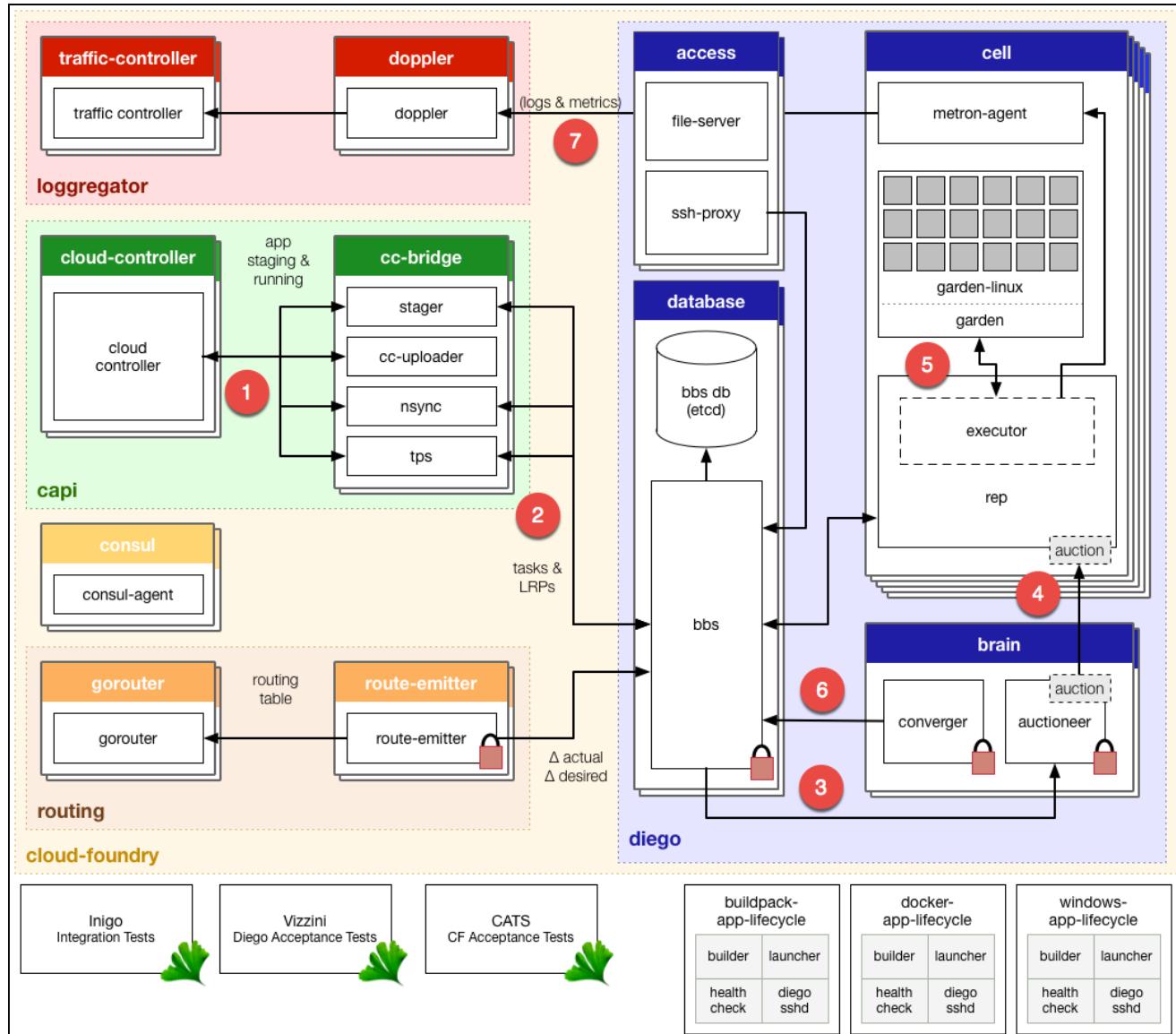
This topic provides an overview of the structure and components of Diego, the container management system for Pivotal Cloud Foundry versions 1.6 and newer.

See the [Using Diego in Pivotal Cloud Foundry](#) topic for information about using Diego in [Pivotal Cloud Foundry](#).

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.



View a larger version of this image at the [Diego Design Notes repo](#).

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 LRPCs to the [Auctioneer](#), part of the [Diego Brain](#).
4. The Auctioneer distributes these Tasks and LRPCs 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 LRPCs, running LRP instances, and in-flight Tasks for the [Converger](#), part of the Diego Brain. The Converger 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 LRPCs. The core consists of the following major areas:

- [Brain](#)
- [Cells](#)
- [Database VMs](#)
- [Access VMs](#)
- [Consul](#)

Diego Brain

Diego Brain components distribute Tasks and LRPCs 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 and the Converger.

Auctioneer

- Uses the [auction package](#) to run Diego Auctions for Tasks and LRPCs
- 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.

Converger

- Uses converge methods from the [Runtime Schema](#) to analyze snapshots from the BBS to ensure consistency and fault tolerance for Tasks and LRPCs
- Acts to keep [DesiredLRP](#) count and [ActualLRP](#) count synchronized in the following ways:
 - If the [DesiredLRP](#) count exceeds the [ActualLRP](#) count, the Converger requests a start auction from the Auctioneer
 - If the [ActualLRP](#) count exceeds the [DesiredLRP](#) count, the Converger sends a stop message to the Rep on the Cell hosting an instance
- Monitors for potentially missed messages, resending them if necessary
- Maintains a lock in the BBS that limits converge methods to one Converger at a time

Refer to the [Converger repo](#) on GitHub for more information.

Diego Cell Components

Diego Cell components manage and maintain Tasks and LRPCs.

Rep

- Represents a Cell in Diego Auctions for Tasks and LRPCs
- Mediates all communication between the Cell and the BBS
- Ensures synchronization between the set of Tasks and LRPCs in the BBS with the containers present on the Cell
- Maintains the presence of the Cell in the BBS
- Runs Tasks and LRPCs 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](#) 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](#) 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](#) 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 LRPCs, 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](#).

Refer to the [Bulletin Board System repo](#) on GitHub for more information.

etcd

- Provides a consistent key-value data store to Diego

Access VMs

File Server

- This “blob store” serves static assets that can include general-purpose [App Lifecycle binaries](#) and application-specific droplets and build artifacts.

Refer to the [File Server repo](#) 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](#), or the [Diego SSH GitHub repo](#) 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 LRPCs
- 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 LRPCs to respond to `cf apps` and `cf app APP_NAME` requests
- Monitors `ActualLRP` activity for crashes and reports them to 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 redeploys 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 underneath](#) 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” - 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] [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.
3. **Warden `deny` rules:** Any Warden Server configuration `deny` rules. Set Warden Server configuration rules in the 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, Converger, and Cell Rep

The function of the Health Monitor (HM9000) component in pre-Diego releases of Cloud Foundry was replaced by the coordinated actions of the [nsync, Converger, and Cell Reps](#). In pre-Diego architecture, the Health Monitor (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

DEA architecture	Diego architecture	Function	△ notes
1.5 and below	1.6 and above	CF version numbers	
Ruby	Go	Source code language	
DEA	Diego Brain	High-level coordinator that allocates processes to containers in application VMs and keeps them running	DEA is part of the Cloud Controller. Diego is outside the Cloud Controller.
DEA Node	Diego Cell	Mid-level manager on each VM that runs apps as directed and communicates “heartbeat”, application status and container location, and other messages	Runs on each VM that hosts apps, as opposed to special-purpose component VMs.
Warden	Garden	Low-level manager and API protocol on each VM for creating, configuring, destroying, monitoring, and addressing application containers	Warden is Linux-only. Garden uses platform-specific Garden-backends to run on multiple OS.
DEA Placement Algorithm	Diego Auction	Algorithm used to allocate processes to VMs	Diego Auction distinguishes between Task and Long-Running Process (LRP) job types
Health Manager	nSync, Converger, and Cell Reps	System that monitors application instances and keeps instance counts in sync with the number that should be	nSync syncs between Cloud Controller and Diego, Converger syncs within Diego, and Cell Reps sync

(HM9000) NATS Message Bus	Bulletin Board System (BBS) and Consul via http/s, and NATS	running Internal communication between components	between cells and the Diego BBS. BBS stores most runtime data; Consul stores control data.
---------------------------------	---	--	---

Understanding Application SSH

Page last updated:

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 LRPCs 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. LRPCs 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 LRPCs 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 LRPCs, 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:

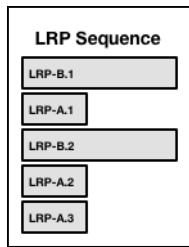
1. Keep at least one instance of each LRP running.
2. Run all of the Tasks in the current batch.
3. Distribute as much of the total desired LRP load as possible over the remaining available VMs, by spreading multiple LRP instances broadly across VMs and their [Availability Zones](#).

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 LRPCs, 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:

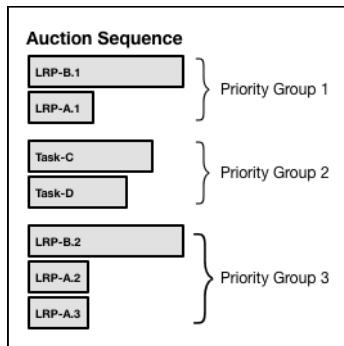
Process	Desired Instances	Load	Jobs
LRP-A	3	2	LRP-A.1 LRP-A.2 LRP-A.3
LRP-B	2	5	LRP-B.1 LRP-B.2

The Auctioneer then builds an ordered sequence of LRP instances by cycling through the list of LRPCs 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 LRPCs like this:



The Auctioneer then builds an ordered sequence for all jobs, both LRP instances and Tasks. Reflecting the auction batch [priority order](#), the first instances of LRP instances are first priority. Tasks are next, in decreasing order of load. Duplicate LRP instances come last.

Adding one-time Task-C (load = 4) and Task-D (load = 3) to the above example, the priority order becomes:



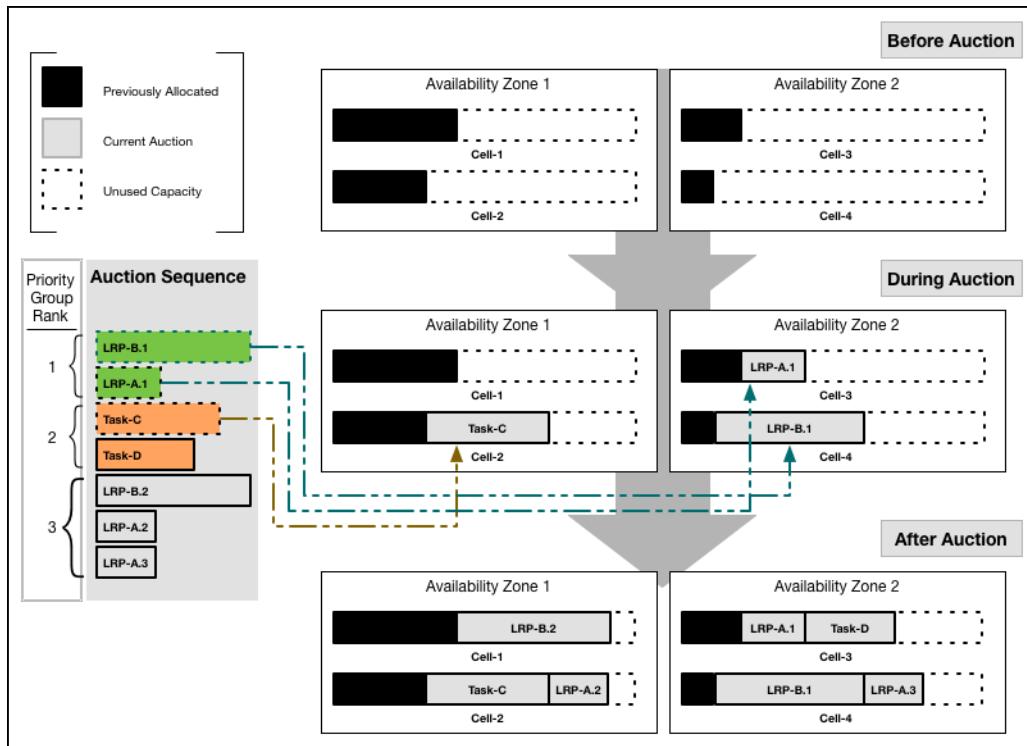
Auctioning the Batch to the Cells

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](#) initiates a new auction when it detects that the actual number of running instances of LRPCs 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 [Converger](#) 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 Converger triggers a new auction. In the case of a surplus of application instances, the Converger 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 Converger 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

- [Isolating a Pivotal Cloud Foundry Deployment with a Private Network](#)
- [Rotating Elastic Runtime MySQL Credentials](#)
- [Understanding the Elastic Runtime Network Architecture](#)
- [Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments](#)
- [Changing the Quota Plan of an Organization with cf CLI](#)
- [Restricting App Access to Internal PCF Components](#)
- [Identifying Elastic Runtime Jobs Using vCenter](#)
- [Understanding the Effects of Single Components on a Pivotal Cloud Foundry Upgrade](#)
- [Configuring Single Sign-On](#)
- [Configuring LDAP](#)
- [Switching Application Domains](#)
- [Configuring SSH Access for PCF](#)
- [Scaling Instances in Elastic Runtime](#)
- [Monitoring App and Service Instance Usage](#)
- [Using Diego in Pivotal Cloud Foundry](#)
- [Deploying Diego for Windows](#)
- [Troubleshooting Diego for Windows](#)
- [The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment](#)
- [Providing a Certificate for your SSL Termination Point](#)

Isolating a Pivotal Cloud Foundry Deployment with a Private Network

Page last updated:

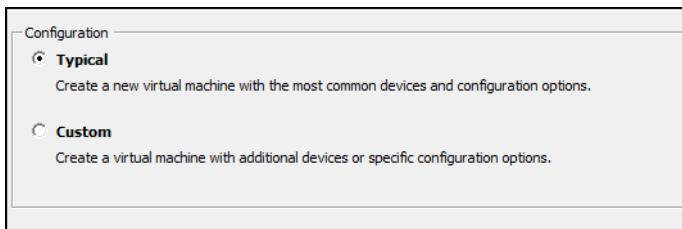
You may want to place your [Pivotal Cloud Foundry](#) (PCF) deployment on a private network, then route requests to PCF from a router or a load balancer on a public network. Isolating PCF in this way increases security and allows you to use as many IP addresses as you need by creating subnets in the private network.

One way to accomplish this is to configure Network Address Translation (NAT) on a VM. The following example explains how to set up a virtual CentOS NAT box in vSphere, and use that to isolate a PCF deployment.

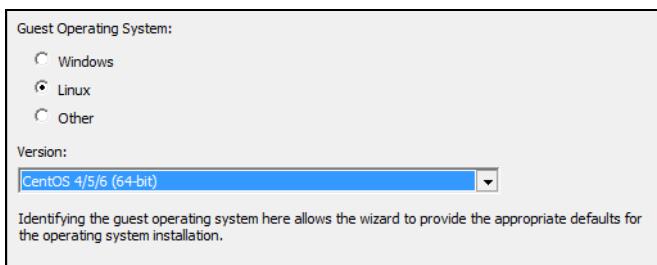
Step 1: Deploy a CentOS image to vSphere

Connect to vCenter using the vSphere client. Assign two network adapters to the VM. This results in the VM having two IP addresses:

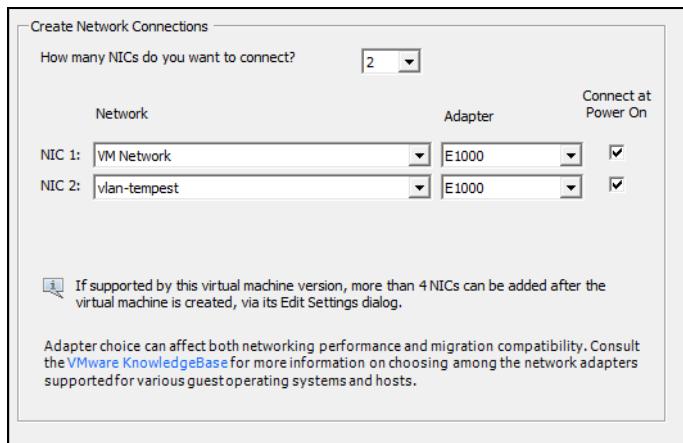
- A public-facing IP address: This IP address connects to the public network VLAN (WAN) through `GATEWAY_EXTERNAL_IP`.
 - An internal VLAN IP address to communicate with other BOSH/Cloud Foundry components: This IP address connects to the private network (LAN) for the Ops Manager deployment using `GATEWAY_INTERNAL_IP`.
1. Download the latest CentOS image from <http://wiki.centos.org/Download>.
 2. In vSphere, select **File > New > Virtual Machine**.
 3. On the **Configuration** page, select the **Typical** configuration and click **Next**.



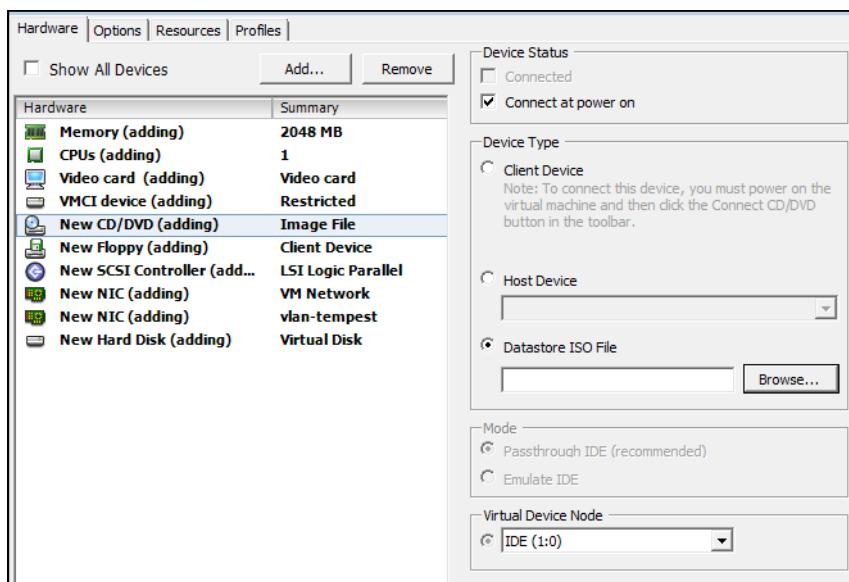
4. On the **Name and Location** page, name the virtual machine and click **Next**.
5. On the **Resource Pool** page, select a resource pool and click **Next**.
6. On the **Storage** page, select a destination storage for the VM and click **Next**.
7. On the **Guest Operating System** page, select **Linux** as the Guest Operating System and **CentOS 4/5/6 (64-bit)** as the version, then click **Next**.



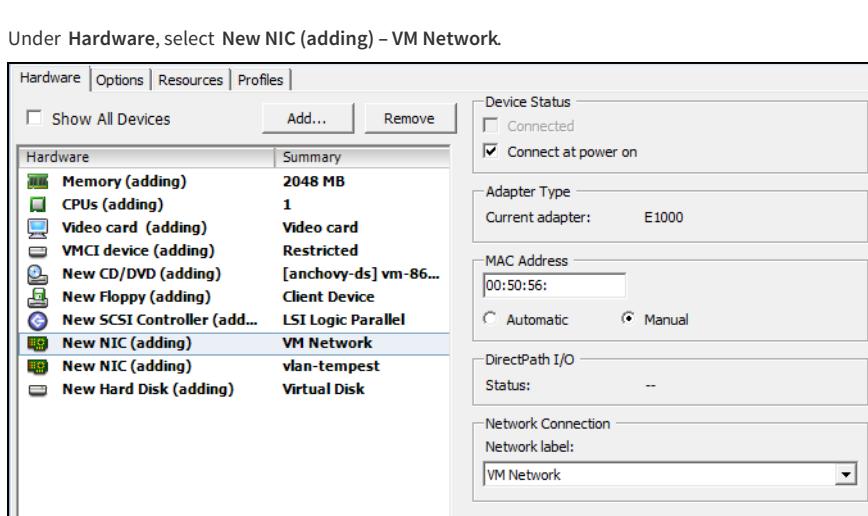
8. On the **Network** page, select **2** from the drop down menu to specify how many NICs you want to connect. Under **Network**, select **VM Network** for NIC 1 and the name of your private network for NIC 2. Then, click **Next**.



9. On the **Create a Disk** page, click **Next**.
10. On the **Ready to Complete** page, check the **Edit the virtual machine settings before completion** checkbox, then click **Continue**.
11. Under **Hardware**, select **New CD/DVD**, then select **Datastore ISO File** and click **Browse**. Browse to **ISO > CentOS-6.5-x86_64-minimal.iso** in your datastore.



12. Under **Device Status**, check the **Connect at power on** checkbox.
13. Under **Hardware**, select **New NIC (adding) – VM Network**.



14. Under **MAC Address**, select **Manual** and specify the MAC address, then click **Finish**.

Step 2: Set up iptables

1. Log in to the CentOS VM as root.
2. Run the following script to set up iptables, replacing the example IPs as appropriate.

```

# the following is an example proxy configuration
sudo vi /etc/sysctl.conf
set net.ipv4.ip_forward=1

# reset
iptables --flush

# example IPs--replace as appropriate
export INTERNAL_NETWORK_RANGE=10.0.0.0/8
export GATEWAY_INTERNAL_IP=10.0.0.1
export GATEWAY_EXTERNAL_IP=88.198.185.190
export PIVOTALCF_IP=10.0.0.10
export HA_PROXY_IP=10.0.0.13

# iptables forwarding rules including loopback
iptables -A FORWARD -i eth1 -j ACCEPT
iptables -A FORWARD -o eth1 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -t nat -A POSTROUTING -d $HA_PROXY_IP -s $INTERNAL_NETWORK_RANGE \
-p tcp --dport 80 -j SNAT --to $GATEWAY_INTERNAL_IP

iptables -t nat -A POSTROUTING -d $HA_PROXY_IP -s $INTERNAL_NETWORK_RANGE \
-p tcp --dport 443 -j SNAT --to $GATEWAY_INTERNAL_IP

iptables -t nat -A PREROUTING -d $GATEWAY_EXTERNAL_IP -p tcp --dport 443 -j DNAT \
--to $HA_PROXY_IP
iptables -t nat -A PREROUTING -d $GATEWAY_EXTERNAL_IP -p tcp --dport 80 -j DNAT \
--to $HA_PROXY_IP

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8443 -j DNAT \
--to $PIVOTALCF_IP:443
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT \
--to $HA_PROXY_IP:80
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 22 -j DNAT \
--to $PIVOTALCF_IP:22
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 8080 -j DNAT \
--to $PIVOTALCF_IP:80

# DNS iptables rules
iptables -A INPUT -p udp --sport 53 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --sport 53 -j ACCEPT
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p udp -j DROP
iptables -A OUTPUT -p udp -j DROP

service iptables save
shutdown -r now

```

Step 3: Deploy the Ops Manager VM Template

1. Download PCF from the Pivotal Network at <https://network.pivotal.io/products/pivotal-cf>.
2. Unzip the downloaded file.
3. Deploy the Ops Manager VM template in the private network using the internal IP address. For more information, see the [Installing PCF Guide](#). Access the deployed VM at port 8443 using https://GATEWAY_EXTERNAL_IP:8443.

Rotating Elastic Runtime MySQL Credentials

Page last updated:

This topic describes how to rotate credentials for Elastic Runtime MySQL. If you are using internal databases in Elastic Runtime and the MySQL for Pivotal Cloud Foundry (MySQL for PCF) product tile, review the notes in this procedure in order to rotate credentials for both products.

Prerequisites

To perform the steps below, you need to obtain the following:

1. Your root CA certificate in a `.crt` file. To retrieve the root CA certificate of your deployment, run the following command:

Note: The Ops Manager API returns the certificate in JSON format with `\n` for every new line. Remove all occurrences of `\n` when you copy the certificate into a `.crt` file.

```
$ curl "https://YOUR-OPSMAN-IP-ADDRESS/api/v0/security/root_ca_certificate"
```

2. Your Elastic Runtime MySQL root password. To retrieve your Elastic Runtime MySQL root password, navigate to the Ops Manager **Installation Dashboard** and select **MySQL > Credentials**. Your Elastic Runtime MySQL root password is called `Mysql Admin Credentials`.

MySQL Server	Vm Credentials	vcap / 37f7dd0c2f25e62f
	Mysql Admin Credentials	root / f07e227e0000-771c-0004

Note: If you use MySQL for PCF®, you also need your MySQL for PCF root password. To retrieve your MySQL for PCF root password, navigate to the Ops Manager **Installation Dashboard** and select **MySQL for Pivotal Cloud Foundry > Credentials**. Your MySQL for PCF root password is called `Mysql Admin Password`.

Rotate Your MySQL Credentials

1. Use `curl` to make a request to the Ops Manager API and pipe the response into `installation_settings_current.json`. For the `OPSMAN_USERNAME` and `OPSMAN_PASSWORD`, use the credentials you used above to log into the Ops Manager **Installation Dashboard**.

```
$ curl -sku OPSMAN_LOGIN:OPSMAN_PASSWORD https://YOUR-OPSMAN-FQDN/api/installation_settings > installation_settings_current.json
```

2. Check to see that the Elastic Runtime MySQL `root password` is in the current installation settings file:

```
$ grep -c YOUR-ERT-MYSQL-ROOT-PASSWORD installation_settings_current.json
```

Note: If you use MySQL for PCF, you should also run the following command: `$ grep -c YOUR-MYSQL-FOR-PCF-ROOT-PASSWORD installation_settings_current.json`

3. Remove the root password from the installation settings file.

```
$ sed -e's/"value": {"identity": "root", "password": "[^"]*"}, } , ("identifier": "mysql_admin")\}\|1/g' installation_settings_current.json > installation_settings_updated.json
```

4. Validate that the root password has been removed from the `installation_settings_updated.json` file.

```
$ grep -c YOUR-ERT-MYSQL-ROOT-PASSWORD installation_settings_updated.json
0
```

Note: If you use MySQL for PCF, you should also run the following command: `$ grep -c YOUR-MYSQL-FOR-PCF-ROOT-PASSWORD installation_settings_updated.json`

5. Upload the updated installation settings. For the `OPSMAN_USERNAME` and `OPSMAN_PASSWORD`, use the credentials you used above to log into the

Ops Manager [Installation Dashboard](#).

```
$ curl -sk -u OPSMAN_USERNAME:OPSMAN_PASSWORD -X POST "https://YOUR-OPSMAN-FQDN/api/installation_settings" -F 'installation[file]=@installation_settings_updated.json'
```

6. Navigate to the Ops Manager [Installation Dashboard](#) and click **Apply Changes**.
7. Once the installation has completed, validate that the Elastic Runtime MySQL password has been changed, using the new password in **Elastic Runtime > Credentials**. Use the IP address for the **MySQL Proxy**, located in the **Status** tab.

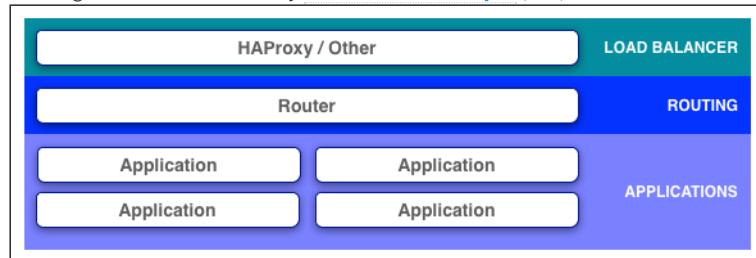
```
$ mysql -uroot -p -h 10.85.21.98
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

 **Note:** If you use MySQL for PCF, you must also validate that the MySQL for PCF root password has been changed. Retrieve the new password from **MySQL > Credentials**. Use the IP address for the MySQL Proxy located in the **Status** tab.

Understanding the Elastic Runtime Network Architecture

Page last updated:

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.

Refer to the [Configuring Pivotal Cloud Foundry SSL Termination](#) topic for more information.

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.

Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments

Page last updated:

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 on the Cloud Controller page in Ops Manager with the HAProxy IP address.

For example, with `cf.example.com` as the main subdomain for your CF install and an HAProxy IP address `172.16.1.1`, you must create an A record in your DNS that serves `example.com` and points `*.cf` to `172.16.1.1`.

Name	Type	Data	Domain
<code>*.cf</code>	A	<code>172.16.1.1</code>	<code>example.com</code>

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 172.16.1.1
$ host anything.example.com
anything.cf.example.com has address 172.16.1.1
```

3. From the PCF Ops Manager Dashboard, click on the Elastic Runtime tile.
4. Select **IPs and Ports**.
5. Enter the IP address for HAProxy in the **HAProxy IPs** field.
6. Leave the **Router IPs** field blank. HAProxy assigns the router IPs internally.
7. Provide your SSL certificate on the **Security Config** page. See [Providing a Certificate for your SSL Termination Point](#).

[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 on the Cloud Controller page in Ops Manager 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 `10.10.1.1`, you must create an A record in your DNS that serves `example.com` and points `*.cf` to `10.10.1.1`.

Name	Type	Data	Domain
<code>*.cf</code>	A	<code>10.10.1.1</code>	<code>example.com</code>

3. From the PCF Ops Manager Dashboard, click on the Elastic Runtime tile.
4. Select **IPs and Ports**.
5. Leave the **HAProxy IPs** field blank.
6. In the **Router IPs** field, enter the static IP address for PCF that you have registered with your load balancer.
7. Provide your SSL certificate on the **Security Config** page. See [Providing a Certificate for your SSL Termination Point](#).

 **Note:** When adding or removing PCF routers, you must update your load balancing solution configuration with the appropriate IP addresses.

Changing the Quota Plan of an Organization with cf CLI

Page last updated:

This page assumes you are using cf CLI v6.

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 identified by unique GUIDs.

Quota plans are not directly associated with user accounts. Instead, every organization has a list of available quota plans, and the account admin assigns a specific quota plan from the list to the organization. Everyone in the organization shares the quotas described by the plan.

Depending on the usage needs of an organization, you may need to change the quota plan. Follow the steps below to change the quota plan of your organization using `cf curl`. Only an Account Administrator can run `cf curl`.

You must have the correct permissions to change the quota plan of an organization. For more information, see [Orgs, Spaces, Roles, and Permissions](#).

Step 1: Find the GUID of your organization

To find the GUID of your organization, replace MY-ORGANIZATION with the name of your organization and run this command:

```
CF_TRACE=true cf org MY-ORGANIZATION
```

This command returns a filtered JSON response listing information about your organization.

Data about your organization shows in two sections: “metadata” and “entity.” The “metadata” section shows the organization GUID, while the “entity” section lists the name of the organization.

 **Note:** Because “metadata” is listed before “entity” in the response, the GUID of the organization is shown six lines above the name.

Example:

```
$ CF_TRACE=true cf org example-org-name | grep -B7 example-org-name

REQUEST:
GET /v2/organizations?q=name%3Aexample-org-name&inline-relations-depth=1 HTTP/1.1
--
"metadata": {
  "guid": "aaaa1234-1111",
  "url": "/v2/organizations/aaaa1234-1111",
  "created_at": "2014-02-06T02:09:09+00:00",
  "updated_at": null
},
"entity": {
  "name": "example-org-name",
```

The GUID of “example-org-name” is *aaaa1234-1111*.

Step 2: Find the GUID of your current quota plan

To find the current quota plan for your organization, replace MY-ORGANIZATION-GUID with the GUID found in Step 1 and run:

```
cf curl /v2/organizations/MY-ORGANIZATION-GUID -X 'GET' | grep
quota_definition_guid
```

This command returns a filtered JSON response showing the quota_definition_guid of your organization.

Example:

```
$ cf curl /v2/organizations/aaaa1234-1111 -X 'GET' | grep quota_definition_guid
"quota_definition_guid": "bbbb6666-2222",
```

The GUID of the current quota plan is *bbbb6666-2222*.

Step 3: View available quota plans and select the quota plan GUID you want

To view all quota plans available to your organization, run:

```
cf curl /v2/quota_definitions -X 'GET'
```

This command returns a JSON response listing every quota plan available to your organization. Data about each plan shows in two sections: “metadata” and “entity.” The “metadata” section shows the quota plan GUID, while the “entity” section lists the name of the plan and the actual usage quotas.

 **Note:** Because “metadata” is listed before “entity” for each quota plan, the GUID of a plan is shown six lines *above* the name.

Example:

```
$ cf curl /v2/quota_definitions -X 'GET'

{
  "total_results": 2,
  "total_pages": 1,
  "prev_url": null,
  "next_url": null,
  "resources": [
    {
      "metadata": {
        "guid": "bbbb6666-2222",
        "url": "/v2/quota_definitions/bbbb6666-2222",
        "created_at": "2013-11-19T18:53:48+00:00",
        "updated_at": "2013-11-19T19:34:57+00:00"
      },
      "entity": {
        "name": "small",
        "non_basic_services_allowed": false,
        "total_services": 10,
        "total_routes": 10,
        "memory_limit": 2048,
        "trial_db_allowed": false
      }
    },
    {
      "metadata": {
        "guid": "cccc0000-3333",
        "url": "/v2/quota_definitions/cccc0000-3333",
        "created_at": "2013-11-19T18:53:48+00:00",
        "updated_at": "2013-11-19T19:34:57+00:00"
      },
      "entity": {
        "name": "bigger",
        "non_basic_services_allowed": true,
        "total_services": 100,
        "total_routes": 100,
        "memory_limit": 10240,
        "trial_db_allowed": true
      }
    }
  ]
}
```

In this example, the GUID of the “small” plan is *bbbb6666-2222* and the GUID of the “bigger” plan is *cccc0000-3333*.

Select the quota plans you want to assign to your organization.

In our example, we want to change from the current plan to the “bigger” plan, GUID`cccc0000-3333`.

Step 4: Change the Quota Plan

Changing the quota plan GUID changes the quota plan used by the organization.

To change the quota plan for your organization, run:

```
cf curl /v2/organizations/MY-ORGANIZATION-GUID -X 'PUT' -d "{\"quota_definition_guid\":\"NEW-QUOTA-PLAN-GUID\"}"
```

Example:

```
$ cf curl /v2/organizations/aaaa1234-1111 -X 'PUT' -d "{\"quota_definition_guid\":\"cccc0000-3333\"}"
```

This command changes the quota plan assigned to the organization to the “bigger” plan, GUIDcccc0000-3333.

Step 5: Verify the change

To verify the change to your quota plan, run the command from Step 2:

```
cf curl /v2/organizations/MY-ORGANIZATION-GUID -X 'GET' | grep quota_definition_guid
```

This command should return a filtered JSON response showing the new quota_definition_guid" of your organization.

Example:

```
$ cf curl /v2/organizations/aaaa1234-1111 -X 'GET' | grep quota_definition_guid
"quota_definition_guid": "cccc0000-3333",
```

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:

Step	Description
1	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.
2	Ensure Access for PCF System Apps: Bind the default ASG to the <code>system</code> 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.
3	Create New ASGs: Block apps from sending traffic to system components, but allow them to send traffic to the services they need.
4	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.
5	Restart your Apps: To apply the ASG changes, you must restart all of the apps in your deployment.

- **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](#) 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:

Layout Name	Layout Description
One Network	<ul style="list-style-type: none"> o One network for Ops Manager and the Ops Manager Director, Elastic Runtime, and services. <div style="border: 1px solid #ccc; padding: 5px; background-color: #ffffcc;"> 💡 Note: 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. </div>
Two Networks	<ul style="list-style-type: none"> o One network for Ops Manager and the Ops Manager Director. o One network for Elastic Runtime and Services.
Three Networks	<ul style="list-style-type: none"> o One network for Ops Manager and the Ops Manager Director. o One network for Elastic Runtime. o One network for all services.
Three or More Networks	<ul style="list-style-type: none"> o One network for Ops Manager and the Ops Manager Director. o One network for Elastic Runtime. o One network for each service.

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 `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:

- o **Two Network Layout:**

```
excluded_networks:  
- YOUR-OPS-MANAGER-CIDR  
- YOUR-ELASTIC-RUNTIME-AND-SERVICES-CIDR
```

- o **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
```

- o **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:

- Replace `SECURITY-GROUP-NAME` with a name of your choice.
- 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:

- [Managing ASGs with the cf CLI](#)
- [Typical ASGs](#)

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:

```
$ cf unbind-staging-security-group default_security_group
```

2. Unbind the default ASG from the running set:

```
$ 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.

Identifying Elastic Runtime Jobs Using vCenter

Page last updated:

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**.

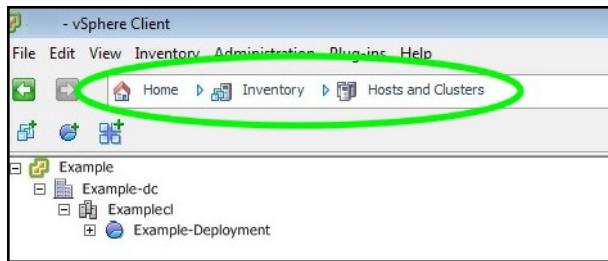
If you have deployed Elastic Runtime to VMware vSphere, you can also identify which Elastic Runtime job corresponds to which VM using the vCenter vSphere client. This option is not available if you have deployed Elastic Runtime to VMware vCloud Air / vCloud.

 **Note:** The CID shown in Ops Manager is the name of the machine in vCenter.

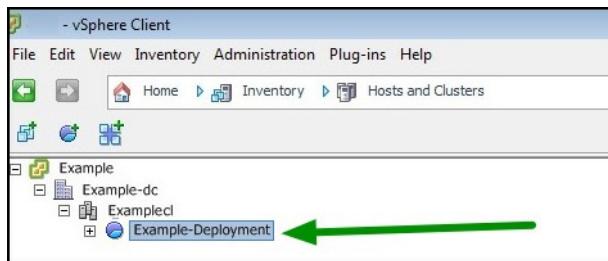
Identifying Elastic Runtime Jobs Using vCenter

1. Launch the vSphere client and log in to the vCenter Server system.

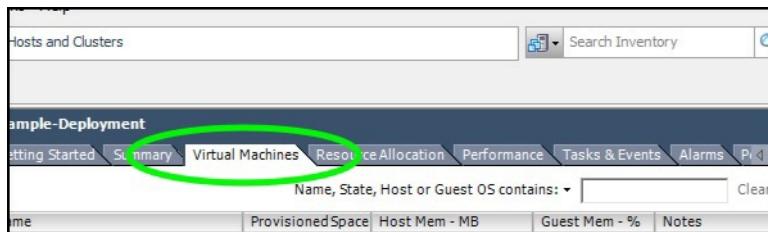
2. Select the **Inventory > Hosts and Clusters** view.



3. Select the Resource Pool containing your Elastic Runtime deployment.



4. Select the **Virtual Machines** tab.



5. Right-click the column label heading and check **job**.

Example-Deployment

Getting Started Summary Virtual Machines Resource Allocation Performance Tasks & Events Alarms Permissions Maps

Name, State, Host or Guest OS contains: Name, Target or Status contains: Name, State, Host or Guest OS contains: Name, Target or Status contains:

Status

- Host
- Provisioned Space
- Used Space
- Host CPU - MHz
- Host Mem - MB
- Guest Mem - %
- Guest OS
- VM Version
- Memory Size
- Reservation - MB
- CPU Count
- NIC Count
- Uptime
- IP Address
- VMware Tools Running status
- VMware Tools Version status
- DNS Name
- EVC Mode
- UUID
- Notes
- Alarm Action
- vSphere HA protection
- Needs Consolidation
- Name**
- compiling
- deployment
- director
- index
- job**

6. The job column displays the Elastic Runtime job associated with each virtual machine.

Name	job
vm-347a89c6-115f-433a-83d1-38cd2c31451b	ccdb
sc-d3520bac-11c2-460f-aa02-60fc11cba375	cloud_controller
vm-d537f816-fc51-450c-8ada-cb536a9e76f5	consoledb
vm-85d43147-7cd2-4cc5-ac6c-47cce18cccd9	dea
vm-5c04a27a-ca70-4f67-a221-767efb76922a	ha_proxy
vm-e9b3e319-690f-44be-87fb-ce97f90c4b2	health_manager
vm-af8bd3d1-7e31-47f3-800f-3f81ff23ed81	loggregator
vm-ee8b5374-bbc9-436b-bfe5-1a7e2b5fee9a	loggregator_trafficcontroller
vm-5896908f-15c5-4989-8c90-36a36e0eb4fd	nats
vm-61fc0d19-01f5-4434-82e0-a11e15b388a8	nfs_server
vm-e88725cd-36fe-4a79-b8ea-f35b2d2cd85b	riak-cs
vm-9ab56b39-4cae-443a-842c-3651c9bf228c	router
vm-2726dcaa-7f92-440c-906e-3e5e544fe628	saml_login
vm-273df76f-7141-432d-9e53-9cb6549d870a	uaa
vm-307a34f4-ed7f-4c48-94d1-bb7a61a695bc	uaadb
vm-e6c2dfcd-ef2e-40d8-af0b-06e5f3e73fa9	
vm-0e9e9384-0057-4efc-9cea-234b5956c510	
vm-5e33e793-0118-44bb-948d-3ea6526b4b2f	

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 24 components that the Ops Manager Director installs. You can specify the number of instances for 11 of the components. We deliver the remaining 13 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 DEA, 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 the 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.

Component	Scalable?	Extended Downtime	Other Limitations and Information
HAProxy	✓	✓	
NATS	✓	✓	
etcd		✓	
Health Manager	✓		Ops Manager operators will see a 5 minute delay in app cleanup during an upgrade.
NFS Server		✓	You cannot push, stage, or restart an app when an upgrade affects the NFS Server.
Cloud Controller Database		✓	
Cloud Controller	✓	✓	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 DEA components. If either of these components are single components, you cannot push, stage, or restart an app during the upgrade.
Clock Global			
Cloud Controller Worker	✓	✓	
Router	✓	✓	
Pivotal Ops Metrics Collector	✓		The Pivotal Ops Metrics tool is a JMX extension for Elastic Runtime that you can install . If you install this tool, Ops Manager operators will see a 5 minute delay in metrics collection during an upgrade. You can disable this component before an upgrade to reduce the overall system downtime.
UAA Database			
UAA	✓		If a user has an active authorization token prior to performing an upgrade, the user can still log in using either a UI or the CLI.

Login	✓	✓	
Apps Manager Database			You can disable this component before an upgrade to reduce the overall system downtime.
MySQL Server			
DEA	✓	✓	Your ability to manage an app when an upgrade affects the DEA depends on the number of instances that you specify for the Cloud Controller and DEA components. If either of these components are single components, you cannot push, stage, or restart an app during the upgrade. The Apps Manager app and App Usage Service components for the Apps Manager Database run in a single DEA instance. You cannot use the Apps Manager or the CLI during the upgrade of the DEA.
Doppler Server			Ops Manager operators will see 2-5 minute gaps in logging.
Loggregator Traffic Controller			Ops Manager operators will see 2-5 minute gaps in logging.
Push Apps Manager and Push App Usage Service errands			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 runs in a single DEA instance and the Apps Manager Database is a single component. If there is an upgrade issue with either Apps Manager Database instance or the DEA instance, the upgrade fails and Ops Manager will not run this errand.

Configuring Single Sign-On

Page last updated:

If your user store is exposed as a SAML Identity Provider for single sign-on (SSO), you can set up SSO to allow users to access the [Pivotal Cloud Foundry](#) (PCF) [Apps Manager](#) without creating a new account or re-entering credentials.

Configure Pivotal Cloud Foundry as a Service Provider

You must configure Pivotal Cloud Foundry (PCF) so that your Identity Provider can recognize PCF as a Service Provider. Follow the instructions below:

1. Log in to Ops Manager.
2. Click the **Elastic Runtime** tile. Select **Cloud Controller** on the **Settings** tab and verify your system domain.

Assign Networks Coordinates Pivotal CF Elastic Runtime application lifecycles

Assign Availability Zones

System Database Config

File Storage Config

IPS and Ports

MySQL Proxy Config

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Errands

Resource Config

Stemcell

System Domain * system.mydomain.com

Apps Domain * apps.mydomain.com

Cloud Controller DB Encryption Key Secret

Maximum File Upload Size (MB) (min: 1024, max: 2048) * 1024

Disable Custom Buildpacks

Default Quota App Memory (MB) (min: 10240, max: 102400) * 10240

Default Quota Service Instances (min: 0, max: 1000) * 100

Save

3. Select **SSO Config**.

Configure Identity Provider

Provider Name

Provider Metadata

(OR) Provider Metadata URL

Proxy IPs Regular Expression *

Save

4. Enter a **Provider Name**. Your provider name appears as a link on your Pivotal login page, which you can access at <https://login.YOUR-SYSTEM-DOMAIN>. The image below shows an example login page with the provider name “Your Provider Name.” If you click on this link, you are redirected to your Identity Provider for authentication.

Welcome!

Email

Password

SIGN IN

or Sign in with:

Your Provider Name

Create account Reset password

5. Populate either **Provider Metadata** or **(OR) Provider Metadata URL**, depending on whether your Identity Provider exposes a Metadata URL. You can do this by either of the following:

- o Download your Identity Provider metadata and paste this XML into the **Provider Metadata** area. Ensure that the XML declaration tag is present at the beginning of the Metadata XML. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor>
</EntityDescriptor>
```

- o If your Identity Provider exposes a Metadata URL, provide the Metadata URL.

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**.

6. (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 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).

7. Click **Save**.
8. Click **Apply Changes** on the Ops Manager home page for the configuration to take effect.

Configure Your Identity Provider

You must configure your Identity Provider to recognize PCF as a Service Provider.

Download the Service Provider Metadata from <https://login.YOUR-SYSTEM-DOMAIN/saml/metadata>. Consult the documentation for 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 Pivotal Cloud Foundry:

Solution Name	Integration Guide
CA Single Sign-On aka CA SiteMinder	PDF
Ping Federate	PDF

 **Note:** Some Identity Providers allow uploads of Service Provider Metadata. Other providers require you to manually enter the Service Provider Metadata into a form.

Configuring LDAP

Page last updated:

This topic describes connecting [Pivotal Cloud Foundry](#) (PCF) Elastic Runtime to LDAP by integrating the Cloud Foundry User Account and Authentication server with LDAP.

The Cloud Foundry User Account and Authentication ([UAA](#)) server provides identity management for Elastic Runtime in three ways:

- Issues tokens for use by client applications when they act on behalf of Elastic Runtime users
- Authenticates users with their Elastic Runtime credentials
- Acts as an SSO service using Elastic Runtime or other credentials

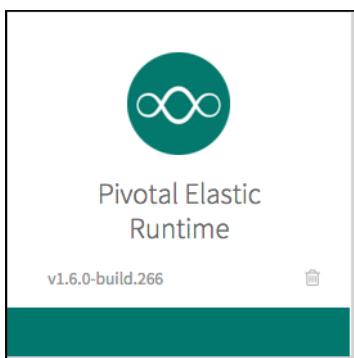
You can integrate the UAA with a Lightweight Directory Access Protocol (LDAP) server. Connecting Elastic Runtime to LDAP allows the UAA to authenticate users using LDAP search and bind operations.

See [Adding Existing LDAP Users to a PCF Deployment](#) for information about managing user identity with LDAP in PCF.

Configuring the LDAP Endpoint

To integrate the UAA with LDAP, configure Elastic Runtime with your LDAP endpoint information as follows:

1. Log into the Pivotal Cloud Foundry Operations Manager web interface.
2. On the Product Dashboard, select **Pivotal Elastic Runtime**.



3. In the left navigation menu, select **LDAP Config**.

- Assign Networks
- Assign Availability Zones
- System Database Config
- File Storage Config
- IPs and Ports
- Security Config
- MySQL Proxy Config
- Cloud Controller
- System Logging
- SSO Config
- LDAP Config
- SMTP Config
- Diego
- Experimental Features
- Errands
- Resource Config
- Stemcell

Configure an LDAP endpoint for the UAA

Server URL

LDAP Credentials

User Search Base

User Search Filter *

Admin Groups*

 No Groups

Enable Admin Groups

Group Search Base

Group Search Filter *

Server SSL Cert

Server SSL Cert AltName

Email Attribute *

4. Enter the following LDAP endpoint configuration information:

- **Server URL:** A URL pointing to the LDAP server. This 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.

- **LDAP Credentials:** The LDAP Distinguished Name (DN) and password for binding to the LDAP Server. Example DN: `cn=administrator,ou=Users,dc=example,dc=com`
- **User Search Base:** The location in the LDAP directory tree from which any LDAP User search begins. The typical User Search Base matches your domain name.

For example, a domain named “cloud.example.com” typically uses the following User Search Base: `ou=Users,dc=example,dc=com`

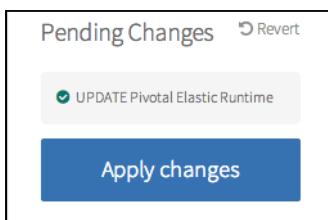
- **User Search Filter:** A string that defines LDAP User search criteria. These search criteria allow LDAP to perform more effective and efficient searches. For example, the standard LDAP search filter `cn=Smith` returns all objects with a common name equal to `Smith`.

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`.

- **Admin Groups:** If you want to map LDAP groups to the Admin role, select **Enable Admin Groups** and complete the following fields. Otherwise, select **No Groups**.
 - **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 Group Search Base: `ou=Groups,dc=example,dc=com`. Review the instructions in the Granting Admin Permissions to an LDAP Group section of the [Creating and Managing Users with the UAA CLI \(UAAC\)](#) topic for more information.
 - **Group Search Filter:** Enter the string that defines LDAP Group search criteria by replacing the `0` in `member={0}` with the user DN.
- **Server SSL Cert:** Paste in the root certificate from your CA certificate or your self-signed certificate. Required only for `ldaps://`.
- **Server SSL AltName:** Enter a Subject Alternative Name for your certificate. Required only for `ldaps://`.
- **Email Attribute:** Enter the name of the attribute that contains the user’s email address.

5. Click **Save**.
6. Click the **Installation Dashboard** link to return to the Installation Dashboard.
7. On the Installation Dashboard, click **Apply Changes**.



Switching Application Domains

Page last updated:

This page assumes you are using cf CLI v6.

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 **Cloud Controller** 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 -n 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 -n myapp
```

Creating route myapp.newapps.mydomain.com for org my-org / space my-space as admin...
OK
Adding route myapp.newapps.mydomain.com to app myapp in org my-org / space my-space as admin...
OK

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:

APPLICATIONS			Learn More
STATUS	APP	INSTANCES	
	myapp myapp.newapps.mydomain...	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 OLD_DOMAIN_TO_DELETE`:

```
$ 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.

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks Coordinates Pivotal CF Elastic Runtime application

Assign Availability Zones

Root Filesystem

System Database Config

File Storage Config

IPs and Ports

Security Config

MySQL Proxy Config

Cloud Controller

System Domain * mysystem.mydomain.com

Apps Domain * newapps.mydomain.com

Cloud Controller DB Encryption Key Secret

Maximum File Upload Size (MB) (min: 1024, max: 2048) * 1024

Disable Custom Buildpacks

Default Quota App Memory (MB) (min: 10240, max: 102400) *

Scaling Instances in Elastic Runtime

Page last updated:

This topic discusses how to scale instances in 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 Instances

1. Navigate to the Pivotal Cloud Foundry Operations Manager Installation Dashboard.
2. Click the Elastic Runtime tile in the Installation Dashboard.
3. Select **Resource Config**.

◀ Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Resource Config

JOB	INSTANCES
NATS	1
consul	1
etcd	1
Diego etcd	1
NFS Server	0
Router	1
MySQL Proxy	0
MySQL Server	0
Cloud Controller Database (Postgres)	0
UAA Database (Postgres)	0
Apps Manager Database (Postgres)	0
Cloud Controller	1
HAProxy	0
Health Manager	1
Clock Global	1
Cloud Controller Worker	1
Collector	0
DEA	0
UAA	1
Diego Brain	1
Diego Cell	3
Doppler Server	1
Loggregator Trafficcontroller	1
Pusher Metrics	0

Assign Networks
Assign Availability Zones
System Database Config
File Storage Config
IPs and Ports
Security Config
MySQL Proxy Config
Cloud Controller
System Logging
SSO Config
LDAP Config
SMTP Config
Diego
Experimental Features
Errands
Resource Config

4. If you are using one of the following configurations in [Pivotal Elastic Runtime](#), enter the values in the corresponding table to scale instances for your particular deployment:

- [Diego with no DEAs](#)
- [External Databases](#)
- [MySQL only](#)
- [External Blobstore](#)
- [External Load Balancer](#)
- [Ops Metrics](#)

Diego with no DEAs

 **Note:** To make Diego highly available, see the [Zero Downtime Deployment and Scaling in CF](#) topic. This scenario is applicable to a basic installation.

If you are using [Diego by default instead of DEAs](#), enter the following values in the Resource Config:

Job	Instance Count	Notes
Diego Cell	≥ 3	
Diego Brain	≥ 1	
Diego Bulletin Board System (BBS)	≥ 1	
DEA	0	
Consul	≥ 1	
Health Manager	0	

External Databases

If you are using an [external database](#), enter the following values in the Resource Config:

Job	Instance Count	Notes
MySQL Server	0	
MySQL Proxy	0	
Apps Manager Database (Postgres)	0	
Cloud Controller Database (Postgres)	0	
UAA Database (Postgres)	0	

MySQL

 **Note:** This scenario is not applicable if you are using a MySQL and Postgres configuration. For example, you have system data from a pre-1.6.0 Elastic Runtime installation.

If you are using a MySQL database, enter the following values in the Resource Config:

Job	Instance Count	Notes
MySQL server	≥ 1	Must be set to an odd number.
MySQL Proxy	≥ 1	
Apps Manager Database (Postgres)	0	
Cloud Controller Database (Postgres)	0	
UAA Database (Postgres)	0	

External Blobstore

If you are using an external [Blobstore](#), enter the following value in the Resource Config:

Job	Instance Count	Notes
NFS	0	

External Load Balancer

If you are using an external load balancer, enter the following values in the Resource Config:

Job	Instance Count	Notes
HAProxy	0	

Router	≥ 1	For Amazon Web Services, set the Elastic Load Balancer name in the Router's "External Load Balancer" field.
Diego Brain	≥ 1	For AWS, if you have the Diego SSH feature enabled, set the SSH ELB name in the Router's "External Load Balancer" field.

Pivotal Ops Metric

If you are using [Ops Metrics](#), enter the following value in the Resource Config:

Job	Instance Count	Notes
Collector	≥ 1	Must have Ops Metric tile added to PCF before you scale this instance count up. See Adding and Deleting Products topic .

5. Enter the suggested values outlined in each scenario above, and click **Save**.

6. Return to the **Installation Dashboard** and click **Apply Changes**.

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 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.

```
{
  "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",
        [...]
        "space_url": "/v2/spaces/a0205ae0-a691-4667-92bc-0d0dd712b6d3",
        "stack_url": "/v2/stacks/86205f38-84fc-4bc2-b2b8-af7f55669f04",
        "routes_url": "/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba/routes",
        "events_url": "/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba/events",
        "service_bindings_url": "/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba/service_bindings",
        "route_mappings_url": "/v2/apps/acf2ce33-ee92-54TY-9adb-55a596a8dcba/route_mappings"
      }
    },
    {
      "metadata": {
        "guid": "79bb58cc-3737-43be-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",
        [...]
        "space_url": "/v2/spaces/a0205ae0-a691-4667-92bc-0d0dd712b6d3",
        "stack_url": "/v2/stacks/86205f38-84fc-4bc2-b2b8-af7f55669f04",
        "routes_url": "/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178/routes",
        "events_url": "/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178/events",
        "service_bindings_url": "/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178/service_bindings",
        "route_mappings_url": "/v2/apps/79bb58cc-3737-4540-ac70-39a2843b5178/route_mappings"
      }
    }
  ]
}
```

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`.

3. The `summary` endpoint under each app-specific URL retrieves the instances and any bound services for that app.

```
$ cf curl /v2/apps/acf2ce75-ee92-4bb6-9adb-55a596a8deba/summary
{
  "guid": "acf2ce75-ee92-4bb6-9adb-55a596a8deba",
  "name": "YOUR-APP",
  "routes": [
    {
      "guid": "7421b6af-75cb-4334-a862-bc5e1ababfe6",
      "host": "YOUR-APP",
      "path": "",
      "domain": {
        "guid": "fb6bd89f-2ed9-49d4-9ad1-97951a573135",
        "name": "YOUR-DOMAIN.io"
      }
    }
  ],
  "running_instances": 5,
  "services": [
    {
      "guid": "b9cdr456-3c61-4f8a-a307-9b4ty836fb2e",
      "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"
      },
      "dashboard_url": "https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/sso/start?serviceUuid=b5cASDF-3c61-4f8a-a307-9bf85j45fb2e",
      "service_plan": {
        "guid": "fbccc3af-3e8d-4ee7-adfe-3f12a137ed66",
        "name": "turtle",
        "service": {
          "guid": "34dbc753-34ed-4cf1-9a87-a255dfea5339b",
          "label": "elephantsql",
          "provider": null,
          "version": null
        }
      }
    }
  ]
}
```

4. To view the `app_usages` 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-a307-9b4ty836fb2e",
        "url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9b4ty836fb2e",
        "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-0d0dd712b6d3",
        "gateway_data": null,
        "dashboard_url": "https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/sso/start?serviceUuid=b9cdr456-3c61-4f8a-a307-9b4ty836fb2e",
        "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-0d0dd712b6d3",
        "service_plan_url": "/v2/service_plans/fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "service_bindings_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9b4ty836fb2e/service_bindings",
        "service_keys_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9b4ty836fb2e/service_keys",
        "routes_url": "/v2/service_instances/b9cdr456-3c61-4f8a-a307-9b4ty836fb2e/routes"
      }
    },
    {
      "metadata": {
        "guid": "78be3399-bdc7-4fbf-a1a4-6858a50d0ff3",
        "url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a50d0ff3",
        "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-0d0dd712b6d3",
        "gateway_data": null,
        "dashboard_url": "https://cloudfoundry.appdirect.com/api/custom/cloudfoundry/v2/sso/start?serviceUuid=78be3399-bdc7-4fbf-a1a4-6858a50d0ff3",
        "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-0d0dd712b6d3",
        "service_plan_url": "/v2/service_plans/fbcec3af-3e8d-4ee7-adfe-3f12a137ed66",
        "service_bindings_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a50d0ff3/service_bindings",
        "service_keys_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a50d0ff3/service_keys",
        "routes_url": "/v2/service_instances/78be3399-bdc7-4fbf-a1a4-6858a50d0ff3/routes"
      }
    }
  ]
}

```

Using Diego in Pivotal Cloud Foundry

Page last updated:

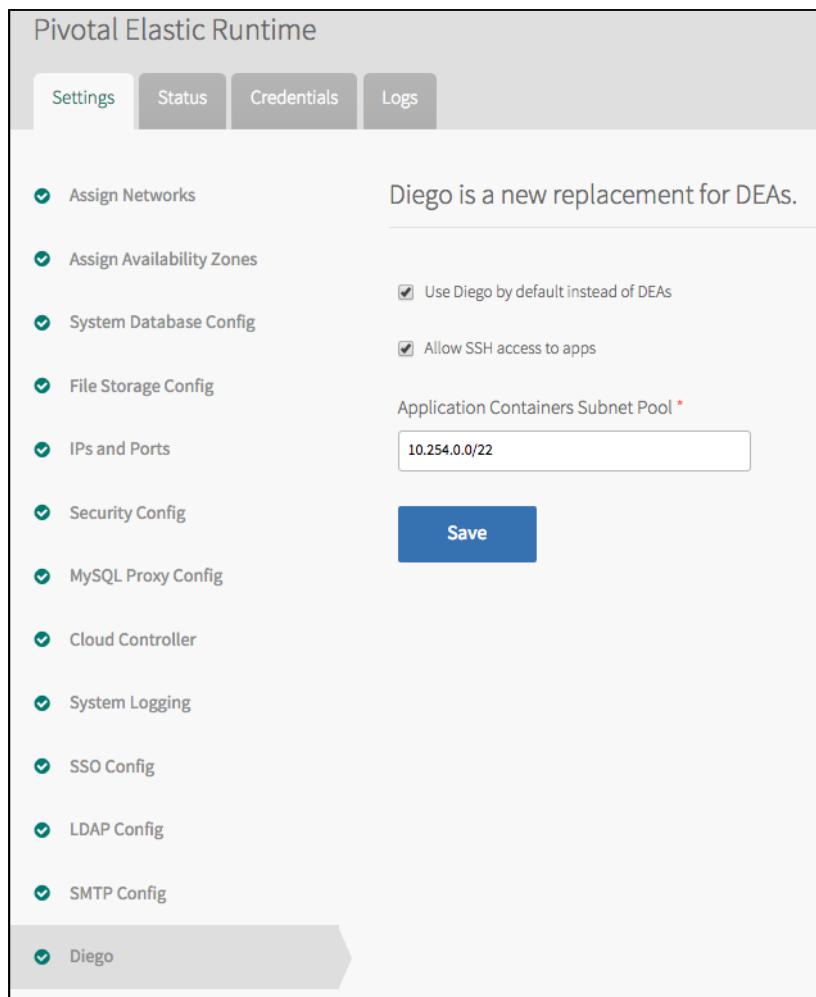
This topic describes using Diego in the [Pivotal Cloud Foundry](#) (PCF) 1.6 release. Diego is the next-generation container scheduler for Elastic Runtime.

In the PCF 1.6 release, Diego replaces the [DEAs](#) and the [Health Manager](#). Diego is installed and enabled in PCF 1.6 by default.

 **Note:** Before you install PCF 1.6, you must uninstall any Diego Beta installations. For more information about upgrading to PCF 1.6, see [Upgrading Operations Manager](#).

Configure Diego in PCF

1. Log in to Pivotal Ops Manager.
2. Click the **Pivotal Elastic Runtime** tile.



3. Select **Diego**.
4. Select the **Use Diego by default instead of DEAs** checkbox to ensure that all applications pushed to your PCF deployment use the Diego container management system. For new installations, this option is selected by default. If you are upgrading to PCF 1.6, you must select this option to automatically enable Diego for newly pushed applications.

 **Note:** If you do not select this option, application developers must explicitly target Diego when pushing their applications.

5. (Optional) Select the **Allow SSH access to apps** checkbox to allow application developers to `ssh` directly into their application instances on Diego. See the [Diego-SSH Overview](#) topic for more details.

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 Cloud Foundry (CF) deployment through Amazon Web Services (AWS) [CloudFormation](#) or by [manually](#) connecting directly to a Windows VM.

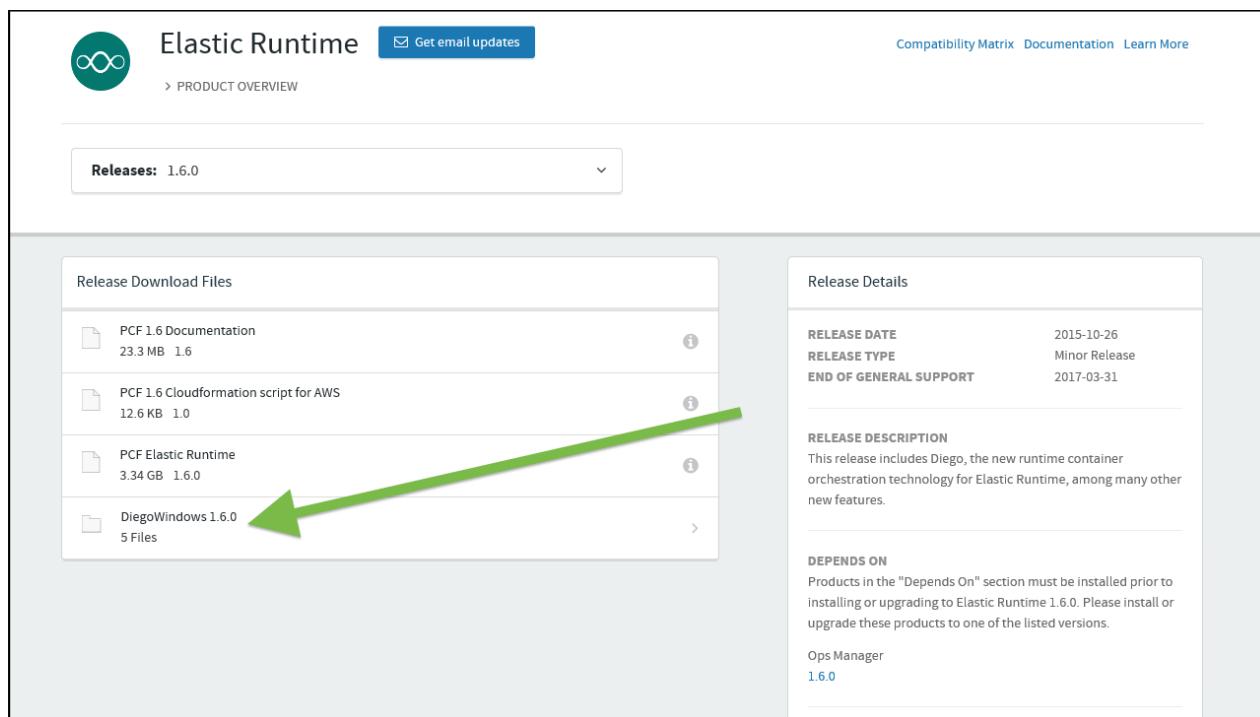
Install through AWS CloudFormation

Prerequisites

- A working Diego deployment

Step 1: Download CloudFormation Template

1. Navigate to the Elastic Runtime product on Pivotal Network (link deprecated). This points to the latest release of the **Elastic Runtime** product.
2. Deprecated: Select **DiegoWindows** from the list of Release Download Files.



The screenshot shows the 'Elastic Runtime' product page on Pivotal Network. The 'Release Download Files' section lists four files:

File	Size	Version
PCF 1.6 Documentation	23.3 MB	1.6
PCF 1.6 Cloudformation script for AWS	12.6 KB	1.0
PCF Elastic Runtime	3.34 GB	1.6.0
DiegoWindows 1.6.0	5 Files	

A green arrow points to the 'DiegoWindows 1.6.0' file. To the right, the 'Release Details' section provides information about the release:

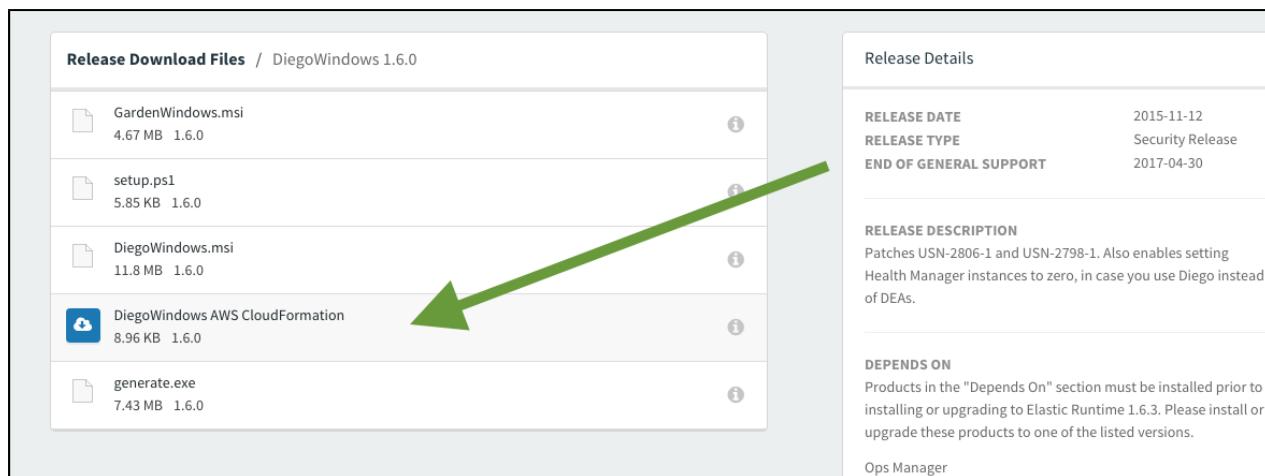
- RELEASE DATE: 2015-10-26
- RELEASE TYPE: Minor Release
- END OF GENERAL SUPPORT: 2017-03-31

RELEASE DESCRIPTION: This release includes Diego, the new runtime container orchestration technology for Elastic Runtime, among many other new features.

DEPENDS ON: Products in the "Depends On" section must be installed prior to installing or upgrading to Elastic Runtime 1.6.0. Please install or upgrade these products to one of the listed versions.

Ops Manager
1.6.0

3. Download the **DiegoWindows AWS CloudFormation.json** file.



Release Download Files / DiegoWindows 1.6.0

GardenWindows.msi	4.67 MB 1.6.0	
setup.ps1	5.85 KB 1.6.0	
DiegoWindows.msi	11.8 MB 1.6.0	
DiegoWindows AWS CloudFormation	8.96 KB 1.6.0	
generate.exe	7.43 MB 1.6.0	

Release Details

RELEASE DATE	2015-11-12
RELEASE TYPE	Security Release
END OF GENERAL SUPPORT	2017-04-30
RELEASE DESCRIPTION	
Patches USN-2806-1 and USN-2798-1. Also enables setting Health Manager instances to zero, in case you use Diego instead of DEAs.	
DEPENDS ON	
Products in the "Depends On" section must be installed prior to installing or upgrading to Elastic Runtime 1.6.3. Please install or upgrade these products to one of the listed versions.	
Ops Manager	

Step 2: Upload Template to AWS

1. Log in to the AWS [CloudFormation console](#).
2. Choose **Create Stack** and upload the DiegoWindows AWS CloudFormation template to Amazon S3.
3. Provide the following information:
 - o **BoshHost**: BOSH director host
 - o **BoshPassword**: Password to use to access the BOSH director
 - o **BoshUserName**: User name to use to access the BOSH director
 - o **CellName**: Name for your cell
 - o **ContainerizerPassword**: Password for the containerizer user. Must be alphanumeric and contain at least one uppercase, one lowercase, and one numeric character.
 - o **DesiredCapacity**: Number of Windows cells to deploy
 - o **Keypair**: A keypair. You must have the private key to this keypair to retrieve the Administrator password to the Windows VMs that are created.
 - o **NATInstance**: The instance ID of the NAT box. To find this instance ID, search for **NAT** in the CloudFormation dropdown.
 - o **SecurityGroup**: Security group ID to use for the Windows cells
 - o **SubnetCIDR**: The IP range of the Windows cell subnet. For example, `10.0.100.0/24`. You should provide a range that does not collide with an existing subnet within the VPC.
 - o **VPCID**: The ID of the VPC where you want the cell and subnet to be created

AWS Services Edit N. Virginia Support

Select Template
Specify Details
 Options
 Review

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

BoshHost	<input type="text"/>	Bosh director (e.g. 127.0.0.1).
BoshPassword	<input type="password"/>	Password for bosh director.
BoshUserName	<input type="text"/>	Username for bosh director.
CellName	<input type="text" value="windows_cell"/>	The name for your cell
ContainerizerPassword	<input type="password"/>	Password for containerizer user e.g. password123!
DesiredCapacity	<input type="text"/>	The number of Windows cells to deploy
KeyPair	<input type="text" value="Search"/>	A keypair that you have the private key to
NATInstance	<input type="text" value="Search by ID, or Name tag value"/>	The instance ID of the NAT box. Search for 'NAT' (case-sensitive) in the dropdown, it will typically be the first result.
SecurityGroup	<input type="text" value="Search by ID, name or Name tag value"/>	Security group ID to use for the Windows cells
SubnetCIDR	<input type="text" value="10.0.100.0/24"/>	The IP range to use for the Windows cell subnet. It should not collide with an existing subnet within the VPC.
VPCID	<input type="text" value="Search by ID, or Name tag value"/>	The vpc in which the cell and subnet will be created
ZZZDiegoWindowsMsiUrl	<input type="text" value="https://github.com/cloudfoundry-incubator/di"/>	URL of the DiegoWindows.msi to install
ZZZGardenWindowsMsiUrl	<input type="text" value="https://github.com/cloudfoundry-incubator/g"/>	URL of the GardenWindows.msi to install
ZZZGenerateUrl	<input type="text" value="https://github.com/cloudfoundry-incubator/di"/>	URL of generate.exe
ZZZSetupPs1Url	<input type="text" value="https://github.com/cloudfoundry-incubator/g"/>	URL of the setup.ps1 to run

The CloudFormation template performs the following functions:

1. Configures the Windows cell for the appropriate availability zone, based on the security group that you provide.
2. Installs the MSI.
3. Registers itself with Diego.

The CloudFormation template succeeds only if all services are up and running after installation. To debug a failed installation, navigate to Advanced Options and set **Rollback on failure** to **No**.

▼ Advanced

You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

Notification options

No notification

New Amazon SNS topic

Topic	<input type="text"/>
Email	<input type="text"/>

Existing Amazon SNS topic

Timeout i Minutes

Rollback on failure i Yes

No



Install Manually

Prerequisites

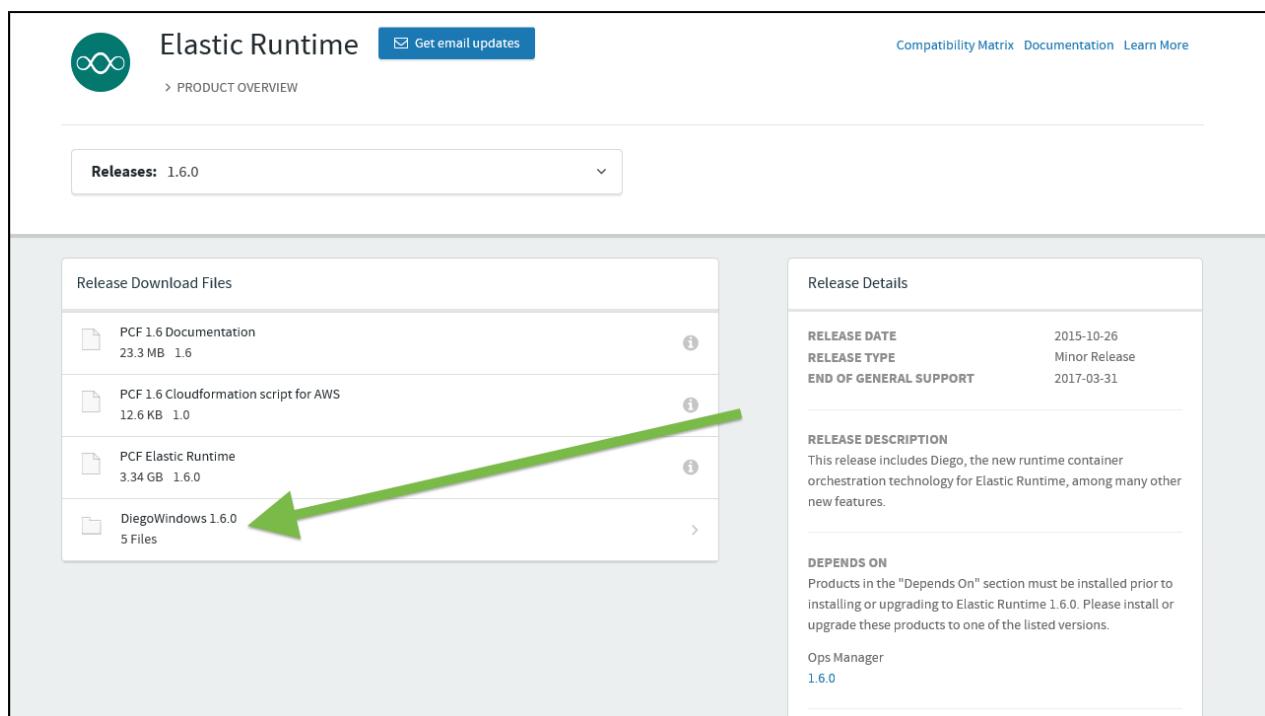
- A working Diego deployment
- A Windows Server 2012R2 VM instance that is routable to your Diego deployment
 - See [recommended instance types](#) in the GitHub Diego release repo for details.
 - If you are creating a new Windows image, and not using one predefined and supplied by your IaaS, we recommend 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

Follow the instructions below to download and configure the Windows cell.

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.



Elastic Runtime

Releases: 1.6.0

Release Download Files

- PCF 1.6 Documentation (23.3 MB)
- PCF 1.6 Cloudformation script for AWS (12.6 KB)
- PCF Elastic Runtime (3.34 GB)
- DiegoWindows 1.6.0 (5 Files)**

Release Details

RELEASE DATE: 2015-10-26
 RELEASE TYPE: Minor Release
 END OF GENERAL SUPPORT: 2017-03-31

RELEASE DESCRIPTION: This release includes Diego, the new runtime container orchestration technology for Elastic Runtime, among many other new features.

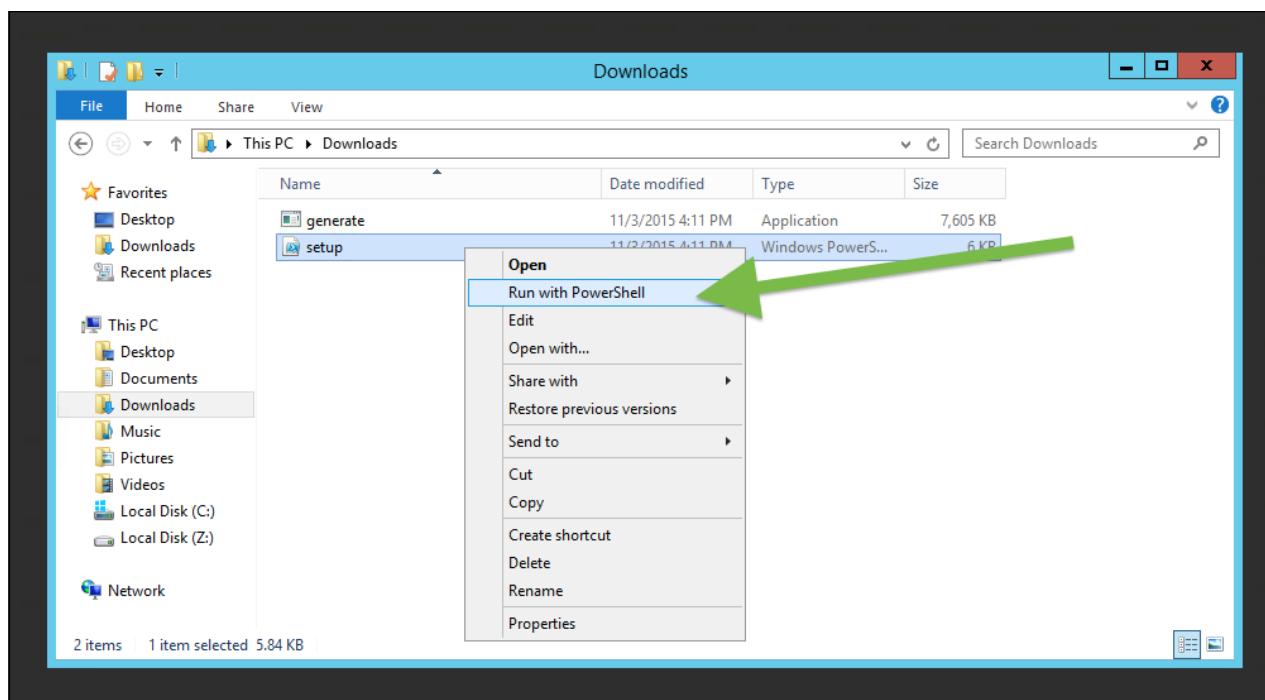
DEPENDS ON: Ops Manager 1.6.0

- 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

- Using **File Explorer**, navigate to the location where you downloaded the **setup.ps1** and **generate.exe** files.
- 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.



Downloads

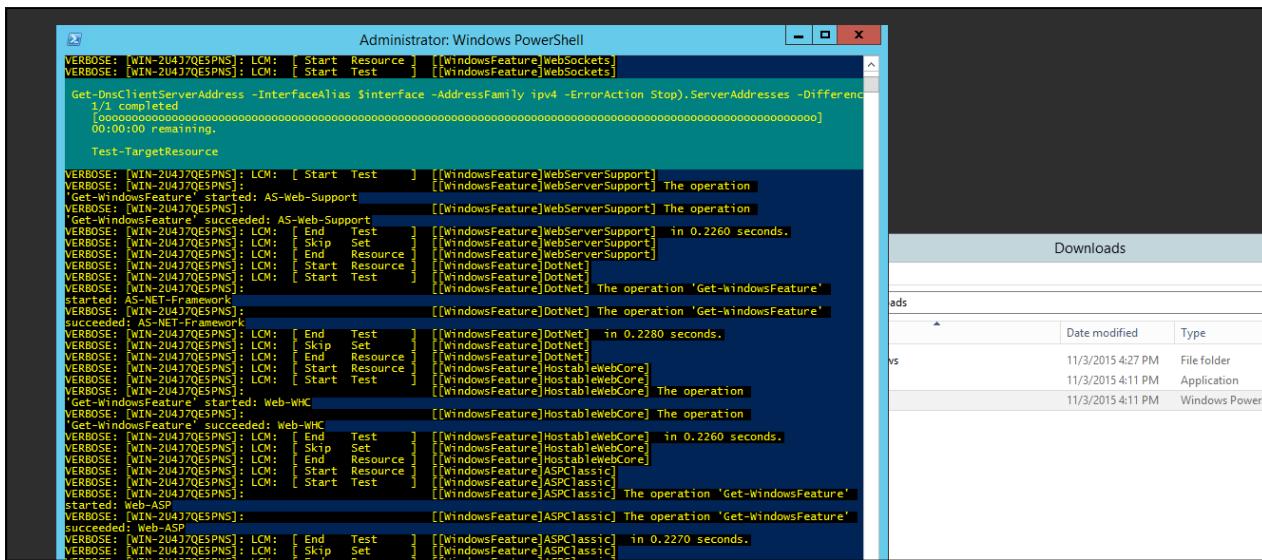
File Home Share View

Downloads

Name	Date modified	Type	Size
generate	11/3/2015 4:11 PM	Application	7,605 KB
setup	11/2/2015 4:11 PM	Windows PowerS...	6 KB

Open

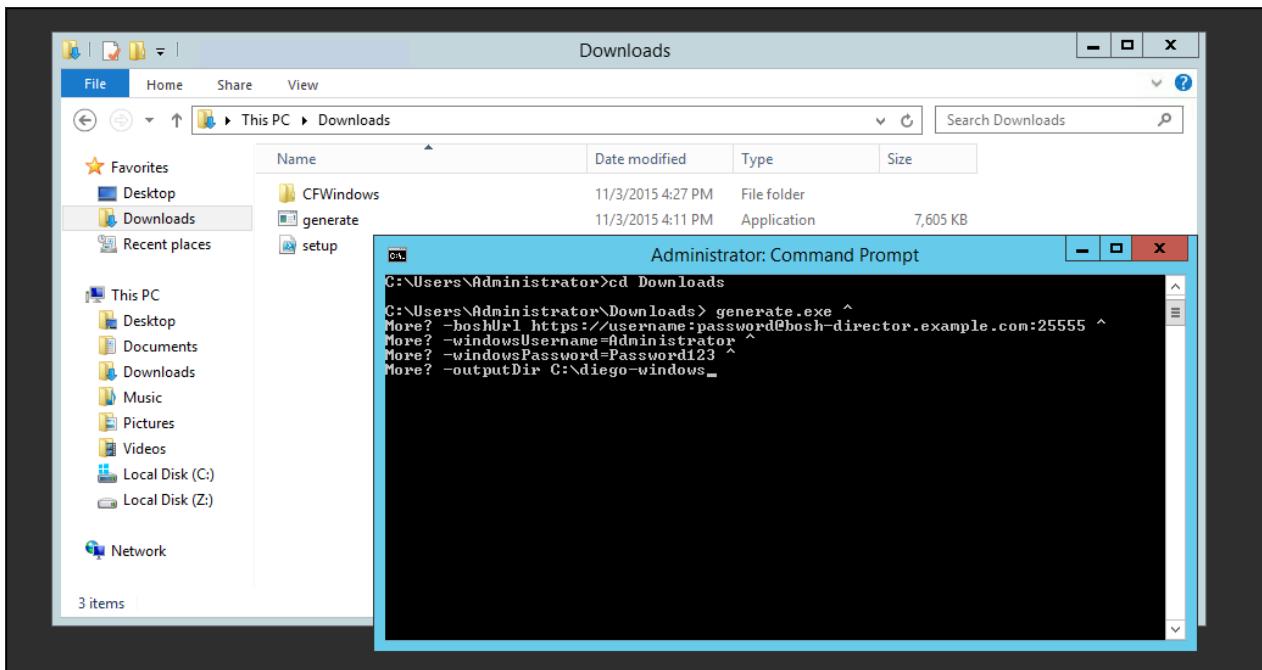
- Run with PowerShell**
- Edit
- Open with...
- Share with
- Restore previous versions
- Send to
- Cut
- Copy
- Create shortcut
- Delete
- Rename
- Properties



Step 3: Run Install Script Generator

- From a command prompt, run `generate.exe` with the following arguments:
 - `boshUrl`: Find your `boshUrl` in the [troubleshooting](#) documentation.
 - `windowsUsername`: Used to run Containerizer. The user provided must be a member of the Administrators group. All other services will run as the SYSTEM user.
 - `windowsPassword`: The password for your specified Windows user.
 - `outputDir`: The directory that will contain the required certificates and a script to run the installers.

```
$ generate.exe ^
-boshUrl https://username:password@bosh-director.example.com:25555 ^
-windowsUsername=Administrator ^
-windowsPassword=Password123 ^
-outputDir C:\diego-windows
```



Step 4: Install MSI

- Download `DiegoWindows.msi` and `GardenWindows.msi` from the same Pivotal Network file group to the `outputDir` that you specified above.

Release Download Files / DiegoWindows 1.6.0

GardenWindows.msi 4.67 MB 1.6.0	
setup.ps1 5.85 KB 1.6.0	
DiegoWindows.msi 11.8 MB 1.6.0	
AWS CloudFormation 8.96 KB 1.6.0	
generate.exe 7.43 MB 1.6.0	

Release Details

RELEASE DATE	2015-10-26
RELEASE TYPE	Minor Release
END OF GENERAL SUPPORT	2017-03-31

RELEASE DESCRIPTION

This release includes Diego, the new runtime container orchestration technology for Elastic Runtime, among many other new features.

DEPENDS ON

Products in the "Depends On" section must be installed prior to installing or upgrading to Elastic Runtime 1.6.0. Please upgrade these products to one of the listed versions.

Ops Manager

Release Download Files / DiegoWindows 1.6.0

Save As

Organize New folder

Name	Date modified	Type	Size
DiegoWindows	11/3/2015 4:11 PM	Windows Installer ...	12,092 KB

File name: GardenWindows

Save as type: Windows Installer Package

Save Cancel

Do you want to run or save GardenWindows.msi (4.67 MB) from dtb5pzswc1e.cloudfront.net?

This type of file could harm your computer.

Run Save Cancel

Release Details

2015-10-26
Minor Release
2017-03-31

Runtime container
Runtime, among many other

must be installed prior to
Elastic Runtime 1.6.0. Please install or
upgrade these products to one of the listed versions.

- From a command prompt, run `install.bat` from the `outputDir`.

Downloads

diego-windows

Name	Date modified	Type	Size
bbs_ca	11/3/2015 4:32 PM	Security Certificate	2 KB
bbs_client	11/3/2015 4:32 PM	Security Certificate	2 KB
bbs_client.key	11/3/2015 4:32 PM	KEY File	2 KB
consul_agent	11/3/2015 4:32 PM	Security Certificate	2 KB
consul_agent.key	11/3/2015 4:32 PM	KEY File	2 KB
consul_ca	11/3/2015 4:32 PM	Security Certificate	2 KB
consul_encrypt.key	11/3/2015 4:32 PM	KEY File	2 KB
DiegoWindows	11/3/2015 4:32 PM	Windows Installer ...	12,092 KB
GardenWindows	11/3/2015 4:32 PM	Windows Installer ...	4,670 KB
install	11/3/2015 4:32 PM	Windows Script Host ...	2 KB

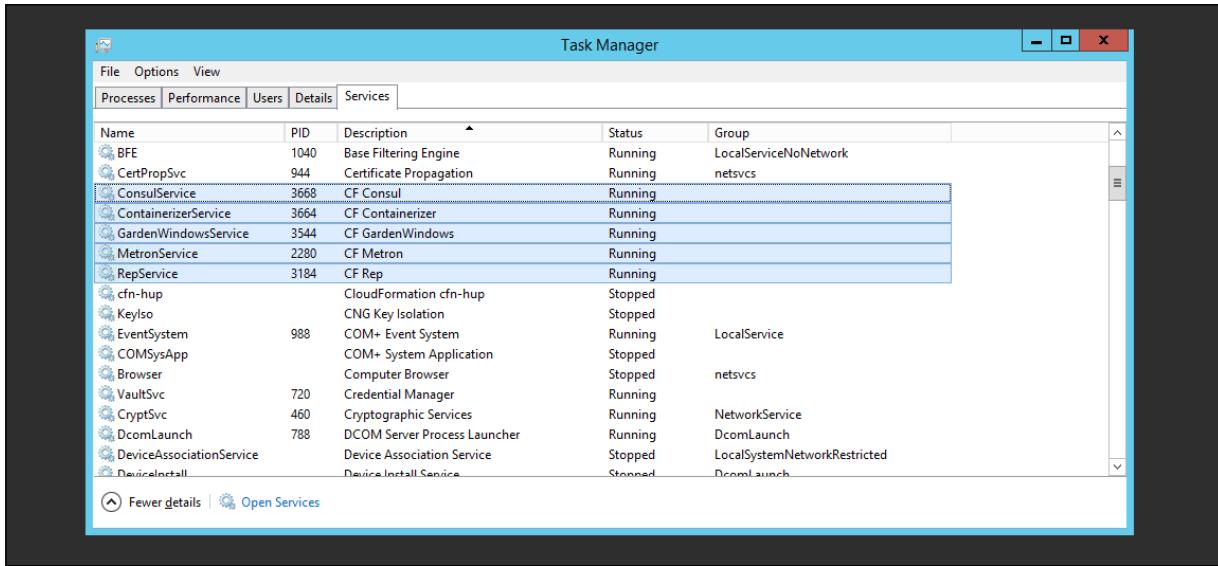
Administrator: Command Prompt

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\Administrator>cd C:\diego-windows
C:\diego-windows>install.bat
```

Step 5: 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`.

Troubleshooting Diego for Windows

Page last updated:

This topic describes how to troubleshoot a Windows cell in a Diego deployment.

Resolve Application Errors

Ensure that your .NET app is ready for deployment. You usually see the following errors after pushing an app.

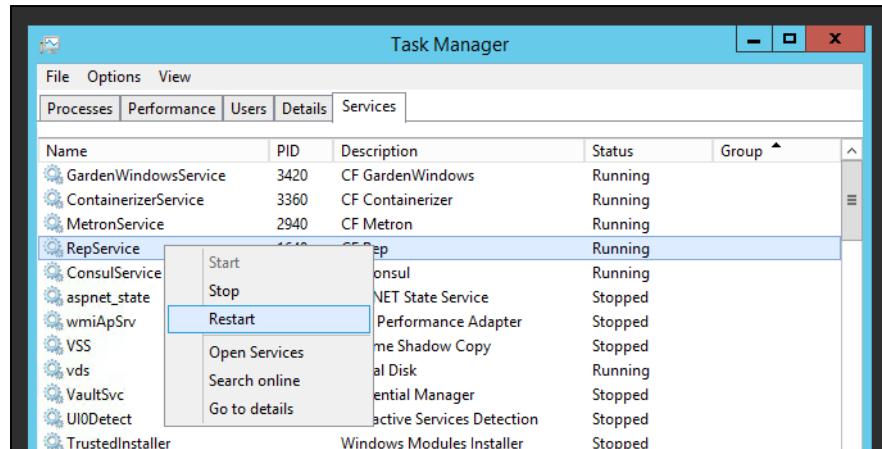
Error

NoCompatibleCell This error usually indicates that the RepService has not yet registered your Windows cell with the rest of your Pivotal Cloud Foundry (PCF) deployment. The RepService attempts to reconnect on an interval, and can sometimes resolve itself within a few minutes.

```
+ cf start nora
Starting app nora in org org / space space as admin...
FAILED
NoCompatibleCell
TIP: use 'cf logs nora --recent' for more information
```

Resolution

Restart the RepService within your cell to trigger an immediate reconnection.



Error

Start Unsuccessful This error usually indicates that your app is misconfigured for your PCF Windows environment, but can also indicate your app does not contain the required .dll's and dependencies.

```
Uploaded build artifacts cache (88B)
Uploaded droplet (1.4M)
Uploading complete

0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 crashed
FAILED
Start unsuccessful
TIP: use 'cf logs nora --recent' for more information
```

```
2016-02-26T15:14:20.66-0500 [APP/0]      OUT Server Started.... press CTRL + C to stop
2016-02-26T15:15:17.30-0500 [CELL/0]      OUT Creating container
2016-02-26T15:15:17.79-0500 [CELL/0]      OUT Successfully created container
2016-02-26T15:15:18.42-0500 [CELL/0]      OUT Starting health monitoring of container
2016-02-26T15:15:20.13-0500 [CELL/0]      ERR Timed out after 1m0s: health check never passed.
2016-02-26T15:15:20.23-0500 [APP/0]      OUT Exit status -26
2016-02-26T15:15:20.25-0500 [CELL/0]      OUT Exit status -26
2016-02-26T15:15:20.71-0500 [API/0]      OUT App instance exited with guid 2cc86330-5500-43f6-8404-d7c3a6d924
48f-c4313b7e4c60", "index"=>0, "reason"=>"CRASHED", "exit_description"=>"2 error(s) occurred:\n\n* 1 error(s)
r(s) occurred:\n\n* cancelled\n* cancelled", "crash_count"=>2, "crash_timestamp"=>1456517720697401194, "versi
```

Resolution

Push your app from a directory containing either a `.exe` binary or a valid `Web.config` file for .NET apps. Alternatively, add the `-p` flag to your `cf push` command and specify the path to the directory that contains the `.exe` or `Web.config` file.

Ensure that your app dependencies are contained in your pushed app.

Find Errors Using Hakim

Hakim is a diagnostic tool that reveals common configuration issues with Windows cells.

Install and Run Hakim

1. Navigate to the [Elastic Runtime product on Pivotal Network](#).
2. Download the `hakim.exe` binary to your Windows cell from your `DiegoWindows` download.
3. In a shell window, navigate to the directory that contains the downloaded binary.
4. Execute the binary. Here is example hakim output:

```
PS C:\Users\Administrator\Downloads> .\hakim.exe
2016/02/26 21:04:35 The following processes are not running: garden-windows.exe
2016/02/26 21:04:36 Failed to create container
Post http://api/containers: dial tcp 127.0.0.1:9241: ConnectEx tcp: No connection could be made because the target machine actively refused it.
```

Resolve Common Errors

Hakim only outputs to the console if it detects errors. Here are some common errors and resolutions:

Error

The following processes are not running This usually indicates a failed deployment.

Resolution

Re-provision your Windows components. If this does not fix this issue, contact support with the exact deployment steps followed and version of PCF deployed.

Error

Failed to resolve consul host This usually indicates interference with DNS resolution on your Windows cell.

Resolution

To resolve this error, set localhost 127.0.0.1 as the primary DNS server for the active network adapter.

Error

Fair Share CPU Scheduling must be disabled

Resolution

You must disable this setting for your Windows cell to function properly. Turn this off through the **Group Policy Management** console, and then restart your Windows cell.

Error

Windows firewall service is not enabled Diego for Windows enforces PCF security group settings for apps running on the cell through the Windows firewall. Apps

can run without this, but security groups do not work correctly and apps have unrestricted network access.

Resolution

Enable the Windows firewall service.

Error

There was an error detecting ntp synchronization on your machine Clock skew with other PCF components can occur if NTP is not configured. Clock skew can result in

odd errors: for example, not receiving any application metrics for apps running on the affected machine.

Resolution

For your Windows cell, use the same NTP server as the rest of your PCF deployment.

Error

Failed to create container This usually indicates an issue with the Windows containerization service.

Resolution

Contact support and provide the full output of this error.

Troubleshoot Other Issues

Look at the **Event Viewer** logs in Windows to troubleshoot other issues:

1. Navigate to **Windows Logs > Application**.
2. Review log messages from the services running in DiegoWindows.
3. To isolate the issue, clear the log, reproduce the issue, and review the latest messages.
4. Include the content of these messages in your support request if you need to contact support.

The screenshot shows the Windows Event Viewer interface. The left pane displays a navigation tree with 'Event Viewer (Local)', 'Custom Views', 'Windows Logs' (selected), and 'Applications and Services Logs'. The 'Windows Logs' section shows 'Application', 'Security', 'Setup', 'System', and 'Forwarded Events'. The 'Applications and Services Logs' section shows 'Subscriptions'. The right pane displays a table of events for the 'Application' log. The table has columns for 'Level', 'Date and Time', 'Source', 'Event ID', and 'Task Category'. The 'Event ID' column is sorted in descending order. The last event in the list is highlighted in blue. The event details are shown in the bottom pane, with the 'General' tab selected. The error message is: 'System.Net.Sockets.SocketException (0x80004005): No such host is known at System.Net.Dns.GetAddrInfo(String name) at System.Net.Dns.InternalGetHostByName(String hostName, Boolean includeIPv6) at System.Net.Dns.GetHostAddresses(String hostNameOrAddress) at System.Net.Sockets.UdpClient.Connect(String hostname, Int32 port) at System.Net.Sockets.UdpClient..ctor(String hostname, Int32 port) at Utilities.Syslog..ctor(String host, Int32 port, String machineName, String sender) at Utilities.Syslog.Build(Dictionary`2 p, String eventSource)'.

Application Number of events: 85				
Level	Date and Time	Source	Event ID	Task Category
Information	3/1/2016 10:56:53 AM	GardenWindowsService	0	None
Information	3/1/2016 10:56:53 AM	RestartManager	10005	None
Warning	3/1/2016 10:56:53 AM	RestartManager	10010	None
Information	3/1/2016 10:56:52 AM	RestartManager	10000	None
Information	3/1/2016 10:56:52 AM	MsInstaller	1040	None
Information	3/1/2016 10:48:49 AM	RestartManager	10001	None
Information	3/1/2016 10:48:50 AM	GardenWindows	0	None
Information	3/1/2016 10:48:50 AM	Containerizer	0	None
Information	3/1/2016 10:48:50 AM	Containerizer	0	None
Information	3/1/2016 10:48:49 AM	GardenWindowsService	0	None
Information	3/1/2016 10:48:49 AM	GardenWindows	0	None
Information	3/1/2016 10:48:49 AM	GardenWindows	0	None
Error	3/1/2016 10:48:49 AM	GardenWindows	0	None
Information	3/1/2016 10:48:49 AM	MsInstaller	1042	None
Information	3/1/2016 10:48:49 AM	MsInstaller	1033	None

Event 0, GardenWindows

General Details

```
System.Net.Sockets.SocketException (0x80004005): No such host is known
at System.Net.Dns.GetAddrInfo(String name)
at System.Net.Dns.InternalGetHostByName(String hostName, Boolean includeIPv6)
at System.Net.Dns.GetHostAddresses(String hostNameOrAddress)
at System.Net.Sockets.UdpClient.Connect(String hostname, Int32 port)
at System.Net.Sockets.UdpClient..ctor(String hostname, Int32 port)
at Utilities.Syslog..ctor(String host, Int32 port, String machineName, String sender)
at Utilities.Syslog.Build(Dictionary`2 p, String eventSource)
```

The Pivotal Cloud Ops Approach to Monitoring a Pivotal Cloud Foundry Deployment

Page last updated:

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

Backend Health		Frontend Health	
NATS - 5m	NATS - 24h	Router - 5m	Router - 24h
100%	100%	100%	100%
Doppler - 5m	Doppler - 2...	DEA - 5m	DEA - 24h
100%	100%	100%	100%
Stats - 5m	Stats - 24h	API - 5m	API - 24h
100%	100%	100%	100%
HM9000 - 5m	HM9000 - 2...	UAA - 5m	UAA - 24h
100%	100%	100%	100%
Bosh - 5m	Bosh - 24h		
100%	100%		
NAT Box - 5m	NAT box - 2...		
1.0	1.0		
ETCD - 5m	ETCD - 24h		
100%	100%		

What we 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:
	<ul style="list-style-type: none"> • NATS • Doppler • Stats • HM9000 • BOSH • NAT Box • ETCD • Router • DEA • API • UAA
	Why we monitor it
	To ensure that all VMs are functioning properly.
	System metric
	<code>bosh.healthmonitor.system.healthy</code>
	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

What we monitor	Requests per second for each of the following components:
	<ul style="list-style-type: none"> • Router • DEA • API

	<ul style="list-style-type: none"> • UAA
Why we monitor it	To track the flow of traffic through the components in the system.
System metric	<code>cf.collector.router.requests(component: app/cloudcontroller/uaa)</code>
Alerts triggered	None
Notes	None

NATS Traffic Delta

What we monitor	<p>Delta of average NATS traffic over the last hour.</p> <p>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.</p>
Why we monitor it	To detect significant drops in NATS traffic. A sudden drop might indicate a problem with the health of the NATS VMs.
System metric	<code>aws.ec2.network_in</code>
Alerts triggered	None
Notes	None

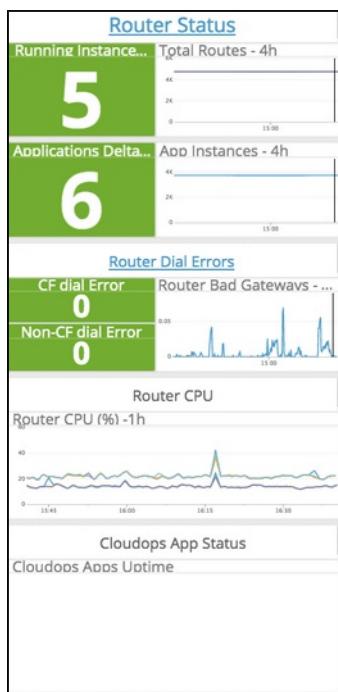
ETCD Leader Uptime

What we monitor	Time since the ETCD leader last was down.
Why we monitor it	When the ETCD leader goes down, it usually indicates a push failure.
System metric	<code>cloudops_tools.etcd_leader_health</code>
Alerts triggered	None
Notes	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.

SSH Attempts

What we monitor	Total SSH attempts. We log the count of connection attempts to our systems on the SSH port (port 22).
Why we monitor it	A spike in SSH attempts is a good indicator of SSH-cracker attacks.
System metric	<code>cloudops_tools.ssh-abuse-monitor</code>
Alerts triggered	None
Notes	<ul style="list-style-type: none"> DEAs send their iptables logs to Logsearch. A Cloud Ops internal app polls Logsearch for first packets and pushes the count to Datadog. 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.

The Router Status Column



App Instance Count

What we monitor	Count of running app instances.
Why we monitor it	Unexpected large fluctuations in app count can indicate malicious user behavior or Cloud Foundry component issues.
System metric	<code>avg:cf.collector.HM9000.HM9000.NumberOfAppsWithAllInstancesReporting</code>
Alerts triggered	running app number change rate
Notes	Spikes in this metric might indicate the need to add more resources. Pay attention to Reservable Stagers and DEA Runners Needed .

Total Routes

What we monitor	Route count from the router, indicated as a delta over the last N minutes.
Why we monitor it	The count on all routers should be the same. If this count differs between routers, it usually indicates a NATS problem.
System metric	<code>cf.collector.router.total_routes</code>
Alerts triggered	prod CF: Number of routes in the router's routing table is too low
Notes	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.

Router Dial Errors

What we monitor	Separate indicators monitor 5xx codes from the routers to backend CF components and user apps, respectively.
Why we monitor it	Indicates failures connecting to components.
System metric	<code>avg:cloudops_tools.app_instance_monitor.router.dial.errors{domain:run.pivotl.io} / avg:cloudops_tools.app_instance_monitor.router.dial.errors{cf_component:false}</code>
Alerts triggered	<ul style="list-style-type: none"> No data for router dial errors Router dial errors for console.run.pivotl.io

	<ul style="list-style-type: none"> Too many router dial errors for cf components
Notes	<ul style="list-style-type: none"> 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 <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.

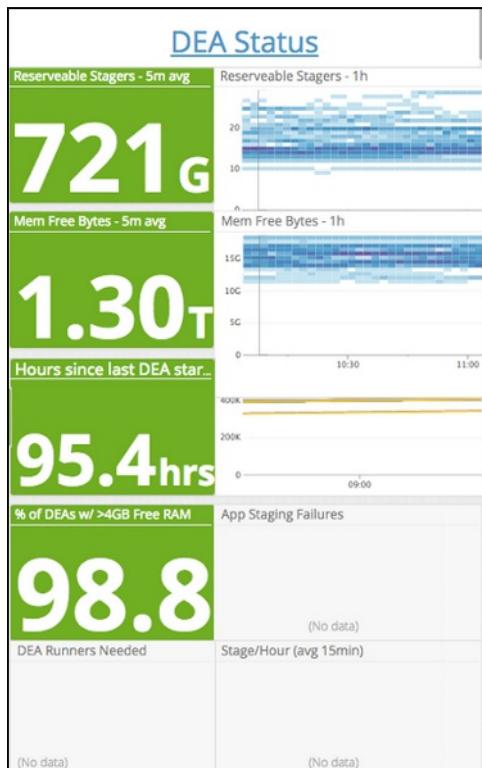
Router CPU

What we monitor	OS-level CPU usage.
Why we monitor it	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.
System metric	<code>bosh.healthmonitor.system.cpu.user{deployment:cf-cfapps-io2,job:}</code>
Alerts triggered	None
Notes	In general, we add routers whenever doing so may resolve issues.

AWS Events

What we monitor	The feed from <code>aws ec2 events</code> .
Why we monitor it	Contains important or critical information from our IaaS about virtual machines, RDS, etc.
System metric	N/A
Alerts triggered	None
Notes	Only applies to Cloud Foundry deployments on AWS.

The DEA Status Column



Reservable Stagers

What we monitor	How many 1GB apps can be staged.
Why we monitor it	Better capacity planning. If this number is too low, users are unable to stage new apps or scale existing apps.
System metric	<code>sum:cf.collector.reservable_stagers</code>
Alerts triggered	Alerts when there are insufficient reservable stagers.
Notes	Used to calculate DEA Runners Needed .

Mem Free Bytes

What we monitor	Total free memory on the DEAs
Why we monitor it	Better capacity planning. If the number is too low, users will not be able to stage more apps or scale staged apps.
System metric	<code>cf.collector.mem_free_bytes{job:deas}</code>
Alerts triggered	None
Notes	Used to calculate DEA Runners Needed .

Hours Since Last DEA Start

What we monitor	Number of hours since the last DEA was started.
Why we monitor it	Frequent DEA restarts might indicate a bug in the DEA. Frequent restarts also directly impact customer-facing performance by causing apps to restage.
System metric	<code>(min:cf.collector.uptime_in_seconds{job:deas}) / 3600</code>
Alerts triggered	None
Notes	None

Percentage of DEAs > 4GB Free RAM

What we monitor	Percentage of existing DEAs available to stage 4 GB apps.
Why we monitor it	To ensure that users can push 4 GB apps.
System metric	<code>avg:cloudbuild_tools.dea_monitor.reservable_stagers_aggregate * 100</code>
Alerts triggered	None
Notes	The <code>cloudbuild_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.

Application Staging Failures

What we monitor	How many app staging requests failed on the runner.
Why we monitor it	To ensure that users are not facing issues staging apps.
System metric	<code>avg:cloudbuild_tools.failed_stages.counts_per_5min</code>
Alerts triggered	None
Notes	The <code>cloudbuild_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.

DEA Runners Needed

What we monitor	Whether any DEA runners are needed.
Notes	Used to calculate DEA Runners Needed .

Why we monitor it	To tell us how many more runners to add
System metric	<code>avg:cloudops_tools.dea_monitor.additional_runners_needed</code>
Alerts triggered	None
Notes	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.

Stages per Hour

What we monitor	How many apps are being staged every hour.
Why we monitor it	This metric is useful for statistical purposes and for debugging issues with the system. If there are no apps being staged for 15 minutes, we might want to check for problems with the system.
System metric	<code>avg:cloudops_tools.stages.counts_per_5min.rollup(sum, 3600)</code>
Alerts triggered	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.
Notes	None

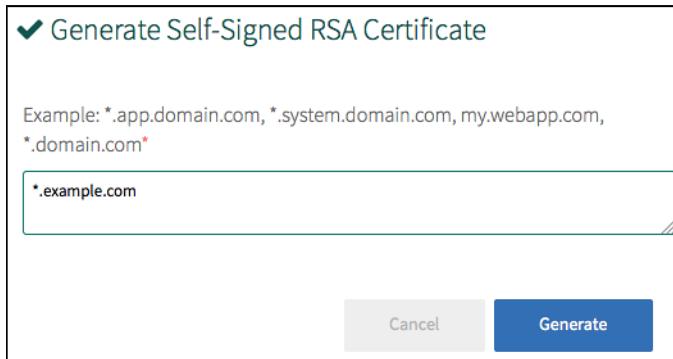
Providing a Certificate for your SSL Termination Point

Page last updated:

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 the [Installing PCF Guide](#) for help installing PCF on your IaaS of choice.

1. In the PCF Ops Manager Installation Dashboard, click the **Elastic Runtime** tile.
2. Select **Security Config**.
3. Enter your SSL Certificate, as follows:
 - o (Option 1) In a production environment, use a signed **SSL Certificate** from a known certificate authority (CA). Copy and paste the values for **Certificate PEM** and **Private Key PEM** from the signed certificate into the appropriate text fields.
 - o (Option 2) In a development or testing environment, you may use a self-signed certificate. To use a self-signed certificate, follow the steps below:
 - Click the **Generate Self-Signed RSA Certificate** link to launch the RSA certificate wizard.
 - Enter your system and app domains in wildcard format. Optionally, also add any custom domains in wildcard format. The example below uses `*.example.com`.

Note: Wildcard DNS records only work for a single domain name component or component fragment. For example, `*.domain.com` works for `apps.domain.com` and `system.domain.com`, but not for `my.apps.domain.com` or `my.system.domain.com`.



- Click **Generate**.

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.

- Elastic Runtime populates the **SSL Certificate** fields with RSA certificate and private key information.
- Select the **Trust Self-Signed Certificates** checkbox.

4. (Optional) Enter a colon-separated list of custom SSL ciphers to pass to HAProxy in the **HAProxy SSL Ciphers** field. If you leave this field empty, PCF uses the default SSL ciphers specified in `cf-release/jobs/haproxy/spec`.
5. 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.
6. Check **Enable cross-container traffic within each DEA** if your installation includes microservices that require cross-container networking, such as [Spring Cloud](#).

Security settings for HAProxy, Router, and DEAs

Assign Networks

Assign Availability Zones

System Database Config

File Storage Config

IPs and Ports

Security Config

MySQL Proxy Config

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Experimental Features

SSL Termination Certificate *

```
-----BEGIN CERTIFICATE-----
MIIDLTCCAHWgAwIBAgIUCE3jK12Al0ZD28
36WvklVVE2wwTkwdQYJKoZlhvcNAQEF
BQAwPjELMAKGA1UEBhMCVVMxEAOBg
NVBAoMBlBpdm90YWwxHTAbBgNVBAM
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEz9fvaybczuZz6kAb+
XgVu3TYEaFOOpAMHDtwJ8OjaTPmKD0j
UHnTQwnrKBJn0fO/QuyInA9GgcMBLI
5L+3s871cW2NGwVzIKjIwxsNO/03ra
-----
```

[Generate Self-Signed RSA Certificate](#)

Trust Self-Signed Certificates

HAProxy SSL Ciphers

Enable cross-container traffic within each DEA

Save

7. Click **Save**.

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 \(UAAC\)](#)
- [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](#)

Managing Custom Buildpacks

Page last updated:

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:

```
$ 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 `cf buildpacks`.

The following example shows the output from running the `cf buildpacks` command after the administrator added a Python buildpack:

```
$ cf buildpacks
Getting buildpacks...

buildpack      position  enabled  locked  filename
ruby_buildpack  1        true     false    buildpack_ruby_v46-245-g2fc4ad8.zip
nodejs_buildpack 2        true     false    buildpack_nodejs_v8-177-g2b0a5cf.zip
java_buildpack  3        true     false    buildpack_java_v2.1.zip
python_buildpack 4        true     false    buildpack_python_v2.7.6.zip
```

Rename a Buildpack

```
$ cf rename-buildpack BUILDPACK_NAME NEW_BUILDPACK_NAME
```

For more information on renaming a buildpack, see the [CLI documentation](#).

Update a Buildpack

```
$ 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](#).

Disabling Custom Buildpacks

Disabling Custom Buildpacks from Ops Manager

You can disable custom buildpacks using your Ops Manager Elastic Runtime tile. From the **Cloud Controller** tab, check the **Disable Custom**

The screenshot shows the 'Pivotal Elastic Runtime' settings page. The 'Cloud Controller' tab is active. On the left, a list of configuration items is shown with checkboxes: 'Assign Networks', 'Assign Availability Zones', 'IPs and Ports', 'MySQL Proxy Config', 'Cloud Controller' (which is checked and highlighted with a grey arrow), 'External Endpoints', 'SSO Config', 'LDAP Config', 'SMTP Config', 'Lifecycle Errands', 'Resource Config', and 'Stemcell'. To the right, there are input fields and dropdowns: 'System Domain *' (empty), 'Apps Domain *' (empty), 'Cloud Controller DB Encryption Key' (set to 'Secret'), 'Maximum File Upload Size (MB) (min: 1024, max: 2048)' (set to '1024'), 'Default Quota App Memory (MB) (min: 10240, max: 102400)' (set to '10240'), and 'Default Quota Service Instances (min: 0, max: 1000)' (set to '100'). A 'Save' button is at the bottom.

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

Page last updated:

This topic outlines how 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-release](#) 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 ['rootfs' 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:
 - o [Buildpack App Lifecycle](#)
 - o [Docker App Lifecycle](#)
 - o [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 rootfs available locally on the Diego cell, at `/var/vcap/packages/pancakes-rootfs/rootfs`. 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:

```
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:

```
access:
- name: ssh_proxy
  release: diego
- name: consul_agent
```

```

release: cf
- name: metron_agent
  release: cf
- name: file_server
  release: diego
- name: pancakes-app-lifecycle
  release: pancakes

```

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-roots` job. For example, in the `preloaded_rootfses:` property under `rep:` 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"

```

Creating and Managing Users with the cf CLI

Page last updated:

Using the cf Command Line Interface (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

FUNCTION	COMMAND	EXAMPLE
Create a new user	cf create-user USERNAME PASSWORD	<code>cf create-user Alice pa55w0rd</code>
Delete a user	cf delete-user USERNAME	<code>cf delete-user Alice</code>

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.

FUNCTION	COMMAND	EXAMPLE
View the organizations belonging to an account	cf orgs	<code>cf orgs</code>
View all users in an organization by role	cf org-users ORGANIZATION_NAME	<code>cf org-users my-example-org</code>
Assign an org role to a user	cf set-org-role USERNAME ORGANIZATION_NAME ROLE	<code>cf set-org-role Alice my-example-org OrgManager</code>
Remove an org role from a user	cf unset-org-role USERNAME ORGANIZATION_NAME ROLE	<code>cf unset-org-role Alice my-example-org OrgManager</code>

App Space Roles

Each app space role applies to a specific app space.

 **Note:** By default, the org manager has app space manager permissions for all spaces within the organization.

Valid [app space roles](#) are SpaceManager, SpaceDeveloper, and SpaceAuditor.

FUNCTION	COMMAND	EXAMPLE

View the spaces in an org	cf spaces	cf spaces
View all users in a space by role	cf space-users ORGANIZATION_NAME SPACE_NAME	cf space-users my-example-org development
Assign a space role to a user	cf set-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE	cf set-space-role Alice my-example-org development SpaceAuditor
Remove a space role from a user	cf unset-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE	cf unset-space-role Alice my-example-org development SpaceAuditor

Creating and Managing Users with the UAA CLI (UAAC)

Page last updated:

Using the UAA Command Line Interface (UAAC), an administrator can create users in the User Account and Authentication (UAAC) server.

 **Note:** The UAAC only creates users in UAA, and does not assign roles in the Cloud Controller database (CCDB). In general, administrators create users with cf Command Line Interface (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:

- [UAAC Overview](#)
- [UAAC Sysadmin Guide](#)
- [Other UAAC Documentation](#)

Create an Admin User

1. Install the UAAC CLI, `uaac`.

```
$ gem install cf-uaac
```

2. Use `uaac target uaa.YOUR-DOMAIN` to target your UAAC server.

```
$ uaac target uaa.example.com
```

3. Record the `uaa:admin:client_secret` from your deployment manifest.

4. Use `uaac token client get admin -s ADMIN-CLIENT-SECRET` to authenticate and obtain an access token for the admin client from the UAAC server. UAAC stores the token in `~/.uaac.yml`.

```
$ uaac token client get admin -s MyAdminPassword
```

5. Use `uaac contexts` to display the users and applications authorized by the UAAC server, and the permissions granted to each user and application.

```
$ uaac contexts

[1]*[admin]
client_id: admin
access_token: aabbCCDD1122334455
token_type: bearer
expires_in: 43200
scope: uaa.admin clients.secret scim.read
jti: 11b1-abed1233
```

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 UAAC 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 UAAC server.

```
$ uaac contexts

[1]*[admin]
client_id: admin
...
scope: uaa.admin clients.secret scim.read
...

$ uaac client update admin --authorities "uaac client get admin | \
awk '/{e=0}/authorities:{e=1;if(e==1){$1=""};print}{" scim.write"

$ uaac token delete
$ uaac token client get admin
```

8. Use `uaac user add NEW-ADMIN-USERNAME -p NEW-ADMIN-PASSWORD --emails NEW-ADMIN-EMAIL` to create an admin user.

```
$ uaac user add Adam -p newAdminSecretPassword --emails newadmin@example.com
```

9. Use `uaac member add GROUP NEW-ADMIN-USERNAME` to add the new admin to the groups `cloud_controller.admin`, `uaa.admin`, `scim.read`, `scim.write`.

```
$ uaac member add cloud_controller.admin Adam
$ uaac member add uaa.admin Adam
$ uaac member add scim.read Adam
$ uaac member add scim.write Adam
```

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`.

```
$ uaac token client get admin -s MyAdminPassword
```

3. Run the following commands to grant all user under the mapped LDAP Group admin permissions:
 - o `uaac group map --name scim.read "GROUP-DISTINGUISHED-NAME"`
 - o `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.

```
$ cf login -u Adam -p newAdminSecretPassword
```

3. Use `cf create-user NEW-USER-NAME NEW-USER-PASSWORD` to create a new user.

```
$ cf create-user Charlie aNewPassword
```

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 -s 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: aabbCCDD1122334455
token_type: bearer
expires_in: 43200
scope: uaa.admin clients.secret password.read
jti: 11b1-abed1233
```

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:

- o 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`.
- o Use `uaac token delete` to delete the local token.
- o 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 '/:{e=0}/{e=1;if(e==1){$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.

```
$ 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 `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. 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: aabbCCDD1122334455
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin clients.secret
  jti: 11b1-abcd1233
```

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:
 - o 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`.
 - o Use `uaac token delete` to delete the local token.
 - o 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 'aabbc11-22a5-87-8056-beaf84'" --attributes emails

resources:
  emails:
    value: user1@example.com
```

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.
- 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 [UAA](#) 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 guide.

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:

```
$ uaac curl https://notifications.user.example.com/notifications -X PUT --data '{
  "source_name": "Cloud Ops Team",
  "notifications": {
    "system-going-down": { "critical": true, "description": "Cloud going down" },
    "system-up": { "critical": false, "description": "Cloud back up" }
  }
}'
```

- `source_name` has “Cloud Ops Team” set as the description.
- `system-going-down` and `system-up` are the notifications set.
- `system-going-down` and `system-up` are made `critical`, so no users can unsubscribe from that notification.

Create a Custom Template

 **Note:** To view a list of templates, you must have the `notifications_templates.read` scope. To create a custom template, you must have the `notification_templates.write` scope.

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.

```
$ uaac 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:

```
$ uaac 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.

```
$ uaac 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 and is experimental.

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](#).

Migrating Apps to Diego

Page last updated:

Introduction

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.

You might want to migrate your apps:

- To take advantage of the Diego architecture.
- Because migration is a prerequisite to upgrading to Pivotal Cloud Foundry (PCF) v1.7.

For information about the differences between Diego and DEA, see the [Differences Between DEA and Diego Architectures](#) topic.

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.

Stage	Who	Does What	Link to Details
1	Operators	Plan the migration schedule.	link
2	Operators	Prepare the deployment for Diego.	link
3	Developers	Determine if changes need to be made to their environment variables to be compatible with Diego.	link
4	Developers	Migrate apps to Diego. The method depends on whether downtime for the app is tolerable.	link
5	Developers	Verify that their apps have been migrated.	link
6	Operators	Confirm that all apps have been migrated and then scale down the DEA components.	link

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.

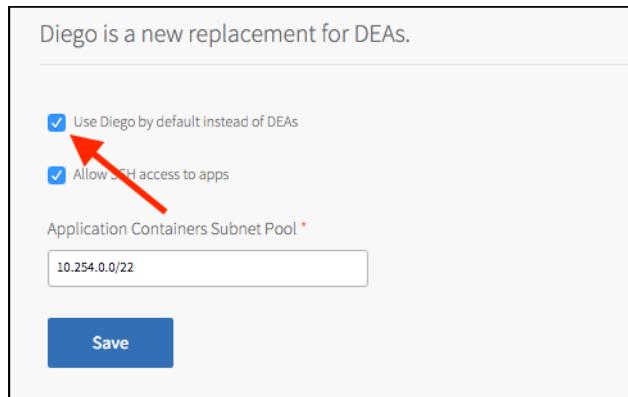
For example, if you have allocated three DEA instances with 32 GB of persistent disk and 32 GB of RAM for each, allocate three Diego Cell instances with 32 GB of persistent disk and 32 GB of RAM for each to have similar capacity on Diego as with DEAs.

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**.



Diego is a new replacement for DEAs.

Use Diego by default instead of DEAs

Allow SSH access to apps

Application Containers Subnet Pool *

10.254.0.0/22

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 worker 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.

If your code uses this environment variable...	Then...
<code>VCAP_APP_HOST</code>	replace it with <code>0.0.0.0</code>
<code>VCAP_APP_PORT</code>	replace it with <code>PORT</code>
<code>VCAP_APPLICATION</code> with any of the following keys: <ul style="list-style-type: none"> o <code>users</code> o <code>started_at_timestamp</code> o <code>state_timestamp</code> o <code>started_at</code> o <code>start</code> 	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 a 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,

```
$ cf dea-apps
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:

 **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 `-o ORG` migrates all apps in an org.
- o `-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`.
- o `-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 a045rw51-o358-4e26-9dfc-4c7365cf987 as example_user
...
Migration to Diego completed: 23 apps, 0 warnings
```

Example 2 — The following example migrates all DEA apps in `space2` to Diego, ten at a time.

```
$ cf migrate-apps -s space2 -p 10 diego
```

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.

```
$ cf diego-apps -o MY-ORG
Getting apps on the DEA runtime as example_user...
OK

name    org    space
diego-app1  MY-ORG  MY-SPACE
diego-app2  MY-ORG  MY-SPACE
```

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 dea-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:

Symptom	Explanation	Solution
Errors related to environment variables	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>SOMEPROP=\$HOME/SOME/PATH</code>	Remove interpolation from any environment variables.
Large applications may fail to stage, typically resulting in the following error: <code>Copying into the container failed</code>	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 <code>cf push -k</code> command to increase your disk quota.
Running the <code>cf files</code> command results in the error: <code>Request failed for app: APP_NAME, instance: INST_NUM and path: PATH</code>	Diego does not support the <code>cf files</code> command.	Run the <code>cf ssh APP_NAME</code> command to list files in your application instances.
Error: <code>Runner error: desire app failed: 503</code>	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 <code>*.my-domain.com</code> 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.

Using the 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 the Apps Manager](#)
- [Understanding Apps Manager Permissions](#)
- [Managing Spaces Using the Apps Manager](#)
- [Managing User Accounts in Spaces Using the Apps Manager](#)
- [Managing User Permissions Using the Apps Manager](#)
- [Monitoring Instance Usage with Apps Manager ↗](#)

Getting Started with the Apps Manager

Page last updated:

The Apps Manager is a tool to help manage organizations, users, applications, and spaces. Complete the following steps to invite new users to the Apps Manager.

1. Log in to the Apps Manager: [How to log in](#).
2. The Apps Manager displays information for your org and all of the spaces in your org.

3. In the Members tab, click **Invite New Members**.

4. Add an email address, assign a team role or roles, then click **Send Invite** to invite a new user to join your org.

Important Tips

- You can remove users from orgs, but not spaces. Instead, revoke the user's permissions in all spaces to effectively remove the user from the org.
- Not all of the Apps Manager pages are Ajax-enabled. Refresh the page to see the latest information.
- Changes to environment variables, as well as service bindings and unbindings, require a `cf restart` to update the application.

Understanding Apps Manager Permissions

Page last updated:

In most cases, the actions available to you on the Apps Manager are a limited subset of the commands available through the CLI. However, depending on your user role and the values that the Admin specifies for the [Apps Manager environment variables](#), you might be able to perform certain org and space management actions on the Apps Manager that usually only an Admin can perform with either the CLI or the Apps Manager.

The table below shows the relationship between specific org and space management actions that you can perform and the users that can perform these actions.

Note that:

- Admins can use either the CLI or the Apps Manager to perform these actions.
- Org Managers, like Admins, can perform all actions using the Apps Manager.
- Space Managers, like Admins and Org Managers, can assign and remove users from spaces using the Apps Manager.
- Apps Manager users of any role can create an org and view org and space users.

Command	CLI	Apps Manager		
	Admin	Admin or Org Manager	Space Manager	Org Auditor or Space Developer or Space Auditor
<code>create-org</code>	X	X	X	X
<code>delete-org</code>	X	X		
<code>rename-org</code>	X	X		
<code>org-users</code>	X	X	X	X
<code>set-org-role</code>	X	X		
<code>unset-org-role</code>	X	X		
<code>space-users</code>	X	X	X	X
<code>set-space-role</code>	X	X	X	
<code>unset-space-role</code>	X	X	X	

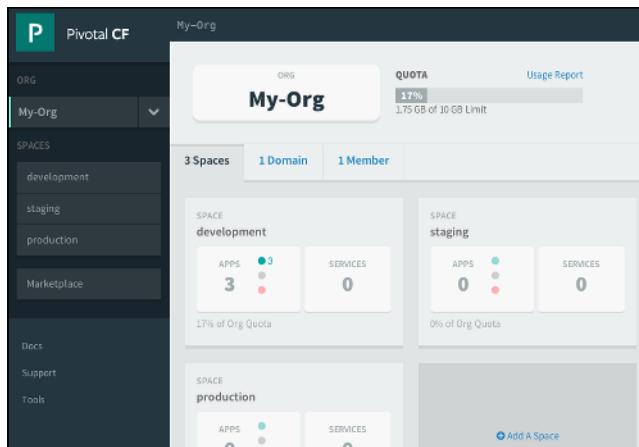
Managing Spaces Using the Apps Manager

Page last updated:

To manage a space in an org, you must have org manager or space manager permissions in that space.

Org managers have space manager permissions by default and can manage user accounts in all spaces associated with the org.

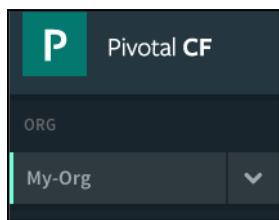
Log in to the Apps Manager: [How to log in.](#)



The screenshot shows the Apps Manager dashboard for the organization 'My-Org'. The top navigation bar shows 'My-Org' and a 'QUOTA' section indicating 17% usage of a 10 GB limit. Below this, there are three tabs: '3 Spaces', '1 Domain', and '1 Member'. The '3 Spaces' tab is selected. The dashboard displays three spaces: 'development', 'staging', and 'production'. The 'development' space is highlighted with a blue border. Each space card shows the number of apps and services. For 'development', there are 3 apps and 0 services, using 17% of the org's quota. For 'staging', there are 0 apps and 0 services, using 0% of the org's quota. For 'production', there are 0 apps and 0 services. A 'Add A Space' button is located at the bottom right of the space cards.

Orgs

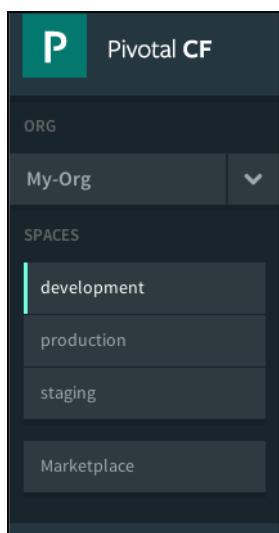
The current org is highlighted in the Apps Manager navigation. The drop-down menu on the right displays other orgs. Use this menu to switch between orgs.



The screenshot shows the Apps Manager navigation bar. The 'ORG' section shows 'My-Org' as the selected organization, indicated by a blue border. A dropdown menu is open to the right of 'My-Org', showing other organization options.

Spaces

The Apps Manager navigation also shows the spaces in the current org. The current space is highlighted.

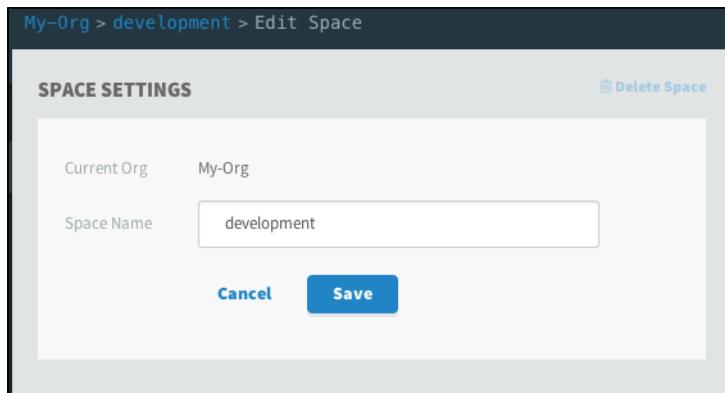


The screenshot shows the Apps Manager navigation bar. The 'ORG' section shows 'My-Org' as the selected organization. The 'SPACES' section shows three spaces: 'development', 'production', and 'staging'. The 'development' space is highlighted with a blue border, indicating it is the current space.

- **Add a Space:** Click the **Add A Space** button.
- **Switch Space:** Select a different space on the Org dashboard or from the navigation.

Edit Space

From the Space page, click the **Edit Space** link.



My-Org > development > Edit Space

SPACE SETTINGS

Current Org My-Org

Space Name development

Cancel Save

- **Delete a Space:** Click the **Delete Space** button in the top right corner.
- **Rename a Space:** Change the text in the **Space Name** field, then click **Save**.

Apps List

The Apps List displays information about the **Status**, **Name**, **Route**, number of **Instances**, and amount of **Memory** for each app in the current space.

APPLICATIONS				LEARN MORE
STATUS	APP	INSTANCES	MEMORY	
100%	hello-world hello-world.ch...	1	128MB	›
100%	spring-music spring-music.cherr...	1	512MB	›

Services View

The Services view lists services bound to apps in the current space.

SERVICES			ADD SERVICE
SERVICE INSTANCE	SERVICE PLAN	BOUND APPS	
mysql-dev Manage Documentation Support Delete	Pivotal MySQL Dev 100mb	3	

- **Add a Service:** Click the **Add Service** button on the right side of the Services view. Clicking the **Add Service** button takes you to the Marketplace.

Managing User Accounts in Spaces Using the Apps Manager

Page last updated:

 **Note:** The procedures described here are not compatible with using LDAP for user identity management. To create and manage user accounts in an LDAP-enabled Cloud Foundry deployment, see [Adding Existing LDAP Users to a Pivotal Cloud Foundry Deployment](#)

To manage user accounts in a space, you must have Space Manager permissions in that space. Org Managers have Space Manager permissions by default and can manage user accounts in all spaces associated with the org.

Space Roles and Permissions

The different user roles are described on the right side of the Teams view.

Space Manager

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 Developer

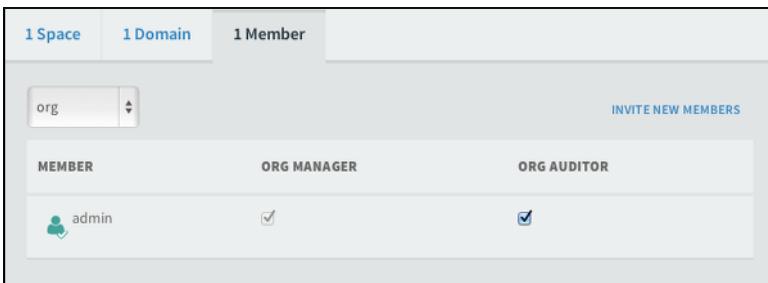
Space Developers can create, delete, and manage applications and services, and have full access to all usage reports and logs. Space Developers can also edit applications, including the number of instances and memory footprint. Assign this role to app developers or other users who need to interact with applications and services.

Space Auditor

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.



The screenshot shows the 'Members' tab of the Org dashboard. At the top, there are three tabs: '1 Space', '1 Domain', and '1 Member'. Below the tabs, there is a dropdown menu set to 'org'. On the right, there is a blue 'INVITE NEW MEMBERS' button. The main area displays a table with three columns: 'MEMBER', 'ORG MANAGER', and 'ORG AUDITOR'. A single member, 'admin', is listed. The 'ORG AUDITOR' column for 'admin' has a checked checkbox.

2. Click **Invite New Members**. The **Invite New Team Member(s)** form appears.

3. In the **Add Email Address(es)** 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 Team Roles** view lists 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 particular 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 User Permissions Using the Apps Manager

Page last updated:

The Apps Manager uses role-based access control, with each role granting permissions in either an organization or an application space. 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.

An administrator can manage user roles in orgs and spaces using the Apps Manager.

To see which permissions each role grants, see [Organizations, Spaces, Roles, and Permissions](#).

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.

MEMBER	ORG MANAGER	ORG AUDITOR
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Bob@example.com	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ClariceP@example.com	<input type="checkbox"/>	<input type="checkbox"/>
EddieG@example.com	<input checked="" type="checkbox"/>	<input type="checkbox"/>
David.Davidson@example.com	<input type="checkbox"/>	<input type="checkbox"/>
Alice@example.com	<input checked="" type="checkbox"/>	<input type="checkbox"/>

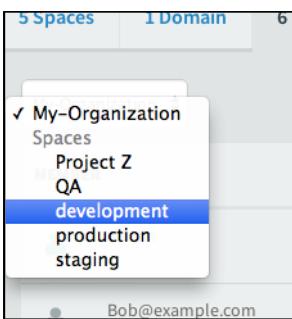
4. The **Members** panel displays all members of the org. Select a checkbox to grant an org role to a user, or deselect a checkbox to revoke a role from a user.

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.

MEMBER	SPACE MANAGER	SPACE DEVELOPER	SPACE AUDITOR
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Bob@example.com	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ClariceP@example.com	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
EddieG@example.com	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
David.Davidson@example.com	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Alice@example.com	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

APP SPACE ROLES

SPACE MANAGER
Can invite/manage users, enable features for a given space

SPACE DEVELOPER
Can create, delete, manage applications and services, full access to all usage reports and logs

SPACE AUDITOR
Read-only access to all space information, settings, reports, logs

cf Command Line Interface (CLI)

Use the cf Command Line Interface (cf CLI) to deploy and manage your applications on Cloud Foundry.

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 Command Line Interface

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](#):

```
$ brew tap cloudfoundry/tap
```

2. Install the cf CLI:

```
$ 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:

```
$ wget -q -O - https://packages.cloudfoundry.org/debian/cli.cloudfoundry.org.key | sudo apt-key add -
```

```
$ 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:

```
$ sudo apt-get update
```

3. Install the cf CLI:

```
$ 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:

```
$ 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:

```
$ 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 `cf CLI` executable.
3. When prompted, click **Install**, then **Close**.
4. To verify your installation, open a terminal window and type `cf`. If your installation was successful, the cf CLI help listing appears.

Mac OS X and Linux Installation

To use the cf CLI installer for Mac OS X, perform the following steps:

1. Open the `.pkg` file.
2. In the installer wizard, click **Continue**.
3. Select an install destination and click **Continue**.
4. When prompted, click **Install**.
5. To verify your installation, open a terminal window and type `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 `sudo`.
 - For Debian/Ubuntu, run the following command:

```
$ dpkg -i path/to/cf-cli-* deb && apt-get install -f
```

- For Red Hat, run the following command:

```
rpm -i path/to/cf-cli-* rpm
```

2. To verify your installation, open a terminal window and type `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](#) 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:

```
$ curl -L "https://cli.run.pivotal.io/stable?release=macosx64-binary&source=github" | tar -zx
```

2. Move it to `/usr/local/bin`, or another location in your `$PATH`:

```
$ mv cf /usr/local/bin
```

3. Confirm your cf CLI version:

```
$ cf --version
```

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](#) 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

Page last updated:

This topic covers how to get started with and configure the cf CLI. This page assumes you have the latest version of the cf CLI. See the [Installing the cf Command Line Interface](#) topic for installation instructions.

Localization

The cf CLI translates terminal output into the language that you select. The default language is `en_US`. Run `cf config --locale YOUR_LANGUAGE` to set the language. For example:

```
cf config --locale pt_BR
```

The cf CLI currently 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`

 **Note:** Localization affects only messages that the cf CLI generates.

Login

The `cf login` command allows you to specify a target API endpoint, [organization](#), and [space](#):

```
$ cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]
```

- **API endpoint:** This is [the URL of the Cloud Controller in your Elastic Runtime instance](#).
- **Username:** Your username.
- **Password:** Your password.
- **Org:** The organization where you want to deploy your application.
- **Space:** The space in the organization where you want to deploy your application.

If you have only one organization and one space, you can omit them because `cf login` targets them automatically.

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, organization, 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

Note: You cannot perform certain tasks in the CLI 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

Commands for listing users:

- `cf org-users` — List users in the organization by role.
- `cf space-users` — List users in the space by role.

Commands for managing roles (admin-only):

- `cf set-org-role` — Assign an organization role to a user. The available roles are “OrgManager”, “BillingManager”, and “OrgAuditor”.
- `cf unset-org-role` — Remove an organization role from a user.
- `cf set-space-role` — Assign a space role to a user. The available roles are “SpaceManager”, “SpaceDeveloper”, and “SpaceAuditor”.
- `cf unset-space-role` — Remove a space role from a user.

Domains, Routes, Organizations, and Spaces

cf CLI treats domains, routes, organizations, and spaces such that:

- All domains are mapped to an org.
- Routes are scoped to spaces and apps.

A route is a URL of the form `HOSTNAME.DOMAIN`. If you do not provide a hostname (also known as subdomain), the URL takes the form `APPNAME.DOMAIN`.

The cf CLI separates between management of private domains and management of shared domains. Only administrators can manage shared domains.

In terms of domains, cf CLI is designed to work with both new and older versions of `cf_release`.

Commands for managing domains:

- `cf create-domain` — Create a domain.
- `cf delete-domain` — Delete a domain.
- `cf create-shared-domain` — Share a domain with all organizations. Admin only.
- `cf delete-shared-domain` — Delete a domain that was shared with all organizations. Admin only.

Commands for managing routes:

- `cf create-route` — Create a route.
- `cf map-route` — Map a route to an application. If the route does not exist, this command creates it and then maps it.
- `cf unmap-route` — Remove a route from an application.
- `cf delete-route` — Delete a route.

Mapping a Route

1. Create a domain in the desired organization using `cf create-domain`, unless the domain already exists in (or has been shared with) the organization.

2. Map the domain to an application using `cf map-route`. Specify a unique hostname for each route that uses the same domain with `-n HOSTNAME`.

Note: You can map a single route to multiple applications in the same space. See [Blue-Green Deployment](#) to learn about an important extension of this technique.

Push

The `cf push` command allows you to push an app or sync changes to an existing app:

```
$ cf push APP [-b URL] [-c COMMAND] [-d DOMAIN] [-i NUM_INSTANCES] [-m MEMORY] /  
[-n HOST] [-p PATH] [-s STACK] [-no-hostname] [-no-route] [-no-start]
```

- `APP` — The name of the application to push is the only required argument, and the only argument that has no flag. Even `APP` can be omitted when you provide the application name in a manifest.
- `-b` — Custom buildpack by name (e.g. `my-buildpack`), Git URL (For example, <https://github.com/cloudfoundry/java-buildpack.git>), or Git URL with a branch or tag: <https://github.com/cloudfoundry/java-buildpack.git#v3.3.0> to select `v3.3.0`.
- `-m` — Memory limit requires a unit of measurement: `M`, `MB`, `G`, or `GB`, in upper case or lower case.
- `-t` — Timeout allows you to give your application more time to start, up to 180 seconds.
- `--no-manifest` — Forces the cf CLI to ignore any existing manifest.
- `--no-hostname` — Makes it possible to specify a route with a domain but no hostname.
- `--no-route` — Removes existing and suppresses new external routes to your application. Use this flag when pushing worker apps.

Other arguments include:

- `-c` — Start command for the application.
- `-d` — Domain, for example, example.com.
- `-f` — Path to manifest.
- `-i` — Number of instances of the application to run.
- `-m` — Memory limit, for example, 256, 1G, 1024M, and so on.
- `-n` — Hostname, for example, `my-subdomain`.
- `-p` — Path to application directory or archive.
- `-s` — Stack to use.
- `-t` — Timeout to start in seconds.
- `--no-hostname` — Map the root domain to this application.
- `--no-manifest` — Ignore manifests if they exist.
- `--no-route` — Do not map a route to this application. Also removes existing routes from previous pushes of this app.
- `--no-start` — Do not start the application after pushing.

 **Note:** To push a sample app, see the [Buildpacks](#) topic.

User-Provided Service Instances

The cf CLI has three ways of creating and updating user-provided service instances: interactively, non-interactively, and in conjunction with third-party log management software as described in [RFC 6587](#). When used with third-party logging, the cf CLI sends data formatted according to [RFC 5424](#).

Once created, user-provided service instances can be bound to an application with `cf bind-service`, unbound with `cf unbind-service`, renamed with `cf rename-service`, and deleted with `cf delete-service`.

The cf create-user-provided-service Command

The alias for `create-user-provided-service` is `cups`.

To create a service instance in interactive mode, use the `-p` option with a comma-separated list of parameter names. The cf CLI then prompts you for each parameter in turn.

```
cf cups SERVICE_INSTANCE -p host, port, dbname, username, password
```

To create a service instance non-interactively, use the `-p` option with a JSON hash of parameter keys and values.

```
cf cups SERVICE_INSTANCE -p '{"username":"admin", "password":"pa55woRD"}'
```

To create a service instance that drains information to third-party log management software, use the `-l SYSLOG_DRAIN_URL` option.

```
cf cups SERVICE_INSTANCE -l syslog://example.com:514
```

The cf update-user-provided-service Command

The alias for `update-user-provided-service` is `uups`. Use `cf update-user-provided-service` to update one or more of the attributes for a user-provided service instance. Attributes that you do not supply are not updated.

To update a service instance non-interactively, use the `-p` option with a JSON hash of parameter keys and values.

```
cf uups SERVICE_INSTANCE -p '{"username":"USERNAME", "password":"PASSWORD"}'
```

To update a service instance that drains information to third-party log management software, use the `-l SYSLOG_DRAIN_URL` option.

```
cf uups SERVICE_INSTANCE -l syslog://example.com:514
```

cf CLI Return Codes

The cf CLI uses exit codes, which is helpful for scripting and for confirming that a command has run successfully. For example, run `echo $?` in the terminal.

If the return code is `0`, the command was successful.

Run cf help

Run `cf help` to view a list of all cf CLI commands and a brief description of each. Run `cf COMMAND -h` to view detailed help, including any alias, for any command. For example, the output below shows all the commands, and the alias, related to `push`:

```
$ cf -h
NAME:
push - Push a new app or sync changes to an existing app

ALIAS:
p

USAGE:
Push a single app (with or without a manifest):
cf push APP [-b BUILDPACK_NAME] [-c COMMAND] [-d DOMAIN] [-f MANIFEST_PATH]
[-i NUM_INSTANCES] [-m MEMORY] [-n HOST] [-p PATH] [-s STACK] [-t TIMEOUT]
[--no-hostname] [--no-manifest] [--no-route] [--no-start]

Push multiple apps with a manifest:
cf push [-f MANIFEST_PATH]

OPTIONS:
-b      Custom buildpack by name (e.g. my-buildpack) or GIT URL
(e.g. https://github.com/cloudfoundry/java-buildpack.git)
-c      Startup command, set to null to reset to default start command
-d      Domain (e.g. example.com)
-f      Path to manifest
-i      Number of instances
-m      Memory limit (e.g. 256M, 1024M, 1G)
-n      Hostname (e.g. my-subdomain)
-p      Path of app directory or zip file
-s      Stack to use
-t      Start timeout in seconds
--no-hostname  Map the root domain to this app
--no-manifest  Ignore manifest file
--no-route     Do not map a route to this app
--no-start     Do not start an app after pushing
```


Using the cf CLI with an HTTP Proxy Server

Page last updated:

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@proxy.example.com:8080`

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:

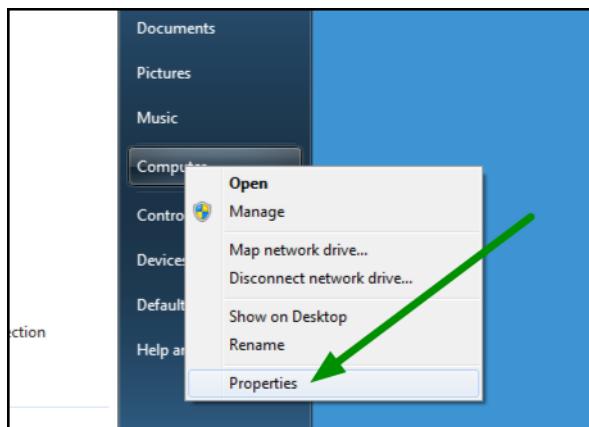
```
$ export https_proxy=http://my.proxyserver.com:8080
```

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:

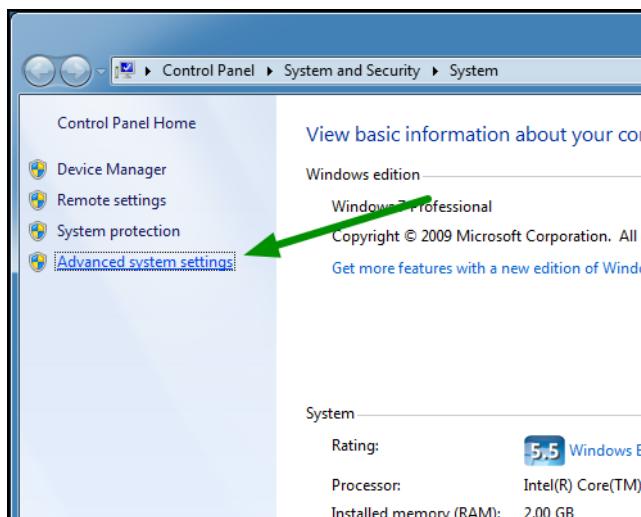
```
https_proxy=http://username:password@hostname:port
export $https_proxy
```

Setting `https_proxy` in Windows

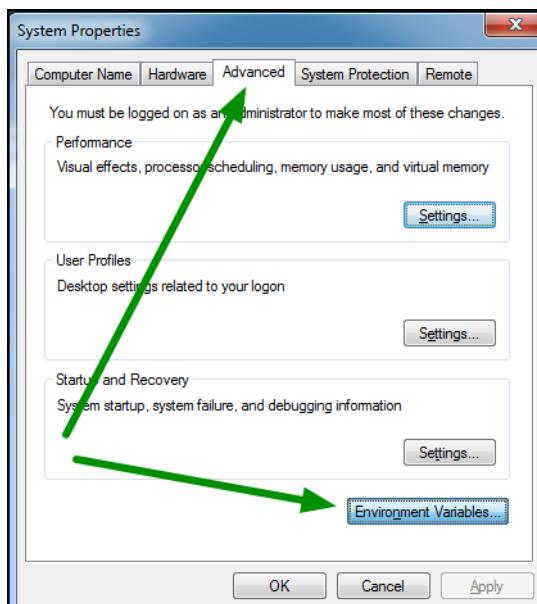
1. Open the Start menu. Right-click **Computer** and select **Properties**.



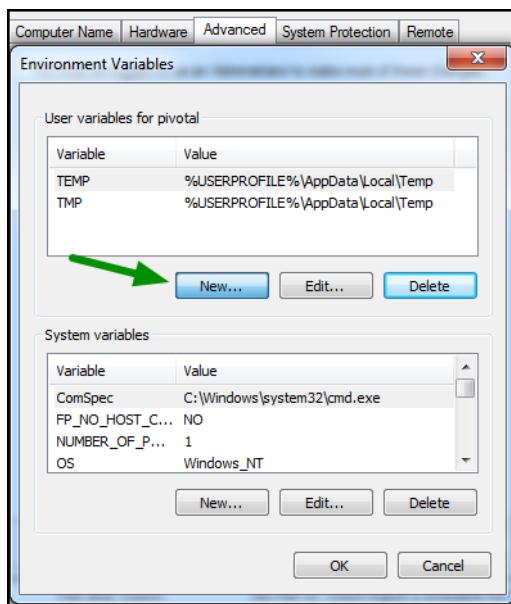
2. In the left pane of the System window, click **Advanced system settings**.



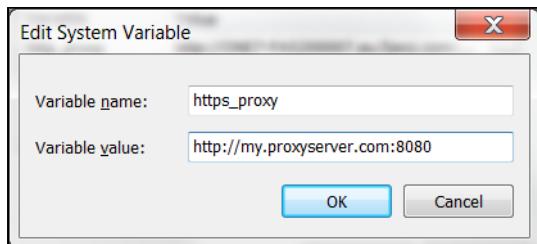
3. In the System Properties window, select the **Advanced** tab, then click **Environment Variables**.



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.



6. Click **OK**.

Using the cf CLI with a Self-Signed Certificate

Page last updated:

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

Page last updated:

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.

A **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 Command Line Interface](#) 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 http://plugins.cloudfoundry.org
OK
http://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  http://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

Page last updated:

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 cf 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

Page last updated:

This page assumes you are using cf CLI v6.

cf push starts your application with a command from one of three sources:

1. The `-c` command-line option, for example:

```
$ cf push my-app -c "node my-app.js"
```

2. The `command` attribute in the application manifest, for example:

```
command: node my-app.js
```

3. The buildpack, which provides a start command appropriate for a particular type of application.

The source that the cf CLI uses depends on factors explained below.

How cf push Determines its Default Start Command

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

To force the cf CLI to use the buildpack start command, specify a start command of `null`.

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 will need 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](#)
 - [Routes and Domains ↗](#)
 - [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](#)
-

Services

- [Services Overview](#)
- [Managing Service Instances](#)
- [Managing Service Keys](#)
- [Delivering Service Credentials to an Application](#)
- [User-Provided Service Instances](#)
- [Using Log Management Services](#)
- [Service-Specific Instructions for Streaming Application Logs](#)
- [Streaming Application Logs to Splunk](#)
- [Configuring Play Framework Service Connections](#)
- [Migrating a Database in Cloud Foundry](#)

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](#).

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

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.com` might be able to access the cookies for an application at `your-app.shared-domain.com`.

Consider whether you want your applications or tools that use cookies to set and store the cookies at the highest available domain.

HTTP Sessions Not Persisted or Replicated

Cloud Foundry supports session affinity or sticky sessions for incoming HTTP requests to applications if a `jsessionid` cookie is used. If multiple instances of an application are running on Cloud Foundry, all requests from a given client will be routed to the same application instance. This allows application containers and frameworks to store session data specific to each user session.

Cloud Foundry does not persist or replicate HTTP session data. If an instance of an application crashes or is stopped, any data stored for HTTP sessions that were sticky to that instance are lost. When a user session that was sticky to a crashed or stopped instance makes another HTTP request, the request is routed to another instance of the application.

Session data that must be available after an application crashes or stops, or that needs to be shared by all instances of an application, should be stored in a Cloud Foundry service. There are several open source projects that share sessions in a data service.

HTTP Headers

HTTP traffic passed from the Cloud Foundry router to an app includes the 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.

Port Limitations

Applications running on Elastic Runtime receive requests through the URLs configured for the application. HTTP requests arrive on ports 80 and 443. Additionally, Elastic Runtime requires a channel for TCP/WebSocket traffic. By default, this is port 443.

Some production load balancers require separate ports for HTTP and TCP/WebSocket traffic. If your load balancer has this requirement, you must do the following in order to access logs from your app with the `cf logs APP_NAME` command:

- Configure a different port for TCP/WebSocket traffic. As of PCF 1.4, you can configure your ports from the Elastic Runtime product tile in Ops Manager.
- Set the value of the `port` sub-key of the `logger_endpoint` key in your manifest.
- Configure your HAProxy or load balancer to receive TCP traffic on the port that you specified.

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:
 - o [Ruby](#)
 - o [Node.js](#)
 - o [Java](#)
 - o [Build Tool Integration](#)
 - o [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](#)
- [Trusted System Certificates](#)

Deploy an Application

Page last updated:

This page assumes that you are using cf CLI v6.

 **Note:** See the [buildpacks](#) documentation for complete deployment guides specific to your application language or framework, such as the [Getting Started Deploying Ruby on Rails Apps](#) guide.

Overview of Deployment Process

You deploy an application to Cloud Foundry by running a `push` command from a Cloud Foundry command line interface (CLI). Refer to the [Install cf CLI v6](#) topic for more information. Between the time that you run `push` and the time that the application is available, Cloud Foundry performs the following tasks:

- Uploads and stores application files
- Examines and stores application metadata
- Creates a “droplet” (the Cloud Foundry unit of execution) for the application
- Selects an appropriate droplet execution agent (DEA) to run the droplet
- Starts the application

An application 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 application. For more information about services, see the [Services Overview](#) topic.

Step 1: Prepare to Deploy

Before you deploy your application to Cloud Foundry, make sure that:

- Your application is *cloud-ready*. Cloud Foundry behaviors related to file storage, HTTP sessions, and port usage may require modifications to your application.
- All required application 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 application directory structure, particularly if your application is large.
- An instance of every service that your application needs has been created.
- Your Cloud Foundry instance supports the type of application you are going to deploy, or you have the URL of an externally available buildpack that can stage the application.

For help preparing to deploy your application, see:

- [Considerations for Designing and Running an Application in the Cloud](#)
- [Buildpacks](#)

Step 2: Know Your Credentials and Target

Before you can push your application 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 application. 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 application using a route, which is a URL made up of a host and a domain.

- The name of an application is the default host for that application, unless you specify the host name with the `-n` flag.
- Every application is deployed to an application 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 application 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 CLI](#) 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 application.
- **Instances:** Generally speaking, the more instances you run, the less downtime your application will experience. If your application is still in development, running a single instance can simplify troubleshooting. For any production application, we recommend a minimum of two instances.
- **Memory Limit:** The maximum amount of memory that each instance of your application 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 application. This start command varies by application 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:** Applications can bind to services such as databases, messaging, and key-value stores. Applications are deployed into application spaces. An application can only bind to a service that has an existing instance in the target application 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 application settings change from push to push, and how command-line options, manifests, and commands like `cf scale` interact.

When you deploy an application while it is running, Cloud Foundry stops all instances of that application and then deploys. Users who try to run the application 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 application 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.

Set Environment Variables

Environment variables are key-value pairs defined at the operating system level. These key-value pairs provide a way to configure the applications running on a system. For example, any application 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 a bash script to the `.profile.d` directory in the source code of your application. Cloud Foundry runs this script when deploying each instance of your application.

 **Note:** You must give your shell scripts a `.sh` extension.

Example `.profile.d/setenv.sh` script:

```
# Set the Java environment path for your applications
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0
export PATH=$PATH:$JAVA_HOME/bin

# Set the default LANG for your applications
export LANG=en_US.UTF-8
```

Step 5: Push the Application

Run the following command to deploy an application without a manifest:

```
cf push APP-NAME
```

If you provide the application name in a manifest, you can reduce the command to `cf push`. See [Deploying with Application Manifests](#).

Since all you have provided is the name of your application, `cf push` sets the number of instances, amount of memory, and other attributes of your application 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 application 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.

```
$ cf push my-app
Creating app my-app in org example-org / space development as a.user@shared-domain.com...
OK

Creating route my-app.shared-domain.com...
OK

Binding my-app.shared-domain.com to my-app...
OK

Uploading my-app...
Uploading app: 560.1K, 9 files
OK

Starting app my-app in org example-org / space development as a.user@shared-domain.com...
----> Downloaded app package (552K)
OK
----> Using Ruby version: ruby-1.9.3
----> Installing dependencies using Bundler version 1.3.2
      Running: bundle install --without development:test --path vendor/bundle --binstubs vendor/bundle/bin --deployment
      Installing rack (1.5.1)
      Installing rack-protection (1.3.2)
      Installing tilt (1.3.3)
      Installing sinatra (1.3.4)
      Using bundler (1.3.2)
      Updating files in vendor/cache
      Your bundle is complete! It was installed into ./vendor/bundle
      Cleaning up the bundler cache.
----> Uploading droplet (23M)

1 of 1 instances running

App started

Showing health and status for app my-app in org example-org / space development as a.user@shared-domain.com...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: my-app.shared-domain.com

      state  since          cpu   memory   disk
#0  running  2014-01-24 05:07:18 PM  0.0%  18.5M of 1G  52.5M of 1G
```

Step 6: (Optional) Configure Service Connections

If you bound a service to the application that you deployed, you might need to configure your application with the service URL and credentials. For more information, see the specific documentation for your application framework:

- [Ruby](#)
- [Node.js](#)
- [Spring](#)
- [Grails](#)

Step 7: Troubleshoot Deployment Problems

If your application does not start on Cloud Foundry, first ensure that your application can run locally.

You can troubleshoot your application 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. Your app must successfully complete each of these phases within the time limits that your administrator establishes for each phase. 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.

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 recommend a minimum speed of 874 KB/s.

 **Note:** Elastic Runtime provides an authorization token that is valid for a minimum of 20 minutes.

- 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.

 **Note:** 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.

Setting	Note
App Package Size	Maximum: 1 GB
Authorization Token Grace Period	Default: 20 minutes, minimum
	cf CLI environment variable

<code>CF_STAGING_TIMEOUT</code>	Default: 15 minutes
<code>CF_STARTUP_TIMEOUT</code>	cf CLI environment variable Default: 5 minutes
<code>cf push -t TIMEOUT</code>	App start timeout maximum Default: 60 seconds
Disk Space Allocation	Default: 1024 MB
Internet Connection Speed	Recommended Minimum: 874 KB/s

Deploying with Application Manifests

Page last updated:

This page assumes that you are using cf CLI v6.

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
```

Note: When you provide an application name at the command line, `cf push` uses that application name whether or not there is a different application name in the manifest. If the manifest describes multiple applications, you can push a single application by providing its name at the command line; the cf CLI does not push the others. Use these behaviors for testing.

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 its URL or name:

```
---  
...  
buildpack: buildpack_URL
```

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 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.

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
```

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"
```

Note: The `-c` option with a value of 'null' forces `cf push` to use the buildpack start command. See [About Starting Applications](#) for more information.

If you override the start command for a Buildpack application, Linux uses `bash -c YOUR-COMMAND` to invoke your application. If you override the start command for a Docker application, Linux uses `sh -c YOUR-COMMAND` to invoke your application. Because of this, if you override a start command, you should prefix `exec` to the final command in your custom composite start command.

An app needs to catch [termination signals](#) and clean itself up appropriately. Because of the way that shells manage process trees, the use of custom composite shell commands, particularly those that create child processes using `&`, `&&`, `||`, etc., can prevent your app from receiving signals that are sent to the top level bash process.

To resolve this issue, you can use `exec` to replace the bash process with your own process. For example:

- `bin/rake cf:on_instance db:migrate && bin/rails server -p $PORT -e $RAILS_ENV` The process tree is bash -> ruby, so on graceful shutdown only the bash process receives the TERM signal, not the ruby process.
- `bin/rake cf:on_instance db:migrate && exec bin/rails server -p $PORT -e $RAILS_ENV` Because of the `exec` prefix included on the final command, the ruby process invoked by rails takes over the bash process managing the execution of the composite command. The process tree is only ruby, so the ruby web server receives the TERM signal and can shutdown gracefully for 10 seconds.

In more complex situations, like making a custom buildpack, you may want to use bash `trap`, `wait`, and backgrounded processes to manage your process tree and shut down apps gracefully. In most situations, however, a well-placed `exec` should be sufficient.

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.

```
---
...
disk_quota: 1024M
```

The command line option that overrides this attribute is `-k`.

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`, in upper case or lower case. For example:

```
---  
...  
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 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`.

```
---  
...  
host: my-app
```

The command line option that overrides this attribute is `-n`.

The hosts attribute

Use the `hosts` 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`.

```
---  
...  
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.

```
---  
...  
no-hostname: true
```

The command line option that corresponds to this attribute is `--no-hostname`.

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`.

```
---  
...  
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.

```
---  
...  
path: path_to_application_bits
```

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:

```
---  
...  
timeout: 80
```

You can increase the timeout length for very large apps that require more time to start. The default timeout is 60 seconds, with an upper bound of 180 seconds.

 **Note:** Administrators can set the upper bound of the `maximum_health_check_timeout` property to any value. Any changes to Cloud Controller properties in the deployment manifest require running `bosh deploy`.

The command line option that overrides the timeout attribute is `-t`.

The no-route attribute

By default, `cf push` assigns a route to every application. But some applications 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 application.

```
---  
...  
no-route: true
```

The command line option that corresponds to this attribute is `--no-route`.

If you find that an application which should not have a route does have one:

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 about these situations, 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:

```
---  
...  
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, Cloud Foundry allows you to operate on environment variables.

- View all variables: `cf env my-app`
- Set an individual variable: `cf set-env my-variable_name my-variable_value`
- Unset an individual variable: `cf unset-env 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.

```
---  
...  
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`:

```
---  
...  
path: ./spark/
```

For `flame`:

```
---  
...  
path: ./flame/
```

The manifest now consists of two applications blocks.

```
---  
# 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.com  
  path: ./spark/  
  services:  
    - mysql-flint-99  
- name: flame  
  memory: 1G  
  instances: 2  
  host: burnin-77  
  domain: shared-domain.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.

```
---  
...  
# all applications use these settings and services  
domain: shared-domain.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:

```
---  
...  
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

```
---  
path: .  
domain: shared-domain.com  
memory: 256M  
instances: 1  
services:  
- singular-backend  
  
# app-specific configuration  
applications:  
- name: springstock  
host: 765shower  
path: ./april/build/libs/april-weather.war  
- name: wintertick  
host: 32flurry  
path: ./december/target/december-weather.war
```

simple-prod-manifest.yml

```
---  
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: 32lblizzard  
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

Page last updated:

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:

This page assumes you are using cf CLI v6.

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

```
$ 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": "fa05c1a9-0fc1-4fb1-ba1-139850dec7a3",
    "application_name": "my-app",
    "application_uris": [
      "my-app.10.244.0.34.xip.io"
    ],
    "application_version": "fb8fbcc6-8d58-479e-bcc7-3b4ce5a7f0ca",
    "limits": {
      "disk": 1024,
      "fds": 16384,
      "mem": 256
    },
    "name": "my-app",
    "space_id": "06450c72-4669-4dc6-8096-45f9777db68a",
    "space_name": "my-space",
    "uris": [
      "my-app.10.244.0.34.xip.io"
    ],
    "users": null,
    "version": "fb8fbcc6-8d58-479e-bcc7-3b4ce5a7f0ca"
  }
}
```

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.

```
PWD=/home/vcap/app
```

TMPDIR

Directory location where temporary and staging files are stored.

```
TMPDIR=/home/vcap/tmp
```

USER

The user account under which the application runs.

```
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.

```
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.

Attribute	Description
<code>application_id</code>	GUID identifying the application.
<code>application_name</code>	The name assigned to the application when it was pushed.
<code>application_uris</code>	The URIs assigned to the application.
<code>application_version</code>	GUID identifying a version of the application. Each time an application is pushed or restarted, this value is updated.
<code>host</code>	Deprecated. IP address of the application instance.
<code>instance_id</code>	Unique ID that identifies the application instance. For instances running on Diego, this is identical to the <code>CF_INSTANCE_GUID</code> variable.
<code>instance_index</code>	Index number of the instance. Identical to the <code>CF_INSTANCE_INDEX</code> variable.
<code>limits</code>	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.
<code>name</code>	Identical to <code>application_name</code> .
<code>port</code>	Port of the application instance. Identical to the <code>PORT</code> variable.
<code>space_id</code>	GUID identifying the application's space.
<code>start</code>	Human-readable timestamp for the time the instance was started. Not provided on Diego Cells.
<code>started_at</code>	Identical to <code>start</code> . Not provided on Diego Cells.
<code>started_at_timestamp</code>	Unix epoch timestamp for the time the instance was started. Not provided on Diego Cells.
<code>state_timestamp</code>	Identical to <code>started_at_timestamp</code> . Not provided on Diego Cells.

<code>uris</code>	Identical to <code>application_uris</code> .
<code>users</code>	Deprecated. Not provided on Diego Cells.
<code>version</code>	Identical to <code>application_version</code> .

For example:

```
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,"start":2013-08-12 00:05:29
+0000,"state_timestamp":1376265929,"limits":{"mem":512,"disk":1024,"fds":16384}
,"application_version":"ab12cd34-5678-abcd-0123-abcdef987654","application_name"
:"styx-james","application_uris":["styx-james.al.app.cf-app.com"],"version":ab1
2cd34-5678-abcd-0123-abcdef987654,"name":my-app,"uris":["my-app.example.com"]
,"users":null}
```

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 of that service 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.

Attribute	Description
<code>name</code>	The name assigned to the service instance by the user
<code>label</code>	The name of the service offering
<code>tags</code>	An array of strings an app can use to identify a service instance
<code>plan</code>	The service plan selected when the service instance was created
<code>credentials</code>	A JSON object containing the service-specific credentials needed to access the service instance.

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](#).

```
VCAP_SERVICES=
{
  "elephantsql": [
    {
      "name": "elephantsql-c6c60",
      "label": "elephantsql",
      "tags": [
        "postgres",
        "postgresql",
        "relational"
      ],
      "plan": "turtle",
      "credentials": {
        "uri": "postgres://seilbmbd:ABcdEF@babar.elephantsql.com:5432/seilbmbd"
      }
    }
  ],
  "sendgrid": [
    {
      "name": "mysendgrid",
      "label": "sendgrid",
      "tags": [
        "smtp"
      ],
      "plan": "free",
      "credentials": {
        "hostname": "smtp.sendgrid.net",
        "username": "QvsXMbJ3rK",
        "password": "HCHMOYluTv"
      }
    }
  ]
}
```

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.

CLI Command	Description
<code>running-environment-variable-group</code> or <code>revg</code>	Retrieves the contents of the running environment variable group
<code>staging-environment-variable-group</code> or <code>sevg</code>	Retrieves the contents of the staging environment variable group
<code>set-staging-environment-variable-group</code> or <code>ssevg</code>	Passes parameters as JSON to create a staging environment variable group
<code>set-running-environment-variable-group</code> or <code>srevg</code>	Passes parameters as JSON to create a running environment variable group

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 87.226.68.130

$ cf sevg
Retrieving the contents of the staging environment variable group as
sampledeveloper@example.com...
OK
Variable Name Assigned Value
HTTP Proxy 27.145.145.105
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,
      "fds": 16384,
      "mem": 256
    },
    "name": "APP-NAME",
    "space_id": "37189599-2407-9946-865e-8ebd0e2df89a",
    "space_name": "dev",
    "uris": [
      "my-app.example.com"
    ],
    "users": null,
    "version": "7d0d64be-7f6f-406a-9d21-504643147d63"
  }
}

Running Environment Variable Groups:
HTTP Proxy: 87.226.68.130

Staging Environment Variable Groups:
EXAMPLE-GROUP: 2001
HTTP Proxy: 27.145.145.105
```

The following examples demonstrate how to set environment variables:

```
$ cf ssevg '{"test":"87.226.68.130","test2":"27.145.145.105"}'
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 87.226.68.130
test2 27.145.145.105

$ cf srevg '{"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

Page last updated:

This page assumes you are using cf CLI v6.

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.

Note: You can adjust the route mapping pattern to display a static maintenance page during a maintenance window for time-consuming tasks, such as migrating a database. In this scenario, the router switches all incoming requests from Blue to Maintenance to Green.

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.

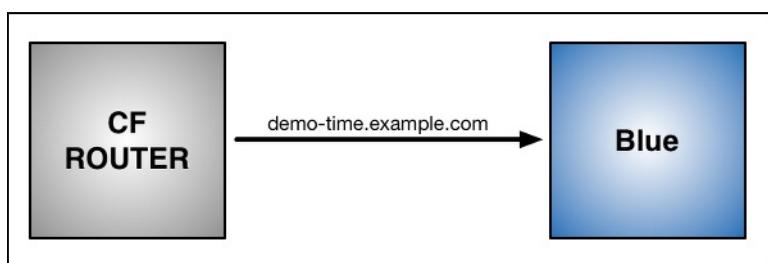
Step 1: Push an App

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.



Step 2: Update App and Push

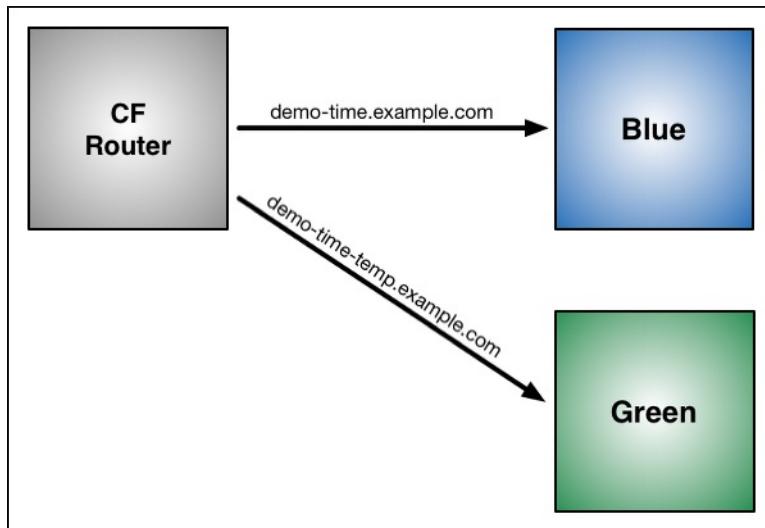
Now make a change to the application. First, replace the word "Blue" on the web page with "Green," then rebuild the source file for the application. Run `cf push` again, but use the name "Green" for the application and provide a different subdomain to create a temporary route:

```
$ 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.



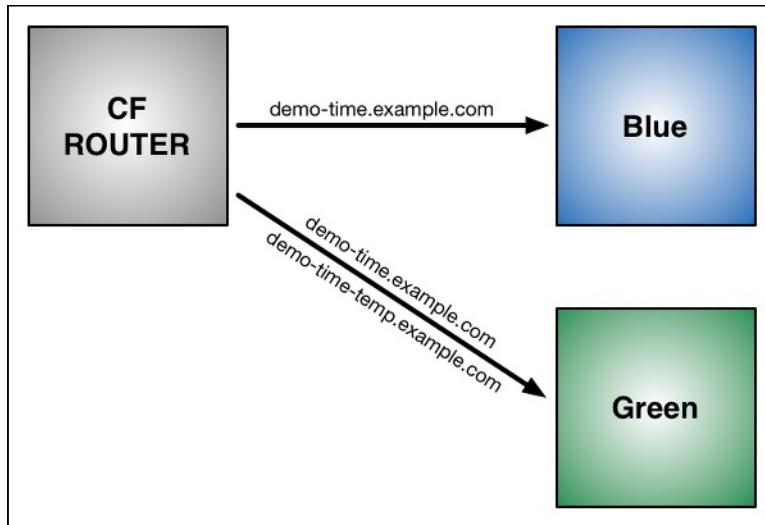
Step 3: Map Original Route to Green

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.

```
$ 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.



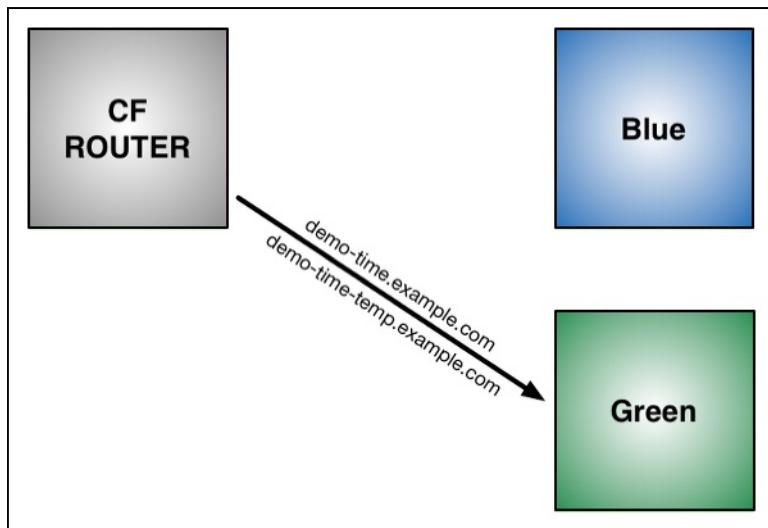
Step 4: Unmap Route to Blue

Once you verify Green is running as expected, stop routing requests to Blue using the `cf unmap-route` command:

```
$ cf unmap-route Blue example.com -n demo-time
Unbinding demo-time.example.com from blue... OK
```

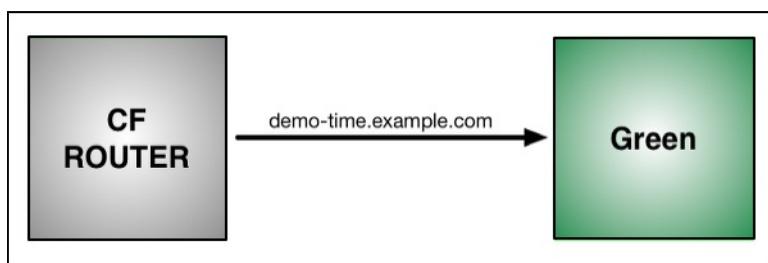
After `cf unmap-route` command:

- The CF Router stops sending traffic to Blue. Instead, it routes all traffic to `demo-time.example.com` to Green:



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.



Application Logging in Cloud Foundry

Page last updated:

This page assumes you are using cf CLI v6.

Loggregator, the Cloud Foundry component responsible for logging, provides a stream of log output from your application 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 application 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. Applications 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 application origin code is `APP` followed by slash and a digit that indicates the application instance.

Many frameworks write to an application log that is separate from `STDOUT` and `STDERR`. This is not supported by Loggregator. Your application must write to `STDOUT` or `STDERR` for its logs to be included in the Loggregator stream. Check the buildpack your application uses to determine whether it automatically ensures that your application 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.

API

Users make API calls to request changes in application state. Cloud Controller, the Cloud Foundry component responsible for the API, logs the actions that Cloud Controller takes in response.

For example:

```
2014-02-13T11:44:52.11-0800 [API]  OUT Updated app with guid e1ca6390-cf78-4fc7-9d86-5b7ed01e9c28 ({"instances":>2})
```

STG

The Droplet Execution Agent emits STG logs when staging or restaging an app. These actions implement the desired state requested by the user. Once the droplet has been uploaded, STG messages end and DEA messages begin.

For example:

```
2014-02-07T10:54:36.80-0800 [STG]  OUT ----> Downloading and installing node
```

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 e1ca6390-cf78-4fc7-9d86-5b7ed01e9c28
```

RTR

The Router emits RTR logs when it routes HTTP requests to the application. Router messages include the application name followed by a Router timestamp and then selections from the HTTP request.

For example:

```
2014-02-13T11:42:31.96-0800 [RTR]  OUT nifty-gui.shared-domain.com - [13/02/2014:19:42:31 +0000]
"GET /favicon.ico HTTP/1.1" 404 23 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36" 10.10.2.142:6609 response_time:0.004092262
app_id:e1ca6390-cf78-4fc7-9d86-5b7ed01e9c28
```

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 application emits logs according to choices by the developer. The digit appended to the code indicates app instance: 0 is the first instance, 1 is the second, and so on.

For example:

```
2014-02-13T11:44:27.71-0800 [App/0]  OUT Express server started
```

Writing to the Log from your Application

Your application 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 application 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 nifty-gui
Connected, tailing logs for app nifty-gui in org janclouduser / space jancloudspace as admin...
2014-02-06T12:00:19.44-0800 [API]
    OUT Updated app with guid c8612fc2-85b1-464c-92f5-d4a1156eacb
    {"route":>"2cef5796b-475a-4615-9c71-75bbe277022e"}
2014-02-06T12:00:27.54-0800 [DEA]
    OUT Got staging request for app with id
    c8612fc2-85b1-464c-92f5-d4a1156eacb
2014-02-06T12:00:28.51-0800 [API]
    OUT Updated app with guid c8612fc2-85b1-464c-92f5-d4a1156eacb
    {"state":>"STARTED"}
...
...
```

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 Router` excludes all Router logs:

```
$ cf logs nifty-gui --recent | grep -v Router
Connected, dumping recent logs for app nifty-gui in org jancloudspace-org / space development
as jancloudspace@shared-domain.com...
2014-02-07T10:54:40.41-0800 [STG]
    OUT ----> Uploading droplet (5.6M)
2014-02-07T10:54:44.44-0800 [DEA]
    OUT Starting app instance (index 0) with guid
    4d397313-20e0-478a-9a74-307446eb7640
2014-02-07T10:54:46.31-0800 [App/0]
    OUT Express server started
2014-02-07T10:57:53.60-0800 [API]
    OUT Updated app with guid 4d397313-20e0-478a-9a74-307446eb7640
    {"instances":>2}
2014-02-07T10:57:53.64-0800 [DEA]
    OUT Starting app instance (index 1) with guid
    4d397313-20e0-478a-9a74-307446eb7640
2014-02-07T10:57:55.88-0800 [App/1]
    OUT Express server started
...
...
```

Troubleshooting Application Deployment and Health

Page last updated:

This page assumes you are using cf CLI v6.

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 recommend 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 `Unable to detect a supported application type`.

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:

```
----> Uploading droplet (23M)
...
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 down
...
0 of 1 instances running, 1 failing
FAILED
Start unsuccessful
```

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 [buildpacks 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 ran `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

Cloud Foundry 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.

Examining 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:

```
$ 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://lraa:c6B-X@p-mysql|provider.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
```

Exploring logs

To explore all logs available in the container, first view the log filenames, then view the log that interests you:

```
$ cf files my-app logs/
Getting files for app my-app in org my-org / space development as a.user@example.com...
OK

staging_task.log      844B
stderr.log            182B
stdout.log            0B

$ cf files my-app logs/stderr.log
Getting files for app my-app in org my-org / space development as a.user@example.com...
OK

[2014-01-27 20:21:58] INFO  WEBrick 1.3.1
[2014-01-27 20:21:58] INFO  ruby 1.9.3 (2013-11-22) [x86_64-linux]
[2014-01-27 20:21:58] INFO  WEBrick::HTTPServer#start: pid=31 port=64391
```

Tracing Cloud Controller REST API Calls

If a command fails or produces unexpected results, re-run it with `CF_TRACE` enabled to view requests and responses between the cf CLI and the Cloud Controller REST API.

For example:

- Re-run `cf push` with `CF_TRACE` enabled:
`CF_TRACE=true cf push APP-NAME`
- 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 `CF_TRACE` 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> 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](#).
- `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 [explore](#) 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](#).

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

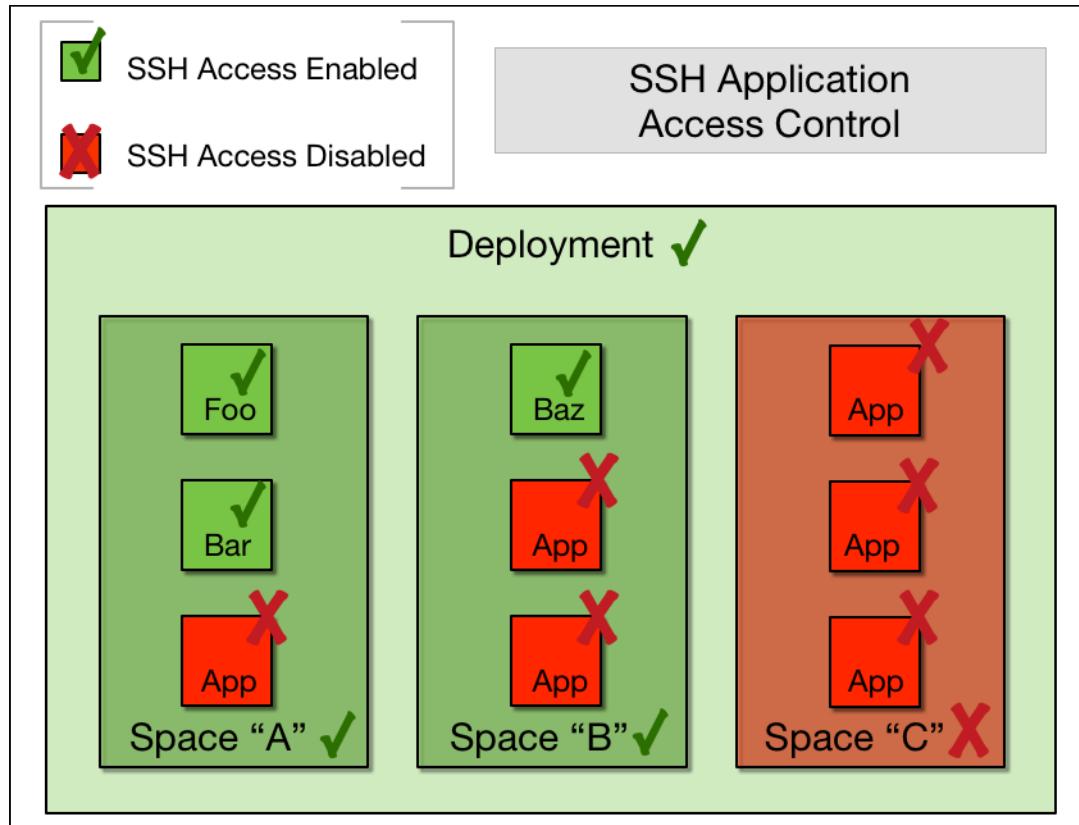
Operators, space administrators, and developers can configure SSH access for Elastic Runtime, spaces, and apps as described in this table:

User Role	Scope of SSH Permissions Control	How They Define SSH Permissions
Operator	Entire deployment	Configure the deployment to allow or prohibit SSH access (one-time)
Space Administrator	Space	cf CLI <code>allow-space-ssh</code> and <code>disallow-space-ssh</code> commands
Developer	Application	cf CLI <code>enable-ssh</code> and <code>disable-ssh</code> commands

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 `app_ssh_oauth_client` identifier in the Cloud Controller `/v2/info` endpoint, and uses this identifier it to query the UAA server for an SSH authorization code. On the target VM side, the SSH proxy contacts the Cloud Controller via the `app_ssh_endpoint` listed in `/v2/info` to confirm permission for SSH access.

Application SSH Commands

cf CLI Command	Purpose
<code>cf enable-ssh</code> <code>cf disable-ssh</code> <code>cf allow-space-ssh</code> <code>cf disallow-space-ssh</code>	Enable and Disable SSH Access
<code>cf ssh-enabled</code> <code>cf space-ssh-allowed</code>	Check SSH Access Permissions
<code>cf ssh</code>	Securely log into an application container.
<code>cf ssh-code</code>	Enable secure login to an application container using non-CF SSH tools like <code>ssh</code> , <code>scp</code> , and <code>sftp</code> .

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

`cf enable-ssh` enables SSH access to all instances of an app:

```
$ cf enable-ssh MY-AWESOME-APP
```

`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

`cf allow-space-ssh` allows SSH access into all apps in a space:

```
$ cf allow-space-ssh SPACE-NAME
```

`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`.

```
$ cf ssh MY-AWESOME-APP -L [LOCAL-NETWORK-INTERFACE:]LOCAL-PORT:REMOTE-HOST-NAME:REMOTE-HOST-PORT
```

- 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 the `/home/vcap/app/.profile.d` directory to make sure they will not perform any actions that are undesirable for your running app.

```
source /home/vcap/app/.profile.d/*.sh
```

If the `.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.

```
$ curl /v2/info
{
...
"app_ssh_endpoint": "ssh.MY-DOMAIN.com:2222",
"app_ssh_host_key_fingerprint": "a6:14:c0:ea:42:07:b2:f7:53:2c:0b:60: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:

```
$ scp -P 2222 -o User=cf:d0a2e11d-e6ca-4120-b32d-140c356906a5/0 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-140c356906a5/0@ssh.MY-DOMAIN.com
The authenticity of host '[ssh.MY-DOMAIN.com]:2222 ([52.12.144.5]:2222)' can't be established.
RSA key fingerprint is a6:14:c0:ea:42:07:b2:f7:53:2c:0b:60:e0:00:21:6c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[ssh.MY-DOMAIN.com]:2222 ([52.12.144.5]:2222)' (RSA) to the list of known hosts.
cf:d0a2e11d-e6ca-4120-b32d-140@ssh.ketchup.cf-app.com's password:
vcap@cc415164kws:~$
```

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 action, but is described here to complete the SSH security picture.

CONTAINER_PORT (required)

`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, `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 -1`.

USER (optional)

`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)

`password` declares the password to use during password authentication with the container's ssh daemon.

PRIVATE_KEY (optional)

`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

```
{
  "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: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-ssh": {
      "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.

```
"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

Page last updated:

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.

 **Note:** Your app must be prepared before you push it. See the [Deploy an Application](#) topic for details on preparing apps for deployment.

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.

```
$ cf create-service-key MY-DB EXTERNAL-ACCESS-KEY
```

2. Retrieve your new service key.

```
$ cf service-key MY-DB EXTERNAL-ACCESS-KEY
Getting key EXTERNAL-ACCESS-KEY for service instance MY-DB as user@pivotal.io...

{
  "hostname": "us-cdbr-iron-east-01.p-mysql.net",
  "jdbcUrl": "jdbc:mysql://us-cdbr-iron-east-03.p-mysql.net/ad_b2fca6t49704585d?user=b5136e448be920&password=231f435o05",
  "name": "ad_b2fca6t49704585d",
  "password": "231f435o05",
  "port": "3306",
  "uri": "mysql://b5136e448be920:231f435o05@us-cdbr-iron-east-03.p-mysql.net:3306/ad_b2fca6t49704585d?reconnect=true",
  "username": "b5136e448be920"
}
```

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 b5136e448be920 -h 0 -p -D ad_b2fca6t49704585d -P 63306
```

- Replace `b5136e448be920` 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_b2fca6t49704585d` 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.

Services Overview

Page last updated:

This documentation is intended for end users of Cloud Foundry and covers provisioning of service instances and integrating them with applications that have been pushed to Cloud Foundry. If you are interested in building Services for Cloud Foundry and making them available to end users, see the [Custom Services](#) documentation.

Services and Service Instances

Cloud Foundry offers a marketplace of services, from which users can provision reserved resources on-demand. Examples of resources services provide include databases on a shared or dedicated server, or accounts on a SaaS application. These resources are known as Service Instances and the systems that deliver and operate these resources are known as Services. Think of a service as a factory that delivers service instances.

For documentation on provisioning service instances and other lifecycle operations, see [Managing Service Instances](#).

 **Note:** For a service to be available in the marketplace, it must be integrated with Cloud Foundry by way of APIs. If you are interested in building Services for Cloud Foundry and making them available to end users, see the [Custom Services](#) documentation.

User-Provided Service Instances

Cloud Foundry enables users to integrate services that are not available in the marketplace with their applications using a feature called [User-Provided Service Instances \(UPSI\)](#).

Service Instance Credentials

Cloud Foundry enables users to automatically provision credentials needed to reach a service instance. These credentials can be managed automatically for use by applications on Cloud Foundry, or managed manually for use by external and local clients.

 **Note:** Not all services support automated generation of credentials. Some services support credentials through application binding only.

Application Binding

Service instance credentials can be delivered automatically to applications running on Cloud Foundry in an environment variable. For more information, see [Delivering Service Credentials to an Application](#).

For details on binding specific to your application development framework, refer to the Service Binding section in the documentation for [your framework's buildpack](#).

Service Keys

Credentials managed manually are known as **Service Keys**. For more information, see [Managing Service Keys](#).

Outbound IP Addresses

To allow an app to communicate with a service external to Elastic Runtime, you may need to configure the service to accept connections from your app based on its outbound IP address.

In your external service configuration, you must do one of the following:

- Whitelist the entire IP range for the Diego cell where the app is deployed.
- Derive the app IP address from its DNS name using a command-line tool such as `dig`, `host`, or `nslookup`. In your external service configuration,

whitelist the IP address or range of the app instance.

Streaming Applications Logs to Log Management Services

To learn how your application logs can be streamed to third-party log management services, see [Log Management Services](#).

User-provided service instances can be used to drain applications logs to a service not available in the marketplace. This is also known as setting up a syslog drain. We've documented instructions for a few providers in the [Service-Specific Instructions for Streaming Application Logs](#) topic.

Manage Application Requests with Route Services

To learn how marketplace services (and user-provided service instances) can be used to perform preprocessing on application requests, see [Manage Application Requests with Route Services](#).

Database Migrations

If your application relies on a relational database, you will need to apply schema changes periodically. For guidance on how to do database migrations on Cloud Foundry-managed services, see [Migrating a Database in Cloud Foundry](#).

Managing Service Instances with the CLI

Page last updated:

This page assumes you are using cf CLI v6.

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](#). If you are interested in building Services for Cloud Foundry and making them available to end users, see the [Custom Services](#) 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.

```
$ cf marketplace
Getting services from marketplace in org my-org / space test as me@example.com...
OK

service      plans      description
p-mysql     100mb, 1gb   A DBaaS
p-riakcs    developer   An S3-compatible object store
```

Creating Service Instances

You can create a service instance with the command: `cf create-service SERVICE PLAN SERVICE_INSTANCE`

- `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.

```
$ cf create-service rabbitmq small-plan my_rabbitmq
Creating service my_rabbitmq in org console / space development as user@example.com...
OK
```

Note: [User Provided Service Instances](#) provide a way for developers to bind applications with services that are not available in their Cloud Foundry marketplace.

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.

```
$ cf create-service my-db-service small-plan my-db -c '{"storage_gb":4}'
Creating service my-db in org console / space development as user@example.com...
OK
```

```
$ cf create-service my-db-service small-plan my-db -c /tmp/config.json
Creating service my-db in org console / space development as user@example.com...
OK
```

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 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.

```
$ cf services  
Getting services in org my-org / space test as user@example.com...  
OK  


| name     | service  | plan      | bound apps | last operation   |
|----------|----------|-----------|------------|------------------|
| mybucket | p-riakcs | developer | myapp      | create succeeded |
| mydb     | p-mysql  | 100mb     |            | create succeeded |


```

Get Details for a Particular Service Instance

Details include dashboard urls, if applicable, and operation start and last updated timestamps.

```
$ cf service mydb  
Service instance: mydb  
Service: p-mysql  
Plan: 100mb  
Description: MySQL databases on demand  
Documentation url:  
Dashboard: https://p-mysql.example.com/manage/instances/cc4eab9d-aff4-4beb-bc46-123f2a02dcf1  
  
Last Operation  
Status: create succeeded  
Message:  
Started: 2015-05-08T22:59:07Z  
Updated: 2015-05-18T22:01:26Z
```

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 applications 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 (e.g., 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
```

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 ?
Deleting service mydb in org my-org / space test as me@example.com...
OK
```

Managing Service Keys

Page last updated:

This page assumes you are using cf CLI v6.

This topic describes generation of credentials for service instances for use by external or local clients. For an overview of services, and documentation on other service management operations, see the [Services Overview](#) topic. If you are interested in building Services for Cloud Foundry and making them available to end users, see the [Custom Services](#) documentation.

 **Note:** Not all services support automated generation of credentials. Some services support credentials through application binding only.

Create a Service Key

Generates credentials for a service instance.

```
$ cf create-service-key myservice mykey
Creating service key mykey for service instance myservice as me@example.com...
OK
```

List Service Keys for a Service Instance

```
$ cf service-keys myservice
Getting service keys for service instance myservice as me@example.com...

name
mykey1
mykey2
```

Get Credentials for a Service Key

```
$ cf service-key myservice mykey
Getting key mykey for service instance myservice as me@example.com...

{
  uri: foo://user2:pass2@example.com/mydb,
  servicename: mydb
}
```

--guid can be provided to display the API guid for the service key.

```
$ cf service-key --guid myservice mykey
Getting key mykey for service instance myservice as me@example.com...

e3696fcb-7a8f-437f-8692-436558e45c7b

OK
```

Delete Service Key

```
$ cf delete-service-key myservice mykey
Are you sure you want to delete the service key mykey ? y
Deleting service key mykey for service instance myservice as me@example.com...

OK
```

Add option `-f` to force deletion without confirmation.

```
$ cf delete-service-key -f myservice mykey
Deleting service key mykey for service instance myservice as me@example.com...
OK
```

Delivering Service Credentials to an Application

Page last updated:

This page assumes you are using cf CLI v6.

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.

```
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@shared-domain.com
OK
```

For more information about application manifests, see [Deploying with Application Manifests](#).

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:

```
$ 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.

```
$ 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.

```
$ 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.

User-Provided Service Instances

Page last updated:

This page assumes you are using cf CLI v6.

User-provided service instances enable developers to use services that 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
```

Proxy Application Requests to a Route Service

User-provided services 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`.

Using Log Management Services

Page last updated:

This page assumes you are using cf CLI v6.

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

The Cloud Foundry marketplace offers several log management services. To use one of these services, create an instance of the service and bind it to your application. See the following topics for more information:

- [Service-Specific Instructions for Streaming Application Logs](#) to learn how to stream application logs to specific services including Papertrail, Splunk, and SumoLogic.
- [Managing Service Instances](#) to learn about creating and binding service instances.

 **Note:** Not all marketplace services support syslog drains. Some services implement an integration with Cloud Foundry that enables automated streaming of application syslogs. If you are interested in building Services for Cloud Foundry and making them available to end users, see the [Custom Services](#) documentation.

Using Other Services

If a compatible log management service is not available in your Cloud Foundry marketplace, use the following procedure to stream your logs to a service of your choice.

 **Note:** Only use this procedure if the service that you want to use is not covered in the [Service-Specific Instructions for Streaming Application Logs](#) topic.

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. See the [Service-Specific Instructions for Streaming Application Logs](#) topic for more information.
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

use `syslog-tls`, you must have an SSL certificate signed by a well-known certificate authority.

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`. After a short delay, logs begin to flow automatically.

Refer to [Delivering Service Credentials to an Application](#) 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

Page last updated:

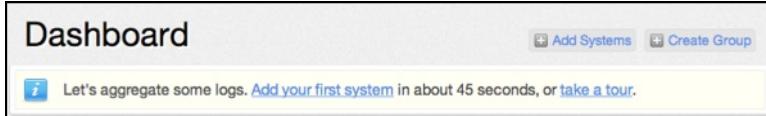
This topic provides instructions for configuring some third-party log management services.

Once you have configured a service, refer to the [Third-Party Log Management Services](#) topic for instructions on binding your application to the service.

Papertrail

From your Papertrail account:

1. Click **Add System**.



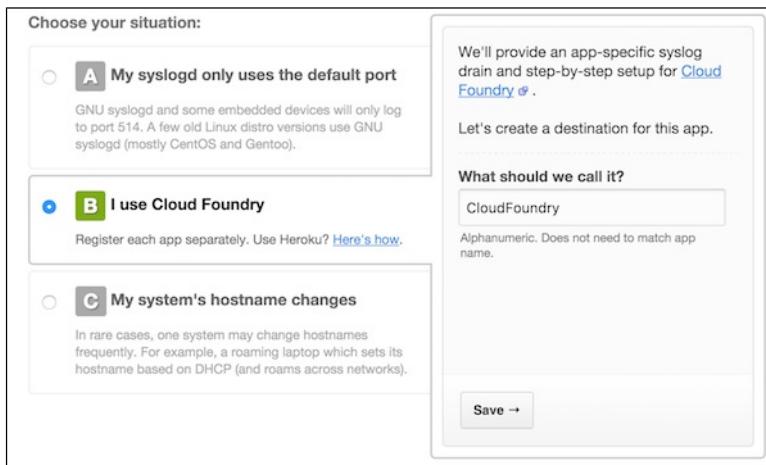
The screenshot shows the Papertrail Dashboard. At the top, there are buttons for 'Add Systems' and 'Create Group'. Below that, a message says 'Let's aggregate some logs. [Add your first system](#) in about 45 seconds, or [take a tour](#)'. There is a small icon of a person with a gear.

2. Click the **Other** link.



The screenshot shows the 'Setup Systems' page. It says 'Your systems will log to **logs2.papertrailapp.com:14608**'. Below that, there is a link '» Other situations: [Port 514](#) | [Other](#)'. The background has a light gray gradient.

3. Select **I use Cloud Foundry**, enter a name, and click **Save**.



The screenshot shows the 'Choose your situation' step. It has three options: **A My syslogd only uses the default port** (describing GNU syslogd and embedded devices), **B I use Cloud Foundry** (selected, with a note about registering each app separately), and **C My system's hostname changes** (describing rare cases like roaming laptops). To the right, there is a panel for creating a destination for the app, asking 'What should we call it?' with a text input field containing 'CloudFoundry' and a note that it's alphanumeric. A 'Save →' button is at the bottom.

4. Record the URL with port that is displayed after creating the system.



The screenshot shows a message box with the text 'CloudFoundry will log to **logs.papertrailapp.com:36129**'.

5. Create the log drain service in Cloud Foundry.

```
$ cf cups my-logs -l syslog://logs.papertrailapp.com:PORT
```

6. Bind the service to an app. Restage or push the app using either the `cf restage APPLICATION-NAME` or `cf push APPLICATION-NAME` command. After a short delay, logs begin to flow automatically.

7. Once Papertrail starts receiving log entries, the view automatically updates to the logs viewing page.

All Systems

Dashboard Events Account Help Me

```

Skiping auto-reconfiguration.
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigure.configure
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} INFO: No beans of type org.springframework.data.mongodb.MongoDbFactory found
in application context. Skipping auto-reconfiguration.
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigure.configure
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} INFO: No beans of type
org.springframework.beans.factory.config.PropertyPlaceholderConfigurer found
in application context. Skipping auto-reconfiguration.
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigure.configure
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} INFO: class org.springframework.amp.rabbit.connection.ConnectionFactory not
in classpath. Skipping auto-reconfiguration for it
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,680 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Id})},methods={DELETE}],params=[],headers=[],consumes=[],produces[],custom[]] onto public void
org.cloudfoundry.samples.music.web.controllers.AlbumController.deleteById(java.lang.String)
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,681 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Id})},methods={PUT}],params[],headers[],consumes[],produces[],custom[]] onto public org.cloudfoundry.samples.music.domain.Album
org.cloudfoundry.samples.music.web.controllers.AlbumController.updateById(java.lang.String)
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,681 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Id})},methods={GET}],params[],headers[],consumes[],produces[],custom[]] onto public
java.lang.Iterable<org.cloudfoundry.samples.music.domain.Album> org.cloudfoundry.samples.music.web.controllers.AlbumController.albums()
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,681 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Id})},methods={PUT}],params[],headers[],consumes[],produces[],custom[]] onto public org.cloudfoundry.samples.music.domain.Album
org.cloudfoundry.samples.music.web.controllers.AlbumController.updateById(java.lang.String)
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,682 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@POST})},params[],headers[],consumes[],produces[],custom[]] onto public org.cloudfoundry.samples.music.domain.Album
org.cloudfoundry.samples.music.web.controllers.AlbumController.createAlbum(org.cloudfoundry.samples.music.domain.Album)
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,684 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Info},methods[],params[],headers[],consumes[],produces[],custom[]] onto public org.cloudfoundry.samples.music.domain.ApplicationInfo
org.cloudfoundry.samples.music.web.controllers.InfoController.info()
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,684 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Id})},params[],headers[],consumes[],produces[],custom[]] onto public
java.util.List<org.springframework.cloud.service.ServiceInfo> org.cloudfoundry.samples.music.web.controllers.InfoController.showServiceInfo()
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,684 INFO RequestMappingHandlerMapping:181 - Mapped
[{\@Path({@Env})},params[],headers[],consumes[],produces[],custom[]] onto public java.lang.String, java.lang.String,
org.cloudfoundry.samples.music.web.controllers.InfoController.showEnvInfo()
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,704 INFO SimpleUrlHandlerMapping:302 - Root mapping to handler of
type [class org.springframework.web.servlet.mvc.ParameterizableViewController]
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,726 INFO SimpleUrlHandlerMapping:315 - Mapped URL path [/assets/**]
onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
Mar 05 14:57:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36,726 INFO SimpleUrlHandlerMapping:315 - Mapped URL path [/**] onto
handler of type [class org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler]
Mar 05 14:57:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:37,040 INFO DispatcherServlet:408 - FrameworkServlet 'appServlet':
initialization completed in 989 ms
Mar 05 14:57:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:37 PM org.apache.coyote.AbstractProtocol start
Mar 05 14:57:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} INFO: Starting ProtocolHandler ["http-nio-0-6399"]
Mar 05 14:57:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:37 PM org.apache.catalina.startup.Catalina start
Mar 05 14:57:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{App/1} INFO: Server startup in 11278 ms
Mar 05 14:59:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{API} Mar 05, 2014 10:59:36 PM org.apache.coyote.AbstractProtocol stop
Mar 05 14:59:36 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{API} INFO: Stopping ProtocolHandler ["http-nio-0-6399"]
Mar 05 14:59:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{DEA} Mar 05, 2014 10:59:37 PM org.apache.catalina.startup.Catalina stop
Mar 05 14:59:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{DEA} INFO: Stopping the Virtual Server (Index 1) with gruul
Mar 05 14:59:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{DEA} Stopped app instance (Index 1) with gruul 4008c968-c093-4bbb-8718-186f147f3e73
Mar 05 14:59:37 CloudFoundry $odc908-c093-4bbb-8718-186f147f3e73/{DEA} Stopped app instance (Index 1) with gruul 4008c968-c093-4bbb-8718-186f147f3e73

```

Example: "access denied" (1.2.3.4 OR redis) -sshd

Search Contrast PAUSE

https://papertrailapp.com/groups/55380/events/entered_on_id=3780034028458926114q-program3A4ad8960-c093-4bbb-8718-186f147f3e73%2f%58App%2f1NSD

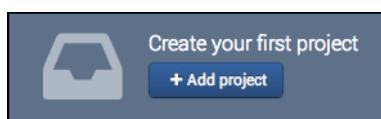
Splunk

See [Streaming Application Logs to Splunk](#) for details.

Splunk Storm

From your Splunk Storm account:

1. Click **Add project**.



2. Enter the project details.

Add project

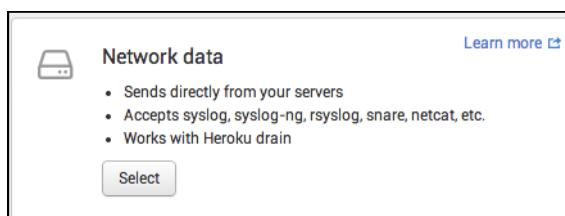
* Project name

* Project time zone

The Storm interface will use this time zone. If data you send to this project does not have a time zone already, it will be assigned this timezone by default [Learn More](#)

Cancel **Continue**

3. Create a new input for Network data.



4. Manually enter the external IP addresses your Cloud Foundry administrator assigns to outbound traffic. If you are using Pivotal Web Services, refer to the [Externally Visible IP Addresses](#) topic.

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.

```
$ cf cups my-logs -l syslog://HOST:PORT
```

7. Bind the service to an app. Restage or push the app using either the `cf restage APPLICATION-NAME` or `cf push APPLICATION-NAME` command. After a short delay, logs begin to flow automatically.

8. Wait for some events to appear, then click **Data Summary**.

9. Click the **loggregator** link to view all incoming log entries from Cloud Foundry.

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.

Add Collector

Select a type of collector:

Installed Collector

Select to install a Collector in your deployment.

Hosted Collector

Select to set up a Collector in the Sumo Logic Cloud.

FAQs

- ▶ What's the difference between an Installed and Hosted Collector?
- ▶ Where should I install an Installed Collector?
- ▶ How do I know if I need more than one Installed Collector?
- ▶ Where does my data go?

Add Collector

Name *

Description

Category

Unless overwritten by Source metadata, the Collector will set the Source category of all messages to this value.

Save | **Cancel**

3. In the new collector's row of the collectors view, click the **Add Source** link.

Manage Collectors and Sources

Show: All Collectors | Running Collectors | Stopped Collectors | Expand: All | None

Name	Type	Status	Source Category	Sources	Last Hour	Messages
Cloud Foundry	Hosted	✓		1	None	Add Source Edit Delete

4. Select **HTTP** source and fill in the details.

Select a type of Source:

Amazon S3

Collects logs from an Amazon S3 bucket.

HTTP

HTTP receiver that collects logs sent to a specific address.

Name*

Description

Source Host

Source Category

Advanced

Filters

Save | **Cancel**

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.

Manage Collectors and Sources

Show: All Collectors | Running Collectors | Stopped Collectors | Expand: All | None

Name	Type	Status	Source Category	Sources	Last Hour	Messages
Cloud Foundry	Hosted	✓		1	None	Add Source Edit Delete
CloudFoundry	HTTP	✓				Show URL Edit Delete

6. Create the log drain service in Cloud Foundry using the displayed URL.

```
$ cf cups my-logs -l HTTP-SOURCE-URL
```

7. Bind the service to an app. Restage or push the app using either the `cf restage APPLICATION-NAME` or `cf push APPLICATION-NAME` command. After a

short delay, logs begin to flow automatically.

8. In the SumoLogic dashboard, click **Manage**, then click **Status** to see a view of log messages received over time.

9. In the SumoLogic dashboard, click on **Search**. Place the cursor in the search box, then press **Enter** to submit an empty search query.

#	Time	Message
1	03/07/2014 11:48:34,000	203 <14>1 2014-03-07T17:48:34+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- 17:48:34,418 INFO DispatcherServlet:480 - FrameworkServlet 'appServlet': initialization completed in 1287 ms Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
2	03/07/2014 11:48:34,000	143 <11>1 2014-03-07T17:48:34+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- INFO: Starting ProtocolHandler ["http-bio-8080"] Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
3	03/07/2014 11:48:34,000	160 <11>1 2014-03-07T17:48:34+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- Mar 07, 2014 5:48:34 PM org.apache.catalina.startup.Catalina start Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
4	03/07/2014 11:48:34,000	126 <11>1 2014-03-07T17:48:34+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- INFO: Server startup in 24233 ms Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
5	03/07/2014 11:48:34,000	158 <11>1 2014-03-07T17:48:34+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- Mar 07, 2014 5:48:34 PM org.apache.coyote.AbstractProtocol start Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
6	03/07/2014 11:48:33,000	180 <11>1 2014-03-07T17:48:33+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- Mar 07, 2014 5:48:33 PM org.apache.catalina.util.SessionGenerator createSecureRandom Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
7	03/07/2014 11:48:33,000	195 <11>1 2014-03-07T17:48:33+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- Mar 07, 2014 5:48:33 PM org.apache.reconfig.org.springframework.cloud.AbstractCloudConnector getServiceInfo Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
8	03/07/2014 11:48:33,000	259 <14>1 2014-03-07T17:48:33+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- 17:48:33,286 INFO XmlBeanDefinitionReader:316 - Loading XML bean definitions from class path resource [META-INF/cloud/cloudfoundry-auto-reconfiguration-context.xml] Host: 99.74.215.164 ▾ Name: Htp Input ▾ Category: Htp Input
9	03/07/2014 11:48:33,000	367 <14>1 2014-03-07T17:48:33+00:00 loggregator 0f65cbbb-aabb-49a8-8073-f604fcc19260 [App/0] -- 17:48:33,907 INFO RequestMappingHandlerMapping:181 - Mapped '{[/{:info}],methods=[],params=[],consumes=[],produces[],consumes=[],headers=[],headers=[],custom=[]}' onto public org.cloudfoundry.samples.music.domain.ApplicationInfo org.cloudfoundry.samples.music.web.controllers.InfoController.info()

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:

This page assumes you are using cf CLI v6.

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{1}\S+\s{1}(\S+)
FORMAT = host::$1

[rfc5424_header]
REGEX = <(d+)>d{1}\s{1}\S+\s{1}\S+\s{1}(\S+)\s{1}(\S+)\s{1}(\S+)
FORMAT = prival::$1 appname::$2 procid::$3 msgid::$4
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

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:

This page assumes you are using cf CLI v6.

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:34/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
```

 **Note:** After this step the database has been migrated but the application itself has not started, because the normal start command is not used.

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`:
`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:

```
---  
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 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.

The following topics contain information that might be relevant to your deployment:

- [Buildpack Detection](#)
- [Custom Buildpacks](#)
- [Packaging Dependencies for Offline Buildpacks](#)
- [Supported Binary Dependencies](#)

System Buildpacks

This table describes Cloud Foundry system buildpacks. Each buildpack row lists supported languages and frameworks and includes a GitHub repo link. Specific buildpack names also link to additional documentation.

Name	Other Supported Languages and Frameworks	GitHub Repo
Java	Grails, Play, Spring, or any other JVM-based language or framework	Java source
Ruby	Ruby, Rack, Rails, or Sinatra	Ruby source
Node.js	Node or JavaScript	Node.js source
Binary	NA	Binary source
Go	NA	Go source
PHP	NA	PHP source
Python	NA	Python source
Staticfile	HTML, CSS, or JavaScript	Staticfile source

Other Buildpacks

If your application uses a language or framework that Cloud Foundry buildpacks do not support, you can try the following:

- Customize an existing buildpack, or create your own [custom buildpack](#).

 **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](#).

- Use a [Cloud Foundry Community Buildpack ↗](#).
- Use a [Heroku Third-Party Buildpack ↗](#).

Buildpack Detection

Page last updated:

When you push an app, Cloud Foundry determines which buildpack to use. Each buildpack has a position in the detection priority list. Cloud Foundry first checks whether the buildpack in position 1 is right for the app. If the position 1 buildpack is not appropriate, Cloud Foundry moves on to the position 2 buildpack. This goes on until Cloud Foundry finds the correct buildpack. Otherwise, `cf push` fails with an Unable to detect a supported application type error.

Each buildpack contains a detect script. Cloud Foundry determines whether a buildpack is appropriate for an app by running that buildpack's detect script against the app.

By default, the three systems buildpacks occupy the first three positions on the detection priority list. Since running detect scripts takes time, an administrator can decrease the average time required to push apps by making sure that the most frequently used buildpacks occupy the lowest positions on the list.

Custom Buildpacks

Page last updated:

Buildpacks are a convenient way of packaging framework and/or runtime support for your application.

For example, by default Cloud Foundry does not support Haskell. Using a buildpack for Haskell allows you to add support for it at the deployment stage. When you push an application written using Haskell, the required buildpack is automatically installed on the Cloud Foundry Droplet Execution Agent (DEA) where the application will run.

 **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](#).

Custom Buildpacks

The structure of a buildpack is straightforward. A buildpack repository contains three main scripts, situated in a folder named `bin`.

bin/detect

The `detect` script is used to determine whether or not to apply the buildpack to an application. The script is called with one argument, the build directory for the application. It returns an exit code of `0` if the application can be supported by the buildpack. If the script does return `0`, it should also print a framework name to STDOUT.

Shown below is an example `detect` script written in Ruby that checks for a Ruby application based on the existence of a `Gemfile`:

```
#!/usr/bin/env ruby

gemfile_path = File.join ARGV[0], "Gemfile"

if File.exist?(gemfile_path)
  puts "Ruby"
  exit 0
else
  exit 1
end
```

Note that cf CLI displays the returned STDOUT “framework name” when printing the details of an application. Buildpack authors are therefore encouraged to include additional details in this returned framework name, such as 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-reconfiguration=1.7.0_RELEASE tomcat-access-logging-support=2.4.0_RELEASE tomcat-in-
```

bin/compile

The `compile` script builds the droplet that will be run by the DEA and will therefore contain all the components necessary to run the application.

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 execution of this script all output sent to STDOUT will be relayed via the cf CLI back to the end user. The generally accepted pattern for this is to break out this functionality into a ‘language_pack’. A good example of this can be seen at https://github.com/cloudfoundry/heroku-buildpack-ruby/blob/master/lib/language_pack/ruby.rb

A simple `compile` script is shown below:

```
#!/usr/bin/env ruby

#sync output
$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 to cache_path
  # install ruby
end
```

bin/release

The `release` script provides feedback metadata back to Cloud Foundry indicating how the application should be executed. The script is run with one argument, the build location of the application. The script must generate a YAML file in the following format:

```
config_vars:
  name: value
default_process_types:
  web: commandLine
```

Where `config_vars` is an optional set of environment variables that will be defined in the environment in which the application is executed. `default_process_types` indicates the type of application being run and the command line used to start it. At this time only `web` type of applications are supported.

The following example shows what a Rack application's `release` script might return:

```
config_vars:
  RACK_ENV: production
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

Partially or completely disconnected environments

Cloud Foundry buildpacks work with limited or no internet connectivity. A Cloud Foundry operator can enable the same support in your custom buildpack by using the [Buildpack Packager](#) application.

Using 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.

Option Flags

`--force-download`

By default, buildpack-packager stores the dependencies that it downloads while building a cached buildpack in a local cache at `~/.buildpack-packager`. This is in order to avoid re-downloading them when repackaging similar buildpacks. Running `buildpack-packager cached` with the `--force-download` option will force 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`

If you would like to include a different manifest file in your packaged buildpack, you may call buildpack-packager with the `--use-custom-manifest` option. buildpack-packager will generate a buildpack with the specified manifest. If you are building a cached buildpack, buildpack-packager will vendor dependencies from the specified manifest as well.

You can find more documentation at the [buildpack-packager Github repo](#).

Uploading the buildpack

Once you have packaged your buildpack using `buildpack-packager`, you can upload the resulting `.zip` file to your instance of Cloud Foundry:

```
cf create-buildpack BUILDPACK_NAME BUILDPACK_ZIP_PATH  
POSITION
```

You can find more documentation at the [buildpack admin guide](#).

Deploying 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:

```
$ 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:

```
$ cf push my-new-app -b https://username:password@github.com/johndoe/my-buildpack.git
```

By default, Cloud Foundry uses the master branch of the buildpack's git repository. You can specify a different branch using the git url as shown in the following example:

```
$ cf push my-new-app -b https://username:password@github.com/johndoe/my-buildpack.git#my-branch-name
```

The application will then be deployed to Cloud Foundry, and the buildpack will be cloned from the repository and applied to the application (provided that the `detect` script returns `0`).

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.git/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.git/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
```

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.git/repo
$ cd

# Builds a package using your script
$ ./package https://our.internal.site/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/dependency/repo/dep1.tgz.... done
  https://our.internal.site/dependency/repo/dep2.tgz.... WARNING: dependency not found!
```

Pros

- 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

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](#).

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
```

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](#)
- [Configure Service Connections for Grails](#)
- [Configure Service Connections for Play](#)
- [Configure Service Connections for Spring](#)
- [Cloud Foundry Eclipse Plugin](#)
- [Cloud Foundry Java Client Library](#)
- [Build Tool Integration](#)

You can find the source for the buildpack on GitHub: <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 `pong_matcher_grails` 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.

Build Tool	Build Script	Documentation
Gradle	<code>build.gradle</code>	Gradle User Guide
Grails	<code>BuildConfig.groovy</code>	Grails: Configuration - Reference Documentation
Maven	<code>pom.xml</code>	Apache Maven Project Documentation

Sample App Step

You can skip this step. The `pong_matcher_grails/app/grails-app/conf/BuildConfig.groovy` file contains the dependencies for the `pong_matcher_grails` 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`, `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.

Step 4: (Optional) Configure a Procfile

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.

When you deploy, Elastic Runtime determines if a Procfile exists and uses the Procfile to configure your app with the specified settings.

In a Procfile, you declare one process type per line and use the following syntax, as shown in the example below:

- `PROCESS_TYPE` is the command name in the format of an alphanumeric string.
- `COMMAND` is the command line to launch the process.

Example process type syntax:

```
PROCESS_TYPE: COMMAND
```

You create a Procfile and add a command line for a `web` process type. Store the Procfile in the root directory of your app. The example shows a process type that starts the launch script created by the build process.

Example Procfile:

```
web: build/install/MY-PROJECT-NAME/bin/MY-PROJECT-NAME ]
```

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 clouduer@example.com...
OK

service    plans    description
...
cleardb   spark, boost, amp   Highly available MySQL for your apps
...
elephantsql  turtle, panda, elephant   PostgreSQL as a Service
...
```

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 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 clouduer@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.

```
---  
applications:  
...  
services:  
- mysql
```

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.

`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.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.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

1. Change to the `app` directory, and run `./grails war` to build the app.

2. Run `cf push pong_matcher_grails -n HOST_NAME` to push the app.

Example: `cf push pong_matcher_grails -n my-grails-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_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.

```
$ 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 1.8.0_25 from https://download.run.pivotal.io/openjdk/lucid/x86_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-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 Tomeat 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 Tomeat 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
#0  running  2014-11-10 05:07:33 PM  0.0%  686.4M of 1G  153.6M of 1G
#1  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](#).

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"}'
```

5. To check the status of the first match request, run:

```
curl -v -X GET  
$HOST/match_requests/firstrequest
```

The last line of the output shows the `match_id`.

6. Replace `MATCH_ID` with the `match_id` value from the previous step in the following command:

```
curl -v -H "Content-Type: application/json" -X POST $HOST/results -d '{ "match_id": "MATCH_ID", "winner": "andrew", "loser": "navratilova" }'
```

You should receive a 201 response.

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 cf Command Line Interface (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 [Understanding Apps Manager Permissions](#) 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

Page last updated:

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 `pong_matcher_groovy` 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 [cf Command Line Interface \(CLI\)](#)
- 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.

Build Tool	Build Script	Documentation
Gradle	<code>build.gradle</code>	Gradle User Guide
Maven	<code>pom.xml</code>	Apache Maven Project Documentation

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.

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:

```
$ 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

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.

When you deploy, Elastic Runtime determines if a Procfile exists and uses the Procfile to configure your app with the specified settings.

In a Procfile, you declare one process type per line and use the following syntax, as shown in the example below:

- `PROCESS_TYPE` is the command name in the format of an alphanumeric string.
- `COMMAND` is the command line to launch the process.

Example process type syntax:

```
PROCESS_TYPE: COMMAND
```

You create a Procfile and add a command line for a `web` process type. Store the Procfile in the root directory of your app. The example shows a process type that starts the launch script created by the build process.

Example Procfile:

```
web: build/install/MY-PROJECT-NAME/bin/MY-PROJECT-NAME
```

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.

```
$ cf marketplace
Getting services from marketplace in org Cloud-Apps / space development as clouduser@example.com...
OK

service    plans    description
...
elephantsql  turtle, panda, elephant  PostgreSQL as a Service
...
rediscloud  30mb, 100mb, 1gb, 10gb, 50gb Enterprise-Class Redis for Developers
...
```

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`. This creates 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.

```
---
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 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.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.

`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.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.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

1. Change to the `app` directory, and run `./gradlew distZip` to build the app.
2. Run `cf push pong_matcher_groovy -n HOST_NAME` to push the app.

Example: `cf push pong_matcher_groovy -n 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.

```
$ cf push pong_matcher_groovy -n groovy-ratpack-app
Using manifest file /Users/example/workspace/pong_matcher_groovy/app/manifest.yml

Creating app pong_matcher_groovy in org Cloud-Apps / space development as clouduer@example.com...
OK

Creating route groovy-ratpack-app.cfapps.io...
OK

Binding groovy-ratpack-app.cfapps.io to pong_matcher_groovy...
OK

Uploading pong_matcher_groovy...
Uploading app files from: /Users/example/workspace/pong_matcher_groovy/app/build/distributions/app.zip
Uploading 138.2K, 18 files
OK

Binding service baby-redis to app pong_matcher_groovy in org Cloud-Apps / space development as clouduer@example.com...
OK

Starting app pong_matcher_groovy in org Cloud-Apps / space development as clouduer@example.com...
OK
----> Downloaded app package (12M)
Cloning into '/tmp/buildpacks/java-buildpack'...
----> Java Buildpack Version: 9e096be | https://github.com/cloudfoundry/java-buildpack#9e096be
      Expanding Open Jdk JRE to .java-buildpack/open_jdk_jre (1.3s)
----> Uploading droplet (49M)

0 of 1 instances running, 1 starting
1 of 1 instances running

App started

Showing health and status for app pong_matcher_groovy in org Cloud-Apps / space development as clouduer@example.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: groovy-ratpack-app.cfapps.io

  state  since          cpu    memory      disk
#0  running  2014-10-28 04:48:58 PM  0.0%  193.5M of 512M  111.7M 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](#).

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"}'
```

5. To check the status of the first match request, run:

```
curl -v -X GET  
$HOST/match_requests/firstrequest
```

The last line of the output shows the `match_id`.

6. Replace `MATCH_ID` with the `match_id` value from the previous step in the following command:

```
curl -v -H "Content-Type: application/json" -X POST $HOST/results -d'{"match_id": "MATCH_ID", "winner": "andrew", "loser": "navratilova"}'
```

You should receive a `201` response.

`Created`

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 cf Command Line Interface (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 [Understanding Apps Manager Permissions](#) 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 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 [cf Command Line Interface \(CLI\)](#)
- 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.

Build Tool	Build Script	Documentation
Gradle	<code>build.gradle</code>	Gradle User Guide
Maven	<code>pom.xml</code>	Apache Maven Project Documentation

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`, `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. 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](#) topic.

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 an app with an artifact.

`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.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.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

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.

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.

```
$ cf push pong_matcher_spring -n spring1119
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 spring1119.cfapps.io...
OK

Binding spring1119.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: spring1119.cfapps.io

  state  since          cpu    memory    disk
#0  running  2014-11-19 12:29:27 PM  0.0%  553.6M of 1G  127.4M of 1G
```

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"}'
```

5. To check the status of the first match request, run:

```
curl -v -X GET  
$HOST/match_requests/firstrequest
```

The last line of the output shows the `match_id`.

6. Replace `MATCH_ID` with the `match_id` value from the previous step in the following command:

```
curl -v -H "Content-Type: application/json" -X POST $HOST/results -d' { "match_id": "MATCH_ID", "winner": "andrew", "loser": "navratilova" }'
```

You should receive a `201 Created` response.

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 cf Command Line Interface (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 [Understanding Apps Manager Permissions](#) 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 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:

This page assumes you are using cf CLI v6.

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:

```
$ gradle 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:

```
$ gradle 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](#)
- [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#0928916a2dd78e9faf9469c558046cef09f60e5d
-> Downloading Open Jdk JRE 1.7.0_51 from
  http://.../openjdk/lucid/x86_64/openjdk-1.7.0_51.tar.gz (0.0s)
  Expanding Open Jdk 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` 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 `path` attribute specifies the current directory, the manifest will include a line like this:

```
path: .
```

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 `path` argument in `manifest.yml` to point to the WAR, and re-push the application.

Debugging 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.

```
$ 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:

```

class NewRelicAgent < JavaBuildpack::Component::VersionedDependencyComponent

# @macro base_component_compile
def compile
  FileUtils.mkdir_p logs_dir

  download_jar
  @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.app_name', "#{application_name}")
  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:

```
System.getenv("VCAP_SERVICES");
```

See the [Cloud Foundry Environment Variables](#) topic for more information.

Configure Service Connections for Grails

Page last updated:

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:

```
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 = 27107
        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.

```
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:

```
beans = {
    cloudFactory(org.springframework.cloud.CloudFactory)
}
```

3. Add the following `imports` to your `DataSource.groovy` file to allow `spring-cloud` API commands:

```
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.

```
def cloud = null
try {
    cloud = new CloudFactory().cloud
} catch (CloudException) {}
```

5. Use code like the following to access the cloud object:

```
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) {}

dataSource {
    pooled = true
    dbCreate = 'update'
    driverClassName = 'com.mysql.jdbc.Driver'
}

environments {
    production {
        dataSource {
            def dbInfo = cloud?.getServiceInfo('myapp-mysql')
            url = dbInfo?.jdbcUrl
            username = dbInfo?.userName
            password = dbInfo?.password
        }
        grails {
            mongo {
                def mongoInfo = cloud?.getServiceInfo('myapp-mongodb')
                host = mongoInfo?.host
                port = mongoInfo?.port
                databaseName = mongoInfo?.database
                username = mongoInfo?.userName
                password = mongoInfo?.password
            }
            redis {
                def redisInfo = cloud?.getServiceInfo('myapp-redis')
                host = redisInfo?.host
                port = redisInfo?.port
                password = redisInfo?.password
            }
        }
    }
    development {
        dataSource {
            url = 'jdbc:mysql://localhost:5432/myapp'
            username = 'sa'
            password = ''
        }
        grails {
            mongo {
                host = 'localhost'
                port = 27107
                databaseName = 'foo'
                username = 'user'
                password = 'password'
            }
            redis {
                host = 'localhost'
                port = 6379
                password = 'password'
            }
        }
    }
}
}

```

Configure Service Connections for Play Framework

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 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.

Configure 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.

```
<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:

```
<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.

Creating Service Beans

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.

```
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:

```
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.

Connecting 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.

Scanning 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:

```
@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:

```
@Autowired DataSource inventoryDb;
```

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:

```
@Autowired @Qualifier("inventory-db") DataSource inventoryDb;  
@Autowired @Qualifier("shipping-db") DataSource shippingDb;
```

Accessing Service Properties

You can expose raw properties for all services and the app through a bean as follows:

```
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 conditionalize 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);
        }
    }

    @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 takes the form `cloud.services.mysql.connection.{property}`. Cloud Foundry uses the following aliases in this case:

- `mysql`
- `postgresql`
- `mongodb`

- redis
- rabbitmq

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 `spring-mysql` to your application. Your application requires a `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.

Property	Description
<code>cloud.application.name</code>	The name provided when the application was pushed to Cloud Foundry.
<code>cloud.provider.url</code>	The URL of the cloud hosting the application, such as <code>cloudfoundry.com</code> .

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 `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:

```
<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: `driverClassName`, `url`, `username`, `password`, `validationQuery`.

Manual Configuration in Java

To configure a database service in Java configuration, create a `@Configuration` class with a `@Bean` method to return a `javax.sql.DataSource` bean. The bean can be created by helper classes in the `spring-cloud` library, as shown here:

```
@Configuration
public class DataSourceConfig {
  @Bean
  public DataSource dataSource() {
    CloudFactory cloudFactory = new CloudFactory();
    Cloud cloud = cloudFactory.getCloud();
    String serviceID = cloud.getServiceID();
    return cloud.getServiceConnector(serviceID, DataSource.class, 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 `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:

```
<mongo:db-factory
  id="mongoDbFactory"
  dbname="pwdtest"
  host="127.0.0.1"
  port="1234"
  username="test_user"
  password="test_pass" />
```

Cloud Foundry creates a `SimpleMongoDbFactory` with its own values for the following properties: `host`, `port`, `username`, `password`, `dbname`.

Manual Configuration in Java

To configure a MongoDB service in Java configuration, create a `@Configuration` class with a `@Bean` method to return an `org.springframework.data.mongodb.MongoDbFactory` bean from Spring Data MongoDB. The bean can be created by helper classes in the `spring-cloud` library, as shown here:

```
@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 `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:

```
<bean id="redis"
  class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
  p:hostName="localhost" p:port="6379" />
```

Cloud Foundry creates a `JedisConnectionFactory` with its own values for the following properties: `host`, `port`, `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 `@Configuration` class with a `@Bean` method to return an `org.springframework.data.redis.connection.RedisConnectionFactory` bean from Spring Data Redis. The bean can be created by helper classes in the `spring-cloud`

library, as shown here:

```

@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, DataSource.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:

```

<rabbit:connection-factory
    id="rabbitConnectionFactory"
    host="localhost"
    password="testpwd"
    port="1238"
    username="testuser"
    virtual-host="virthost" />

```

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:

```

@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**.

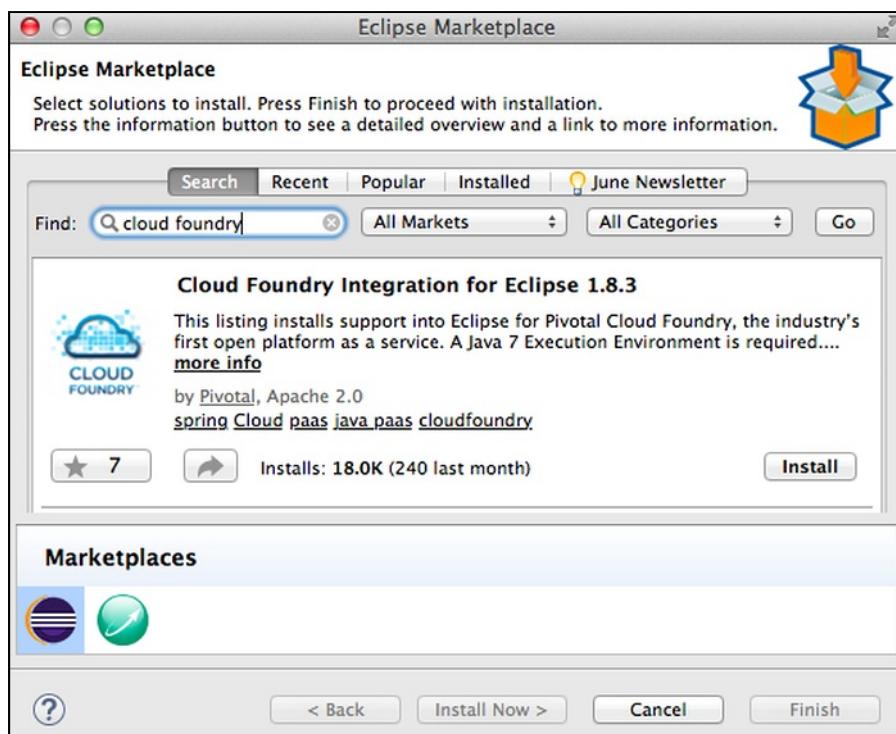
Follow the installation instructions appropriate for your environment:

- [Install to Eclipse from Marketplace](#)
- [Install to STS from IDE Extensions Tab](#)
- [Install from a Local Repository](#)

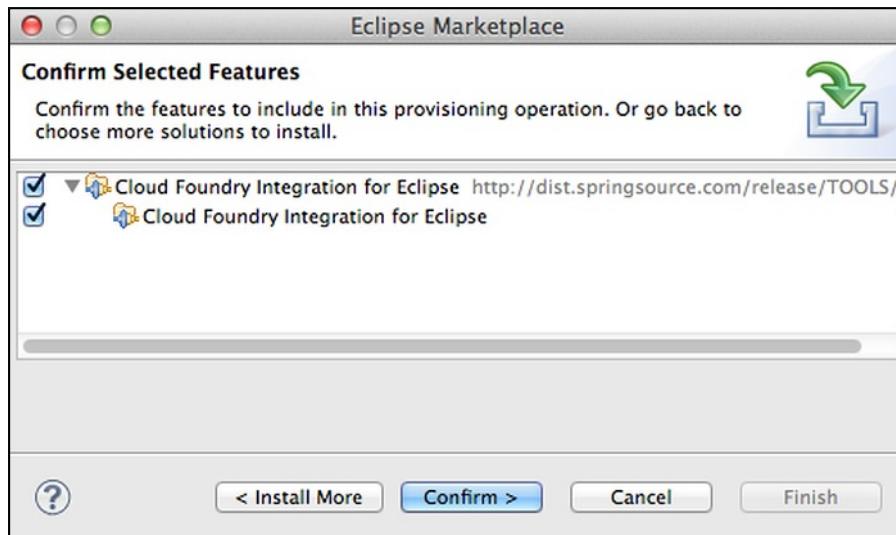
Install to Eclipse from Marketplace

Follow the instructions below to install the Cloud Foundry Eclipse Plugin to Eclipse from the Eclipse Marketplace.

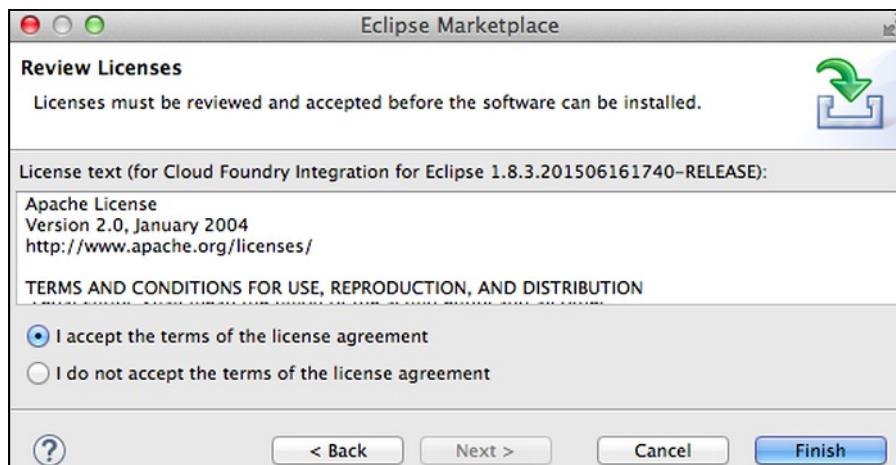
1. Start Eclipse.
2. From the Eclipse **Help** menu, select **Eclipse Marketplace**.
3. In the Eclipse Marketplace window, enter “Cloud Foundry” in the **Find** field. Click **Go**.
4. In the search results, next to the listing for Cloud Foundry Integration, click **Install**.



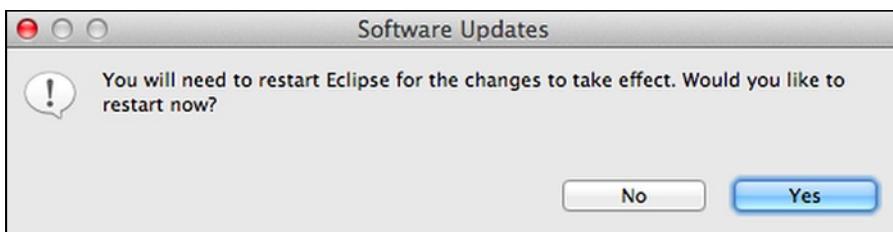
5. In the Confirm Selected Features window, click Confirm.



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.



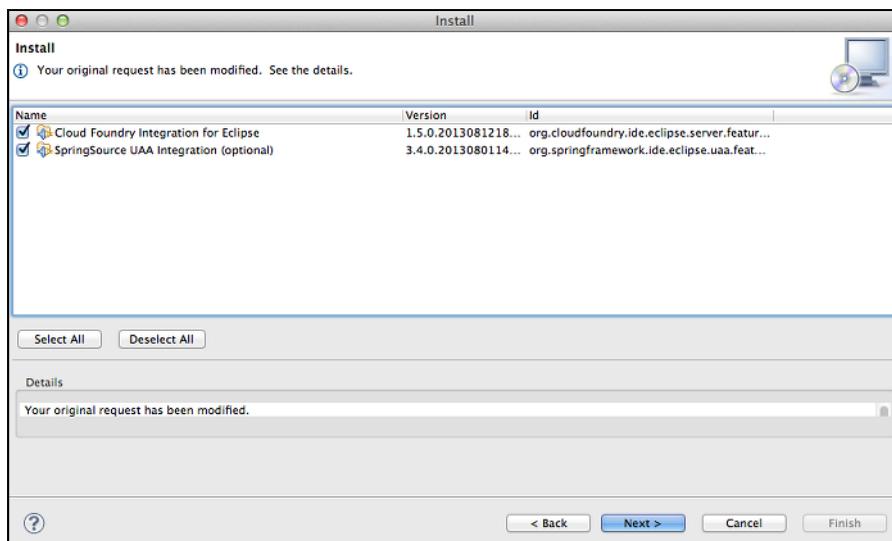
Install to STS from IDE Extensions Tab

Follow these instructions to install the Cloud Foundry Eclipse Plugin to Spring Tool Suite (STS) from the **IDE Extensions** tab.

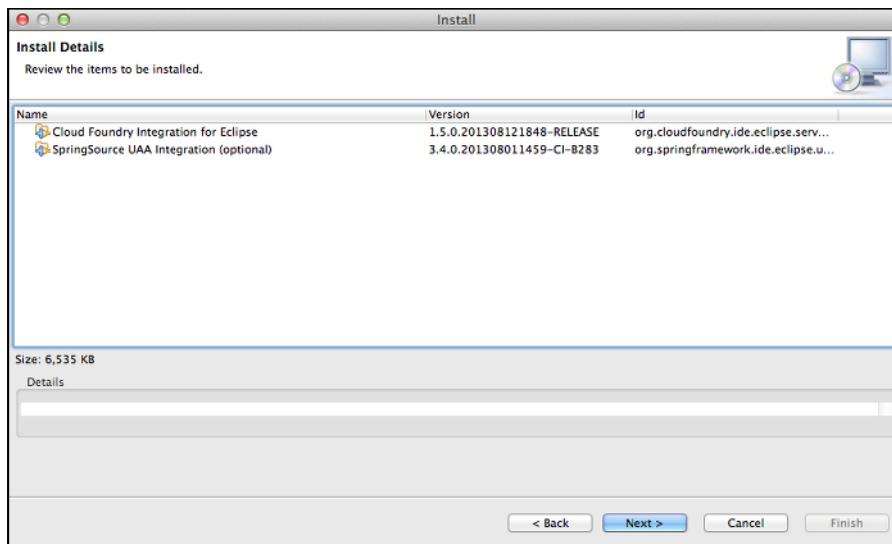
1. Start STS.
2. On the STS Dashboard, click **IDE Extensions**.

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**.



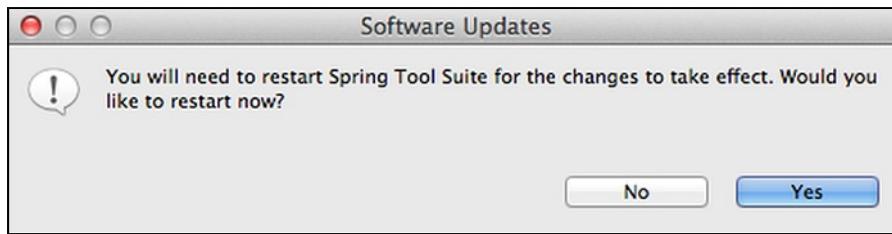
6. In the **Install Details** window, click **Next**.



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).

1. Browse to <https://github.com/cloudfoundry/eclipse-integration-cloudfoundry/blob/master/updatesites.md> and download a release update site zip file.
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:
 - o Download archived source code for released versions of the plugin from <https://github.com/SpringSource/eclipse-integration-cloudfoundry/releases>
 - o Clone the project repository:

```
$ git clone https://github.com/SpringSource/eclipse-integration-cloudfoundry
```

2. Unzip the downloaded archive. In a terminal, run the following command:

```
$ mvn -Pe37 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**.

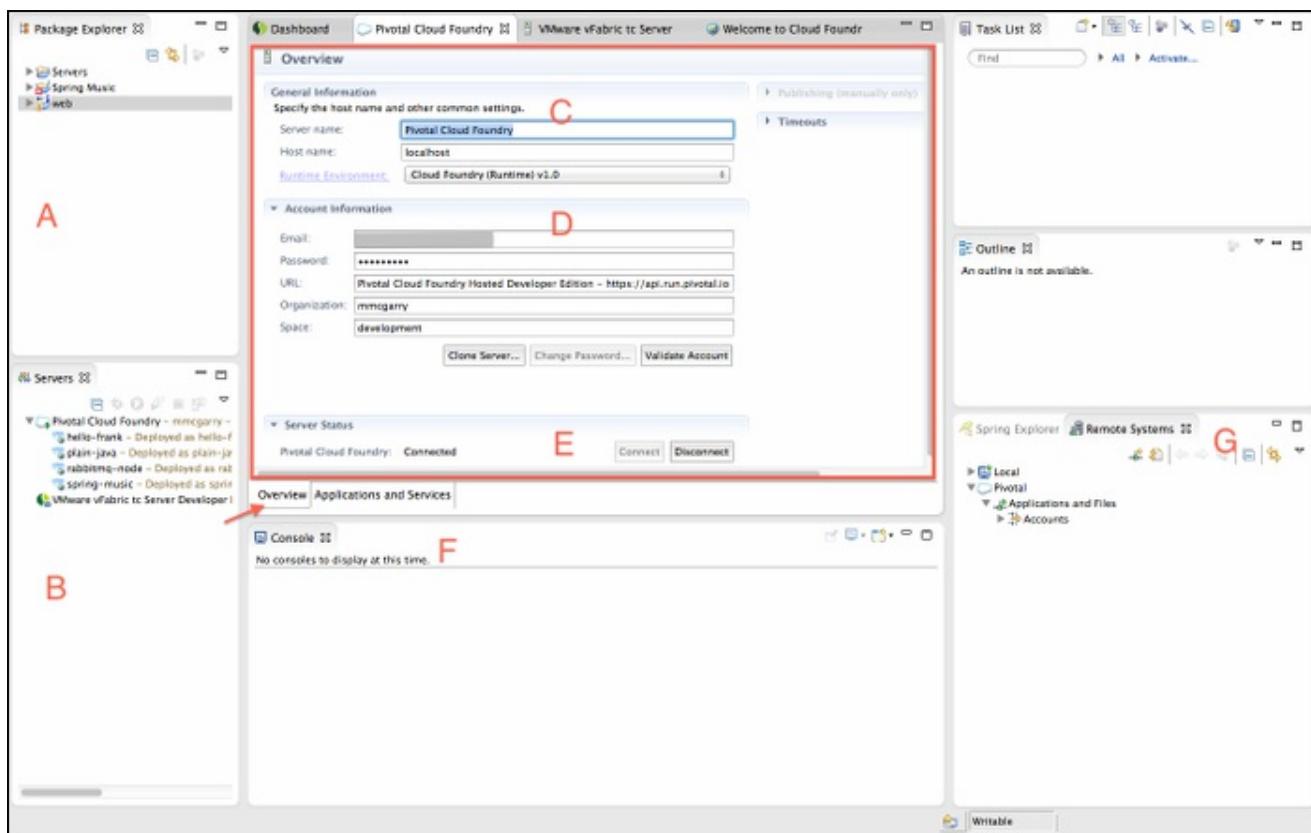
The Cloud Foundry editor, outlined in red in the screenshot below, is the primary plugin user interface. Some workflows involve interacting with standard elements of the Eclipse user interface, such as the **Project Explorer** and the **Console** and **Servers** views.

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 following panes and views are present when the **Overview** tab is selected:

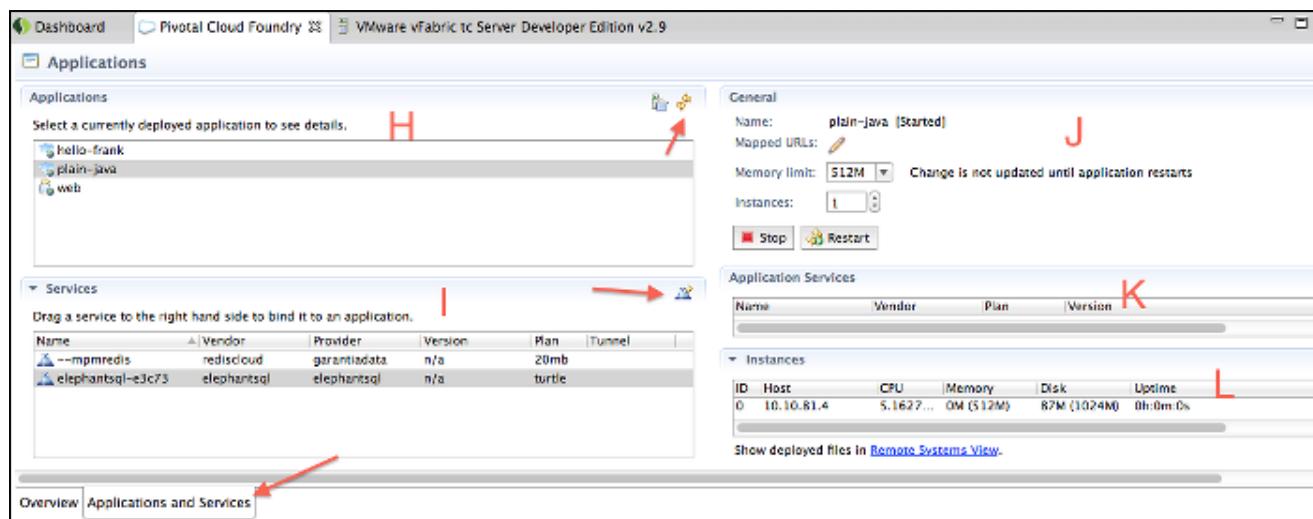
- A – The **Package Explorer** view lists the projects in the current workspace.
- B – The **Servers** view lists server instances configured in the current workspace. A server of type **Pivotal Cloud Foundry®** represents a targeted space in a Cloud Foundry instance.
- C – The **General Information** 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 following 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](#).

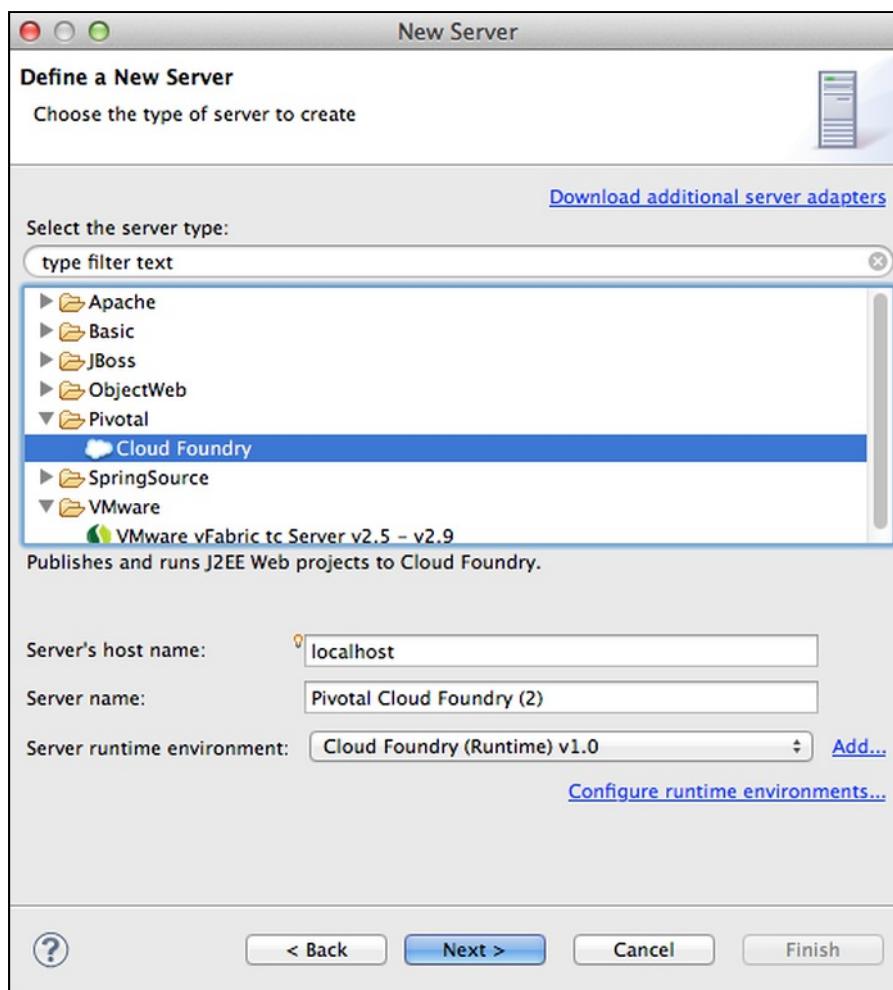


Create a Cloud Foundry Server

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**.

Note: Do not modify default values for **Server host name** or **Server Runtime Environment**. These fields are not used

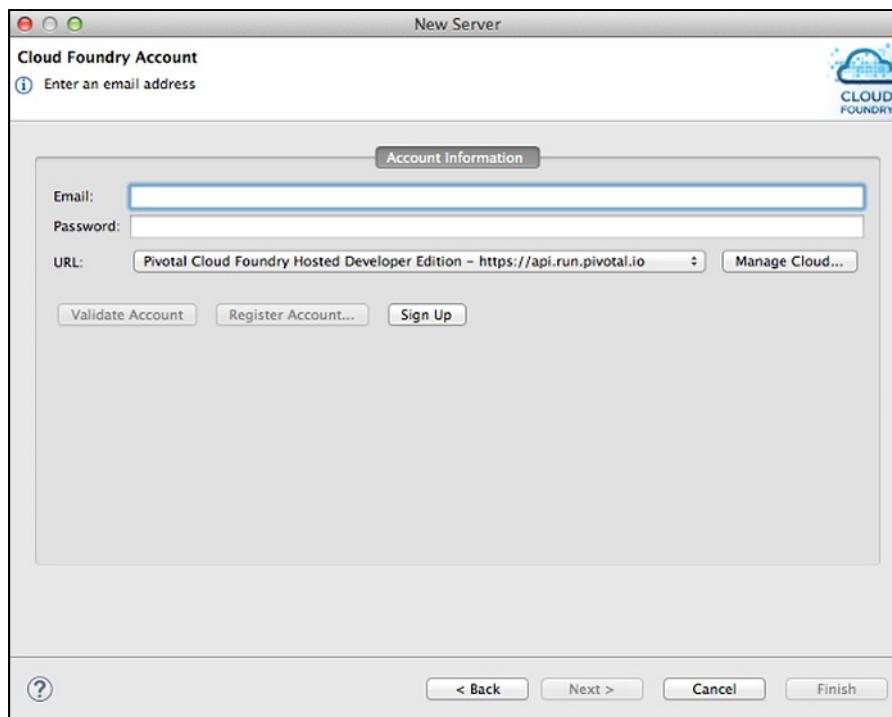


3. In the **Cloud Foundry Account** window, if you already have a Pivotal Cloud Foundry® Hosted Developer Edition account, enter your email account and password credentials and click **Validate Account**.

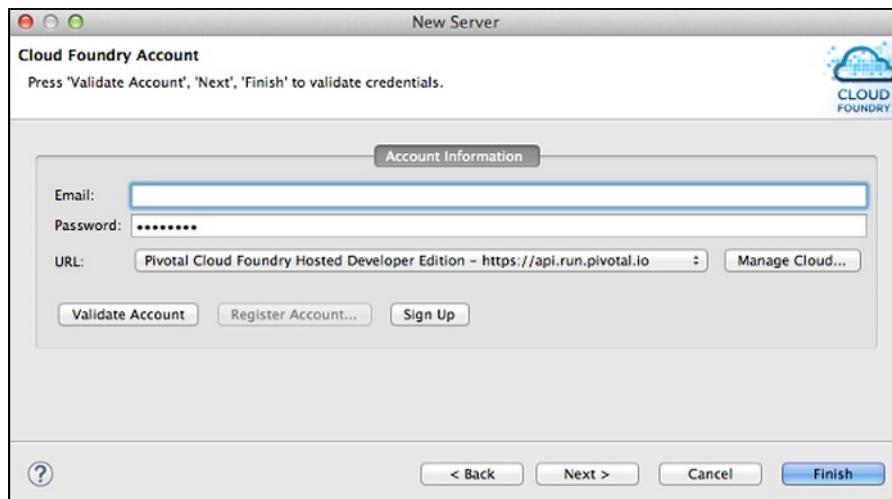
Note: By default, the **URL** field points to the Pivotal Cloud Foundry® Hosted Developer Edition URL of <https://api.run.pivotal.io>. If you have a Pivotal Elastic Runtime account, refer to the [Logging into the Apps Manager](#) topic to determine your Pivotal Elastic Runtime URL. Click **Manage Cloud...** 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.

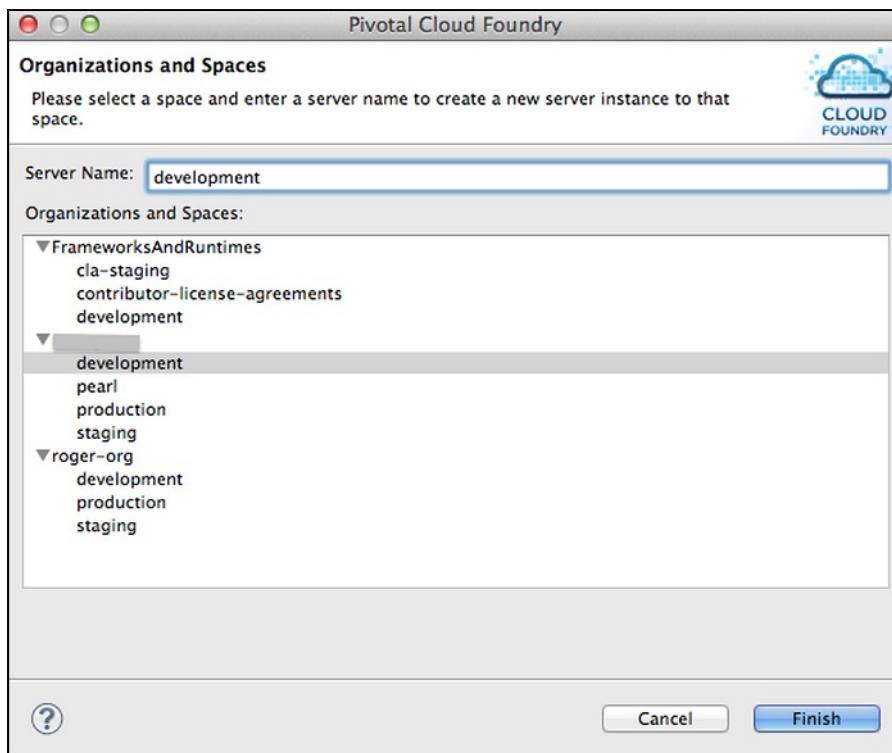


4. The **Cloud Foundry Account** window is refreshed and displays a message indicating whether or not your credentials were valid. Click **Next**.



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.



- 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](#).

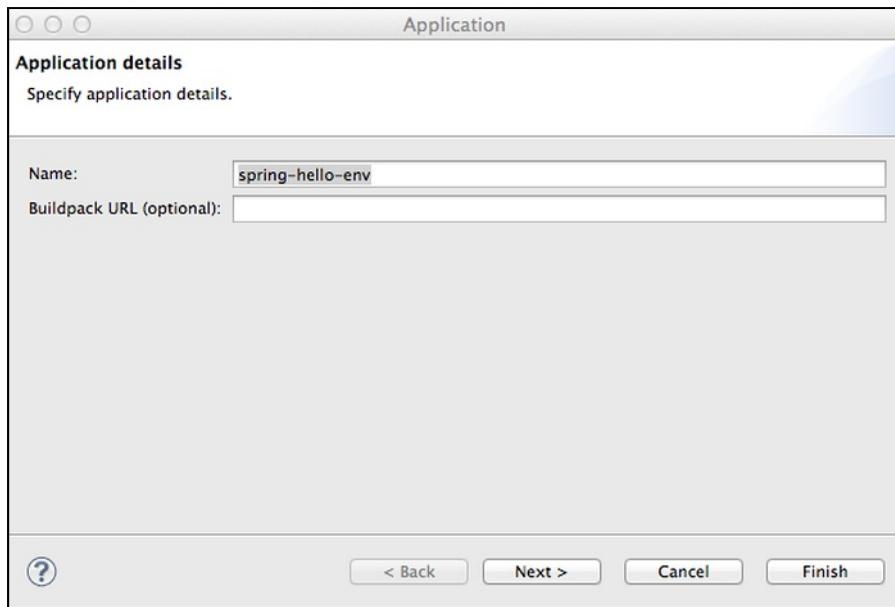
Deploy an Application

To deploy an application to Cloud Foundry using the plugin:

- 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.
- 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.

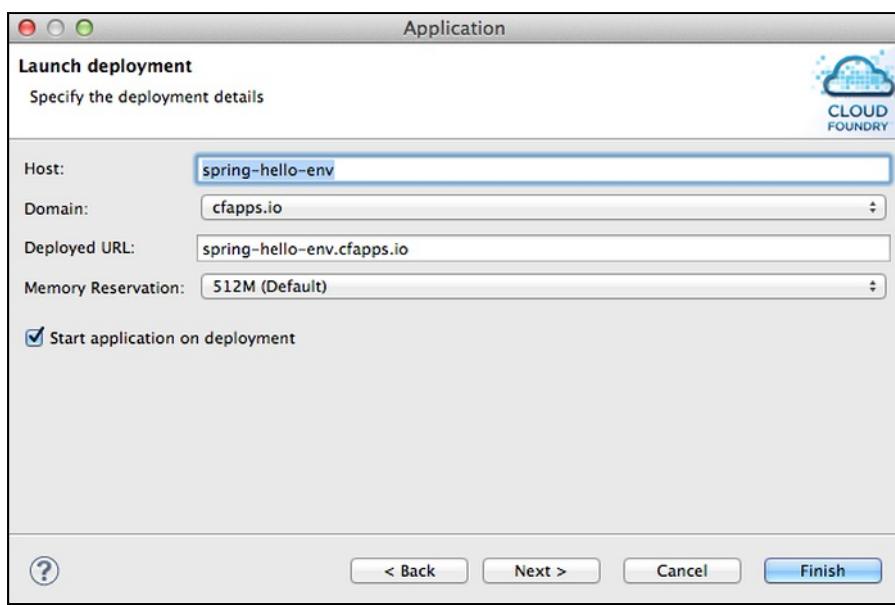
Domain — Contains the default domain. If you have mapped custom domains to the target space, they appear in the pull-down list.

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 `map domain` command to do so.

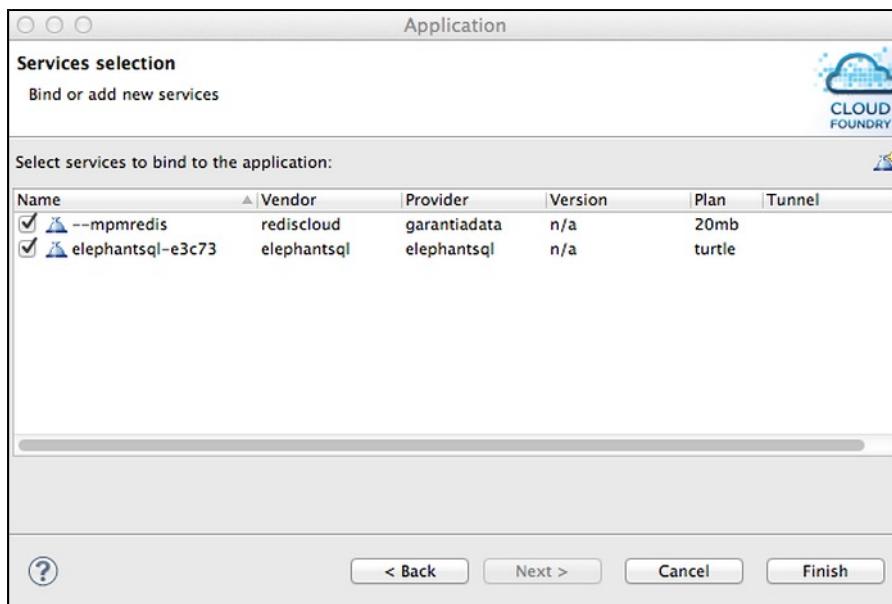
Deployed URL — By default, contains the value of the **Host** and **Domain** fields, separated by a period (.) character.

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](#).



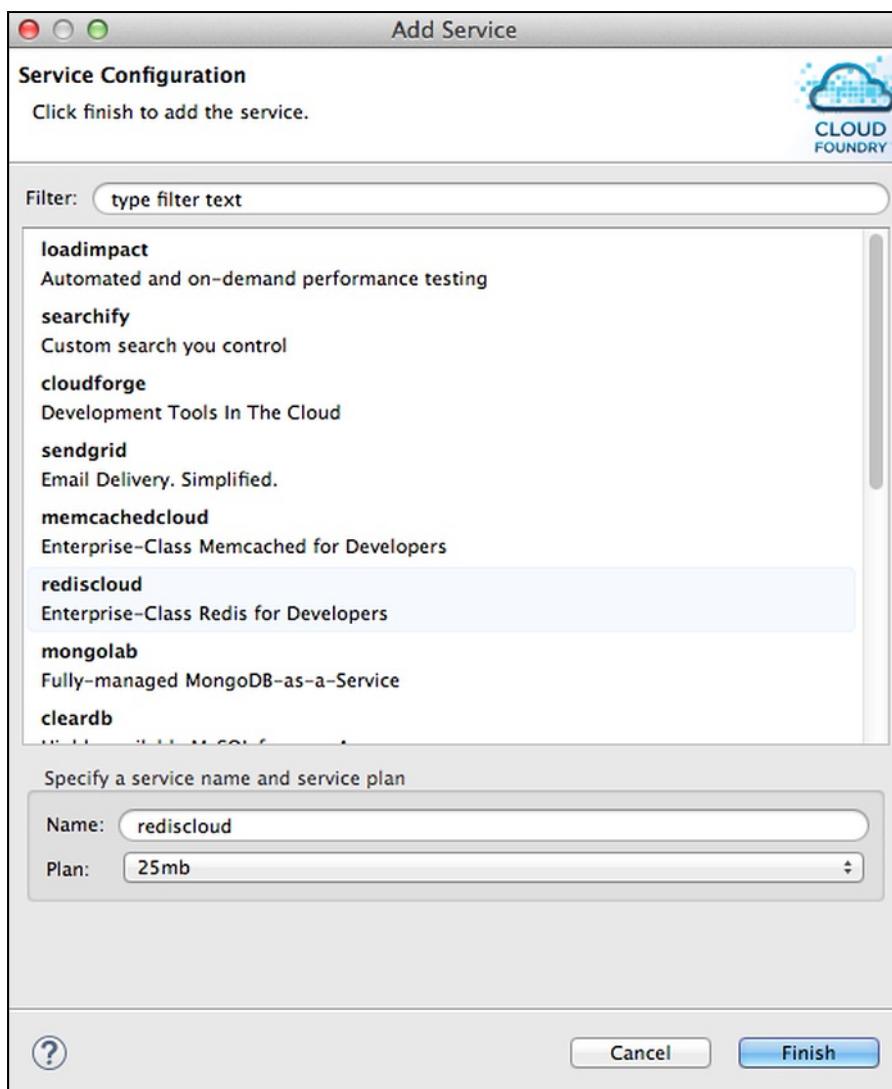
As the deployment proceeds, progress messages appear in the **Console** view. When deployment is complete, the application is listed in the **Applications** pane.

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

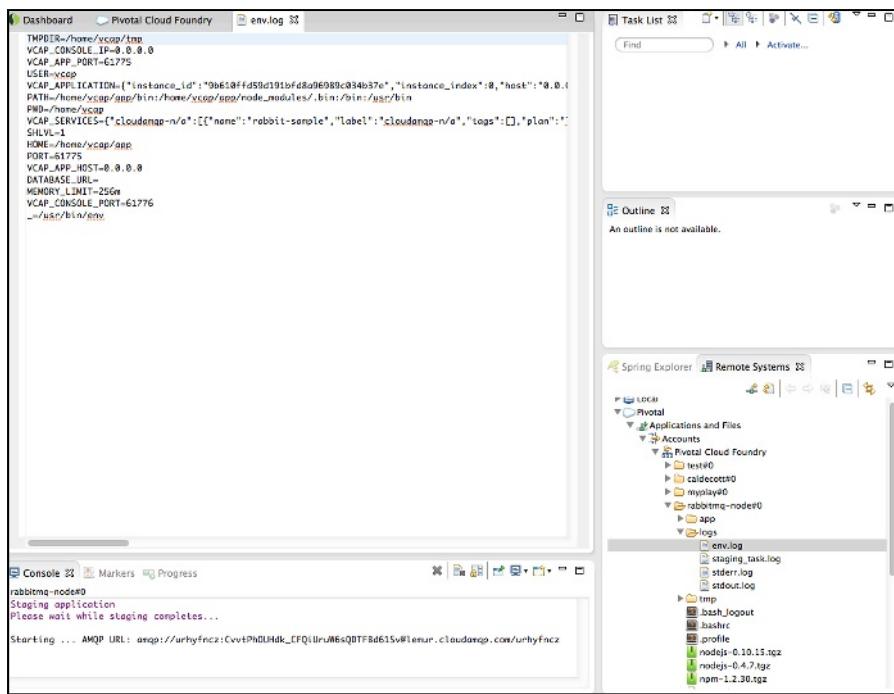
You can bind a service to an application when you deploy it. To bind a service to an application that is already deployed, drag the service from the **Services** pane to the **Application Services** pane. (See the area labelled “G” in the screenshot in the [Applications and Services](#) above.)

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 in 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](#).

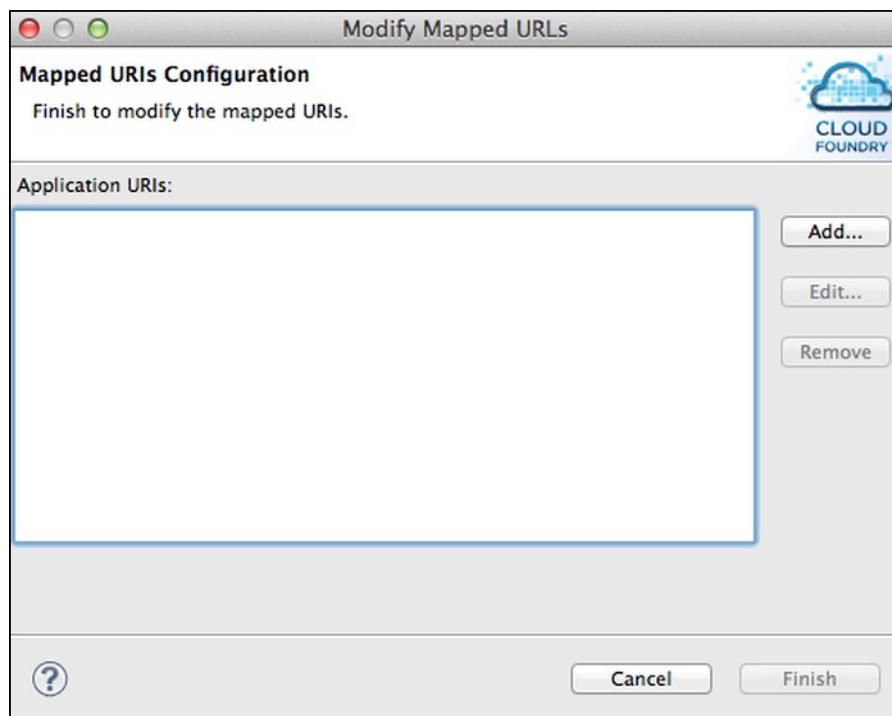
Push Application Changes

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 URLs 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.

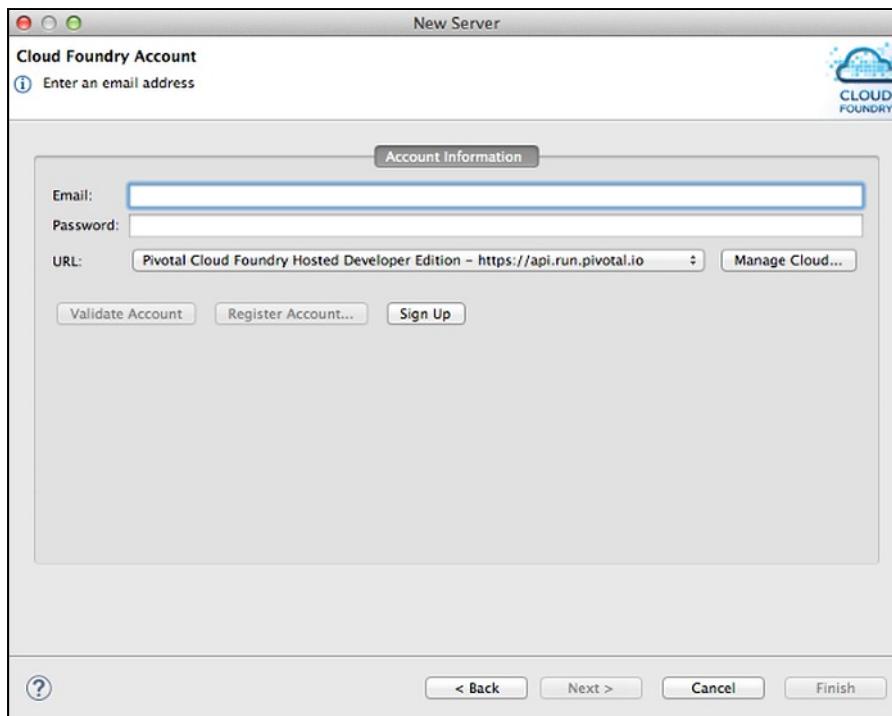
Follow the step below to clone a server:

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**.

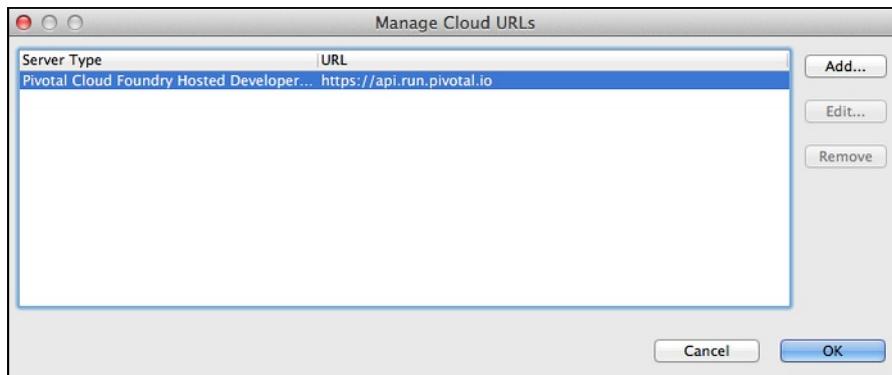
Add a Cloud Foundry Instance URL

You can configure the plugin to work with any Cloud Foundry instances to which you have access. To do so:

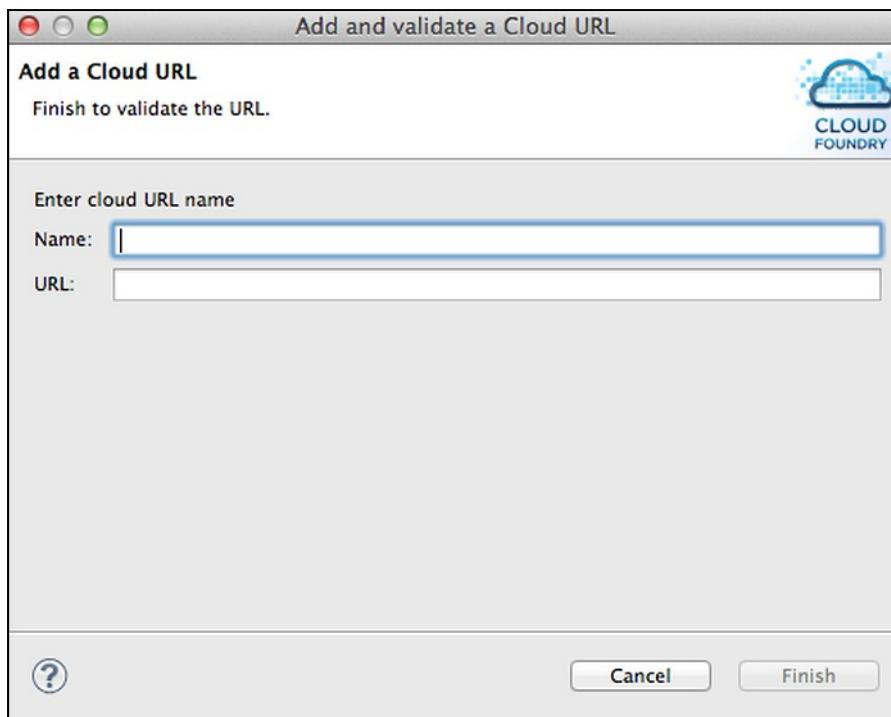
1. Perform steps 1 and 2 of [Create a Cloud Foundry Server](#).
2. In the **Cloud Foundry Account** window, enter the email account and password that you use to log on to the target instance, then click **Manage Cloud URLs**



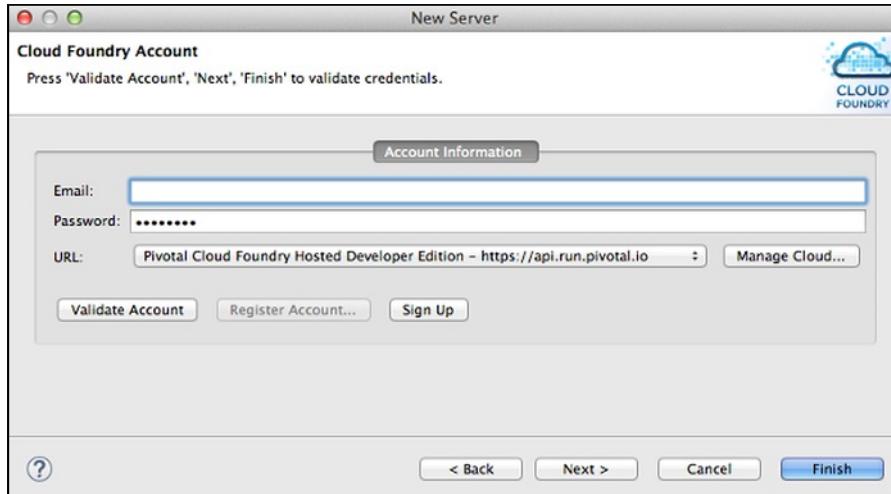
3. In the **Manage Cloud URLs** window, click **Add**.



4. In the **Add a Cloud URL** window, enter the name and URL of the target cloud instance and click **Finish**.

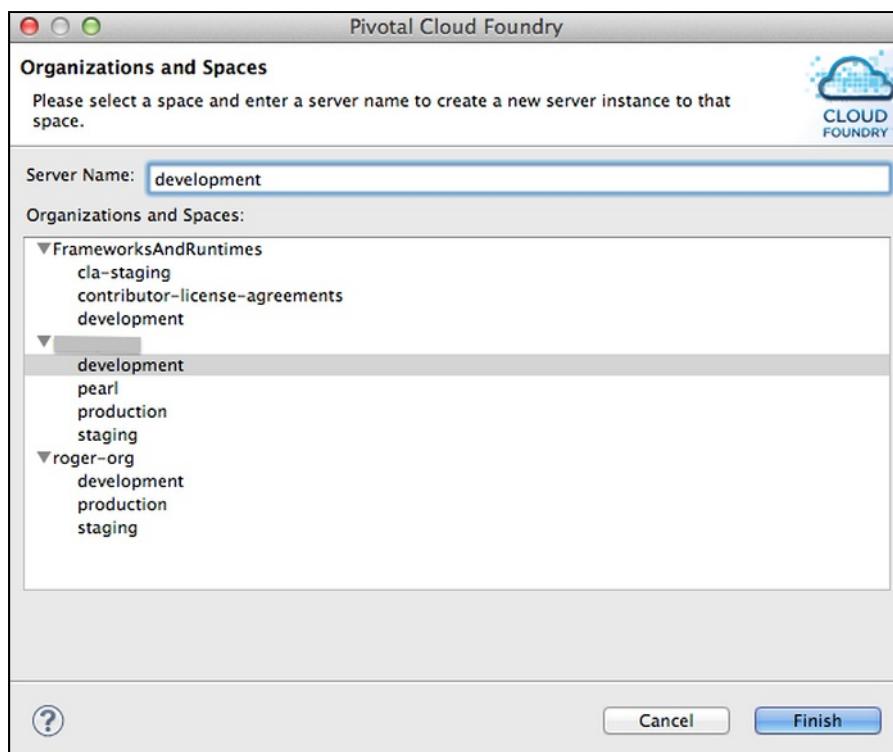


5. The new cloud instance should appear in the list on the **Manage Cloud URLs** window. Click **OK** to proceed.
6. In the **Cloud Foundry Account** window, click **Validate Account**.
7. The **Cloud Foundry Account** window is refreshed and displays a message indicating whether or not your credentials 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.

Getting the Library

The Cloud Foundry Java Client Library is available in Maven Central. The library can be added as a dependency to Maven or Gradle project using the following information.

Maven

The `cloudfoundry-client-lib` dependency can be added to your `pom.xml` as follows:

```
<dependencies>
  <dependency>
    <groupId>org.cloudfoundry</groupId>
    <artifactId>cloudfoundry-client-lib</artifactId>
    <version>1.1.2</version>
  </dependency>
</dependencies>
```

Gradle

To use the Java client library in a Gradle project, add the `cloudfoundry-client-lib` dependency to your `build.gradle` file:

```
repositories {
  mavenCentral()
}

dependencies {
  compile 'org.cloudfoundry:cloudfoundry-client-lib:1.1.2'
}
```

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%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%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 on the Cloud Foundry Java Client Library, view the [source](#) on GitHub. The [domain package](#) shows the objects that can be queried and inspected.

The source for the Cloud Foundry [Maven Plugin](#) is also a good example of using the Java client library.

Build Tool Integration

Page last updated:

This page assumes you are using cf CLI v6 and 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:

```
<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:

```
<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.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.

```
<settings>
  ...
  <servers>
    ...
    <server>
      <id>cloud-foundry-credentials</id>
      <username>my-name@email.com</username>
      <password>s3cr3t</password>
    </server>
    ...
  </servers>
  ...
</settings>
```

2. Add a server configuration element referencing the ID to the `pom.xml` file:

```
<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:

Maven Goal	Cloud Foundry Command	Syntax
cf:login	login -u USERNAME	\$ mvn cf:login
cf:logout	logout	\$ mvn cf:logout
cf:app	app APPNAME	\$ mvn cf:app [-Dcf.appname=APPNAME]
cf:apps	apps	\$ mvn cf:apps
cf:target	api	\$ mvn cf:target
cf:push	push	\$ mvn cf:push [-Dcf.appname=APPNAME] [-Dcf.path=PATH] [-Dcf.url=URL] [-Dcf.no-start=BOOLEAN]
cf:start	start APPNAME	\$ mvn cf:start [-Dcf.appname=APPNAME]
cf:stop	stop APPNAME	\$ mvn cf:stop [-Dcf.appname=APPNAME]
cf:restart	restart APPNAME	\$ mvn cf:stop [-Dcf.appname=APPNAME]
cf:delete	delete APPNAME	\$ mvn cf:delete [-Dcf.appname=APPNAME]
cf:scale	scale APPNAME -i INSTANCES	\$ mvn cf:scale [-Dcf.appname=APPNAME] [-Dcf.instances=INTEGER]
cf:env	env APPNAME	\$ mvn cf:env [-Dcf.appname=APPNAME]
cf:services	services	\$ mvn cf:services
cf:create-services	create-service SERVICE PLAN SERVICE_INSTANCE	\$ mvn cf:create-services
cf:delete-services	delete-service SERVICE_INSTANCE	\$ mvn cf:delete-service

cf:bind-services	bind-service APPNAME SERVICE_INSTANCE	\$ mvn cf:bind-services
cf:unbind-services	unbind-service APPNAME SERVICE_INSTANCE	\$ mvn cf:unbind-services

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.com"
    memory = 512
    instances = 1
    env = [ "key": "value" ]
    serviceInfos {
        "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
```

(Note there are no quotes around either the username or password.)

Command-Line Usage

Key functionality available with the Cloud Foundry Gradle plugin:

Gradle Task	Cloud Foundry Command	Syntax
cfLogin	login -u USERNAME	\$ gradle cfLogin
cfLogout	logout	\$ gradle cfLogout
cfApp	app APPNAME	\$ gradle cfApp [-PcfApplication=APPNAME]
cfApps	apps	\$ gradle cfApps
cfTarget	api	\$ gradle cfTarget
cfPush	push	\$ gradle cfPush [-PcfApplication=APPNAME] [-PcfUri=URL] [-PcfStartApp=BOOLEAN]
cfStart	start APPNAME	\$ gradle cfStart [-PcfApplication=APPNAME]
cfStop	stop APPNAME	\$ gradle cfStop [-PcfApplication=APPNAME]
cfRestart	restart APPNAME	\$ gradle cfStop [-PcfApplication=APPNAME]
cfDelete	delete APPNAME	\$ gradle cfDelete [-PcfApplication=APPNAME]
cfScale	scale APPNAME -i INSTANCES	\$ gradle cfScale [-PcfApplication=APPNAME] [-PcfInstances=INTEGER]
cfEnv	env APPNAME	\$ gradle cfEnv [-PcfApplication=APPNAME]
cfServices	services	\$ gradle cfServices
cfCreateService	create-service SERVICE PLAN SERVICE_INSTANCE	\$ gradle cfCreateServices
cfDeleteServices	delete-service SERVICE_INSTANCE	\$ gradle cfDeleteServices
cfBind	bind-service APPNAME SERVICE_INSTANCE	\$ gradle cfBind
cfUnbind	unbind-service APPNAME SERVICE_INSTANCE	\$ gradle cfUnbind

Ruby Buildpack

Page last updated:

Use the Ruby buildpack with Ruby, Rack, Rails or Sinatra applications.

See the following topics:

- [Getting Started Deploying Ruby Apps](#)
- [Getting Started Deploying Ruby on Rails Apps](#)
- [Deploy a Sample Ruby on Rails App](#)
- [Configure a Production Server for Ruby Apps](#)
- [Configure Rake Tasks for Deployed Apps](#)
- [Tips for Ruby Developers](#)
- [Environment Variables Defined by the Ruby Buildpack](#)
- [Configure Service Connections for Ruby](#)

The source for the buildpack is [here ↗](#).

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. Application logging is a separate concern.

If you are deploying a Ruby, Rack, or Sinatra application, your application must write to STDOUT or STDERR to include its logs in the Loggregator stream. See [Application Logging in Cloud Foundry](#).

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.
- Rails 3.x: The buildpack automatically installs the `rails_12factor` gem if it is not present and issues a warning message.
- 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_log_stdout` plugin, refer to the [Github README ↗](#).

For more information about the `rails_12factor` gem, refer to the [Github README ↗](#).

Getting Started Deploying Ruby Apps

Page last updated:

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 [cf Command Line Interface \(CLI\)](#) 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](#) 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 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 cloudunder@example.com...
OK

service    plans  description
...
cleardb    spark, boost, amp, shock  Highly available MySQL for your Apps
...
elephantsql  turtle, panda, hippo, elephant  PostgreSQL as a Service
...
mongolab    sandbox  Fully-managed MongoDB-as-a-Service
...
...
```

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`. This creates a service instance named `redis` that uses the `rediscloud` service and the `30mb` plan, as the example below shows.

```
$ cf create-service rediscloud 30mb redis
Creating service redis in org Cloud-Apps / space development as cloudunder@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.

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 [Configure 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.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.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 pongmatch-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.

```
$ cf push pong_matcher_ruby -n pongmatch-ex12
Using manifest file /Users/clouduser/workspace/pong_matcher_ruby/manifest.yml

Creating app pong_matcher_ruby in org Cloud-Apps / space development as clouduser@example.com...
OK

Creating route pongmatch-ex12.shared-domain.com
  Binding pongmatch-ex12.shared-domain.com to pong_matcher_ruby...
OK

Uploading pong_matcher_ruby...
Uploading app files from: /Users/clouduser/workspace/pong_matcher_ruby
Uploading 8.8K, 12 files
OK
Binding service redis to app pong_matcher_ruby in org Cloud-Apps / space development as clouduser@example.com...
OK

Starting app pong_matcher_ruby in org cf-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 pong_matcher_ruby in org Cloud-Apps / space development as clouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: pongmatch-ex12.cfapps.io

  state  since        cpu   memory      disk
#0  running  2014-12-09 10:04:40 AM  0.0%  35.2M of 256M  45.8M of 1G
```

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 cf Command Line Interface (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 [Understanding Apps Manager Permissions](#) 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

Page last updated:

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 [cf Command Line Interface \(CLI\)](#)

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 [Configure a Production Server](#) topic.

[Production Server](#) topic for help configuring a server other than WEBrick.

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.com`. For example: `cf push my-app` creates the URL `my-app.shared-domain.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.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 `cf` Command Line Interface (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

Page last updated:

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.

```
$ 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.

```
$ cf login -a YOUR-API-ENDPOINT
```

2. Use your credentials to log in, and to select a [Space and Org](#).

 **Note:** The API endpoint must be entered in the following format: `https://api.IP-ADDRESS`.

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.

```
$ cf create-service elephantsql turtle rails-postgres
Creating service rails-postgres in org YOUR-ORG / space development as clouduser@example.com...
OK
```

The manifest for the rails_sample_app contains a `services` sub-block in the `applications` block. The cf Command Line Interface tool (cf CLI) binds the service to the app.

```
...
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.com`, running `cf push rails_sample_app` creates the URL `rails_sample_app.shared-domain.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 cloudunder@example.com...
OK

Using route rails_sample_app.shared-domain.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 cloudunder@example.com...
OK

Starting app rails_sample_app in org Cloud-Apps / space development as cloudunder@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 cloudunder@example.com...
OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: rails_sample_app.shared-domain.com

  state  since          cpu  memory      disk
#0  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: Verifying 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.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 a Production Server for Ruby Apps

Page last updated:

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.

To instruct Elastic Runtime to use a web server other than WEBrick, you configure a text file called a *Procfile*. 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.

When you deploy, Elastic Runtime determines if a Procfile exists and uses the Procfile to configure your app with the specified production web server settings.

In a Procfile, you declare one process type per line and use the following syntax, as shown in the example below:

- `PROCESS_TYPE` is the command name in the format of an alphanumeric string. Specifically for RoR web apps, you can declare `web` and `worker` process types.
- `COMMAND` is the command line to launch the process.

Example process type syntax:

```
PROCESS_TYPE: COMMAND
```

To set a different production web server for your app:

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.

Example Puma config file:

```
# 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. The example below 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. Refer to your web server documentation for how to configure the specific command for a process type.

Example Procfile:

```
web: bundle exec puma -e $RAILS_ENV -p 1234 -S ~/puma -C config/puma.rb
```

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:

```
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.

```
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 assumes you are using cf CLI v6.

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:

```
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`:

```
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`.

```
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 Rake task to limit an idempotent command to the first instance of a deployed application:

```
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
```

2. Add the task to the `manifest.yml` file, referencing the idempotent command `rake db:migrate` with the `command` attribute.

```
---
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`.

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:

Library	Description
Delayed::Job	A direct extraction from Shopify where the job table is responsible for a multitude of core tasks.
Resque	A Redis-backed library for creating background jobs, placing those jobs on multiple queues, and processing them later.
Sidekiq	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.

For other alternatives, see 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
```

```
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
```

```
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
```

```
nil
```

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`:

```
---  
applications:  
- name: sidekiq  
  memory: 256M  
  instances: 1  
  host: sidekiq  
  domain: ${target-base}  
  path: .  
  services:  
  - sidekiq-mysql:  
  - sidekiq-redis:
```

Worker Manifest: Save this as `worker-manifest.yml`:

```
---  
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 -i 2
```

Use `rails_serve_static_assets` on Rails 4

By default Rails 4 returns a 404 if an asset is not handled via an external proxy such as Nginx. The `rails_serve_static_assets` 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 `rails_serve_static_assets` gem enables this behavior by setting the `config.serve_static_assets` option to `true`, so you do not need to configure it manually.

Additional Ruby Buildpack Information

For 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.

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>.

```
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://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 -rbundler/setup
```

Environment Variables Defined by the Ruby Buildpack

Page last updated:

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. Useful for Rails applications, where “assets” and “development” gem groups typically contain gems that are not needed when the app runs in production.
See [this](#) blog post for more information.
`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://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 -rbundler/setup`

Configure Service Connections for Ruby

Page last updated:

This page assumes that you are using cf CLI v6.

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-app-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/exampledbs"  
      }  
    }  
  ]  
}
```

Based on this `VCAP_SERVICES`, the Ruby buildpack creates the following `DATABASE_URL` environment variable:

```
DATABASE_URL = postgres://exampleuser:examplepass@babar.elephantsql.com:5432/exampledbs
```

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.

Configuring Non-Rails Applications

Non-Rails applications can also access the `DATABASE_URL` variable.

If you have more than one service with credentials, only the first will be populated into `DATABASE_URL`. To access other credentials, you can inspect the `VCAP_SERVICES` environment variable.

```
vcap_services = JSON.parse(ENV['VCAP_SERVICES'])
```

Use the hash key for the service to obtain the connection credentials from `VCAP_SERVICES`.

- For services that use the [v2 format](#), the hash key is the name of the service.
- For services that use the [v1 format](#), the hash key is formed by combining the service provider and version, in the format PROVIDER-VERSION.

For example, the service provider “p-mysql” with version “n/a” forms the hash key `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 [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 `cf env` command to view the Cloud Foundry environment variables for your application. `cf env` displays the following environment variables:

- The `VCAP_SERVICES` variables existing in the container environment
- The user-provided variables set using the `cf set-env` command

```
$ 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://lraa: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

Use the `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 `cf logs APPNAME` to direct current logging to standard output:

```
$ cf logs my-app
Connected, tailing logs for app my-app in org My-Org / space development as admin...
1:27:19.72 [App/0] OUT [CONTAINER] org.apache.coyote.http11.Http11Protocol INFO Starting ProtocolHandler ["http-bio-61013"]
1:27:19.77 [App/0] OUT [CONTAINER] org.apache.catalina.startup.Catalina INFO Server startup in 10427 ms
```

Run `cf logs APPNAME --recent` to dump the most recent logs to standard output:

```
$ cf logs my-app --recent
Connected, dumping recent logs for app my-app in org My-Org / space development as admin...
1:27:15.93 [App/0] OUT 15,935 INFO EmbeddedDatabaseFactory:124 - Creating embedded database 'SkyNet'
1:27:16.31 [App/0] OUT 16,318 INFO LocalEntityManagerFactory:287 - Building TM container EntityManagerFactory for unit 'default'
1:27:16.50 [App/0] OUT 16,505 INFO Version:37 - HCANN001: Hibernate Commons Annotations {4.0.1.Final}
1:27:16.51 [App/0] OUT 16,517 INFO Version:41 - HHH412: Hibernate Core {4.1.9.Final}
1:27:16.95 [App/0] OUT 16,957 INFO SkyNet-Online:73 - HHH268: Transaction strategy: org.hibernate.internal.TransactionFactory
1:27:16.96 [App/0] OUT 16,963 INFO InitiateTerminatorT1000Deployment:48 - HHH000397: Using TranslatorFactory
1:27:17.02 [App/0] OUT 17,026 INFO Version:27 - HV001: Hibernate Validator 4.3.0.Final
```

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:

Use the Node.js buildpack with Node or JavaScript applications.

See the following topics:

- [Tips for Node.js Developers](#)
- [Environment Variables Defined by the Node Buildpack](#)
- [Configure Service Connections for Node](#)

You can find the source for the buildpack on GitHub: <https://github.com/cloudfoundry/heroku-buildpack-nodejs> ↗

Buildpack Logging and Application Logging

The buildpack only runs during the staging process, and therefore 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.

Once staging is done, the buildpack stops logging. Application logging is a separate concern.

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.

Tips for Node.js Applications

Page last updated:

This page assumes you are using cf CLI v6.

This topic provides Node-specific information to supplement the general guidelines in the [Deploy an Application](#) topic.

Application Package File

Cloud Foundry expects a `package.json` in your Node.js application. You can specify the version of Node.js you want to use in the `engine` node of your `package.json` file. As of July, 2015, and buildpack version 1.5.0, Cloud Foundry uses Node.js version 0.12.7 by default. See the GitHub [Node.js buildpack page](#) for current information.

Example `package.json` file:

```
{  
  "name": "first",  
  "version": "0.0.1",  
  "author": "Demo",  
  "dependencies": {  
    "express": "3.4.8",  
    "consolidate": "0.10.0",  
    "express": "3.4.8",  
    "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 application should listen on. In order to also run your application locally, you may want to make port 3000 the default:

```
app.listen(process.env.PORT || 3000);
```

Application Start Command

Node.js applications require a start command. You can specify a Node.js application's web start command in a Procfile or in the application deployment manifest.

You will be asked if you want to save your configuration the first time you deploy. This will save a `manifest.yml` in your application with the settings you entered during the initial push. Edit the `manifest.yml` file and create a start command as follows:

```
---  
applications:  
- name: my-app  
  command: node my-app.js  
  ... the rest of your settings ...
```

Alternately, specify the start command with `cf push -c`.

```
$ cf push my-app -c "node my-app.js"
```

Application Bundling

You do not need to run `npm install` before deploying your application. Cloud Foundry will run it for you when your application is pushed. If you would prefer to run `npm install` and create a `node_modules` folder inside of your application, this is also supported.

Solving Discovery Problems

If Cloud Foundry does not automatically detect that your application is a Node.js application, 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:

```
---  
applications:  
- 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
```

Binding 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 of `0.12.7`. To specify the versions of Node.js and npm an application requires, edit the application'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 application 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:/usr/bin
```

Environment Variables Defined by the Node Buildpack

Page last updated:

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
```

Configure Service Connections for Node.js

Page last updated:

This page assumes that you are using cf CLI v6.

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>

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 PostgreSQL, 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](#) module
- Mongo via the [mongodb](#) and [mongoose](#) modules
- MySQL via the [mysql](#) module
- Postgres via the [pg](#) module
- Redis via the [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:

```
{  
  "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.

For more information, see the [buildpack documentation](#).

Usage

Unlike most other Cloud Foundry buildpacks, you must specify the binary buildpack in order to use it in 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:

```
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(":"+os.Getenv("PORT"), nil)
}
```

Your binary should run without any additional runtime dependencies on the cflinuxfs2 or lucid64 root filesystem (rootfs). 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 rootfs, 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](#).

Go Buildpack

Page last updated:

Pushing Apps

The Go buildpack will be automatically detected if your app has been packaged with [godep](#) using `godep save`. It will also be automatically detected if your app has a `vendor/` directory and your app has any files ending with `.go`.

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, we recommend only specifying 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
```

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`:

```
{
  "ImportPath": "go_app",
  "GoVersion": "go1.6",
  "Deps": []
}
```

An example `manifest.yml`:

```
---  
applications:  
- name: my-app-name  
env:  
  GO15VENDOREXPERIMENT: 0
```

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` :

If you are using the `vendor/` directory for dependencies, you can set the Go version with the `GOVERSION` environment variable.

Go 1.6

- [sample app ↗](#).

An example `manifest.yml` :

```
---  
applications:  
- name: my-app-name  
  command: example-project  
  env:  
    GOVERSION: go1.6  
    GOPACKAGENAME: github.com/example-org/example-project
```

Passing a symbol and string to the linker

This buildpack supports the go [linker's][[https://golang.org/cmd/ld/ ↗](https://golang.org/cmd/ld/)] 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.

Disconnected environments

To use this buildpack on Cloud Foundry, where the Cloud Foundry instance limits some or all internet activity, please read the [Disconnected Environments documentation ↗](#).

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 ↗](#).

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.

See the following topics:

- [Composer](#)
- [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>

Buildpack Logging and Application Logging

The buildpack only runs during the staging process, and therefore 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.

Once staging is done, the buildpack stops logging. Application logging is a separate concern.

Your application must write to STDOUT or STDERR for its logs to be included in the Loggregator stream. See [Application Logging in Cloud Foundry](#).

Composer

Page last updated:

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:

Example	Expected Version
5.3.*	latest 5.4.x release (5.3 is not supported)
>=5.3	latest 5.4.x release (5.3 is not supported)
5.4.*	latest 5.4.x release
>=5.4	latest 5.4.x release
5.5.*	latest 5.5.x release
>=5.5	latest 5.5.x release
5.4.x	specific 5.4.x release that is listed
5.5.x	specific 5.5.x release that is listed

Configuration

The buildpack runs with a set of default values for Composer. You can adjust these values by adding a `.bp-config/options.json` file to your application and setting any of the following values in it.

Variable	Explanation
COMPOSER_VERSION	The version of Composer to use. It defaults to the latest bundled with the buildpack.
COMPOSER_INSTALL_OPTIONS	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.
COMPOSER_VENDOR_DIR	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> .
COMPOSER_BIN_DIR	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> .
COMPOSER_CACHE_DIR	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.

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.

Vendor 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 `PHP_RCI` must be different for staging and runtime. The buildpack takes care of setting these variables, which means user values for these variables are ignored.

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.

Variable	Explanation
WEB_SERVER	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> .
HTTPD_VERSION	Sets the version of Apache HTTPD to use. Currently the build pack supports the latest stable version. This value will default to the latest release that is supported by the build pack.
ADMIN_EMAIL	The value used in HTTPD's configuration for ServerAdmin
NGINX_VERSION	Sets the version of Nginx to use. By default, the buildpack uses the latest stable version.
PHP_VERSION	Sets the version of PHP to use.
PHP_EXTENSIONS	A list of the extensions to enable. <code>bz2</code> , <code>zlib</code> , <code>curl</code> , and <code>mcrypt</code> are enabled by default.
PHP_MODULES	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>cgi</code> , and <code>pear</code> .
ZEND_EXTENSIONS	A list of the Zend extensions to enable. Nothing is enabled by default.
APP_START_CMD	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.
WEBDIR	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.
LIBDIR	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.
HTTP_PROXY	The buildpack downloads uncached dependencies using HTTP. If you are using a proxy for HTTP access, set its URL here.
HTTPS_PROXY	The buildpack downloads uncached dependencies using HTTPS. If you are using a proxy for HTTPS access, set its URL here.
ADDITIONAL_PREPROCESS_CMDS	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.

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](#)

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](#) 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.

PHP Modules

The following modules can be included by adding it to the `PHP_MODULES` list: - `cli`, installs `php` and `phar` - `fpm`, installs `PHP-FPM` - `cgi`, installs `php-cgi` - `pear`, 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](#), [Nginx](#), [PHP](#), and [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.

The [Development Documentation](#) explains how to write extensions.

Deploying and Developing PHP Apps

Page last updated:

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 [cf Command Line Interface \(CLI\)](#) 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 Feb 27 21:40 images
-rw-r--r-- 1 daniel staff 0B Feb 27 21:39 index.php
drwxr-xr-x 3 daniel staff 102B Feb 27 21:40 lib

./lib:
total 0
-rw-r--r-- 1 daniel staff 0B 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 Feb 27 21:48 htdocs

./htdocs: <-- create the htdocs directory and put your files there
total 0
-rw-r--r-- 1 daniel staff 0B Feb 27 21:40 images
-rw-r--r-- 1 daniel staff 0B Feb 27 21:39 index.php
drwxr-xr-x 3 daniel staff 102B 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 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 Feb 27 21:48 htdocs
drwxr-xr-x 3 daniel staff 102B Feb 27 21:51 lib

./htdocs:
total 0
-rw-r--r-- 1 daniel staff 0B Feb 27 21:40 images
-rw-r--r-- 1 daniel staff 0B Feb 27 21:39 index.php
drwxr-xr-x 3 daniel staff 102B Feb 27 21:48 lib

./htdocs/lib: <-- public lib directory
total 0
-rw-r--r-- 1 daniel staff 0B Feb 27 21:48 controller.php

./lib: <-- protected lib directory
total 0
-rw-r--r-- 1 daniel staff 0B 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`.

(i.e. become public).

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 `cf files` command, as the following example shows:

```
$ cf files APP app/.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](#).

You can find current information about this buildpack on the PHP buildpack [release page](#) 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:

About the Python Buildpack

For information about using and extending the Python buildpack in Cloud Foundry, see the [python-buildpack Github repo](#).

You can find current information about this buildpack on the Python buildpack [release page](#) in GitHub.

This buildpack uses a default Python version of `2.7.10`. To change the default version, specify the `version` key in your `runtime.txt` file.

Staticfile Buildpack

Page last updated:

Use the Staticfile buildpack with front-end only web apps or demos.

You only need to create a `Staticfile` file for Cloud Foundry to detect this buildpack:

```
$ touch Staticfile
$ cf push my-site -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

Alternate root folder

By default, the buildpack will serve `index.html` and all other assets from the root folder of your project.

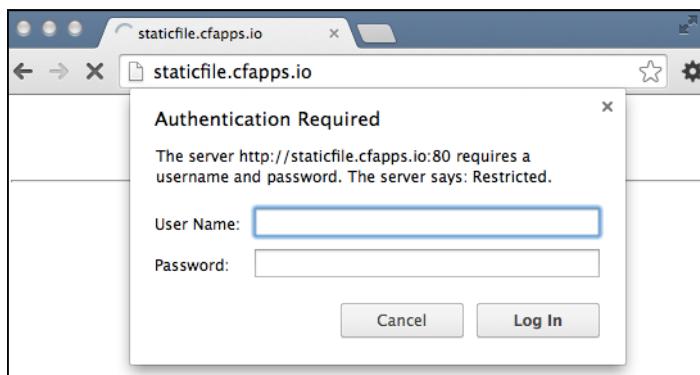
In many cases, you may have an alternate folder where your HTML/CSS/JavaScript files are to be served from, such as `dist/` or `public/`.

To configure the buildpack add the following line to your `Staticfile`:

```
root: dist
```

Basic authentication

Protect your website with a user/password configured via environment variables.



Convert the username / password to the required format: <http://www.htaccesstools.com/htpasswd-generator/>

For example, username `bob` and password `bob` becomes `bob:$apr1$DuUQEQt8$ZccZCHQEINSjrg.ewrSF0`.

Create a file in the root of your application `Staticfile.auth`. This becomes the `.htpasswd` file for nginx to protect your site. It can include one or more user/password lines.

```
bob:$apr1$DuUQEQt8$ZccZCHQEINSjrg.ewrSF0
```

Push your application to apply changes to basic auth. Remove the file and push to disable basic auth.

Directory Index

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.

Index of /

```
../  
subdir/  
find-me-too.html  
find-me.html
```

Add a line to your `Staticfile` that begins with `directory:`

```
directory: visible
```

Server Side Includes (SSI)

Add the following line to your `Staticfile` to enable support for [SSI](#):

```
ssi: enabled
```

GZip Static

The `gzip_static` and `gunzip` modules are set to `on` by default. Together, these modules enable Nginx to properly serve compressed files to clients. They also decompress files for clients that do not support compressed content or responses.

If you want to disable these modules, specify the following in your custom `nginx.conf`:

```
gunzip off;  
gzip_static off;
```

Advanced Nginx configuration

You can customize the Nginx configuration further, by adding `nginx.conf` and/or `mime.types` to your root folder.

If the buildpack detects either of these files, they will be used in place of the built-in versions.

For 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.

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](#)
- [Supporting Multiple Cloud Foundry Instances](#)

Overview

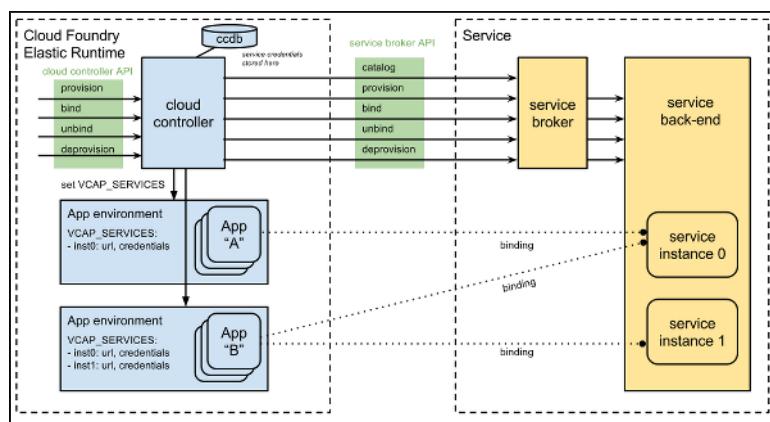
Page last updated:

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 v2.7

Page last updated:

Document Changelog

[v2 API Change Log ↗](#)

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 v2.6

1. [Asynchronous operations](#) for [provisioning](#), [updating](#), and [deprovisioning](#) are no longer experimental.
2. Added support for querying `last_operation` status at new [endpoint](#) `GET /v2/service_instances/:guid/last_operation`

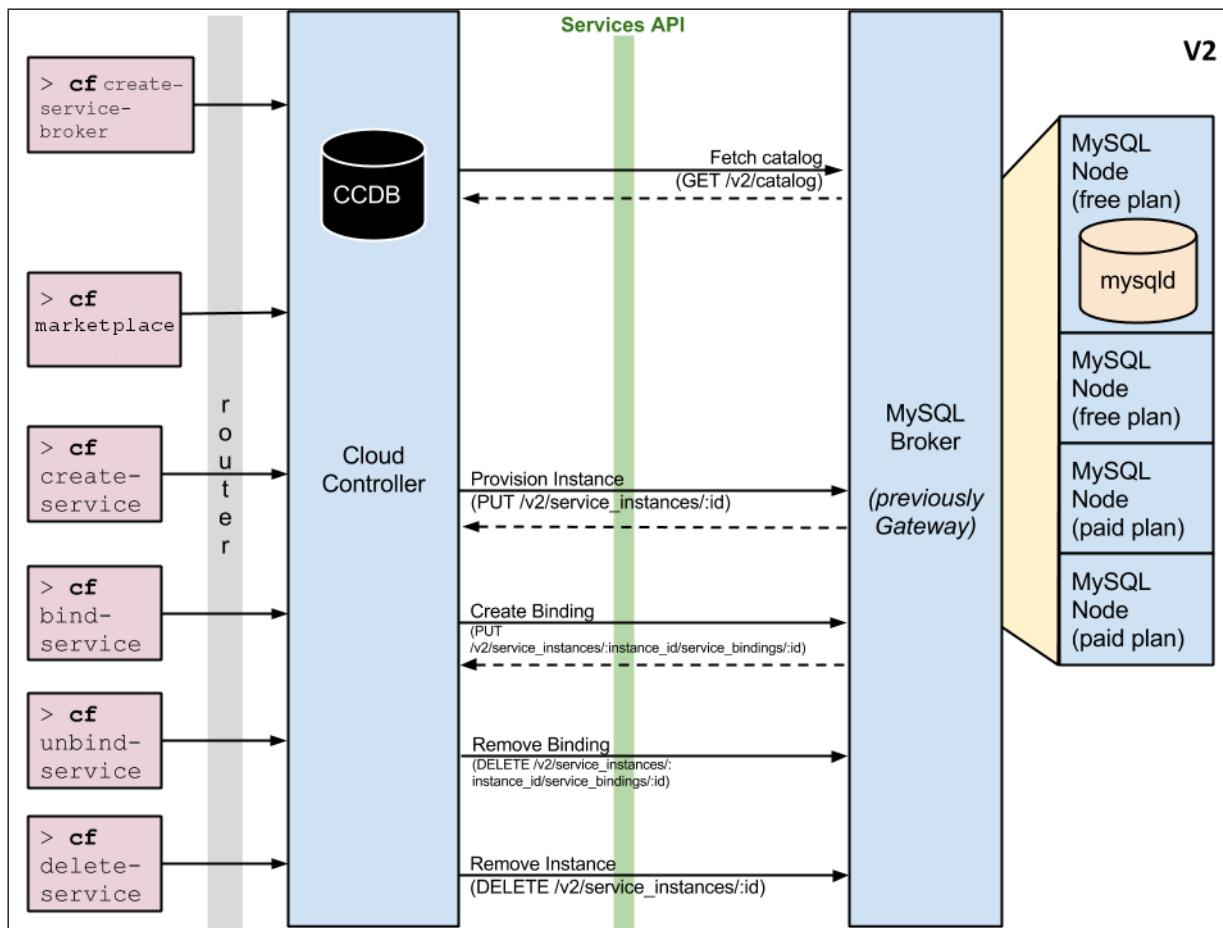
Dependencies

v2.7 of the services API has been supported since:

- [Final build 220 ↗](#) of [cf-release ↗](#)
- v2.37.0 of the Cloud Controller API
- CLI [v6.12.1 ↗](#)

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.



API Version Header

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:  
2.7
```

Authentication

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.

Catalog Management

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 plan 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: 2.7" http://username:password@broker-url/v2/catalog
```

Response

Status Code	Description
200 OK	The expected response body is below.

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.

Response field	Type	Description
services*	array-of-objects	Schema of service objects defined below:
id*	string	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.
name*	string	The CLI-friendly name of the service that will appear in the catalog. All lowercase, no spaces.
description*	string	A short description of the service that will appear in the catalog.
bindable*	boolean	Whether the service can be bound to applications.
tags	array-of-strings	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.
metadata	object	A list of metadata for a service offering. For more information, see Service Metadata .
requires	array-of-strings	A list of permissions that the user would have to give the service, if they provision it. The only permission currently supported is <code>syslog_drain</code> ; for more info see Application Log Streaming .
plan_updateable	boolean	Whether the service supports upgrade/downgrade for some plans. Please note that the misspelling of the attribute <code>plan_updatable</code> to <code>plan_updateable</code> was done by mistake. We have opted to keep that misspelling instead of fixing it and thus breaking backward compatibility.
plans*	array-of-objects	A list of plans for this service, schema defined below:

Response field	Type	Description
	string	An identifier used to correlate this plan in future requests to the catalog. This must be unique within Cloud Foundry, using a GUID is recommended.
name*	string	The CLI-friendly name of the plan that will appear in the catalog. All lowercase, no spaces.
description*	string	A short description of the service that will appear in the catalog.
metadata	object	A list of metadata for a service plan. For more information, see Service Metadata .
free	boolean	This field allows the plan to be limited by the non_basic_services_allowed field in a Cloud Foundry Quota, see Quota Plans . Default: true
dashboard_client	object	Contains the data necessary to activate the Dashboard SSO feature for this service
id	string	The id of the OAuth2 client that the service intends to use. The name may be taken, in which case the API will return an error to the operator
secret	string	A secret for the dashboard client
redirect_uri	string	A domain for the service dashboard that will be whitelisted by the UAA to enable SSO

* Fields with an asterisk are required.

```
{
  "services": [
    {
      "id": "service-guid-here",
      "name": "mysql",
      "description": "A MySQL-compatible relational database",
      "bindable": true,
      "plans": [
        {
          "id": "plan1-guid-here",
          "name": "small",
          "description": "A small shared database with 100mb storage quota and 10 connections"
        },
        {
          "id": "plan2-guid-here",
          "name": "large",
          "description": "A large dedicated database with 10GB storage quota, 512MB of RAM, and 100 connections",
          "free": false
        }
      ],
      "dashboard_client": {
        "id": "client-id-1",
        "secret": "secret-1",
        "redirect_uri": "https://dashboard.service.com"
      }
    }
  ]
}
```

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 `201 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 `accepts_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

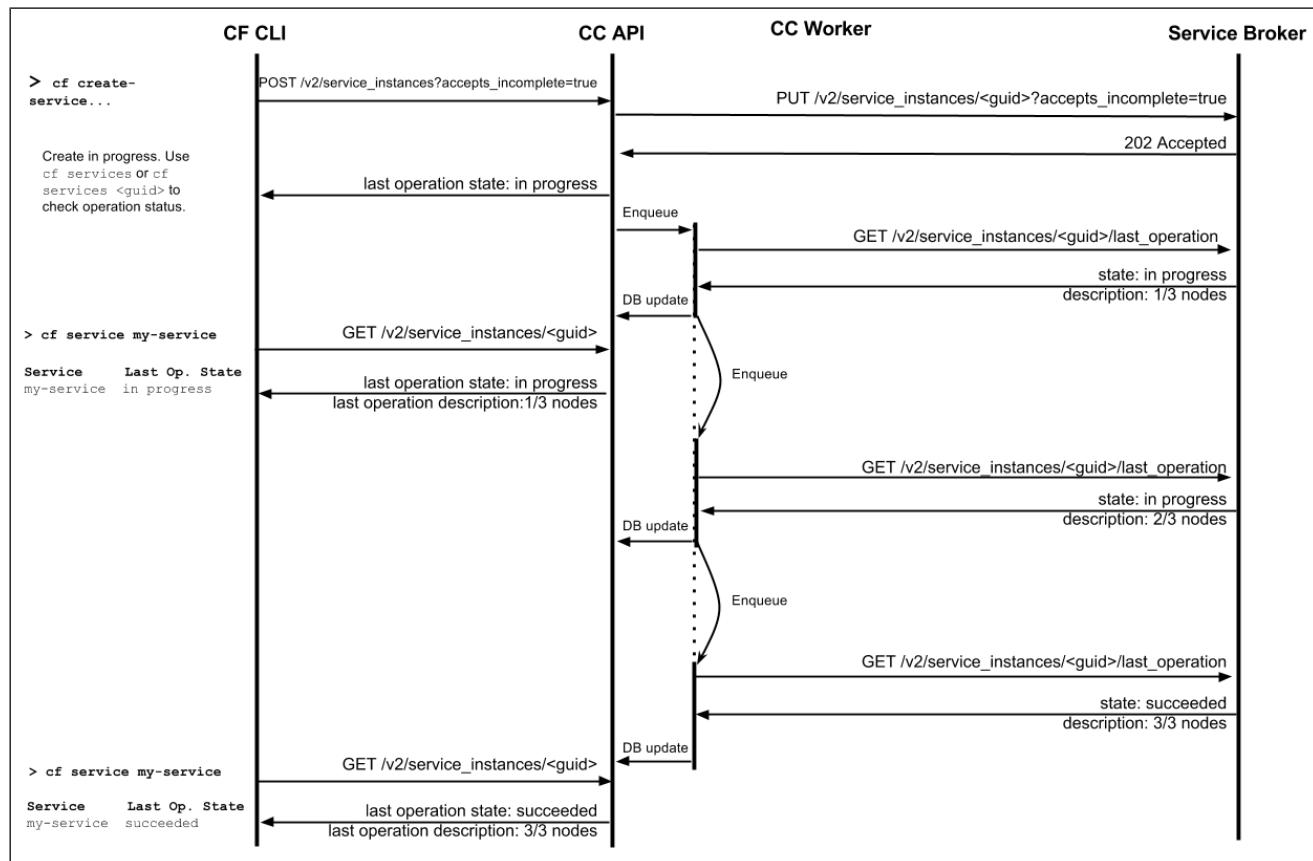
feature is considered stable.

If the `accepts_incomplete=true` parameter is not included, and the broker can not 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 can not guarantee completion of the operation with the response should implement support for asynchronous provisioning. Support for asynchronous or synchronous 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

Status Code	Description
200 OK	The expected response body is below.
410 GONE	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 <code>{}</code> . 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.

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.

Response field	Type	Description
state*	string	Valid values are <code>in progress</code> , <code>succeeded</code> , and <code>failed</code> . While <code>"state": "in progress"</code> , Cloud Foundry will continue polling. A response with <code>"state": "succeeded"</code> or <code>"state": "failed"</code> will cause Cloud Foundry to cease polling.
description	string	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.

```
{
  "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

Request field	Type	Description
organization_guid*	string	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.
plan_id*	string	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.
service_id*	string	The ID of the service within the catalog above. While not strictly necessary, some brokers might make use of this ID.

Request field	Type	Description
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.

```
{
  "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-guid-here",
  "parameters": {
    "parameter1": 1,
    "parameter2": "value"
  }
}' -X PUT -H "X-Broker-API-Version: 2.7" -H "Content-Type: application/json"
```

In this case, `:instance_id` refers to the service instance id generated by Cloud Controller

Response

Status Code	Description
201 Created	Service instance has been created. The expected response body is below.
200 OK	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.
202 Accepted	Service instance creation is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.
409 Conflict	Should be returned if the requested service instance already exists. The expected response body is <code>{}</code> .
422 Unprocessable Entity	Should be returned if the broker only supports asynchronous provisioning for the requested plan and the request did not include <code>?accepts_incomplete=true</code> . The expected response body is: <code>{ "error": "AsyncRequired", "description": "This service plan requires client support for asynchronous service operations." }</code> , as described below.

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 ([8](#)). 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](#).

Response field	Type	Description
		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 ("9189kdfsk0vfnku" in the

Response field	String Type	example below). For information on how users can authenticate with service dashboards via SSO, see Dashboard Single Sign-On .
----------------	-------------	---

```
{
  "dashboard_url": "http://example-dashboard.com/9189kdfsk0vfnku"
}
```

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

Request Field	Type	Description
service_id*	string	The ID of the service within the catalog above. While not strictly necessary, some brokers might make use of this ID.
plan_id	string	ID of the new plan from the catalog.
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.
previous_values	object	Information about the instance prior to the update.
previous_values.plan_id	string	ID of the plan prior to the update.
previous_values.service_id	string	ID of the service for the instance.
previous_values.organization_id	string	ID of the organization containing the instance.
previous_values.space_id	string	ID of the space containing the instance..
accepts_incomplete	boolean	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.

```
{
  "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"
  }
}
```

cURL

```
$ 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: 2.7" -H "Content-Type: application/json"
```

Response

Status Code	Description
200 OK	New plan is effective. The expected response body is <code>{}</code> .
202 Accepted	Service instance update is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.
422 Unprocessable entity	May be returned if the particular plan change requested is not supported or if the request can not currently be fulfilled due to the state of the instance (eg. 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 <code>422</code> 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: <code>{ "error": "AsyncRequired", "description": "This service plan requires client support for asynchronous service operations." }</code> <code>.</code>

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 services deliver value just from being provisioned. Brokers that offer services that are not bindable should declare them as such using `bindable: false` in the [Catalog](#). Brokers that do not offer any bindable services do not need to implement the endpoint for bind requests.

When the broker receives a bind request from the Cloud Controller, it should return information which helps an application to utilize the provisioned resource. This information is generically referred to as `:credentials`. Applications should be issued unique credentials whenever possible, so one application's access can be revoked without affecting other bound applications. For more information on credentials, see [Binding Credentials](#).

In addition, the bind operation can enable streaming of application logs from Cloud Foundry to a consuming service instance. For details, see [Application Log Streaming](#).

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

Request Field	Type	Description
<code>service_id</code> *	string	ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.
<code>plan_id</code> *	string	ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.
<code>app_guid</code>	string	GUID of the application that you want to bind your service to. Will be included when users bind applications to service instances.
<code>parameters</code>	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.

```
{
  "plan_id": "plan-guid-here",
  "service_id": "service-guid-here",
  "app_guid": "app-guid-here",
  "parameters": {
    "parameter1-name-here": 1,
    "parameter2-name-here": "parameter2-value-here"
  }
}
```

cURL

```
$ curl http://username:password@broker-url/v2/service_instances/:instance_id/service_bindings/:binding_id -d {
  "plan_id": "plan-guid-here",
  "service_id": "service-guid-here",
  "app_guid": "app-guid-here",
  "parameters": {
    "parameter1-name-here": 1,
    "parameter2-name-here": "parameter2-value-here"
  }
}' -X PUT
```

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

Status Code	Description
201 Created	Binding has been created. The expected response body is below.
200 OK	May be returned if the binding already exists and the requested parameters are identical to the existing binding. The expected response body is below.

Status Code	Description
409 Conflict	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 .
422 Unprocessable Entity	Should be returned if the broker requires that <code>app_guid</code> be included in the request body. The expected response body is: <code>{ "error": "RequiresApp", "description": "This service supports generation of credentials through binding an application only." }</code>

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](#).

Response Field	Type	Description
credentials	object	A free-form hash of credentials that the bound application can use to access the service. For more information, see Binding Credentials .
syslog_drain_url	string	A URL to which Cloud Foundry should drain logs for the bound application. <code>requires:syslog_drain</code> must be declared in the catalog endpoint or Cloud Foundry will consider the response invalid. For details, see Application Log Streaming .

```
{
  "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.

Query-String Field	Type	Description
service_id*	string	ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.
plan_id*	string	ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.

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: 2.7"
```

Response

Status Code	Description
200 OK	Binding was deleted. The expected response body is <code>{}</code> .
410 Gone	Should be returned if the binding does not exist. The expected response body is <code>{}</code> .

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.

Query-String Field	Type	Description
service_id*	string	ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.
plan_id*	string	ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.
accepts_incomplete	boolean	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.

cURL

```
$ curl 'http://username:password@broker-url/v2/service_instances/:instance_id?service_id=service-id-here&plan_id=plan-id-here' -X DELETE -H "X-Broker-API-Version: 2.7"
```

Response

Status Code	Description
200 OK	Service instance was deleted. The expected response body is <code>{}</code> .
202 Accepted	Service instance deletion is in progress. This triggers Cloud Controller to poll the Service Instance Last Operation Endpoint for operation status.
410 Gone	Should be returned if the service instance does not exist. The expected response body is <code>{}</code> .
422 Unprocessable Entity	Should be returned if the broker only supports asynchronous deprovisioning for the requested plan and the request did not include <code>?accepts_incomplete=true</code> . The expected response body is: <code>{ "error": "AsyncRequired", "description": "This service plan requires client support for asynchronous service operations." }</code> , 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.`

Response Field	Type	Description
description	string	An error message explaining why the request failed. This message will be displayed to the user who initiated the request.

```
{
  "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 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).

Status code	Result	Orphan mitigation
200	Success	
200 with malformed response	Failure	
201	Success	
201 with malformed response	Failure	Yes
All other 2xx	Failure	Yes
408	Failure due to timeout	Yes
All other 4xx	Broker rejects request	
5xx	Broker error	Yes
Timeout	Failure	Yes

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-Spaced 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 do to 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-service-offering` requires cf-release v160 and edge of cf CLI 6.0.2.

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 the 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
```

`purge-service-instance` requires cf-release v218 and cf CLI 6.14.0.

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.cf-app.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.cf-app.com/v2/catalog: 404 Not Found

Server error, status code: 500, error code: 10001, message:
The service broker API returned an error from
http://github-broker.primo.cf-app.com/v2/catalog: 500 Internal Server Error
```

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 [{"bullets"=>["bullet1", "bullet2"]}]]
```


Access Control

Page last updated:

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": "http://support.cloudfoundry.com",
  "version": 2,
  "description": "Cloud Foundry sponsored by Pivotal",
  "authorization_endpoint": "https://login.system-domain.com",
  "token_endpoint": "https://uaa.system-domain.com",
  "api_version": "2.13.0",
  "logging_endpoint": "wss://loggregator.system-domain.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-riakes
  service plan    access  orgs
  p-riakes  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.

```
$ cf help service-access
NAME:
  service-access - List service access settings

USAGE:
  cf service-access [-b BROKER] [-e SERVICE] [-o ORG]

OPTIONS:
  -b  access for plans of a particular broker
  -e  access for plans of a particular service offering
  -o  plans accessible by a particular 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.

```
$ cf enable-service-access p-riakcs
Enabling access to all plans of service p-riakcs for all orgs as admin...
OK

$ cf service-access
getting service access as admin...
broker: p-riakcs
  service plan    access orgs
  p-riakcs developer all
```

An admin can use `enable-service-access` 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`)

```
$ cf help enable-service-access
NAME:
  enable-service-access - Enable access to a service or service plan for one or all orgs

USAGE:
  cf enable-service-access SERVICE [-p PLAN] [-o ORG]

OPTIONS:
  -p  Enable access to a particular service plan
  -o  Enable access to a particular organization
```

Disable Access to Service Plans

```
$ cf disable-service-access p-riakcs
Disabling access to all plans of service p-riakcs for all orgs as admin...
OK

$ cf service-access
getting service access as admin...
broker: p-riakcs
  service plan    access orgs
  p-riakcs developer none
```

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:

```
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:

```
$ 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:

```
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.

```
$ cf curl /v2/service_plans/1113aa0-124e-4af2-1526-6bfaef61b111 -X 'PUT' -d '{"public":true}'

{
  "metadata": {
    "guid": "1113aa0-124e-4af2-1526-6bfaef61b111",
    "url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfaef61b111",
    "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": true,
    "service_url": "/v2/services/d9011411-1463-477c-b223-82e04996b91f",
    "service_instances_url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfaef61b111/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](#)

```
cf curl -X 'GET'  
/v2/service_plans
```

To find the organization GUIDs, run:

```
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:

```
$ cf curl /v2/organizations?q=name:my-org

{
  "metadata": {
    "guid": "c54bf317-d791-4d12-89f0-b56d0936cf0",
    "url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0",
    "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": "52c5413c-869f-455a-8873-7972ecb85ca8",
    "status": "active",
    "quota_definition_url": "/v2/quota_definitions/52c5413c-869f-455a-8873-7972ecb85ca8",
    "spaces_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/spaces",
    "domains_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/domains",
    "private_domains_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/private_domains",
    "users_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/users",
    "managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/managers",
    "billing_managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/billing_managers",
    "auditors_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/auditors",
    "app_events_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cf0/app_events"
  }
}
```

In this example, the GUID of my-org is `c54bf317-d791-4d12-89f0-b56d0936cf0`.

To make a private plan available to a specific organization, run:

```
cf curl /v2/service_plan_visibilities -X POST -d
'{"service_plan_guid":"SERVICE_PLAN_GUID","organization_guid":"ORG_GUID"}'
```

Example:

```
$ cf curl /v2/service_plan_visibilities -X 'POST' -d '{"service_plan_guid":"1113aa0-124e-4af2-1526-6bfaef61b111","organization_guid":"aaaa1234-da91-4f12-8ffa-b51d0336aaaa"}'

{
  "metadata": {
    "guid": "99993789-a368-483e-ae7c-ebe79e199999",
    "url": "/v2/service_plan_visibilities/99993789-a368-483e-ae7c-ebe79e199999",
    "created_at": "2014-02-12T21:03:42+00:00",
    "updated_at": null
  },
  "entity": {
    "service_plan_guid": "1113aa0-124e-4af2-1526-6bfaef61b111",
    "organization_guid": "aaaa1234-da91-4f12-8ffa-b51d0336aaaa",
    "service_plan_url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfaef61b111",
    "organization_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc"
  }
}
```

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/1113aa0-124e-4af2-1526-6bfaef61b111 -X 'PUT' -d '{"public":false}'

{
  "metadata": {
    "guid": "1113aa0-124e-4af2-1526-6bfaef61b111",
    "url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfaef61b111",
    "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/1113aa0-124e-4af2-1526-6bfaef61b111/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:

```
$ cf curl /v2/service_plan_visibilities/99993789-a368-483e-ae7c-ebe79e199999 -X DELETE
```

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

Broker API Field	Type	Description	CC API Field	Pivotal CLI	Pivotal Apps Manager
name	CLI string	A short name for the service to be displayed in a catalog.	label	X	X
description	string	A short 1-line description for the service, usually a single sentence or phrase.	description	X	X
metadata.displayName	string	The name of the service to be displayed in graphical clients	extra.displayName		X
metadata.imageUrl	string	The URL to an image.	extra.imageUrl		X
metadata.longDescription	string	Long description	extra.longDescription		X
metadata.providerDisplayName	string	The name of the upstream entity providing the actual service	extra.providerDisplayName		X
metadata.documentationUrl	string	Link to documentation page for service	extra.documentationUrl		X
metadata.supportUrl	string	Link to support for the service	extra.supportUrl		X

Plan Metadata Fields

Broker API Field	Type	Description	CC API Field	Pivotal CLI	Pivotal Apps Manager
name	CLI string	A short name for the service plan to be displayed in a catalog.	name	X	
description	string	A description of the service plan to be displayed in a catalog.	description		

Broker API Field metadata.bullets	array-of- strings	Description Features of this plan, to be displayed in a bulleted-list	CC API Field extra.bullets	Pivotal CLI	Pivotal Apps Manager
metadata.costs	cost object	<p>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).</p> <p>Each object must provide the following keys:</p> <pre>amount: { usd: float }, unit: string</pre> <p>This indicates the cost in USD of the service plan, and how frequently the cost is occurred, such as “MONTHLY” or “per 1000 messages”.</p>	extra.costs		X
metadata.displayName	string	Name of the plan to be display in graphical clients.	extra.displayName		X

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.

```
{
  "services": [
    {
      "id": "766fa866-a950-4b12-adff-c11fa4cf8fdc",
      "name": "cloudamqp",
      "description": "Managed HA RabbitMQ servers in the cloud",
      "requires": [
        ],
      "tags": [
        "amqp",
        "rabbitmq",
        "messaging"
      ],
      "metadata": {
        "displayName": "CloudAMQP",
        "imageUrl": "https://d3na3ni6eqf5j.cloudfront.net/app_resources/18492/thumbs_112/img9069612145282015279.png",
        "longDescription": "Managed, highly available, RabbitMQ clusters in the cloud",
        "providerDisplayName": "84codes AB",
        "documentationUrl": "http://docs.cloudfoundry.com/docs/dotcom/marketplace/services/cloudamqp.html",
        "supportUrl": "http://www.cloudamqp.com/support.html"
      },
      "dashboard_client": {
        "id": "p-mysql-client",
        "secret": "p-mysql-secret",
        "redirect_uri": "http://p-mysql.example.com/auth/create"
      },
      "plans": [
        {
          "id": "024f3452-67f8-40bc-a724-a20c4ea24b1c",
          "name": "bunny",
          "description": "A mid-sized plan",
          "metadata": {
            "bullets": [
              "20 GB of messages",
              "20 connections"
            ],
            "costs": [
              {
                "amount": {
                  "usd": "99.0"
                },
                "unit": "MONTHLY"
              },
              {
                "amount": {
                  "usd": "0.99"
                },
                "unit": "1GB of messages over 20GB"
              }
            ],
            "displayName": "Big Bunny"
          }
        }
      ]
    }
  ]
}
```

Example Cloud Controller Response Body

```
{
  "metadata": {
    "guid": "bc8748f1-fe05-444d-ab7e-9798e1f9aef6",
    "url": "/v2/services/bc8748f1-fe05-444d-ab7e-9798e1f9aef6",
    "created_at": "2014-01-08T18:52:16+00:00",
    "updated_at": "2014-01-09T03:19:16+00:00"
  },
  "entity": {
    "label": "cloudamqp",
    "provider": "cloudamqp",
    "url": "http://adgw.a1.cf-app.com",
    "description": "Managed HA RabbitMQ servers in the cloud",
    "long_description": null,
    "version": "n/a",
    "info_url": null,
    "active": true,
    "bindable": true,
    "unique_id": "18723",
    "extra": {
      "displayName": "CloudAMQP",
      "imageUrl": "https://d33na3ni6eqf5j.cloudfront.net/app_resources/18723/thumbs_112/img9069612145282015279.png",
      "longDescription": "Managed, highly available, RabbitMQ clusters in the cloud",
      "providerDisplayName": "84codesAB",
      "documentationUrl": null,
      "supportUrl": null
    },
    "tags": [
      "amqp",
      "rabbitmq"
    ],
    "requires": [
    ],
    "documentation_url": null,
    "service_plans": [
      {
        "metadata": {
          "guid": "6c4903ab-14ce-41de-adb2-632cf06117a5",
          "url": "/v2/services/6c4903ab-14ce-41de-adb2-632cf06117a5",
          "created_at": "2013-11-01T00:21:25+00:00",
          "updated_at": "2014-01-09T03:19:16+00:00"
        },
        "entity": {
          "name": "bunny",
          "free": true,
          "description": "Big Bunny",
          "service_guid": "bc8748f1-fe05-444d-ab7e-9798e1f9aef6",
          "extra": {
            "bullets": [
              "20 GB of messages",
              "20 connections"
            ],
            "costs": [
              {
                "amount": {
                  "usd": 99.0
                },
                "unit": "MONTHLY"
              },
              {
                "amount": {
                  "usd": 0.99
                },
                "unit": "1GB of messages over 20GB"
              }
            ],
            "displayName": "Big Bunny"
          },
          "unique_id": "addonOffering_1889",
          "public": true
        }
      }
    ]
  }
}
```

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:

```
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:

```
{  
  "services": [  
    {  
      "id": "44b26033-1f54-4087-b7bc-da9652c2a539",  
      ...  
      "dashboard_client": {  
        "id": "p-mysql-client",  
        "secret": "p-mysql-secret",  
        "redirect_uri": "http://p-mysql.example.com"  
      }  
    }  
  ]  
}
```

The `dashboard_client` field is a hash containing three fields:

- o `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).
- o `secret` is the shared secret your dashboard will use to authenticate with the token server (UAA).
- o `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/`.

- 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-cf.com/info
{"name":"vcap","build":"2222","support":"http://support.example-cf.com","version":2,"description":"Cloud Foundry sponsored by Pivotal","authorization_endpoint":"https://login.example-cf.com","token_endpoint":"https://uaa.example-cf.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 ([UAU docs](#), [RFC docs](#)).

- 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](#).
- UAU 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.
- 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
```

- 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.
- UAU 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 issued. If valid, UAA responds back with an access token and a refresh token.

Checking User Permissions

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'\n-H 'Authorization: bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJc2VyX2lkIjoid'\nhttp://api.cloudfoundry.com/v2/service_instances/44b26033-1f54-4087-b7bc-da9652c2a539/permissions
```

Response:

```
{\n  \"manage\": true\n}
```

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.

Scope	Permissions
<code>openid</code>	Allows access to basic data about the user, such as email addresses
<code>cloud_controller_service_permissions.read</code>	Allows access to the CC endpoint that specifies whether the user can manage a given service instance

Additional Scopes

Dashboards with extended capabilities may need to request these additional scopes:

Scope	Permissions
<code>cloud_controller.read</code>	Allows read access to all resources the user is authorized to read
<code>cloud_controller.write</code>	Allows write access to all resources the user is authorized to update / create / delete

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 OmniAuth Strategy](#). See this [OmniAuth wiki page](#) for instructions on how to create your own strategy.

The UAA OmniAuth 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 Boot 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.

 **Note:** If you provide a service that supports a connection string, provide the `uri` key for buildpacks and application libraries to use.

CREDENTIALS	DESCRIPTION
uri	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.
hostname	FQDN of the server host
port	Port of the server host
name	Name of the service instance
vhost	Name of the messaging server virtual host - a replacement for a <code>name</code> specific to AMQP providers
username	Server user
password	Server password

The following is an example output of `ENV["VCAP_SERVICES"]`.

 **Note:** Depending on the types of databases you are using, each database might return different credentials.

```
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:IwN6dCdZmeQD400ZPKpu1Y0aLx1he8wo@lemur.cloudamqp.com/ksvyjmiv"
      }
    }
  ],
  name: "cloudamqp-9dbc6",
  label: "cloudamqp",
  plan: "lemur",
  credentials: {
    uri: "amqp://vhuklnxa:91NFxpTuJsAdTts98vQIdKHW3MojyMyV@lemur.cloudamqp.com/vhuklnxa"
  }
  ],
  rediscloud: [
    {
      name: "rediscloud-1",
      label: "rediscloud",
      plan: "20mb",
      credentials: {
        port: "6379",
        ...
      }
    }
  ]
}
```

```
        host: "pub-redis-6379.us-east-1-2.3.ec2.redislabs.com",
        password: "1M5zd3QfWi9nUyya"
    },
],
}
```

Application Log Streaming

Page last updated:

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`s

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

Page last updated:

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

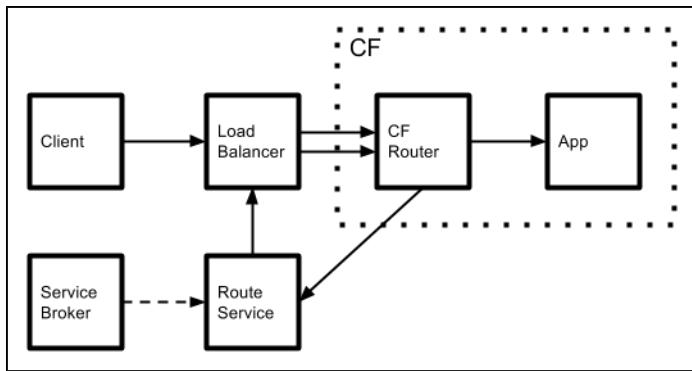
Cloud Foundry application developers may wish to apply transformation or processing to requests before they reach an application. Common examples of use cases are authentication, rate limiting, and caching services. Route Services are a new kind of Marketplace Service that developers can use to apply 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.



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:

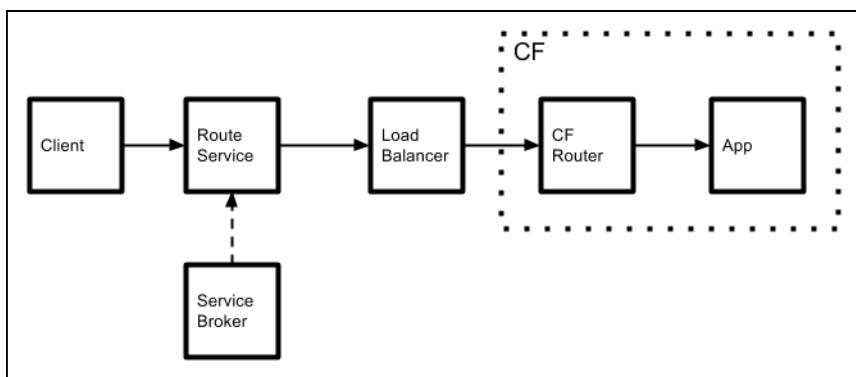
- Developers can use a Service Broker to dynamically configure how the route service processes traffic to specific applications.
- Adding route services requires no manual infrastructure configuration.
- Traffic to apps that do not use the service makes fewer network hops; requests for those apps do not pass through the route service.

Disadvantages::

- 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:

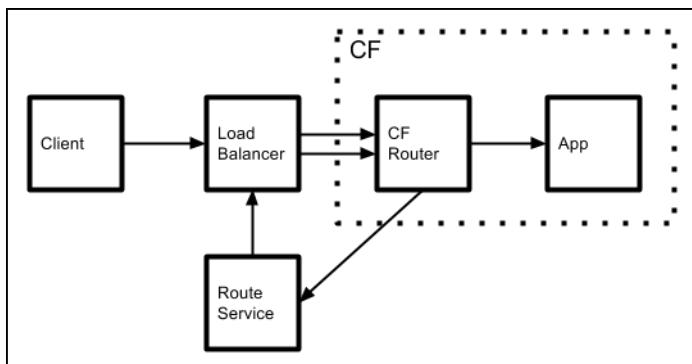
- Developers can use a Service Broker to dynamically configure how the route service processes traffic to specific applications.
- Traffic to apps that use the route service takes fewer network hops.

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:

cf create-user-provided-service SERVICE_INSTANCE -r
ROUTE_SERVICE_URL

cf bind-route-service DOMAIN SERVICE_INSTANCE [--hostname
HOSTNAME]

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:

Route Services Architecture	Fulfils CF Service Instance Responsibilities	Fulfils CF Broker Responsibilities
Fully-Brokered	Yes	Yes
Static Brokered	No	Yes
User-Provided	Yes	No

Enabling Route Services in Cloud Foundry

To enable support for Route Services in a Cloud Foundry deployment, the operator must provide a passphrase used by GoRouter to generate a key, which in turn is used to encrypt a header sent with the forwarded request to the route service. When this header is present in the request forwarded from the route service, Gorouter forwards the request to the application. See [Headers](#) below for details. The passphrase is configured in the cf-release manifest.

```
properties:
  router:
    route_services_secret: YOUR-SECRET-PASSPHRASE
```

Note: The `route_services_secret` property should be a strong passphrase. See the [GoRouter spec](#) in cf-release for details.

Route Service instances should send requests to the value of `x-cf-forwarded-url`, obeying the scheme. The scheme is `https` by default; for environments that don't support TLS termination, this property can be set to `false`.

```
properties:
  router:
    route_services_recommend_https: true
```

The CF router will only forward requests to Route Services over SSL. By default, certificates provided by Route Services must be signed by a trusted CA. If they are not, the CF router will reject the request. In development environments this concern may be unreasonable. To disable SSL cert validation, modify the following property:

```
properties:
  router:
    ssl_skip_validation: true
```

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.

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.

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](#).

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

Testing

The following instructions show how to use the [Logging Route Service](#) described in [Examples](#) to verify that when a route service is bound to a route, requests for that route are proxied to the route service.

Requires CLI version 6.16 or above.

1. Push the [Logging Route Service](#) as an app.

```
$ 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.

```
$ 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`.

```
$ 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`.

```
$ cf bind-route-service cf.example.com mylogger --hostname spring-music
```

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
```

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](#)
- [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 Pivotal Ops Metrics](#)
- [Deploying Pivotal Ops Metrics](#)
- [Using Pivotal Ops Metrics](#)

Overview of the Loggregator System

Page last updated:

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.

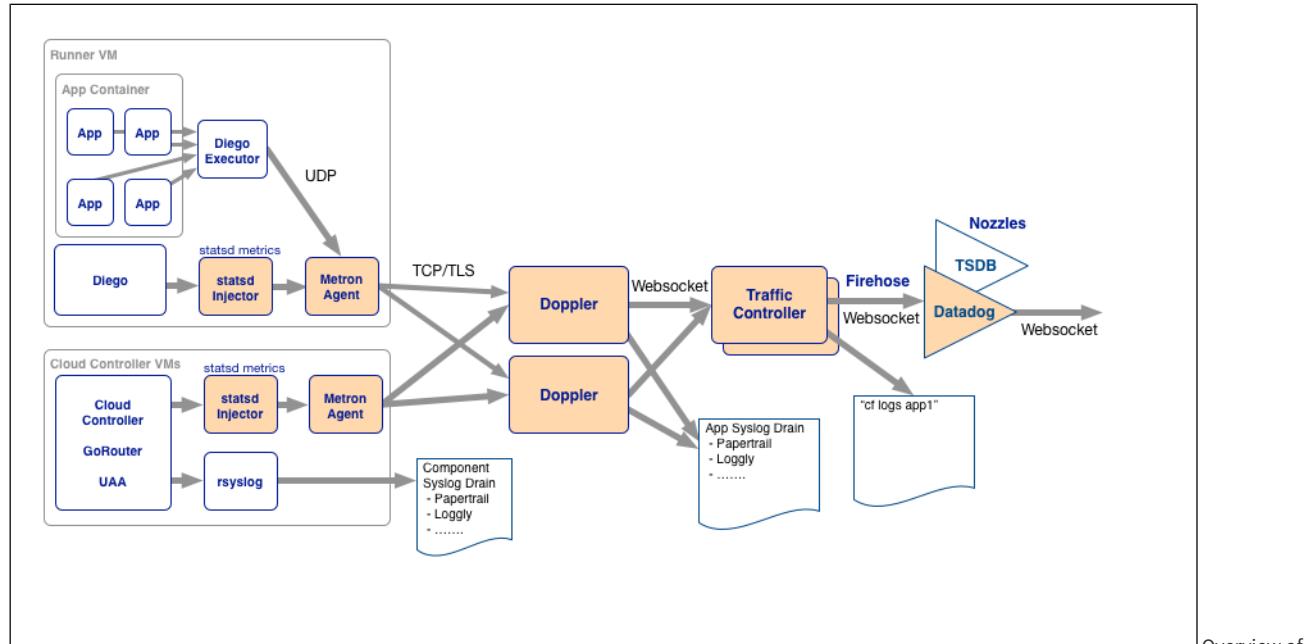
[Loggregator on GitHub](#)

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.10.244.0.34.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](#) publishes metrics coming from the Firehose to Datadog.
- The [Syslog nozzle](#) filters out log messages coming from the Firehose and sends it to a syslog server.

See our [Nozzle Tutorial](#).

Loggregator Guide for Cloud Foundry Operators

Page last updated:

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.

Cloud Foundry System Metrics

Page last updated:

This topic lists and describes the metrics available for system components. These metrics are streamed from the Loggregator [Firehose](#).

Cloud Controller

Metric Name	Description
failed_job_count.cc-api_z1-0	Number of failed jobs in the cc-api_z1-0 queue. Updated every 30 seconds per VM
failed_job_count.cc-api_z2-0	Number of failed jobs in the cc-api_z2-0 queue. Updated every 30 seconds per VM
failed_job_count.cc-api_z2-1	Number of failed jobs in the cc-api_z2-1 queue. Updated every 30 seconds per VM
failed_job_count.cc-generic	Number of failed jobs in the cc-generic queue. Updated every 30 seconds per VM
failed_job_count.total	Number of failed jobs in all queues. Updated every 30 seconds per VM
http_status.1XX	Number of HTTP response status codes of type 1xx (informational)
http_status.2XX	Number of HTTP response status codes of type 2xx (success)
http_status.3XX	Number of HTTP response status codes of type 3xx (redirection)
http_status.4XX	Number of HTTP response status codes of type 4xx (client error)
http_status.5XX	Number of HTTP response status codes of type 5xx (server error)
job_queue_length.cc-api_z1-0	Number of background jobs in the cc-api_z1-0 queue that have yet to run for the first time. Updated every 30 seconds per VM
job_queue_length.cc-api_z1-1	Number of background jobs in the cc-api_z1-1 queue that have yet to run for the first time. Updated every 30 seconds per VM
job_queue_length.cc-api_z2-0	Number of background jobs in the cc-api_z2-0 queue that have yet to run for the first time. Updated every 30 seconds per VM
job_queue_length.cc-api_z2-1	Number of background jobs in the cc-api_z2-1 queue that have yet to run for the first time. Updated every 30 seconds per VM
job_queue_length.cc-generic	Number of background jobs in the cc-generic queue that have yet to run for the first time. Updated every 30 seconds per VM
job_queue_length.total	Total number of background jobs in the queues that have yet to run for the first time. Updated every 30 seconds per VM
log_count.all	Total number of log messages, sum of messages of all severity levels. Updated every 30 seconds per VM
log_count.debug	Number of log messages of severity “debug”. Updated every 30 seconds per VM
log_count.debug1	Number of log messages of severity “debug1”. Updated every 30 seconds per VM
log_count.debug2	Number of log messages of severity “debug2”. Updated every 30 seconds per VM
log_count.error	Number of log messages of severity “error”. Updated every 30 seconds per VM
log_count.fatal	Number of log messages of severity “fatal”. Updated every 30 seconds per VM
log_count.info	Number of log messages of severity “info”. Updated every 30 seconds per VM
log_count.off	Number of log messages of severity “off”. Updated every 30 seconds per VM
log_count.warn	Number of log messages of severity “warn”. Updated every 30 seconds per VM
requests.completed	Number of requests that have been processed
requests.outstanding	Number of requests that are currently being processed
tasks_running.count	Number of currently running tasks. Updated every 30 seconds per VM. This metric is only seen in version 3 of the Cloud Foundry API
tasks_running.memory_in_mb	Memory being consumed by all currently running tasks. Updated every 30 seconds per VM. This metric is only seen in version 3 of the Cloud Foundry API
thread_info.event_machine.connection_count	Number of open connections to event machine. Updated every 30 seconds per VM
thread_info.event_machine.resultqueue.num_waiting	Number of scheduled tasks in the result. Updated every 30 seconds per VM
thread_info.event_machine.resultqueue.size	Number of unscheduled tasks in the result. Updated every 30 seconds per VM

Metric Name	Description
thread_info.event_machine.threadqueue.num_waiting	Number of scheduled tasks in the threadqueue. Updated every 30 seconds per VM
thread_info.event_machine.threadqueue.size	Number of unscheduled tasks in the threadqueue. Updated every 30 seconds per VM
thread_info.thread_count	Total number of threads that are either runnable or stopped. Updated every 30 seconds per VM
total_users	Total number of users ever created, including inactive users. Updated every 10 minutes per VM
vcap_sinatra.recent_errors	50 most recent errors. DEPRECATED
vitals.cpu	Percentage of CPU used by the Cloud Controller process. Updated every 30 seconds per VM
vitals.cpu_load_avg	System CPU load averaged over a minute according to the OS. Updated every 30 seconds per VM
vitals.mem_bytes	The RSS bytes (resident set size) or real memory of the Cloud Controller process. Updated every 30 seconds per VM
vitals.mem_free_bytes	Total memory available according to the OS. Updated every 30 seconds per VM
vitals.mem_used_bytes	Total memory used (active + wired) according to the OS. Updated every 30 seconds per VM
vitals.num_cores	The number of CPUs of a host machine. Updated every 30 seconds per VM
vitals.uptime	The uptime of the Cloud Controller process in seconds. Updated every 30 seconds per VM

[Top](#)

DEA

Default Origin Name: dea

Metric Name	Description
available_disk_ratio	Percentage of remaining disk that includes staging and running apps, includes overcommit
available_memory_ratio	Percentage of remaining memory that includes staging and running apps, includes overcommit
avg_cpu_load	Average CPU over the past minute
dea_registry_born	The number of born instances
dea_registry_crashed	The number of crashed instances
dea_registry_evacuating	The number of evacuating instances
dea_registry_resuming	DEPRECATED
dea_registry_running	The number of running instances
dea_registry_starting	The number of starting instances
dea_registry_stopped	The number of stopped instances
dea_registry_stopping	The number of stopping instances
instances	Total number of “running” instances, regardless of state
remaining_disk	Remaining disk that includes staging and running apps, includes overcommit
remaining_memory	Remaining memory that includes staging and running apps, includes overcommit
reservable_stagers	The number of stagings that could occur given the current memory and disk available
total_warden_response_time_in_ms	DEPRECATED
uptime	How long the DEA has been running
warden_error_response_count	DEPRECATED
warden_request_count	DEPRECATED

[Top](#)

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](#)
- [tps_listener](#)
- [tps_watcher](#)

Default Origin Name: auctioneer

Metric Name	Description
AuctioneerFetchStatesDuration	Time the auctioneer took to fetch state from all the cells when running its auction. Emitted during each auction
AuctioneerLRPAuctionsFailed	Cumulative number of LRP instances that the auctioneer failed to place on Diego cells. Emitted during each auction
AuctioneerLRPAuctionsStarted	Cumulative number of LRP instances that the auctioneer successfully placed on Diego cells. Emitted during each auction
AuctioneerTaskAuctionsFailed	Cumulative number of Tasks that the auctioneer failed to place on Diego cells. Emitted during each auction
AuctioneerTaskAuctionsStarted	Cumulative number of Tasks that the auctioneer successfully placed on Diego cells. Emitted during each auction
LockHeld.v1-locks-auctioneer_lock	Whether an auctioneer holds the auctioneer lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active auctioneer
LockHeldDuration.v1-locks-auctioneer_lock	Time the active auctioneer has held the auctioneer lock. Emitted periodically by the active auctioneer
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: bbs

Metric Name	Description
BBSSMasterElected	Emitted once when the BBS is elected as master
ConvergenceLRPDuration	Time the BBS took to run its LRP convergence pass. Emitted every time LRP convergence runs
ConvergenceLRPPreProcessingActualLRPsDeleted	Cumulative number of times the BBS has detected and deleted a malformed ActualLRP in its LRP convergence pass. Emitted periodically
ConvergenceLRPPreProcessingMalformedRunInfos	Cumulative number of times the BBS has detected a malformed DesiredLRP RunInfo in its LRP convergence pass. Emitted periodically
ConvergenceLRPPreProcessingMalformedSchedulingInfos	Cumulative number of times the BBS has detected a malformed DesiredLRP SchedulingInfo in its LRP convergence pass. Emitted periodically
ConvergenceLPRRuns	Cumulative number of times BBS has run its LRP convergence pass. Emitted periodically
ConvergenceTaskDuration	Time the BBS took to run its Task convergence pass. Emitted every time Task convergence

Metric Name	Description
ConvergenceTaskRuns	Cumulative number of times the BBS has run its Task convergence pass. Emitted periodically
ConvergenceTasksKicked	Cumulative number of times the BBS has updated a Task during its Task convergence pass. Emitted periodically
ConvergenceTasksPruned	Cumulative number of times the BBS has deleted a malformed Task during its Task convergence pass. Emitted periodically
CrashedActualLRPs	Total number of LRP instances that have crashed. Emitted periodically
CrashingDesiredLRPs	Total number of DesiredLRPs that have at least one crashed instance. Emitted periodically
Domain.cf-apps	Whether the 'cf-apps' domain is up-to-date, so that CF apps from CC have been synchronized with DesiredLRPs for Diego to run. 1 means the domain is up-to-date, no data means it is not. Emitted periodically
Domain.cf-tasks	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 periodically
ETCDLeader	Index of the leader node in the etcd cluster. Emitted periodically
ETCDRaftTerm	Raft term of the etcd cluster. Emitted periodically
ETCDReceivedBandwidthRate	Number of bytes per second received by the follower etcd node. Emitted periodically
ETCDReceivedRequestRate	Number of requests per second received by the follower etcd node. Emitted periodically
ETCDSentBandwidthRate	Number of bytes per second sent by the leader etcd node. Emitted periodically
ETCDSentRequestRate	Number of requests per second sent by the leader etcd node. Emitted periodically
ETCDWatchers	Number of watches set against the etcd cluster. Emitted periodically
LockHeld.v1-locks-bbs_lock	Whether a BBS holds the BBS lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active BBS server
LockHeldDuration.v1-locks-bbs_lock	Time the active BBS has held the BBS lock. Emitted periodically by the active BBS server
LRPsClaimed	Total number of LRP instances that have been claimed by some cell. Emitted periodically
LRPsDesired	Total number of LRP instances desired across all LRPs. Emitted periodically
LRPsExtra	Total number of LRP instances that are no longer desired but still have a BBS record. Emitted periodically
LRPsMissing	Total number of Tasks running on cells. Emitted periodically
LRPsRunning	Total number of LRP instances that are running on cells. Emitted periodically
LRPsUnclaimed	Total number of LRP instances that have not yet been claimed by a cell. Emitted periodically
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
MetricsReportingDuration	Time the BBS took to emit metrics about etcd. Emitted periodically
MigrationDuration	Time the BBS took to run migrations against its persistence store. Emitted each time a BBS becomes the active master
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process
RequestCount	Cumulative number of requests the BBS has handled through its API. Emitted periodically
RequestLatency	Time the BBS took to handle requests to its API endpoints. Emitted when the BBS API handles requests
TasksCompleted	Total number of Tasks that have completed. Emitted periodically
TasksPending	Total number of Tasks that have not yet been placed on a cell. Emitted periodically
TasksResolving	Total number of Tasks locked for deletion. Emitted periodically

Metric Name	Description
-------------	-------------

Default Origin Name: cc_uploader

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: file_server

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: garden_linux

Metric Name	Description
BackingStores	Number of container backing store files
DepotDirs	Number of directories in the Garden depot
LoopDevices	Number of attached loop devices
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
MetricsReporting	How long it took to emit the BackingStores, DepotDirs, and LoopDevices metrics
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: nsync_bulker

Metric Name	Description
DesiredLRPSyncDuration	Time the nsync-bulker took to synchronize CF apps and Diego DesiredLRPs. Emitted periodically
LockHeld.v1-locks-nsync_bulker_lock	Whether an nsync-bulker holds the nsync-bulker lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active nsync-bulker
LockHeldDuration.v1-locks-nsync_bulker_lock	Time the active nsync-bulker has held the convergence lock. Emitted periodically by the active nsync-bulker
LRPsDesired	Cumulative number of LRPCs desired through the nsync API. Emitted on each request desiring a new LRP
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use

memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
NsyncInvalidDesiredLRPsFound	Number of invalid DesiredLRPs found during nsync-bulker periodic synchronization
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: nsync_listener

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: rep

Metric Name	Description
CapacityRemainingContainers	Remaining number of containers this cell can host. Emitted periodically
CapacityRemainingDisk	Remaining amount of disk available for this cell to allocate to containers. Emitted periodically
CapacityRemainingMemory	Remaining amount of memory available for this cell to allocate to containers. Emitted periodically
CapacityTotalContainers	Total number of containers this cell can host. Emitted periodically
CapacityTotalDisk	Total amount of disk available for this cell to allocate to containers. Emitted periodically
CapacityTotalMemory	Total amount of memory available for this cell to allocate to containers. Emitted periodically
CM	
ContainerCount	Number of containers hosted on the cell. Emitted periodically
GardenContainerCreationDuration	Time the rep Garden backend took to create a container. Emitted after every successful container creation
LogMessage	
logSenderTotalMessagesRead	
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process
RepBulkSyncDuration	Time the cell rep took to synchronize the ActualLRPs it has claimed with its actual garden containers. Emitted periodically by each rep
UnhealthyCell	Whether the cell has failed to pass its healthcheck against the garden backend. 0 signifies healthy, and 1 signifies unhealthy. Emitted periodically

Default Origin Name: route_emitter

Metric Name	Description
ConsulDownMode	Whether a route-emitter is operating normally: <input type="checkbox"/> if the route-emitter is healthy, and <input checked="" type="checkbox"/> when the consul servers are either down or in a bad state.
	Whether a route-emitter holds the route-emitter lock: 1 means the lock is held, and 0 means the lock was lost.

Metric Name	Description
LockHeld.v1-locks-route_emitter_lock	Time the active route-emitter has held the route-emitter lock. Emitted periodically by the active route-emitter
LockHeldDuration.v1-locks-route_emitter_lock	Time the active route-emitter has held the route-emitter lock. Emitted periodically by the active route-emitter
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
MessagesEmitted	The cumulative number of registration messages that this process has sent
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process
RouteEmitterSyncDuration	Time the active route-emitter took to perform its synchronization pass. Emitted periodically
RoutesRegistered	Cumulative number of route registrations emitted from the route-emitter as it reacts to changes to LRPs
RoutesSynced	Cumulative number of route registrations emitted from the route-emitter during its periodic route-table synchronization
RoutesTotal	Number of routes in the route-emitter's routing table. Emitted periodically
RoutesUnregistered	Cumulative number of route unregistrations emitted from the route-emitter as it reacts to changes to LRPs

Default Origin Name: ssh_proxy

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: stager

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process
StagingRequestFailedDuration	Time the failed staging task took to run. Emitted each time a staging task fails
StagingRequestsFailed	Cumulative number of failed staging tasks handled by each stager. Emitted every time a staging task fails
StagingRequestsSucceeded	Cumulative number of successful staging tasks handled by each stager. Emitted every time a staging task completes successfully
StagingRequestSucceededDuration	Time the successful staging task took to run. Emitted each time a staging task completes successfully
StagingStartRequestsReceived	Cumulative number of requests to start a staging task. Emitted by a stager each time it handles a request

Default Origin Name: tps_listener

Metric Name	Description

Metric Name	Description
memoryStats.lastGCPauseTimeNS	the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: tps_watcher

Metric Name	Description
LockHeld.v1-locks-tps_watcher_lock	Whether a tps-watcher holds the tps-watcher lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active tps-watcher
LockHeldDuration.v1-locks-tps_watcher_lock	Time the active tps-watcher has held the convergence lock. Emitted periodically by the active tps-watcher
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

[Top](#)

DopplerServer

Default Origin Name: DopplerServer

Metric Name	Description
dropsondeListener.currentBufferCount	DEPRECATED
dropsondeListener.receivedByteCount	DEPRECATED in favor of DopplerServer.udpListener.receivedByteCount
dropsondeListener.receivedMessageCount	Lifetime number of messages received by Doppler
dropsondeUnmarshaller.containerMetricReceived	Lifetime number of ContainerMetric messages unmarshalled
dropsondeUnmarshaller.counterEventReceived	Lifetime number of CounterEvent messages unmarshalled
dropsondeUnmarshaller.errorReceived	Lifetime number of Error messages unmarshalled
dropsondeUnmarshaller.heartbeatReceived	DEPRECATED
dropsondeUnmarshaller.httpStartReceived	Lifetime number of HttpStart messages unmarshalled
dropsondeUnmarshaller.httpStartStopReceived	Lifetime number of HttpStartStop messages unmarshalled
dropsondeUnmarshaller.httpStopReceived	Lifetime number of HttpStop messages unmarshalled
dropsondeUnmarshaller.logMessageTotal	Lifetime number of LogMessage messages unmarshalled
dropsondeUnmarshaller.unmarshalErrors	Lifetime number of errors when unmarshalling messages
dropsondeUnmarshaller.valueMetricReceived	Lifetime number of ValueMetric messages unmarshalled
httpServer.receivedMessages	Number of messages received by Doppler's internal MessageRouter
LinuxFileDescriptor	Number of file handles for the Doppler's process
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use

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
messageRouter.numberOfDumpSinks	Instantaneous number of dump sinks known to the SinkManager
messageRouter.numberOfFirehoseSinks	Instantaneous number of firehose sinks known to the SinkManager
messageRouter.numberOfSyslogSinks	Instantaneous number of syslog sinks known to the SinkManager
messageRouter.numberOfWebSocketSinks	Instantaneous number of websocket sinks known to the SinkManager
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
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
tcpListener.receivedMessageCount	Lifetime number of messages received by Doppler's TCP Listener
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
tlsListener.receivedMessageCount	Lifetime number of messages received by Doppler's TLS Listener
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"
TruncatingBuffer.totalDroppedMessages	Lifetime total number of messages intentionally dropped by Doppler from all of its sinks due to back pressure
listeners.totalReceivedMessageCount	Total number of messages received across all of Doppler's listeners (UDP, TCP, TLS)
numCpus	Number of CPUs on the machine
numGoRoutines	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

[Top](#)

Etcd

Visit [etcd stats API](#)

Default Origin Name: etcd

Metric Name	Description
CompareAndDeleteFail	CompareAndDeleteFail operation count
CompareAndDeleteSuccess	CompareAndDeleteSuccess operation count
CompareAndSwapFail	CompareAndSwapFail operation count
CompareAndSwapSuccess	CompareAndSwapSuccess operation count
CreateFail	CreateFail operation count
CreateSuccess	CreateSuccess operation count
DeleteFail	DeleteFail operation count

Metric Name	Description
DeleteSuccess	DeleteSuccess operation count
EtcdIndex	X-Etcd-Index value from the /stats/store endpoint
ExpireCount	ExpireCount operation count
Followers	Number of etcd followers
GetsFail	GetsFail operation count
GetsSuccess	GetsSuccess operation count
IsLeader	1 if the current server is the leader, 0 if it is a follower
Latency	Current latency to a specific follower
RaftIndex	X-Raft-Index value from the /stats/store endpoint
RaftTerm	X-Raft-Term value from the /stats/store endpoint
ReceivedAppendRequests	Number of append requests this node has processed
ReceivingBandwidthRate	Number of bytes per second this node is receiving (follower only)
ReceivingRequestRate	Number of requests per second this node is receiving (follower only)
SendingBandwidthRate	Number of bytes per second this node is sending (leader only). This value is undefined on single member clusters.
SendingRequestRate	Number of requests per second this node is sending (leader only). This value is undefined on single member clusters.
SentAppendRequests	Number of requests that this node has sent
SetsFail	SetsFail operation count
SetsSuccess	SetsSuccess operation count
UpdateFail	UpdateFail operation count
UpdateSuccess	UpdateSuccess operation count
Watchers	Watchers operation count

[Top](#)

HM9000

HM9000 metrics have the following origin names:

- [analyzer](#)
- [apiserver](#)
- [evacuator](#)
- [fetcher](#)
- [listener](#)
- [sender](#)
- [shredder](#)

Default Origin Name: analyzer

Metric Name	Description
LockHeld.hm9000.analyzer	Whether an analyzer holds the lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active analyzer
LockHeldDuration.hm9000.analyzer	Time the active analyzer has held the analyzer lock. Emitted periodically by the active analyzer
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
NumberOfAppsWithAllInstancesReporting	The number of desired apps for which all instances are reporting. The state of the instance is irrelevant: STARTING/RUNNING/CRASHED all count
NumberOfAppsWithMissingInstances	The number of desired apps for which an instance is missing, i.e. the instance is not heartbeating at all

Metric Name	Description
NumberOfCrashedIndices	The number of indices reporting as crashed. Because of the restart policy an individual, index might have very many crashes associated with it.
NumberOfCrashedInstances	The number of instances reporting as crashed
NumberOfDesiredApps	The number of apps that should be running
NumberOfDesiredAppsPendingStaging	The number of apps that are staging
NumberOfDesiredInstances	The number of apps instances that should be running
NumberOfMissingIndices	The number of missing instances. These are instances that are desired but are not heartbeating at all.
NumberOfRunningInstances	The number of instances in the STARTING or RUNNING state
NumberOfUndesiredRunningApps	The number of undesired apps with at least one instance reporting as STARTING or RUNNING
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: apiserver

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: evacuator

Metric Name	Description
LockHeld.hm9000.evacuator	Whether an evacuator holds the lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active evacuator
LockHeldDuration.hm9000.evacuator	Time the active evacuator has held the evacuator lock. Emitted periodically by the active evacuator
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

Default Origin Name: fetcher

Metric Name	Description
DesiredStateSyncTimeInMillisconds	DEPRECATED

Default Origin Name: listener

Metric Name	Description
ActualStateListenerStoreUsagePercentage	DEPRECATED
LockHeld.hm9000.listener	Whether a listener holds the lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active listener
LockHeldDuration.hm9000.listener	Time the active listener has held the listener lock. Emitted periodically by the active listener
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use

MemoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process
ReceivedHeartbeats	Number of DEA heartbeats received within an interval
SavedHeartbeats	Number of heartbeats saved to etcd within an interval

Default Origin Name: sender

Metric Name	Description
StartCrashed	Number of instances crashed in the current analysis
StartMissing	Number of instances missing in the current analysis
StartEvacuating	Number of evacuating crashed in the current analysis
StopExtra	Number of extra instances stopped in the current analysis
StopDuplicate	Number of duplicate instances crashed in the current analysis
StopEvacuationComplete	Number of instances evacuated in the current analysis

Default Origin Name: shredder

Metric Name	Description
LockHeld.hm9000.shredder	Whether an shredder holds the lock: 1 means the lock is held, and 0 means the lock was lost. Emitted periodically by the active shredder
LockHeldDuration.hm9000.shredder	Time the active shredder has held the shredder lock. Emitted periodically by the shredder analyzer
memoryStats.lastGCPauseTimeNS	Duration of the last garbage collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active goroutines in the process

[Top](#)

Metron Agent

Default Origin Name: MetronAgent

Metric Name	Description
MessageAggregator.counterEventReceived	Lifetime number of CounterEvents aggregated in Metron
MessageAggregator.httpStartReceived	Lifetime number of HTTPStart aggregated in Metron
MessageAggregator.httpStartStopEmitted	Lifetime number of HTTPStartStop events emitted by Metron (created by combining HTTPStart and HTTPStop events)
MessageAggregator.httpStopReceived	Lifetime number of HTTPStop aggregated in Metron
MessageAggregator.httpUnmatchedStartReceived	Lifetime number of HTTPStart events for which no HTTPStop was received
MessageAggregator.httpUnmatchedStopReceived	Lifetime number of HTTPStop events for which no HTTPStart was received
MessageAggregator.uncategorizedEvents	Lifetime number of non-(CounterEvent HTTPStart HTTPStop) events processed by aggregator
MessageBuffer.droppedMessageCount	Lifetime number of intentionally dropped messages from Metron's batch writer buffer. Batch writing is performed over TCP/TLS only.
DopplerForwarder.sentMessages	Lifetime number of messages sent to Doppler regardless of protocol

Metric Name	Description
dropsondeAgentListener.currentBufferCount	Instantaneous number of Dropsonde messages read by UDP socket but not yet unmarshalled
dropsondeAgentListener.receivedByteCount	Lifetime number of bytes of Dropsonde messages read by UDP socket
dropsondeAgentListener.receivedMessageCount	Lifetime number of Dropsonde messages read by UDP socket
dropsondeMarshaller.containerMetricMarshalled	Lifetime number of ContainerMetric messages marshalled
dropsondeMarshaller.counterEventMarshalled	Lifetime number of CounterEvent messages marshalled
dropsondeMarshaller.errorMarshalled	Lifetime number of Error messages marshalled
dropsondeMarshaller.heartbeatMarshalled	Lifetime number of Heartbeat messages marshalled
dropsondeMarshaller.httpStartMarshalled	Lifetime number of HttpStart messages marshalled
dropsondeMarshaller.httpStartStopMarshalled	Lifetime number of HttpStartStop messages marshalled
dropsondeMarshaller.httpStopMarshalled	Lifetime number of HttpStop messages marshalled
dropsondeMarshaller.logMessageMarshalled	Lifetime number of LogMessage messages marshalled
dropsondeMarshaller.marshalErrors	Lifetime number of errors when marshalling messages
dropsondeMarshaller.valueMetricMarshalled	Lifetime number of ValueMetric messages marshalled
dropsondeUnmarshaller.containerMetricReceived	Lifetime number of ContainerMetric messages unmarshalled
dropsondeUnmarshaller.counterEventReceived	Lifetime number of CounterEvent messages unmarshalled
dropsondeUnmarshaller.errorReceived	Lifetime number of Error messages unmarshalled
dropsondeUnmarshaller.heartbeatReceived	DEPRECATED
dropsondeUnmarshaller.httpStartReceived	Lifetime number of HttpStart messages unmarshalled
dropsondeUnmarshaller.httpStartStopReceived	Lifetime number of HttpStartStop messages unmarshalled
dropsondeUnmarshaller.httpStopReceived	Lifetime number of HttpStop messages unmarshalled
dropsondeUnmarshaller.logMessageTotal	Lifetime number of LogMessage messages unmarshalled
dropsondeUnmarshaller.unmarshalErrors	Lifetime number of errors when unmarshalling messages
dropsondeUnmarshaller.valueMetricReceived	Lifetime number of ValueMetric messages unmarshalled
legacyAgentListener.currentBufferCount	Instantaneous number of Legacy messages read by UDP socket but not yet unmarshalled
legacyAgentListener.receivedByteCount	Lifetime number of bytes of Legacy messages read by UDP socket
legacyAgentListener.receivedMessageCount	Lifetime number of Legacy messages read by UDP socket
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCpus	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
tcp.sendErrorCount	Lifetime number of errors if writing to Doppler over TCP fails
tcp.sentByteCount	Lifetime number of sent bytes to Doppler over TCP
tcp.sentMessageCount	Lifetime number of sent messages to Doppler over TCP
tls.sendErrorCount	Lifetime number of errors if writing to Doppler over TLS fails
tls.sentByteCount	Lifetime number of sent bytes to Doppler over TLS
tls.sentMessageCount	Lifetime number of sent messages to Doppler over TLS
udp.sendErrorCount	Lifetime number of errors if writing to Doppler over UDP fails
udp.sentByteCount	Lifetime number of sent bytes to Doppler over UDP
udp.sentMessageCount	Lifetime number of sent messages to Doppler over UDP

[Top](#)

Routing

Routing Release metrics have following origin names:

- [gorouter](#)
- [routing_api](#)
- [tcp_emitter](#)
- [tcp_router](#)

Default Origin Name: gorouter

Metric Name	Description
bad_gateways	Total number of bad gateways
latency	Time the Gorouter took to handle requests to its application endpoints. Emitted when the Gorouter handles requests
latency.{component}	Time the Gorouter took to handle requests from each component to its endpoints. Emitted when the Gorouter handles requests
logSenderTotalMessagesRead	Count of application log messages
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
ms_since_last_registry_update	Time in millisecond since the last route register has been received
numCpus	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
registry_message.{components}	Total number of route register messages received for each component
rejected_requests	Total number of bad requests received on gorouter
requests.{component_name}	Total number of requests received for each component
responses	Total number of HTTP responses
responses.2xx	Total number of 2xx HTTP responses
responses.3xx	Total number of 3xx HTTP responses
responses.4xx	Total number of 4xx HTTP responses
responses.5xx	Total number of 5xx HTTP responses
responses.other	Total number of other(non-(2xx-5xx)) HTTP responses
routed_app_requests	The collector sums up requests for all dea-{index} components for its output metrics.
total_requests	Total number of requests received
total_routes	Total number of routes registered

Default Origin Name: routing_api

Metric Name	Description
key_refresh_events	Total number of events when fresh token was fetched from UAA
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCpus	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
total_http_routes	Number of TCP routes in the routing table. Emitted periodically

Metric Name	Description
total_http_subscription	Number of HTTP routes subscriptions
total_tcp_routes	Number of TCP routes in the routing table. Emitted periodically
total_tcp_subscriptions	Number of TCP routes subscriptions
total_token_errors	Total number of UAA token errors
udp.sentByteCount	Lifetime number of sent bytes to Doppler over UDP
udp.sentMessageCount	Lifetime number of sent messages to Doppler over UDP

Default Origin Name: `tcp_emitter`

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCpus	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
udp.sentByteCount	Lifetime number of sent bytes to Doppler over UDP
udp.sentMessageCount	Lifetime number of sent messages to Doppler over UDP

Default Origin Name: `router_configurer` (bosch job `tcp_router`)

Metric Name	Description
{session_id}.ConnectionTime	Average connection time to backend in current session
{session_id}CurrentSessions	Total number of current sessions
AverageConnectTimeMs	Average backend response time (in ms)
AverageQueueTimeMs	Average time spent in queue (in ms)
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCpus	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
TotalBackendConnectionErrors	Total number of backend connection errors
TotalCurrentQueuedRequests	Total number of requests unassigned in queue
udp.sentByteCount	Lifetime number of sent bytes to Doppler over UDP
udp.sentMessageCount	Lifetime number of sent messages to Doppler over UDP

[Top](#)

Syslog Drain Binder

Default Origin Name: `syslog_drain_binder`

Metric Name	Description
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use

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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
pollCount	Number of times the syslog drain binder has polled the cloud controller for syslog drain bindings
totalDrains	Number of syslog drains returned by cloud controller

[Top](#)

Traffic Controller

Default Origin Name: LoggregatorTrafficController

Metric Name	Description
dopplerProxy.containermetricsLatency	Duration for serving container metrics via the containermetrics endpoint (milliseconds)
dopplerProxy.recentlogsLatency	Duration for serving recent logs via the recentLogs endpoint (milliseconds)
memoryStats.lastGCPauseTimeNS	Duration of the last Garbage Collector pause in nanoseconds
memoryStats.numBytesAllocated	Instantaneous count of bytes allocated and still in use
memoryStats.numBytesAllocatedHeap	Instantaneous count of bytes allocated on the main heap and still in use
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
numCPUS	Number of CPUs on the machine
numGoRoutines	Instantaneous number of active Goroutines in the Doppler process
Uptime	Uptime for the Traffic Controller's process
LinuxFileDescriptor	Number of file handles for the TrafficController's process

[Top](#)

User Account and Authentication (UAA)

Default Origin Name: uaa

Metric Name	Description
audit_service.principal_authentication_failure_count	Number of failed non-user authentication attempts since the last startup
audit_service.principal_not_found_count	Number of times non-user was not found since the last startup
audit_service.user_authentication_count	Number of successful authentications by the user since the last startup
audit_service.user_authentication_failure_count	Number of failed user authentication attempts since the last startup
audit_service.user_not_found_count	Number of times the user was not found since the last startup
audit_service.user_password_changes	Number of successful password changes by the user since the last startup
audit_service.user_password_failures	Number of failed password changes by the user since the last startup

[Top](#)

Deploying a Nozzle to the Loggregator Firehose

Page last updated:

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.

Note: 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](#).

```
$ 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.

```
$ 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.

```
$ 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.

```
$ 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.

```
$ 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.

```
$ bosh status --uuid
```

5. In the compiled nozzle deployment manifest, locate the `director_uuid` property. Replace `PLACEHOLDER-DIRECTOR-UUID` with your BOSH director UUID.

```
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:
`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.

```
$ bosh deployment tmp/manifest_bosh_lite.yml
Deployment set to '/Users/example_user/workspace/example-nozzle-release/templates/tmp/manifest_bosh_lite.yml'
```

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
Syncing blobs...
...
```

Step 8: Upload Nozzle BOSH Release

Run `bosh upload release` to upload the release that you created in [Step 7: Create Nozzle BOSH Release](#).

```
$ bosh upload release
Acting as user 'admin' on 'Bosh Lite Director'

Copying packages
-----
example-nozzle
golang1.4

Copying jobs
-----
example-nozzle

Generated /var/folders/4n/qs1rjbmd1c5gfb78m3_06j6r0000gn/T/d20151009-71219-17a5m49/d20151009-71219-rts928/release.tgz
Release size: 59.2M

Verifying release...
...
Release info
-----
Name: nozzle-test
Version: 0+dev.2

Packages
- example-nozzle (b0944f95eb5a332e9be2adfb4db1bc88f9755894)
- golang1.4 (b68dc9557ef296cb21c577c31ba97e2584a5154b)

Jobs
- example-nozzle (112e01c6ee91e8b268a42239e58e8e18e0360f58)

License
- none

Uploading release
```

Step 9: Deploy Nozzle

Run `bosh deploy` to deploy the nozzle.

```
$ bosh deploy
Acting as user 'admin' on deployment 'example-nozzle-lite' on 'Bosh Lite Director'
Getting deployment properties from director...
Unable to get properties list from director, trying without it...
Cannot get current deployment information from director, possibly a new deployment
Please review all changes carefully

Deploying
-----
Are you sure you want to deploy? (type 'yes' to continue):yes
```

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 `./templates/bosh-lite-stub.yml`.

```
$ 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 21:29:50 2015 from 192.168.50.1
```

2. Use the `cat` command to output the `stdout` log file.

```
$ cat /var/vcap/sys/log/example-nozzle/example-nozzle.stdout.log
===== Streaming Firehose (will only succeed if you have admin credentials)
origin:"DopplerServer" eventType:ValueMetric timestamp:1443046217700750747 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910193187 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910360012 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910252169 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910294255 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910318582 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910339088 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910379199 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046218910394886 deployment:"cf-warden" job:"loggregator_trafficcontroller_z1" index:"0" ip:"10.244.0.146" counterEvent:
origin:"router_0" eventType:HttpStartStop timestamp:1443046219105062148 deployment:"cf-warden" job:"router_z1" index:"0" ip:"10.244.0.22" httpStartStop: peerType:Client method:GET
origin:"api_z1_0" eventType:HttpStartStop timestamp:1443046219109842455 deployment:"cf-warden" job:"api_z1" index:"0" ip:"10.244.0.134" httpStartStop: peerType:Server method:GET
origin:"router_0" eventType:HttpStartStop timestamp:1443046219110064368 deployment:"cf-warden" job:"router_z1" index:"0" ip:"10.244.0.22" httpStartStop: peerType:Client method:GET
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177165446 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177288325 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177346726 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177274975 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177310389 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177330214 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177353454 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177360052 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"syslog_drain_binder" eventType:ValueMetric timestamp:1443046219177481456 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" valueMetric:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219880585603 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219880895040 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219881017888 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219881011670 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219880929574 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219881004417 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"DopplerServer" eventType:CounterEvent timestamp:1443046219880929568 deployment:"cf-warden" job:"doppler_z1" index:"0" ip:"10.244.0.142" counterEvent:
origin:"MetronAgent" eventType:CounterEvent timestamp:1443046220058280679 deployment:"cf-warden" job:"api_z1" index:"0" ip:"10.244.0.134" counterEvent:
```

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:

- Ops Metrics
- 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 Command Line Interface](#) 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.

```
$ cf add-plugin-repo CF-Community http://plugins.cloudfoundry.org
```

2. Run `cf install-plugin PLUGIN-NAME -r PLUGIN-REPO` to install the Firehose plugin from the CF Community plugin repository.

```
$ 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](#).

```
$ 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.

```
$ 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.

```
$ cf uninstall-plugin FirehosePlugin
```

Routing

Understand Routing Components and Concepts

- [\(Go\)Router](#)
- [Route Services](#)
- [Providing a Certificate for Your SSL/TLS Termination Point](#)
- [Understanding Availability Zones in VMWare Installations ↗](#)
- [Understanding the Elastic Runtime Network Architecture](#)

Manage and Troubleshoot Routing

- [Routes and Domains ↗](#)
- [Managing Cloud Foundry Domains and Routes ↗](#)
- [Enabling IPv6 for Hosted Applications ↗](#)
- [Manage Application Requests with Route Services ↗](#)
- [Securing Traffic into Cloud Foundry ↗](#)
- [Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments](#)
- [Using Your Own Load Balancer](#)
- [Creating a Proxy ELB for Diego SSH without CloudFormation ↗](#)

Pivotal Cloud Foundry Release Notes and Known Issues

Release Notes

- [Pivotal Elastic Runtime Release Notes](#)
- [Pivotal Operations Manager Release Notes](#)
- [Pivotal Cloud Foundry Ops Metrics Release Notes](#)
- [Pivotal Cloud Foundry Metrics Release Notes](#)

Known Issues

- [Pivotal Elastic Runtime Known Issues](#)
- [Pivotal Operations Manager Known Issues](#)
- [Pivotal Cloud Foundry Ops Metrics Known Issues](#)
- [Pivotal Cloud Foundry Metrics Known Issues](#)

Pivotal Elastic Runtime v1.6.0.0 Release Notes

About Updating to Elastic Runtime v1.6.48 or Later

If you are currently on Elastic Runtime v1.6.44 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 v1.6.23 or later.

Releases

1.6.80

- Bumps UAA to v2.7.4.18.

Component	Version
Stemcell	3312.24
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11.0
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.79

- Bumps UAA to v2.7.4.17.

Component	Version
Stemcell	3312.24
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11.0
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

Component	Version
-----------	---------

1.6.78

- Bumps the stemcell to v3312.24.

Component	Version
Stemcell	3312.24
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11.0
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.77

- Bumps UAA to v2.7.4.15.

Component	Version
Stemcell	3312.23
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11.0
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.76

- Bumps UAA to v2.7.4.14.
- Bumps HAProxy to v1.5.19 and PCRE to v8.40 to patch some security vulnerabilities.

Component	Version
Stemcell	3312.23
cf	225*
cf-autoscaling	36.5

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

Component	Version
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.75

- Bumps the stemcell to version 3312.23.

Component	Version
Stemcell	3312.23
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.74

- Bumps rootfs to v1.60.0 with stack 1.111.0 for low/medium security fixes
- Bumps UAA to v2.7.4.13 to patch session fixation bug

Component	Version
Stemcell	3312.22
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.60.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.73

- Bumps the stemcell version to 3312.22.

Component	Version
Stemcell	3312.22
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.56.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.72

- Configures the CAPI debug servers to bind to the loopback device instead of the open interface.

Component	Version
Stemcell	3263.21
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.56.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.71

- Bumps the rootfs to v1.56 which contains stack version 1.107.0.

Component	Version
Stemcell	3263.21
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.56.0
consul	101*
diego	0.1446.0*

Component	Version
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.70

- Bumps the stemcell to version 3263.21.

Component	Version
Stemcell	3263.21
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.69

- Secures the router debug servers by ensuring they bind to the loopback device.

Component	Version
Stemcell	3263.20
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

** Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.*

1.6.68

- Bumps the stemcell to version 3263.20.

Component	Version
Stemcell	3263.20
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.6.67

- Allows external databases to be configured with unique user accounts.

Component	Version
Stemcell	3263.17
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.6.66

- Adds configurable audit logging for the Internal MySQL database. Configuration options can be found on the Internal MySQL page.

Component	Version
Stemcell	3263.17
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1

* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

Component	Version
push-apps-manager-release	397*
* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.	

1.6.65

- Bumps the stemcell to version 3263.17 as the 3233 stemcell line is deprecated.
- 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.

Component	Version
Stemcell	3263.17
cf	225*
cf-autoscaling	36.5
cf-mysql	24.11
cflinuxfs2-rootfs	1.46.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	33
notifications-ui	26
push-apps-manager-release	397*
* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.	

1.6.64

- Bumps the stemcell to version 3233.12 to cover some low vulnerabilities.

Component	Version
Stemcell	3233.12
cf	225*
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.43.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	32
notifications-ui	26
push-apps-manager-release	397*
* Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.	

1.6.63

- Bumps the garden-runc release to version 1.1.1 to address CVE-2016-9962. For more details, please see pivotal.io/security.

Component	Version
Stemcell	3233.10
cf	225*
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.43.0
consul	101*
diego	0.1446.0*
etcd	48*
garden-runc	1.1.1
notifications	32
notifications-ui	26
push-apps-manager-release	397*

*Components marked with an asterisk have been patched to resolve security vulnerabilities or fix component behavior.

1.6.62

- Resolves an issue with the Diego Cell download cache that could result in unexpected behavior.

Component	Version
Stemcell	3233.10
cf	225
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.43.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	32
notifications-ui	26
push-apps-manager-release	397

1.6.61

- Bumps the stemcell to 3233.10 in order to address a memory usage issue in rsyslog.

Component	Version
Stemcell	3233.10
cf	225
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.43.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	32
notifications-ui	26

Component	Version
push-apps-manager-release	

1.6.60

- Bumps the stemcell to 3233.8 and the rootfs to 1.94.0 in order to address a vulnerability in APT (USN-3156-1).
- Bumps the Golang buildpack to 1.7.16 to bring in support for Golang 1.7.
- 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.

Component	Version
Stemcell	3233.8
cf	225
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.43.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	32
notifications-ui	26
push-apps-manager-release	397

1.6.59

- Patches Cloud Controller to redact service broker credentials from logs when binding a service to an application.
- Patches a vulnerability in the Autoscaling service that allowed unprivileged users to impersonate other users with unauthenticated tokens.

Component	Version
Stemcell	3233.6
cf	225
cf-autoscaling	36.2
cf-mysql	24.11
cflinuxfs2-rootfs	1.40.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	31
notifications-ui	26
push-apps-manager-release	397

1.6.58

- Bumps the stemcell to 3233.6 to address the Linux kernel vulnerability described in USN-3151-2.
- Bumps the rootfs to version 1.91.0.
- Bumps UAA to 2.7.4.12, 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.
- Patches Cloud Controller to redact logging of application environment variables.
- Patches Cloud Controller to prevent disabled buildpacks from being used pushing an application.
- Patches Cloud Controller to prevent SpaceAuditors from being able to stage applications.

Component	Version
Stemcell	3233.6
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.40.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.57

- Resolves an issue with the Diego Cell download cache that could result in unexpected behavior.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.56

- 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](#).

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0

Component	Version
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.55

- 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.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.54

- Reduced logging output in UAA of audit events.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.53

- Bumps the internal MySQL release version to use a newer version of MariaDB. The newer version of MariaDB addresses the following vulnerabilities: CVE-2016-6663 and CVE-2016-6664.

- Bumps the version of PCRE used in both haproxy and nginx to address multiple CVEs.
- See <https://pivotal.io/security> for more information.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.11
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.52

- Patches UAA to address CVE-2015-3192. For more information, please see <https://pivotal.io/security>.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.10
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.51

- Includes a patch to the Diego container runtime to improve the performance of the application of security group rules during application startup.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.10
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2

Component	Version
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.50

- 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.
- 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.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.10
cflinuxfs2-rootfs	1.35.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.2
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.49

- 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.
- Addresses an issue where iptables performance was negatively impacted for applications running on DEA VMs.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.10
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	1.0.0
notifications	24
notifications-ui	22
push-apps-manager-release	397

1.6.48

- Fixes [CVE-2016-5195](#) for more information see <https://pivotal.io/security>.
- Fixes previously broken migration that would not allow customers to upgrade from version 1.6.45.

- Fixed an inconsistency between the hostname and the contents of /etc/hosts that would cause some java apps to crash.
- UAA now allows for a custom regular expression to be set for Proxy IP's.

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	0.9.2
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.47

- Bumps the required stemcell version to 3233.3

Component	Version
Stemcell	3233.3
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	0.9.2
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.46

- Fixed an inconsistency between the hostname and the contents of /etc/hosts that would cause some java apps to crash.
- UAA now allows for a custom regular expression to be set for Proxy IP's.

Component	Version
Stemcell	3233.2
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48

Component	Version
garden-runc	0.9.2
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.45

 **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.
- Elastic Runtime v1.8.8 uses Garden-runC instead of Garden-Linux as the Garden backend for our container technology.
- Documentation for the Garden-runC runtime can be found at <https://github.com/cloudfoundry/garden-runc-release>.
- 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 downloader.
 - 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.

Component	Version
Stemcell	3233.2
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48
garden-runc	0.9.2
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.44

- Includes a fix for the Critical Vulnerability issue CVE-2016-6655. See <https://pivotal.io/security> for more information.

Component	Version
Stemcell	3232.21
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48

Component	Version
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.43

- 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.

Component	Version
Stemcell	3232.21
cf	225
cf-autoscaling	28.3
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	101
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.42

- Fixes 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.

Component	Version
Stemcell	3232.21
cf	225
cf-autoscaling	28.2
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.41

- 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>.

Component	Version
Stemcell	3232.21
cf	225

Component	Version
cf-mysql	24.8
cflinuxfs2-rootfs	1.33.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.40

- Patches CVE-2016-6651. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.17
cf	225
cf-autoscaling	28.2
cf-mysql	24.6
cflinuxfs2-rootfs	1.26.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.39

- Introduces a field in Experimental Features which allows configuring the CF CLI's CF_DIAL_TIMEOUT value for the errands which are pushing Apps Manager and running smoke tests, so that you can increase the timeout for high-latency environments.

Component	Version
Stemcell	3232.17
cf	225
cf-autoscaling	28.2
cf-mysql	24.6
cflinuxfs2-rootfs	1.26.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.38

- 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.

Component	Version
Stemcell	3232.17
cf	225
cf-autoscaling	28.2
cf-mysql	24.6
cflinuxfs2-rootfs	1.26.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.342.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.37

- 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>.

Component	Version
Stemcell	3232.17
cf	225
cf-autoscaling	28.2
cf-mysql	24.6
cflinuxfs2-rootfs	1.26.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.36

- Addresses an issue with using an LDAP server over SSL for user authentication.
- 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.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28.2
cf-mysql	24.6
cflinuxfs2-rootfs	1.19.0
consul	87
diego	0.1446.0

Component	Version
garden-linux	0.338.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.35

 **Note:** Do not upgrade to this release if you are using LDAP with SSL enabled. The next patch version will address this issue.

- Updates Consul to address an issue where the loss of a leader Consul node in the Consul cluster can lead to loss of application routes.
- Fixes an issue where Doppler servers did not add instance IDs to the syslog drain messages for applications deployed to Diego.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28
cf-mysql	24
cflinuxfs2-rootfs	1.18.0
consul	87
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.34

- 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 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 this [Pivotal Tracker story](#) for details.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28
cf-mysql	24
cflinuxfs2-rootfs	1.18.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.33

- Patches CVE-2016-5006, USN-3012-1, USN-3010-1, and CVE-2016-4450. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28
cf-mysql	24
cflinuxfs2-rootfs	1.18.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	24
notifications-ui	10
push-apps-manager-release	397

1.6.32

- Patches CVE-2016-0926. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28
cf-mysql	24
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.31

- Patches USN-3020-1. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.12
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.30

- Patches CVE-2016-0928. Additional info can be found at <https://pivotal.io/security>.
- Addresses an issue with rare occurrences of application containers being assigned duplicate IP addresses.

Component	Version
Stemcell	3232.8
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.29

- Patches CVE-2016-4468. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.8
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.28

- Patches USN-3001-1. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3232.8
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10

push-apps-manager-release	397
---------------------------	-----

1.6.27

- 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>.
- Addresses an issue with rare occurrences of application containers being assigned duplicate IP addresses.

Component	Version
Stemcell	3232.6
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.15.0
diego	0.1446.0
etcd	48
garden-linux	0.338.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.26

- Updates your Diego Cell and Diego BBS jobs to not use static IPs. This is useful if you want to scale up Diego jobs and do not have enough IP addresses reserved on your network.

Component	Version
Stemcell	3232.4
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.3.0
diego	0.1446.0
etcd	48
garden-linux	0.333.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.25

- 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](https://pivotal.io/security) topic of our documentation.
- Patches USN-2977-1. Additional info can be found at <https://pivotal.io/security>.
- Increases the robustness of etcd.
- Patches a regression bug that was introduced by Elastic Runtime 1.6.24 which reverted a patch for Diego introduced in Elastic Runtime 1.6.21 that fixed descriptor limits on listeners.

Component	Version
Stemcell	3232.4

Component	Version
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.3.0
diego	0.1446.0
etcd	48
garden-linux	0.333.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.24

- Patches USN-2977-1. Additional info can be found at <https://pivotal.io/security>. Also increases the robustness of etcd.

Component	Version
Stemcell	3232.4
cf	225
cf-autoscaling	28
cf-mysql	23
cflinuxfs2-rootfs	1.3.0
diego	0.1446.0
etcd	48
garden-linux	0.333.0
notifications	19
notifications-ui	10
push-apps-manager-release	397

1.6.23

- 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>.

Component	Version
Stemcell	3146.11
cf	225
etcd	45
notifications-ui	10
push-apps-manager-release	397
cf-autoscaling	28
cf-mysql	23
diego	0.1446.0
garden-linux	0.333.0
cflinuxfs2-rootfs	1.3.0
notifications	19

1.6.22

- Addresses a race condition experienced in some multi-node Diego DB (BBS) clusters.
- Adds the Cloud Controller statsd metrics to the firehose output.

Component	Version
Stemcell	3146.10
cf	225
push-apps-manager-release	397
cf-autoscaling	28
cflinuxfs2-rootfs	0.2.0
etcd	45
cf-mysql	23
diego	0.1446.0
garden-linux	0.333.0
notifications	19
notifications-ui	10

1.6.21

- Fixes an issue with connection flooding in the Loggregator traffic controller.
- Includes fixes for NOAA and file descriptor limits on listeners.

 **Known Issue:** A race condition experienced in some 3-node Diego DB clusters will be fixed in a follow-on release.

Component	Version
Stemcell	3146.10
cf	225
garden-linux	0.333.0
notifications-ui	10
diego	0.1446.0
etcd	27
cf-mysql	23
cf-autoscaling	28
push-apps-manager-release	397
notifications	19

1.6.20

- Patches CVE-2016-0781 and CVE-2016-2165. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.10
cf	225
notifications	19
push-apps-manager-release	397
etcd	27
notifications-ui	10
cf-autoscaling	28
garden-linux	0.333.0
diego	0.1446.0
cf-mysql	23

1.6.19

- Patches USN-2939-1. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.10
cf	225
diego	0.1446.0
notifications	19
push-apps-manager-release	397
etcd	27
notifications-ui	10
cf-autoscaling	28
cf-mysql	23
garden-linux	0.333.0

1.6.18

- Patches USN-2929-1. Additional info can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.10
cf	225
cf-autoscaling	28
diego	0.1446.0
garden-linux	0.333.0
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-mysql	23
etcd	27

1.6.17

- Patches USN-2900-1 (this one is a critical GNU C lib (glibc) CVE), [CVE-2016-0761](https://pivotal.io/security) (critical Cloud Foundry Garden CVE with respect to Docker Host File management), USN-2910-1 (high CVE in the Linux kernel), USN-2897-1, and USN-2896-1.
- Fixes an IP allocation issue with v1.6.16 where Cloud Controller would sometimes try to take the IP address of another job.
- Fixes an issue with 1.6.15 where UAA startup fails for installs that use LDAP, when upgrading from Elastic Runtime v1.5.x or v1.6.0-1.6.14.

Component	Version
Stemcell	3146.9
cf	225
cf-mysql	23
garden-linux	0.333.0
etcd	27
push-apps-manager-release	397
diego	0.1446.0
notifications	19
cf-autoscaling	28
notifications-ui	10

1.6.15

- Fixes a critical Linux kernel bug that led to unkillable AUFS container processes that make the Diego Cells and DEAs unresponsive. The bug was introduced in the previous kernel releases that addressed CVEs. The stemcell in this version of Elastic Runtime patches the bug.
- Updates the UAA, Diego, Consul, and etcd releases for increased stability and lifecycle operations. The Diego release update (now version 0.1446) allows for larger customized buildpacks to be used in your deployment.
- Addresses known issues around applications with long file names failing to stage on the Windows 2012 R2 stack. File paths are still subject to the Microsoft Windows MAX_PATH limit of 260 characters. Note that the Diego operating directory adds approximately 40 characters to the path length, in addition to anything in the application directory.

Known Issue: There is known issue with UAA startup when LDAP is configured, please prefer to the Known Issues section for more details and remediation.

Component	Version
Stemcell	3146.7
cf	225
garden-linux	0.330.0
push-apps-manager-release	397
diego	0.1446.0
etcd	18
notifications	19
cf-autoscaling	28
cf-mysql	23
notifications-ui	10

1.6.14

- Patches [CVE-2016-0732](#), [USN-2882-1](#), [USN-2879-1](#), [USN-2875-1](#), and [USN-2874-1](#). Additional info can be found at <https://pivotal.io/security>.
- It was also discovered that several CVE patches since early Dec 2015, while resolved in the BOSH stemcell, were not applied to the root filesystem of application containers running on Diego in PCF Elastic Runtime. PCF Elastic Runtime versions 1.6.5 through 1.6.13 therefore had versions of this root filesystem on Diego that remained vulnerable to several of the CVEs identified and addressed from December 2015 through January 2016. It is recommended that 1.6.13 and below installations of Elastic Runtime running Diego upgrade to 1.6.14 or higher versions to address these CVEs.

Component	Version
Stemcell	3146.6
cf	225
etcd	18
push-apps-manager-release	397
notifications-ui	10
cf-autoscaling	28
notifications	19
garden-linux	0.330.0
cf-mysql	23
diego	0.1441.0

1.6.13

- Patches [USN-2871-1](#). Additional information can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.5

Component	Version
etcd	18
cf-mysql	23
notifications	19
cf-autoscaling	28
notifications-ui	10
garden-linux	0.330.0
push-apps-manager-release	397
diego	0.1441.0

1.6.12

- Patches [CVE-2016-0715](#). Additional information can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.3
cf	225
push-apps-manager-release	397
cf-mysql	23
etcd	18
notifications-ui	10
cf-autoscaling	28
diego	0.1441.0
notifications	19
garden-linux	0.330.0

1.6.11

- Patches [CVE-2016-0708](#), [USN-2869-1](#), [USN-2868-1](#), [USN-2865-1](#), and [USN-2861-1](#). Additional information can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146.3
cf	225
push-apps-manager-release	397
cf-mysql	23
etcd	18
notifications-ui	10
cf-autoscaling	28
diego	0.1441.0
notifications	19
garden-linux	0.330.0

1.6.10

- This patch is only applicable for deployments that have not yet migrated to Diego cells from DEAs. In this patch release, the notifications and autoscale applications that come with Elastic Runtime were fixed to deploy to DEAs if you chose not to use Diego as the default runtime backend for your platform.
- All the system applications for Elastic Runtime will be deployed to whichever backend, Diego or DEAs, you select as the default in the Elastic Runtime tile. No manual migration should be necessary.

Component	Version
Stemcell	3146.2
cf	225
etcd	18
cf-mysql	23
diego	0.1441.0
notifications	19
cf-autoscaling	28
notifications-ui	10
garden-linux	0.330.0
push-apps-manager-release	397

1.6.9

- Patches [USN-2857-1](#), [USN-2842-1](#), [USN-2842-2](#), [USN-2837-1](#), [USN-2836-1](#), [USN-2835-1](#), [USN-2834-1](#), [USN-2830-1](#), and [USN-2829-1](#). Additional information can be found at <https://pivotal.io/security>.
- There is also a fix to ensure that Consul server is launched before the etcd and Diego BBS jobs, so that the Consul DNS service is ready before those jobs start.
- By default the Postgres DBs included in Elastic Runtime now default to zero instances, since we encourage every 1.6.x deployment of Elastic Runtime to not use the Postgres DBs and instead only use MySQL.

Component	Version
Stemcell	3146.2
cf	225
etcd	18
cf-mysql	23
diego	0.1441.0
notifications	19
cf-autoscaling	28
notifications-ui	10
garden-linux	0.330.0
push-apps-manager-release	397

1.6.8

- Patches a Garden CVE, [CVE-2015-5350](#).
- Fixes a few issues on Garden containers, such as the inability to run Docker images that are larger than 120 MB and a network communication issue with the Openstack GRE Tunnel network for Pivotal Cloud Foundry deployments to Openstack (the Garden network MTU was lowered from 1500 to 1454 to address this). This also addresses a DNS resolution issue with Consul that causes several CF CLI command requests to take longer than expected.
- As a result of these fixes, the OSS Cloud Foundry release for Garden was bumped to version 0.330.

Component	Version
Stemcell	3146
cf	225
cf-autoscaling	28
etcd	18
notifications-ui	10
push-apps-manager-release	397
garden-linux	0.330.0
diego	0.1441.0

Component	Version
cf-mysql	23

1.6.7

- Fixes an issue for applications running on Windows containers that have very long file paths in their application project. You don't need to upgrade to this version unless you use Windows .NET applications with long file paths.

Component	Version
Stemcell	3146
cf	225
cf-autoscaling	28
garden-linux	0.327.0
etcd	18
notifications-ui	10
push-apps-manager-release	397
diego	0.1441.0
notifications	19
cf-mysql	23

1.6.6

- Includes CVE patches for [USN-2821-1](#) and [USN-2820-1](#). Additional information can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3146
cf	225
cf-mysql	23
notifications-ui	10
cf-autoscaling	28
garden-linux	0.327.0
notifications	19
etcd	18
push-apps-manager-release	397
diego	0.1441.0

1.6.5

- Includes CVE patches for [USN-2815-1](#), [USN-2812-1](#) and [USN-2810-1](#). Additional information can be found at <https://pivotal.io/security>
- Moves Apps Usage service to an HTTPS route instead of plain HTTP.

Component	Version
Stemcell	3144
cf	225
diego	0.1441.0
etcd	18
notifications	19
cf-autoscaling	28
push-apps-manager-release	397

Component	Version
garden-linux	0.327.0
notifications-ui	10

1.6.4

- Includes CVE patches for [USN-2788-1](#), [USN-2787-1](#) and [USN-2788-2](#). Additional information can be found at <https://pivotal.io/security>.
- Includes a migration from btrfs to aufs as the filesystem for application containers on Diego, for improved performance. It also fixes an issue with Cloud Controller returning a 503 error code whenever a user requests the status of an application that is still staging.
- Updates the Docker registry API to v2 instead of v1.

Component	Version
Stemcell	3140
cf	225
diego	0.1441.0
garden-linux	0.327.0
etcd	18
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-autoscaling	28
cf-mysql	23

1.6.3

- Includes CVE patches for [USN-2806-1](#) and [USN-2798-1](#). Additional information can be found at <https://pivotal.io/security>.
- Enables setting the Health Manager instance count to zero, since Health Manager is not used by the Diego runtime components. If you are using DEAs, you should continue to keep Health Manager around.

Component	Version
Stemcell	3130
cf	222
diego	0.1437.0
garden-linux	0.308.0
etcd	16
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-autoscaling	28
cf-mysql	23

1.6.2

- Fixes an issue with application security groups not enforcing expected egress traffic rules for applications running on Diego.

Component	Version
Stemcell	3112
cf	222
diego	0.1437.0
garden-linux	0.308.0

Component	Version
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-autoscaling	28
cf-mysql	23

1.6.1

- Includes CVE patches for [USN-2778-1](#). Additional information can be found at <https://pivotal.io/security>.

Component	Version
Stemcell	3112
cf	222
diego	0.1437.0
garden-linux	0.308.0
etcd	16
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-autoscaling	28
cf-mysql	23

Changes since v1.5.0.0:

Elastic Runtime

New Features

Default Stack: cflinuxfs2

This release completely removes the lucid64 stack from [Pivotal Cloud Foundry](#) (PCF). After upgrading to this release, apps that have not been migrated to cflinuxfs2 will fail to start.

Operators are encouraged to migrate all apps to cflinuxfs2, as mentioned in the v1.4 release notes, before upgrading to v1.6.

Further details can be found [here](#).

Cloud Foundry Runtime - Diego

Diego is the new application runtime that Pivotal Cloud Foundry will use by default to run your apps.

Diego enables many new features and enhancements, such as allowing the deployment of thousands of applications and resurrecting crashed applications within seconds. It also supports new workloads, such as .NET applications on Windows, Docker on Linux, and enables SSH access to containers.

Details about how Diego works can be found [here](#). Details about its architecture can be found [here](#)

To enable SSH access to your application containers on Diego, you will need to make sure that your external load-balancer, if you have one, is listening on port 2222 and forwarding to port 2222 on the Diego Brain. Also, if you are using an external load-balancer for SSH access, separate of the one you have

pointed to the Router for general access to Cloud Foundry, you will need to add ssh.(your-domain) to your DNS, pointing to this load-balancer. Details about SSH access to application containers on Diego can be found [here](#). Instructions for setting up the SSH ELB for AWS can be found [here](#).

There is a new field, “Application Containers Subnet Pool”, which allows you to configure a specific subnet pool for your applications running on Diego. You can leave this field as its default value unless you specifically want to use a different subnet, say if you have a third-party service or other VMs running with the same IPs in your network. This feature is only usable on Diego, not on DEAs. You can only specify a single CIDR subnet, no “excluded IPs” can be specified.

Windows 2012 stack

By instantiating and setting up Windows cells in your Cloud Foundry environment, users will be able to deploy Windows .NET applications with the same familiar `cf push` commands they are used to. These applications run inside Windows containers on Windows 2012 R2 servers within your environment.

Diego must be enabled for applications to run on the new Windows 2012R2 stack. It can either be enabled by default or must be enabled for the individual application before that app can be started. Further documentation can be found [here](#).

At this time, Windows cells cannot be managed by BOSH, but server deployment can easily be integrated with any number of existing infrastructure tools.

Docker

Docker support for Pivotal Cloud Foundry is in beta currently. To enable Docker support in your deployment, you can use a CLI command as a user with the admin role: `cf enable-feature-flag diego_docker`. Further details about Docker on Diego can be found [here](#).

Security Configuration of Runtime

This section allows you to configure security settings for your Pivotal Cloud Foundry Elastic Runtime components, such as HAProxy, Router, and the DEA/Diego Cells.

The SSL Termination Certificate input now applies to both HAProxy (if you use this component) and the Cloud Foundry Router. This field is not optional, even if you use an external load-balancer instead of HAProxy, since it will be applied to the Router. It does not have to be the same certificate as the one used by your external load-balancer, as this certificate is only used for SSL traffic between the load-balancer and Router. You can enable SSL termination on the Router with a checkbox in this same section, “Enable TLS on the Router”. More details [here](#) about securing connections to the Router.

If you have multiple domains to map to the Router, such as separate system and apps domains, you can only use one SSL certificate. The Router does not yet support multiple certificates. However, one SSL certificate with multiple domains attributed to it is acceptable.

The “Enable cross-container traffic” checkbox now controls the restriction of cross-container traffic for both DEAs and Diego Cells, depending on which runtime backend platform is chosen. Pivotal does not recommend that you select this checkbox within multi-tenant environments. The feature enables using application containers to connect to other application containers directly, without going through the router. This setup may be desired for microservices, which may be optionally used with Pivotal Spring Cloud Services.

The Security Configuration section also includes fields that allow specifying the HAProxy and Router ciphers, both of which are optional fields.

Other Runtime Features

S3-Compatible Filestore Configuration

It is now possible to configure four different S3 buckets to comprise the Cloud Controller’s filesystem, instead of just one.

You can choose to use four different buckets if you are installing Pivotal Cloud Foundry for the first time, but not for an upgrade (i.e., from Pivotal Cloud Foundry v1.5 to v1.6). This is because the data will not be automatically migrated from the one bucket configuration in 1.5 to separate buckets in 1.6. It is recommended to use separate buckets for new deployments. Existing deployments are supported with one bucket.

Experimental Features

There are no experimental features being introduced in PCF Elastic Runtime for v1.6.

Improved MySQL Service

MySQL, used by some of the Pivotal Cloud Foundry system applications in the past, is now available as a data store for every Pivotal Cloud Foundry system component and application, including Cloud Controller and UAA. This is an improvement over the non-highly-available Postgresql databases that these components and applications relied upon in previous releases.

You can choose to use this MySQL service if you are installing Pivotal Cloud Foundry for the first time, but not for an upgrade (e.g.: from Pivotal Cloud Foundry v1.5 to v1.6). This is because the data will not be automatically migrated from an existing database to a new one. The Pivotal team is currently working on the migration scripts that will allow for the migration of all databases in Postgres to the HA MySQL service. We will announce when these scripts are ready for use, the timeframe of which is currently aiming for sometime during the first half of 2016.

If you do install Pivotal Cloud Foundry on the MySQL databases only, then you can scale down the Apps Manager Database (Postgres), Cloud Controller Database (Postgres), and UAA Database (Postgres) to zero instances. Those components will instead use the MySQL cluster.

API/cf CLI

Note: These features are available via API only; cf CLI support coming soon. - Org Managers and Space Managers can now manage roles without requiring admin privileges. - Max number of private domains can be specified in the Organization Quota. - Max number of app instances can be specified in the Organization Quota and Space Quota. - Admins can purge a single service instance and its bindings. This is a more targeted purge to resolve a single service instance than the purge service offering.

Apps Manager

This release adds support for the Diego Runtime, and features numerous fixes and enhancements for it.

Two new environment variables are introduced and one is removed in v1.6:

- The new `ENABLE_INTERNAL_USER_STORE` environment variable controls org and space user invitations, new user registrations, and password updates to the PCF internal user store. The default is set to `false`, which means that none of the features are exposed in Apps Manager. Set to `true` to expose these features.
- The new `ENABLE_NON_ADMIN_ROLE_MANAGEMENT` environment variable allows org managers and spaces managers to manage user roles regardless of where these users are created. The default is set to `false`, which means that none of the features are exposed in Apps Manager. Set to `true` to expose these features.
- The previous `ENABLE_NON_ADMIN_USER_MANAGEMENT` environment variable that controlled many of these same features is removed in v1.6.

DEAs

Multiple stability and performance improvements were made to DEAs. They can now handle many more simultaneous connections, for instance.

Identity (aka UAA Server)

- UAA now enforces HTTPS only communication. This was an experimental feature in PCF 1.5.x
- Added support for UAA Log Rotation via BOSH
- SAML SSO Integration Updates
 - Allow Identity Provider Metadata with missing XML Declaration tag
 - Strict checking for duplicate SAML Entity IDs
- Token Format Updates
 - UAA tokens now contain an origin field which signifies the Identity Provider used for authentication. If SAML IDP is used for authentication, the Identity Provider alias is set in as the origin value.
 - Support for nonce parameter in OAuth requests to prevent against CSRF attacks

Logging and Metrics

- Diego metrics now flow through the Loggregator subsystem.

- Collector is deprecated.
- DEA and HM9000 continue to use Collector.
- A varz nozzle is available for directing metrics coming through Loggregator to Collector.
- There is a new field, “Syslog Drain Buffer Size”, which allows you to configure the number of messages that your Doppler server will keep before starting to drop messages. Note that a larger buffer may overload your system.

Buildpacks

Smaller, More Secure Buildpacks

Buildpacks in PCF 1.6 are smaller than in PCF 1.5, since older, unsupported versions of third-party dependencies are no longer being packaged.

An example of this is Ruby 1.9.x, which reached [End Of Life on February 23, 2015](#) and was removed in [ruby-buildpack v1.4.0](#).

PCF 1.6 also is only providing the **last two** patch releases on any `major.minor` branch, with the goal of allowing an upgrade path, while also encouraging developers to upgrade to the latest security releases whenever possible.

The smaller buildpacks sizes also means a faster PCF installation, as well as improved staging performance.

For any customers who want to customize the binaries that are shipped with a buildpack, the tooling for generating custom buildpacks has been open-sourced:

- [buildpack-packager](#)
- [binary-builder](#)

Binary Buildpack

PCF 1.6 also provides a “binary buildpack”, for customers to run precompiled binaries, for those occasions when developers want a bit more control over the executables that are running in the container.

The [binary-builder](#) tooling can also be used to help build executables for a specific rootfs in a maintainable manner.

Security - CVE fixes have been implemented since v1.5.0.0 and released via security patches.

- Canonical Ubuntu USN-2765-1
- Canonical Ubuntu USN-2756-1
- Canonical Ubuntu USN-2751-1
- Canonical Ubuntu USN-2740-1
- Canonical Ubuntu USN-2739-1
- Canonical Ubuntu USN-2738-1
- Canonical Ubuntu USN-2726-1
- Canonical Ubuntu USN-2722-2
- Canonical Ubuntu USN-2718-1
- Canonical Ubuntu USN-2710-1
- Canonical Ubuntu USN-2710-2
- Canonical Ubuntu USN-2698-1
- Canonical Ubuntu USN-2696-1
- Canonical Ubuntu USN-2694-1
- Canonical Ubuntu USN-2690-1
- Canonical Ubuntu USN-2684-1
- Canonical Ubuntu USN-2670-1

- Canonical Ubuntu USN-2639-1
- CVE 2015-3281
- CVE 2015-1420
- CVE 2015-1330

Other Bug Fixes

We fixed an issue with the v1.5.6 patch for OpenStack deployments. The BOSH stemcell v3094, which this version of Elastic Runtime references, has a limitation affecting OpenStack users only:

- Elastic Runtime 1.5.6 on OpenStack does not work with S3/Swift blobstores.
- Elastic Runtime 1.5.6 on OpenStack users must configure their object storage to use the internal blobstore option.
- vSphere, AWS and vCloud users are not affected.
- This is fixed in v1.6.0

Component	Version
Stemcell	3100
cf	222
diego	0.1437.0
garden-linux	0.308.0
etcd	16
push-apps-manager-release	397
notifications	19
notifications-ui	10
cf-autoscaling	28
cf-mysql	23

Pivotal Cloud Foundry Ops Manager v1.6 Release Notes

 **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.

About Updating to Ops Manager v1.6.23 or Later

If you are currently on Ops Manager v1.6.22 or earlier, you should update your Ops Manager. Later versions include a major stemcell upgrade to 3151.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.

v1.6.31 patch release:

- Patches USN-3249-2. Additional information can be found at <https://pivotal.io/security>.

Versions
Stemcell: 3151.14
vSphere CPI: 37
AWS CPI: 44
OpenStack CPI: 23

v1.6.30 patch release:

- Patches USN-3220-2. Additional information can be found at <https://pivotal.io/security>.

Versions
Stemcell: 3151.12
vSphere CPI: 37
AWS CPI: 44
OpenStack CPI: 23

v1.6.29 patch release:

- Patches USN-3208-2. Additional information can be found at <https://pivotal.io/security>.

Versions
Stemcell: 3151.11
vSphere CPI: 37
AWS CPI: 44
OpenStack CPI: 23

v1.6.28 patch release:

- Patches USN-3161-2
- Patches USN-3169-2
- Patches USN-3172-1

Versions
Stemcell: 3151.9
vSphere CPI: 37

Versions 44

OpenStack CPI: 23

v1.6.27 patch release:

- Patches USN-3156-1.

Versions

Stemcell: 3151.6

vSphere CPI: 18

AWS CPI: 44

OpenStack CPI: 23

v1.6.26 patch release:

- Patches USN-3151-2. Additional information can be found at <https://pivotal.io/security>.

Versions

Stemcell: 3151.5

vSphere CPI: 18

AWS CPI: 44

OpenStack CPI: 23

v1.6.25 patch release:

- Patches USN-3106-2. Additional information can be found at <https://pivotal.io/security>.

Versions

Stemcell: 3151.3

vSphere CPI: 18

AWS CPI: 44

OpenStack CPI: 23

v1.6.24 patch release:

- Patches USN-3099-2. Additional information can be found at <https://pivotal.io/security>.
- Ops Manager can now be deployed on AWS M4 machine types.

Versions

Stemcell: 3151.2

vSphere CPI: 18

AWS CPI: 44

OpenStack CPI: 23

v1.6.23 patch release:

- Bumped Linux kernel to v4.4

Versions

OpenShift: 3.151.1
vSphere CPI: 18
AWS CPI: 44
OpenStack CPI: 23

v1.6.22 patch release:

- Patches USN-3087-2. Additional information can be found at <https://pivotal.io/security>.

Versions
Stemcell: 3146.23
vSphere CPI: 18
AWS CPI: 44
OpenStack CPI: 23

v1.6.21 patch release:

Versions
Stemcell: 3146.21
vSphere CPI: 18
AWS CPI: 44
OpenStack CPI: 23

v1.6.20 patch release:

- Ops Manager user sessions will now timeout after 20 minutes of inactivity
- Modified drcrypt scripts on the Ops Manager VM to prompt for decryption passphrase

Versions
Stemcell: 3146.20
vSphere CPI: 18
AWS CPI: 44
OpenStack CPI: 23

v1.6.19 patch release:

- Patches USN-3064-1. Additional information can be found at <https://pivotal.io/security>.

Versions
Stemcell: 3146.20
vSphere CPI: 18
AWS CPI: 44
OpenStack CPI: 23

v1.6.18 patch release:

- Patches USN-3020-1. Additional information can be found at <https://pivotal.io/security>.

v1.6.17 patch release:

- Patches CVE-2016-0897 and CVE-2016-0927. Additional information can be found at <https://pivotal.io/security>.
- 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 not boot up. Additional information can be found at <http://docs.pivotal.io/pivotalcf/1-6/customizing/vsphere.html>

v1.6.16 patch release:

Patches USN-3001-1. Additional information can be found at <https://pivotal.io/security>.

v1.6.15 patch release:

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>.

v1.6.13 patch release:

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>.

v1.6.12 patch release:

Ops Manager 1.6.12 includes a fix for Ubuntu Security Notice USN-2932-1. (Embedded stemcell 3146.10, Ops Manager build 2981f)

v1.6.11 patch release:

Ops Manager 1.6.11 includes a fix for Ubuntu Security Notice USN-2910-1. (Embedded stemcell 3146.9, Ops Manager build 567cac)

v1.6.10 patch release:

Ops Manager 1.6.10 includes a fix for Ubuntu Security Notice USN-2900-1 for CVE-2015-7547. (Embedded stemcell 3146.8, Ops Manager build 8d3baac)

v1.6.9 patch release:

Ops Manager 1.6.9 includes a fix for Pivotal CVE-2016-0883, Ubuntu Security Notice USN-2882-1, USN-2879-1, USN-2875-1, and USN-2874-1. (Embedded stemcell 3146.6, Ops Manager build 1468ac3)

v1.6.8 patch release:

Ops Manager 1.6.8 includes patches for Ubuntu Security Notice USN-2870-2. (Embedded stemcell 3146.5, Ops Manager build b69cad)

v1.6.7 patch release:

Ops Manager 1.6.7 includes patches for Ubuntu Security Notice USN-2869-1. (Embedded stemcell 3146.3, Ops Manager build 234cf4d)

v1.6.6 patch release:

Ops Manager 1.6.6 includes patches for Ubuntu Security Notice USN-2857-1. (Embedded stemcell 3146.2, Ops Manager build 236fc1)

v1.6.5 patch release:

Ops Manager 1.6.5 includes patches for a bug that can render important information in log files as a series of asterisks. (Embedded stemcell 3146 - same as 1.6.4, Ops Manager build c79b317)

v1.6.4 patch release:

Ops Manager 1.6.4 includes patches for Ubuntu Security Notices USN-2821 and USN-2820-1. (Embedded stemcell 3146, Ops Manager build 511c28f)

v1.6.3 patch release:

Ops Manager 1.6.3 includes patches for Ubuntu Security Notices USN-2815-1, USN-2812-1 and USN-2810-1. (Embedded stemcell 3144, Ops Manager build c47f94)

v1.6.2 patch release:

Ops Manager 1.6.2 includes patches for Ubuntu Security Notices USN-2806-1 and USN-2798-1. (Embedded stemcell 3130, Ops Manager build 6042e5)

v1.6.1 patch release:

Ops Manager 1.6.1 is the security patch rollup for November 2015. (Embedded stemcell 3112)

Version 1.6.0 (changes since v1.5.5)

Major Features

- EBS encryption is now available for AWS deployments. This enables AWS customers to have encrypted data at rest on persistent disks.

API Changes

- Previously, Ops Manager provided an API endpoint called `/api/users` that allowed for the creation of multiple user accounts, though only the first one was functional. To correct this, `/api/users` has been replaced by `/api/setup` that provides the same functionality for creating the first user, but does not attempt to create secondary, tertiary, etc., users, since these user accounts would not be functional. Please see API documentation at <http://your-ops-man-ip/docs> for more information. **Please note that the API is not officially supported and may change without warning in the future.**

Security Features

- Multiple CVE fixes per pivotal.io/security.
- VM passwords are now scrubbed from the output logs and replaced with asterisks.

Minor Features

- On vSphere, the operator is now able to split availability zones by resource pool in the same cluster.

Bug Fixes

- Credentials no longer shows incorrect credentials when “Use default BOSH password” is selected.

- Upgrades no longer fail if product version contains a dash.
- A migration that exists but is null no longer results in a 500 error.

Product Author Features

- New product author documentation has been made available here: <http://docs.pivotal.io/pivotalcf/packaging/index.html>

Notes

Embedded stemcell in 1.6.0 is 3100

Pivotal Cloud Foundry Ops Metrics v1.6.X Release Notes

Version 1.6.33

Release Date: April 26, 2017

- Ops Metrics 1.6.33 - now with stemcell 3263.24
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.32

Release Date: March 31, 2017

- Ops Metrics 1.6.32 - now with stemcell 3263.22
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.31

Release Date: March 10, 2017

- Ops Metrics 1.6.31 - now with stemcell 3263.21
- Maintenance update of the following product dependencies:
 - Updates the OpenJDK version used in the product to v1.8.0.121
 - Updates the golang version used in the product to v1.8
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.29

Release Date: February 24, 2017

- Ops Metrics 1.6.29 - now with stemcell 3263.20
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.28

Release Date: January 24, 2017

- Ops Metrics 1.6.28 - now with stemcell 3263.17
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.27

Release Date: December 15, 2016

- Ops Metrics 1.6.27 - Fixes a potential issue in v1.6.25 and v1.6.26 where the tile may fail to perform as expected if the optional Enable SSL feature was active. Although no customer issues were reported, v1.6.27 is a corrective patch to prevent any possible performance risk when using the Enable SSL feature
- Stemcell remains 3263.13
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.26

Release Date: December 14, 2016

- Ops Metrics 1.6.26 - now with stemcell 3263.13
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.25

Release Date: December 12, 2016

- Maintenance update of the following product dependencies:
 - SLF4J now v1.7.21
 - Logback now v1.1.7
 - Guava now v20.0
 - Bouncy Castle bcprov-jdk15on now v1.55
- Stemcell remains 3263.12
- For Known Issues, please see the [Ops Metrics Known Issues](#) documentation

Version 1.6.24 and Earlier

Known issues

For Known Issues, please see the [Ops Metrics Known Issues](#) documentation. For Ops Metrics 1.6.X there is one important install ordering dependency before enabling the OpenTSDB firehose nozzle job.

Major Features

- The addition of CF loggregator data via the firehose is still available, but disabled by default.
 - This allows the user to install Ops Metrics without Elastic Runtime to monitor Bosh data.
 - The nozzle should not be enabled until after the user successfully deploys Elastic Runtime (see known issues)
- Ops Metrics still supports the firehose in addition to Bosh and the Collector for data.
- Ops Metrics now asks Elastic Runtime for the port that firehose data comes from.

Bug Fixes

- Firehose data ingest disabled by default to avoid deadlock condition if Ops Metrics installed before Elastic Runtime.
- Firehose no longer defaults to using 4443 to read data, and instead asks Elastic Runtime for the port the firehose comes from.

Notes

- Upgrading from Ops Metrics 1.4.X to 1.6.X will leave the firehose nozzle job disabled (since it did not exist previously).
- Upgrading from Ops Metrics 1.5.X to 1.6.X will enable the firehose nozzle by default (since it was enabled before upgrading).
 - If you are upgrading to resolve an installation error with 1.5.X, immediately disable the firehose job before deploying, and enable once you have deployed Elastic Runtime.
- Stemcell for 1.6.24 is 3263.12

Pivotal Cloud Foundry Metrics Release Notes

Version 1.0.18

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3263.8

Version 1.0.17

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3233.4

Version 1.0.16

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3233.2

Version 1.0.15

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3233.1

Version 1.0.14

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3232.21

Version 1.0.13

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Release bumps required CLI to CLI v6.21.1 and configures CLI environment variable CF_DIAL_TIMEOUT to 30 seconds to reduce chance of CLI errors when network connection attempts include slow DNS resolution times

Version 1.0.11

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3232.17

Version 1.0.10

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- BUG FIX: Corrects issue in 1.0.9 where the UI errand was not properly bumped to CLI v6.19, causing UI push to fail in some environments.

Version 1.0.9

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3232.12

Version 1.0.8

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- This release fixes a route issue in prior versions of PCF Metrics v1.0 where you could receive a “404 Not Found: Requested route does not exist in PCF Metrics” error.

Version 1.0.4

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Update to use Stemcell 3146.11
- Change list of apps available to a user and prevent overload of Cloud Controller API
- The route for the PCF Metrics application is registered as `metrics.<system-domain>` in this v1.0.4 release. The application can be accessed at `https://metrics.<system-domain>`
- The install will create a small deployment suitable for hundreds of apps. This is not configured for thousands of apps.

Version 1.0.3

Known issues

For Known Issues, please see [PCF Metrics Known Issues](#).

Notes

- Updated to use stemcell 3146.5
- This beta requires Elastic Runtime v1.6.16 or greater.
- The route for the PCF Metrics application is registered as `metrics.<system-domain>` in this v1.0.3 release. The application can be accessed at `https://metrics.<system-domain>`
- The install will create a small deployment suitable for hundreds of apps. This is not configured for thousands of apps.

Pivotal Elastic Runtime v1.6 Known Issues

New Issues

- **Elastic Runtime 1.6.35 to 1.6.42:** 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 1.6.24 upgrades:** Upgrading Elastic Runtime to or from this version may not be accepted by bosh. If you have Elastic Runtime 1.6.24 added to your PCF environment but not successfully installed, you should remove it. If it is installed, you can upgrade to later versions but you may have to `bosh delete` a release package that bosh believes has a conflicting file hash.
- **Elastic Runtime 1.6.15 UAA with LDAP startup:** There's a UAA startup issue for customers using the LDAP user store when they upgrade from Elastic Runtime 1.5.x or v1.6.0-1.6.14 to 1.6.15. The workaround is to manually delete the LDAP provider from the UAA DB by issuing the following query `DELETE FROM identity_provider WHERE type='ldap'`, and then upgrade to 1.6.15. This issue was resolved in Elastic Runtime 1.6.16.
- **Zombie processes:** With the Elastic Runtime 1.6.13, 1.6.14, and 1.5.13 releases, there's a linux kernel bug that affects Garden, Docker, and almost certainly Warden for Cloud Foundry deployments on stemcell 3146.5. It manifests in containers failing to be killed properly leaving unkillable zombie processes around consuming lots of CPU and some memory. After a while, Diego cells / DEAs may experience slower performance and eventually may even become unresponsive. The current workaround is to restart these cells / DEAs before they slow down so far as to become unresponsive, or once they become unresponsive. This issue was resolved in Elastic Runtime 1.6.15 and 1.5.14.
- **Diego BBS failure:** Etcd may sometimes experience orphaned processes that leads to the Diego BBS job failing. This can be remedied by `killall etcd` on the Diego BBS VMs and subsequent redeployment. This will be fixed in a future 1.6.x release soon.
- **Version label in tile:** The [Pivotal Cloud Foundry](#) (PCF) Elastic Runtime tile v.1.6.0 will show “1.6.0-build.315” in the tile version. This is fine.
- **v.1.5.x to v.1.6.0 upgrades:** Before upgrading to Elastic Runtime v.1.6.0 from v.1.5.x, make sure you have followed [these instructions](#).
- **No Notifications and Autoscale services on DEAs** The Notifications and Autoscale services in Elastic Runtime will always deploy on Diego, even if DEAs are selected as the default backend, in versions 1.6.0 to 1.6.9 (this was fixed in patch 1.6.10). The Apps Manager and Apps Usage service will deploy to Diego only if the “Use Diego by default” checkbox is enabled in the configuration of Elastic Runtime.
- **Diego cell push timeout with no `-b`:** 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.
- **Slow deploys when Diego cells have high instance counts:** Cloud Foundry installations in which Diego cells host over 30 app containers may experience degraded performance. This behavior manifests as:
 - Slow deployments, in which some Diego cells take up to 15 minutes to drain.
 - Reduced app performance and increased crash rates during rolling deployment.
 - Possible brief outages of single-instance apps during rolling deployment.

This issue stems from the scheduling algorithm used in PCF 1.6 Diego, and is resolved in PCF 1.7 and later.

- **Slow Diego cell performance:** Diego cells may sometimes experience slow performance for various actions like staging new applications. If you experience slow performance, you can SSH into the Ops Manager VM and `bosh-recreate` the affected Diego Cell VM.
 - **Docker library image names:** Docker library images require you to specify `library/` in front of the docker image name. E.g. `cf p jetty -o library/jetty`
 - **Diego cell disk space with lots of Docker:** Diego Cells may run out of disk space if they are using a lot of different Docker images with different layers.
 - **`cf files` on Diego:** The CLI command for viewing application files, `cf files`, does not work with applications on Diego.
 - **Diego BBS handshake error in logs:** The Diego BBS may output a TLS handshake error every three seconds in the sys logs. This error is fine to ignore, and will be fixed in a future release of Diego.
 - **Space Developer role to SSH:** To SSH into an application container on Diego, your CF user must have the Space Developer role attached to it for the application space.
 - **Smoke Test failures:** The Smoke Tests may occasionally fail at the logging test suite. The Smoke Tests errand is fine to re-run in case of any failures.
 - **Postgres version:** The version of Postgres has been upgraded to 9.4.2. If you have Postgresql databases in your deployment, Cloud Controller and UAA will be unavailable during the upgrade. In case that upgrade fails, there are mitigations [documented here](#).
 - **Blobstore space:** After you upgrade to PCF 1.6, you may want to reclaim some blobstore space due to some orphaned blobs. To reclaim space, call [this end point](#) to delete all buildpack caches from the blobstore after the v1.6 upgrade is complete.
 - **CCDB events table migration:** This upgrade includes a migration that modifies the events table on the Cloud Controller database. This table may be very large, and the migration may cause the upgrade to fail at the Cloud Controller job if it takes too long. If this happens, do not restart the deploy until the migration has finished running. The deploy can be restarted once the `space_id` foreign key constraint has been removed from the events table.
- To avoid the possibility of the migration causing a failure, truncate the events table before the deployment starts. The data in the events table are audit and log data, and PCF can function without it.
- **.NET security:** .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.

- **Windows container permissions:** At this time, the container accounts on Windows cells must have permissions to log on locally, which may not be the default.
- **Windows filename length:** 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.
- **Blue-Green deploy from Apps Manager:** Apps Manager now has a zero-downtime blue green deploy. As of v1.6, any `NON_ADMIN` environment variable changed from the default setting will not be applied to the second app version. You will need to set `NON-ADMIN` environment variables on both versions in order to keep the configuration consistent.
- **Diego Beta tile:** If you have installed the Diego Beta tile in a 1.5.x Elastic Runtime environment, it will need to be uninstalled prior to upgrade. The Windows MSI provided with the 1.6 Elastic Runtime release will not work with the older Diego Beta tile.

Existing Issues

- 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.
- Recent versions of the cf CLI, including the version shipped with v1.6, will report a version mismatch with Elastic Runtime. You can ignore this message, and there is no need to upgrade the CLI. This error will be fixed in a future version of the cf CLI.

Pivotal Cloud Foundry Ops Manager v1.6.0 Known Issues

New Issues

Some of these issues may be fixed in subsequent patch releases to 1.6. Consult the Ops Manager 1.6 [Release Notes](#) for more information.

- All 1.6 versions of Ops Manager through 1.6.4 contain a bug that can render important information in log files as a series of asterisks. The bug stems from an over-aggressive algorithm to address any risk of credentials appearing in log files. The bug is fixed in version 1.6.5.
- **If you change the password in Ops Man 1.6.x, you must remember your original password.** Because of a bug in Ops Man 1.6, you must remember your original password (the one you chose at installation time, or retained at upgrade time) in order to upgrade to later versions of PCF such as 1.7 or beyond. Immediately before performing an upgrade, you must change your password back to its original value.
- If EBS encryption is initially disabled, and then later enabled, the change will not apply to the Director's persistent disk.
- The new "Trusted Certificates" feature does not insert the certificate (or certificate chain) that you provide into your application containers.
- The new "Trusted Certificates" feature does not insert the certificate (or certificate chain) that you provide into UAA's JVM store (which would be necessary to enable UAA -> SASL trust).
- Once the above issues with "Trusted Certificates" are resolved, "Trusted Certificates" will be moved out of the Experimental Features section.
- The "Generate Self-Signed Certificate" link no longer generates a self-signed certificate. (The certificate it generates is signed by Ops Manager's internal CA, which is generated once at installation time for each Ops Manager and persists across exports/imports.) The link will be renamed "Generate Certificate" in future versions of Ops Manager.

Existing Issues

- Ops Manager 1.5 does not support AWS S3 buckets in eu-central-1 (Frankfurt) due to an issue with Amazon's authentication APIs in this region. Therefore, Pivotal does not recommend customers use the eu-central-1 region for deployments that require high performance.
- On AWS environments, it is only possible to use "light-bosh" stemcells.
- If you are using OpenStack Juno, make sure you have applied this patch: <https://bugs.launchpad.net/horizon/+bug/1394051>
- DNS and Gateway ICMP errors will occur on AWS even if ICMP is allowed.
- Resource Pools cannot be deleted from Availability Zones. If a resource pool is inadvertently deleted, run BOSH recreate.
- Compiled Packages are missing from installation exports, which can cause a failure if the product contained and referenced them. A fix is to SSH into the Ops Manager VM and remove references to compiled_packages in every file located in the `/var/tempest/workspace/metadata` directory.
- Unnecessary job instances must be reduced manually (MySQL, HA Proxy if RDS & ELBs are used).

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.

Pivotal Cloud Foundry Ops Metrics v1.6.X Known Issues

New Issues for 1.6.X

Some of these issues may be fixed in subsequent patch releases to 1.6. Consult the Ops Metrics 1.6 [Release Notes](#) - any additional fixes will be added immediately.

For releases earlier than 1.6.17:

- An increase in disconnects between the OpenTSDB Firehose Nozzle and the Firehose, resulting in a need to restart the nozzle, may be due to the NOAA client becoming too far out-of-date. Version 1.6.17 updates the NOAA client and other service related dependencies.

For all 1.6.X releases:

- To fix the possibility of installation deadlock with Elastic Runtime 1.6.X, the OpenTSDB Firehose Nozzle job (responsible for receiving Diego metric data) has been disabled by default.
- There is nothing that prevents the user from turning on the Nozzle deployment when Elastic Runtime is not present.
 - Enabling the nozzle when Elastic Runtime is not deployed or enabled will produce the following error:
 - “RuntimeError - unknown product ‘cf’ in ((..cf.cloud_controller.system_domain.value))”
 - To fix this, reduce the number of nozzle counts to zero until Elastic Runtime is enabled.
- Because deploying the OpenTSDB firehose nozzle is now optional, smoke tests for Elastic Runtime have been disabled until a later release.

For the 1.6.0 release:

- Leaving the port blank in Elastic Runtime would cause the OpenTSDB firehose nozzle to try to listen to port 0. To fix, enter port 443 in Elastic Runtime or use Ops Metrics 1.6.1.

Pivotal Cloud Foundry Metrics Release Notes

Version 1.0.18

Known Issues

- For applications that have had no activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A workaround for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller. If you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege for specific spaces, you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.17

Known Issues

- For applications that have had no activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A workaround for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller. If you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege for specific spaces, you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.16

Known Issues

- For applications that have had no activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see

any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.

- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A work around for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller. If you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege for specific spaces, you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.15

Known Issues

- For applications that do not have any activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A work around for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller. If you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege for specific spaces, you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.14

Known Issues

- For applications that do not have any activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A work around for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller. If you have cloud.controller.admin

privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege for specific spaces, you can only see the apps in those spaces in PCF Metrics UI.

- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.13

Known Issues

- For applications that do not have any activity in the past 24 hours, which means no app events or HTTP requests have been generated, users do not see any data in the UI. Users must refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. Adjust them manually to fit your environment.
- All v1.0.x tiles - if the apps dropdown is grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A work around for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if some apps do not appear in the drop down, even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller so if you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege to certain spaces then you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.11

Known Issues

- Due to a stemcell upgrade issue in earlier versions of Ops Manager, you must be on Ops Manager 1.7.8 or later to upgrade to PCF Metrics 1.1
- Due to a known limitation with deep pagination in Elastic Search, if the results of your log query exceed 10k, we are capping the result set that returns to the UI at 10k and disabling the needle navigation interaction. We are working to overcome this issue in the next version of the product. In the meantime, if you are impacted by the 10k result cap on your logs data, you can do the following to ensure more targeted queries:
 - reduce your viewed timeslice to an hour or a minute
 - use the available search filters
 - potentially change the timestamp filter from newest to oldest
- On the Explore view, when using the needle navigation interaction on the metrics graphs to jump to a particular point in time of your application logs, you may occasionally see the needle snap to a slightly different time stamp than the time selected. This can occur when there are fewer log lines available for the chosen time, and the needle instead snaps to a nearby time. If you experience this issue, we suggest that you scroll within the logs view to the desired time, at which point you should also see the needle move itself to the same time stamp.
- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.

Version 1.0.10

Known Issues

- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.
- All v1.0.x tiles - if you see apps dropdown grayed out:
 - Because of an earlier CLI bug, we have seen PCF metrics apps missing system org guid during tile upgrade/uninstall process which crashes PCF Metrics UI and makes the apps dropdown inaccessible. This is fixed in 1.1 Metrics tile. A work around for this to remove the user's access to system org using CLI.
- All v1.0.x tiles - if you don't see some apps in the dropdown even though they are accessible in apps manager:
 - If you have apps manager/console admin privilege, you may be able to see apps in apps manager without having space developer role privilege to the spaces in which those apps reside. Metrics however follows permissions directly from cloud controller so if you have cloud.controller.admin privilege, you can see all apps in all spaces in all orgs in PCF Metrics UI. If you have space developer privilege to certain spaces then you can only see the apps in those spaces in PCF Metrics UI.
- All v1.0.x tiles - SSO authentication is not supported. This feature is added in the 1.1 Metrics tile.

Version 1.0.9

Known Issues

- BUG IDENTIFIED: An errand was missed when requiring the CLI v6.19 bump, leaving the UI push errand still on v6.18.1. This bug has caused the UI push to fail in some environments. Metrics v1.0.10 corrects.
- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.

Version 1.0.8

Known Issues

- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.
- No direct upgrade path. Operators can delete the installed tile and install the new tile. This does result in downtime and data loss.

Version 1.0.4

Known Issues

- Route mappings for metrics. and metrics./v2 may be overwritten or incorrectly mapped during redeploys.
- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.
- No direct upgrade path. Operators can delete the installed tile and install the new tile. This does result in downtime and data loss.

Version 1.0.3

Known Issues

- Not all applications metrics are accessible to admin users.
- For applications that do not have any activity in the past 24 hours (which means no app events or http requests have been generated), users will not see any data in the UI. They will also need to refresh their browser after metrics are generated by the use of the app.
- While installing the PCF Metrics tile, the default Resource Config settings for each resource pool may be over-sized for POC/Beta purposes. You may adjust them manually to fit your environment.