

Table of Contents

Table of Contents	1
Pivotal Cloud Foundry Documentation	5
Getting Started with Pivotal Cloud Foundry	6
Using Operations Manager	10
Prerequisites to Deploying Operations Manager and Elastic Runtime	12
Preparing Your Firewall for Deploying PCF	15
Using Your Own Load Balancer	17
Using Ops Manager Resurrector on VMware vSphere	19
Connecting Elastic Runtime to LDAP	21
Understanding Availability Zones and Regions in Pivotal Cloud Foundry	24
Configuring Network Isolation Options in Ops Manager	26
Configuring AWS for PCF	36
Deploying Operations Manager to vSphere	52
Provisioning a Virtual Disk in vSphere	55
Deploying Operations Manager to vCloud Air and vCloud	57
Understanding the Ops Manager Interface	66
Adding and Deleting Products	67
Configuring Ops Manager Director for AWS	69
Configuring Ops Manager Director for VMware vSphere	76
Configuring Ops Manager Director for vCloud Air and vCloud	82
Configuring Elastic Runtime for AWS	87
Configuring Elastic Runtime for vSphere and vCloud	94
Creating New Elastic Runtime User Accounts	104
Logging into the Apps Manager	105
Controlling Apps Manager User Activity with Environment Variables	106
Configuring Your App Autoscaling Instance	107
Managing Scheduled Scaling in the App Autoscaling Service	110
Backing Up Pivotal Cloud Foundry	114
Upgrading Operations Manager	123
Upgrading Products in a PCF Deployment	126
Monitoring VMs in PCF	127
Deploying Pivotal Ops Metrics	129
Using SSL with a Self-Signed Certificate in Pivotal Ops Metrics	134
Using Pivotal Ops Metrics	136
PCF Troubleshooting Guide	138
Troubleshooting Ops Manager for VMware vSphere	145
Advanced Troubleshooting with the BOSH CLI	148
PCF Security Overview and Policy	154
Cloud Foundry Concepts	156
Cloud Foundry Overview	157
Cloud Foundry Components	158
(Go)Router	160
Getting started	160
User Account and Authentication (UAA) Server	162
Cloud Controller	168

Droplet Execution Agent	170
Warden	173
Messaging (NATS)	178
How Applications Are Staged	180
Zero Downtime Deployment and Scaling in CF	181
Orgs, Spaces, Roles, and Permissions	184
Understanding Cloud Foundry Security	186
Stacks	191
Glossary	193
Operator's Guide	194
Isolating a PCF Deployment with a Private Network	195
Understanding the Elastic Runtime Network Architecture	198
Load Balancer	198
Router	198
Configuring PCF SSL Termination	199
Changing the Quota Plan of an Organization with cf CLI	201
Identifying Elastic Runtime Jobs Using vCenter	204
Understanding the Effects of Single Components on a PCF Upgrade	206
Configuring Single Sign-On	208
Getting Started with the Notifications Service	211
Administering Cloud Foundry	213
Adding Buildpacks to Cloud Foundry	214
Creating and Managing Users with the cf CLI	216
Creating and Managing Users with the UAA CLI (UAAC)	218
Application Security Groups	222
Using the Apps Manager	224
Getting Started with the Apps Manager	225
Understanding Apps Manager Permissions	226
Managing Spaces Using the Apps Manager	227
Managing User Accounts in Spaces Using the Apps Manager	229
Managing User Permissions Using the Apps Manager	231
Controlling Apps Manager User Activity with Environment Variables	233
Developer Guide	234
Considerations for Designing and Running an Application in the Cloud	236
Deploy an Application	238
Deploying a Large Application	242
Creating Domains and Routes	244
Domains and Shared Domains	249
Deploying with Application Manifests	251
Cloud Foundry Environment Variables	259
Using Blue-Green Deployment to Reduce Downtime and Risk	266
Application Logging in Cloud Foundry	269
Troubleshooting Application Deployment and Health	272
Identifying the API Endpoint for your Elastic Runtime Instance	277
cf Command Line Interface (CLI)	278
Installing the cf Command Line Interface	279
Getting Started with the cf CLI	280

Using the cf CLI with an HTTP Proxy Server	285
Using cf CLI Plugins	288
About Starting Applications	290
Scaling an Application Using cf scale	291
Services Overview	292
Adding a Service	294
Binding a Service Instance	296
Managing Service Instances with the CLI	297
Third-Party Managed Service Instances	299
User-Provided Service Instances	301
Configuring Play Framework Service Connections	304
Migrating a Database in Cloud Foundry	305
Using Third-Party Log Management Services	307
Configuring Selected Third-Party Log Management Services	309
Integrating Cloud Foundry with Splunk	314
Buildpacks	316
Buildpack Detection	317
Custom Buildpacks	318
Packaging Dependencies for Offline Buildpacks	321
Java Buildpack	323
Getting Started Deploying Grails Apps	324
Getting Started Deploying Ratpack Apps	331
Getting Started Deploying Spring Apps	338
Tips for Java Developers	345
Configure Service Connections for Grails	351
Configure Service Connections for Play Framework	354
Configure Service Connections for Spring	355
Cloud Foundry Eclipse Plugin	362
Cloud Foundry Java Client Library	377
Build Tool Integration	379
Node.js Buildpack	383
Tips for Node.js Applications	384
Environment Variables Defined by the Node Buildpack	387
Configure Service Connections for Node.js	388
Ruby Buildpack	390
Getting Started Deploying Ruby Apps	391
Getting Started Deploying Ruby on Rails Apps	396
Deploy a Sample Application	398
Configure a Production Server for Ruby Apps	400
Configure Rake Tasks for Deployed Apps	401
Tips for Ruby Developers	402
Environment Variables Defined by the Ruby Buildpack	408
Configure Service Connections for Ruby	409
Services	412
Overview	413
Service Broker API	414
Managing Service Brokers	425

Access Control	428
Catalog Metadata	433
Binding Credentials	437
Dashboard Single Sign-On	439
Example Service Brokers	443
Application Log Streaming	444
Packaging Pivotal Cloud Foundry Products	445
Understanding Lifecycle Errands	462
Decrypting and Encrypting Installation Files	464
Pivotal Cloud Foundry Release Notes and Known Issues	466
Pivotal Elastic Runtime v1.4.0.0 Release Notes	467
Ops Manager 1.3 Release Notes	475
Pivotal Cloud Foundry Apps Manager v1.4.0.0 Release Notes	477
Pivotal Elastic Runtime v1.4 Known Issues	478
Ops Manager 1.3 Known Issues	479
Pivotal Cloud Foundry Apps Manager v1.4.0.0 Known Issues	480

Pivotal Cloud Foundry Documentation

1. [Getting Started](#)
A quick guide to installing and getting started with [Pivotal Cloud Foundry](#) (PCF). If you are new to PCF, start here.
2. [Using Ops Manager](#)
A guide to using the Pivotal Cloud Foundry Operations Manager interface to manage your PCF PaaS.
3. [Elastic Runtime Concepts](#)
An explanation of the components in Pivotal Cloud Foundry Elastic Runtime and how they work.
4. [Operating Elastic Runtime](#)
A guide to running the Elastic Runtime component of PCF.
5. [Administering Elastic Runtime](#)
A guide to managing Elastic Runtime at the administrator level.
6. [Using the Apps Manager](#)
A guide to using the web-based Apps Manager application for managing users, organizations, spaces, and applications.
7. [Deploying Applications](#)
A guide for developers on deploying and troubleshooting applications running in Elastic Runtime (Cloud Foundry).
8. [Buildpacks](#)
A guide to using system buildpacks and extending your Elastic Runtime with custom buildpacks.
9. [Custom Services](#)
A guide to extending your Elastic Runtime with custom services.
10. [Packaging Pivotal Cloud Foundry](#)
A guide to packaging PCF products.
11. [Release Notes](#)
Release notes and known issues for Pivotal Operations Manager, Pivotal Elastic Runtime and Pivotal MySQL Developer.

Getting Started with Pivotal Cloud Foundry

Welcome to Pivotal Cloud Foundry!

This guide is intended to walk you through deploying the installation virtual machine, setting up your PaaS, targeting Pivotal Cloud Foundry Elastic Runtime, and pushing your first app. If you experience a problem while following the steps below, try checking the Known Issues, or refer to the Troubleshooting Guide for more tips. 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.

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 Installation Virtual Machine

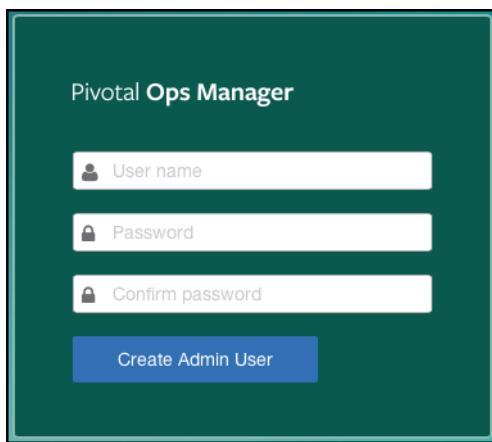
Download the PCF .ova file and deploy it. The procedure you follow depends on the IaaS you use:

- [Deploying Operations Manager to vSphere](#)
- [Deploying Operations Manager to vCloud Air and vCloud](#)
- [Deploying Operations Manager to Amazon Web Services](#)

Step 3: Set Up Your PaaS

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. Browse to the interface IP address you specified in Step 2.



2. Create a **User name** and **Password** and log in to access the interface.

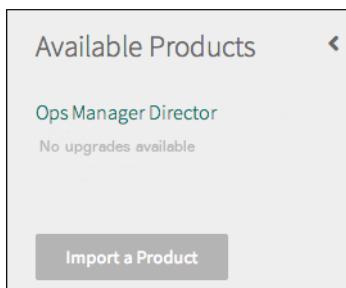
Note: On your first login attempt, an error message that the connection is untrusted appears because you are attempting to connect securely to a website with a self-signed certificate. Add Ops Manager as an exception to bypass this message on subsequent logins.

Import Products

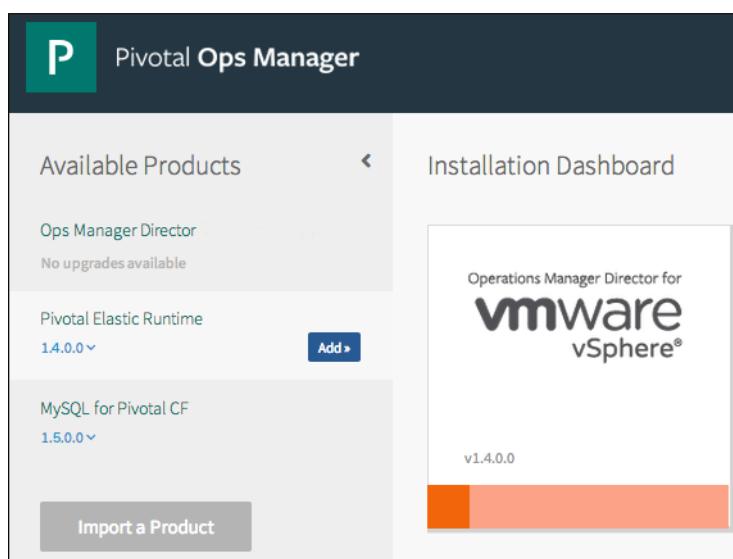
1. Download Elastic Runtime and MySQL for PCF at [Pivotal Network](#).
 - Elastic Runtime: Click the **Pivotal Cloud Foundry** banner to access the PCF product page. Use the drop-down menu to select an Elastic Runtime release.
 - MySQL for PCF: Select the **All Products** tab to view available products. 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.



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](#)
- [Configuring Ops Manager Director for Amazon Web Services](#)

Configure Elastic Runtime

Refer to one of the following topics for help configuring Elastic Runtime:

- [Configuring Elastic Runtime for AWS](#)
- [Configuring Elastic Runtime for vSphere and VCloud Air](#)

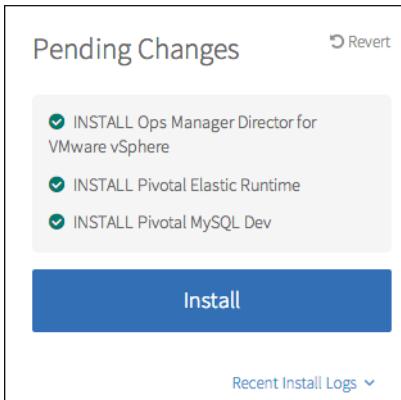
Configure MySQL

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**.

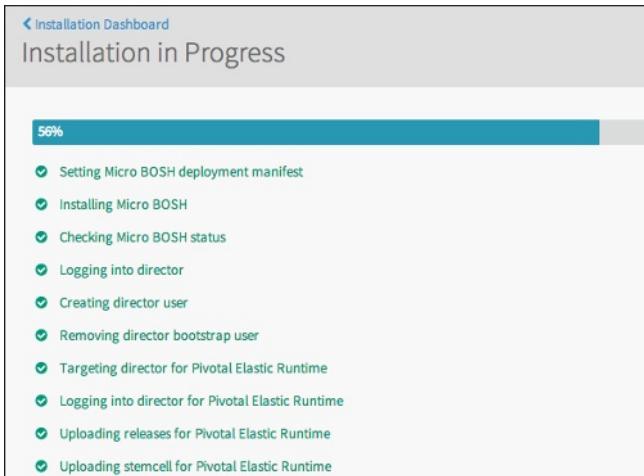
3. Click the **Installation Dashboard** link to return to the Installation Dashboard.

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. The following sections will help you start using your PaaS.

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 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 Operations 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

Operations Manager is a web application that you use to deploy and manage a [Pivotal Cloud Foundry](#) PaaS. This is a guide to deploying and using Ops Manager.

Before You Deploy

- [Prerequisites to Deploying Operations Manager and Elastic Runtime](#)
- [Preparing Your Firewall for Deploying PCF](#)
- [Using Your Own Load Balancer](#)
- [Using Ops Manager Resurrector on VMware vSphere](#)
- [Connecting Elastic Runtime to LDAP](#)
- [Understanding Availability Zones and Regions in Pivotal Cloud Foundry](#)
- [Configuring Network Segmentation in Ops Manager](#)

Deploying Operations Manager

- [Configuring AWS for PCF](#)
- [Deploying Operations Manager to vSphere](#)
- [Provisioning a Virtual Disk in vSphere](#)
- [Deploying Operations Manager to vCloud Air and vCloud](#)

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_IP_Address/docs.

 **Warning:** The Ops Manager API is an experimental feature that is not fully implemented and could change without notice. Pivotal is developing an officially supported Ops Manager API that will replace many of these endpoints in a subsequent release.

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

Backing Up and Upgrading

- [Backing Up Pivotal Cloud Foundry](#)

- [Upgrading Operations Manager](#)
- [Upgrading Products in a PCF Deployment](#)

Monitoring, Logging, and Troubleshooting

- [Monitoring VMs in PCF](#)
- [Deploying Pivotal Ops Metrics](#)
- [Using SSL with a Self-Signed Certificate in Pivotal Ops Metrics](#)
- [Using Pivotal Ops Metrics](#)
- [PCF Troubleshooting Guide](#)
- [Troubleshooting Ops Manager for VMware vSphere](#)
- [Advanced Troubleshooting with the BOSH CLI](#)
- [PCF Security Overview and Policy](#)

Prerequisites to Deploying Operations Manager and Elastic Runtime

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

vSphere/vCenter Requirements

- vSphere 5.0, 5.1, or 5.5
- vSphere editions: standard and above
- HTTPS access to vCenter on TCP ports 443, 902, and 903.
- A configured vSphere cluster:
 - If you enable vSphere DRS (Distributed Resource Scheduler) for the cluster, you must set the Automation level to **Partially automated** or **Fully automated**. If you set the Automation level to **Manual**, the BOSH automated installation will fail with a `power_on_vm` error when BOSH attempts to create virtual 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:** 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

- 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
- 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

- Amazon Web Services account

OpenStack

Command line install available only through Pivotal Professional Services.

General Requirements

 **Note:** Ops Manager fails to create VMs using the Cisco NSX 1000v switch. This is a known issue. Pivotal

recommends using a [vSphere Distributed Switch](#).

- 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.

- One IP address for each instance.
- An additional IP address for each instance that requires static IPs.

 **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 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)	Dynamic IP	Static IP
Ops Manager Director	1	4	3072	19456	20480		✓
HAProxy	1	1	1024	5120	0	✓	✓
Message Bus (NATS)	1	1	1024	5120	0	✓	✓
etcd Leader	1	1	1024	5120	1024	✓	✓
etcd	2	1	1024	5120	1024	✓	
Health Manager	1	1	1024	5120	0	✓	
Blob Store (NFS Server)	1	1	1024	5120	102400	✓	✓
Cloud Controller Database	1	1	1024	5120	2048	✓	✓
Cloud Controller	1	1	4096	13312	0	✓	
Clock Global	1	1	1024	5120	0	✓	
Cloud Controller Worker	1	1	1024	5120	0	✓	
Router	1	1	1024	5120	0	✓	✓
Collector	0	1	1024	5120	0	✓	
OAuth2 Server Database (UAA Database)	1	1	1024	5120	8192	✓	✓
OAuth2 Server (UAA)	1	1	1024	5120	0	✓	
Login Server (SAML Login)	1	1	1024	5120	0	✓	
Apps Manager Database	1	1	1024	5120	1024	✓	✓
Application Execution (DEA)	1	2	16384	35840	0	✓	
App Log Aggregator (Loggregator Server)	1	1	1024	5120	0	✓	✓

App Log Aggregator Traffic Controller (Loggregator Traffic Controller)	1	1	1024	5120	0	✓	✓
Compilation	5	2	1024	9216	0	✓	
Each Post-Install Errand	1	1	1024	4096	0	✓	
TOTALS	26	27	42 GB	165 GB	123 GB	22	11

 **Note:** Elastic Runtime deploys two post-install errands: Push Console and Smoke Test.

[Return to the Getting Started Guide](#)

Preparing Your Firewall for Deploying PCF

This topic describes how to configure your firewall for [Pivotal Cloud Foundry](#) and how to troubleshoot issues you might encounter with DNS resolution.

Configuring Your Firewall for PCF

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

- **25555**: Routes to the Ops Manager VM
- **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

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

The following iptables script configures an example firewall.

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

```
sudo vi /etc/sysctl.conf
net.ipv4.ip_forward=1

iptables --flush
iptables --flush -t nat
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

# 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 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

service iptables save
shutdown -r now
```

Verifying 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 firewall might prevent Elastic Runtime from accessing `*.xip.io` and the tests will fail without warning.

To verify that Elastic Runtime can correctly resolve DNS entries:

1. SSH into the Pivotal Ops Manager VM.

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

2. Run any of the following network administration commands with the IP address of the VM:

- `nslookup`
- `dig`
- `host`
- The appropriate `traceroute` command for your OS.

3. Review the output of the command and fix any blocked routes.

If the output displays an error message, review the firewall logs to determine which blocked route or routes you need to clear.

4. Repeat steps 1-3 with the Ops Manager Director VM and the HAProxy VM.

Using Your Own Load Balancer

This guide describes how to use your own load balancer and forward traffic to your Elastic Runtime router IP address.

Pivotal Cloud Foundry (PCF) deploys with a single instance of HAProxy for use in lab and test environments. Production environments should use a highly-available customer-provided load balancing solution that does the following:

- Provides load balancing to each of the PCF Router IPs
- Supports SSL termination with wildcard DNS location
- Adds appropriate `x-forwarded-for` and `x-forwarded-proto` HTTP headers to incoming requests
- **(Optional)** Supports WebSockets

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 you assigned to PCF.

Step 3: Configure Pivotal Ops Manager and Ops Manager Director

1. Configure your Pivotal Operations Manager and Ops Manager Director as described in [Getting Started with Pivotal Cloud Foundry](#), then add Elastic Runtime. Do not click **Install** after adding Elastic Runtime.
2. In Pivotal Operations Manager, click the **Elastic Runtime** tile.
3. In the left column, select **HAProxy** if not already selected.
4. In the **HAProxy IPs** field, delete any existing IP addresses. This field should be blank.



5. If you are using a signed certificate from a known certificate authority, install the SSL Certificate on your load balancer. If you are not using a signed certificate, generate a self-signed RSA certificate as described in [Generate an RSA Certificate](#), then install this certificate on your load balancer.
6. In the left column, select **Router IPs**.

7. In the **Router IPs** field, enter the IP address(es) for PCF that you registered with your load balancer in Step 2. Save this change.

<input checked="" type="checkbox"/> System Database Config	Router IPs
	192.168.0.1
<input checked="" type="checkbox"/> File Storage Config	

8. Click **Install**. Complete installation of your selected products as described in the [Install Products](#) section of the PCF Getting Started Guide.

Using Ops Manager Resurrector on VMware vSphere

The Ops Manager Resurrector increases Pivotal Cloud Foundry Elastic Runtime availability in the following ways:

- Reacts to hardware failure and network disruptions by restarting virtual machines on active, stable hosts
- Detects operating system failures by continuously monitoring virtual machines and restarting them as required
- Continuously monitors the BOSH Agent running on each virtual machine and restarts the VMs as required

The Ops Manager Resurrector continuously monitors the status of all virtual machines in an Elastic Runtime deployment. The Resurrector also monitors the BOSH Agent on each VM. If either the VM or the BOSH Agent fail, the Resurrector restarts the virtual machine on another active host.

Limitations

The following limitations apply to using the Ops Manager Resurrector:

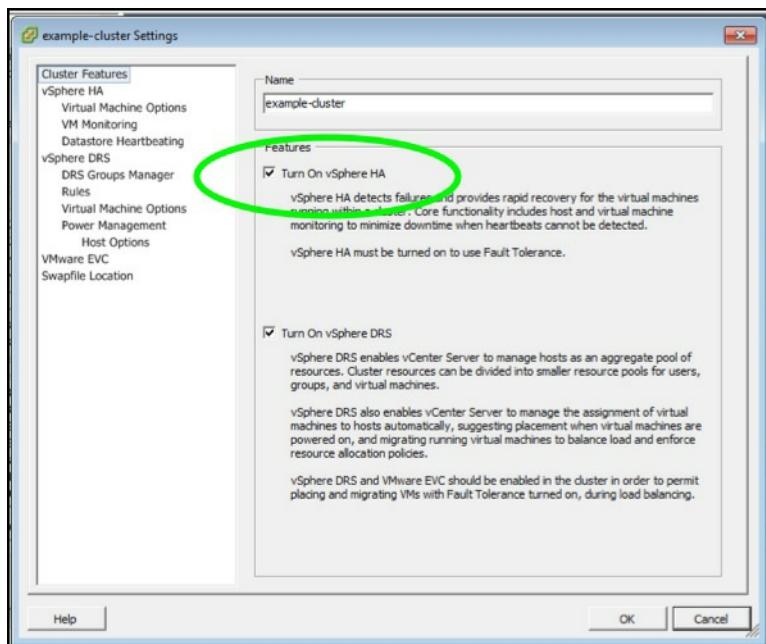
- The Resurrector does not monitor or protect the Ops Manager VM or the BOSH Director VM.
- The Resurrector might not be able to resolve issues caused by the loss of an entire host.
- The Resurrector does not monitor or protect data storage.

For increased reliability, Pivotal recommends that you use vSphere High Availability to protect all of the VMs in your deployment, and that you use a highly-available storage solution.

Enabling vSphere High Availability

Follow the steps below to enable vSphere High Availability:

1. Launch the vSphere Management Console.
2. Right-click the cluster that contains the [Pivotal Cloud Foundry](#) deployment and select **Edit Settings**.
3. Check the **Turn on vSphere High Availability** checkbox.



4. Click **OK** to enable vSphere High Availability on the cluster.

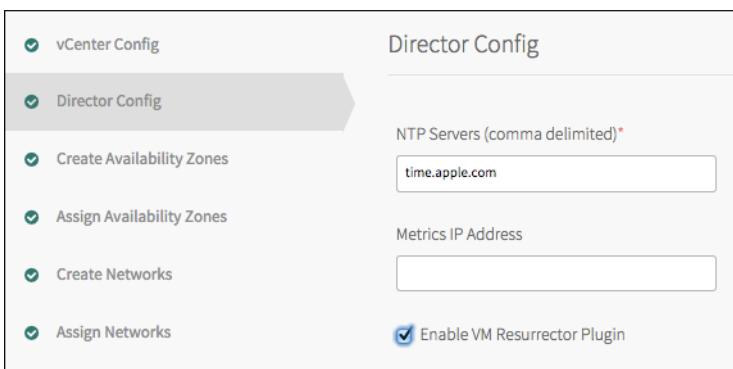
Enabling Ops Manager Resurrector

To enable the Ops Manager Resurrector:

1. Log into the Ops Manager web interface.
2. On the Product Dashboard, select **Ops Manager Director**.



3. In the left navigation menu, select **System Settings**.
4. Check **Enable VM resurrector plugin**. Click **Save**.



5. Ops Manager Resurrector is now enabled.

Connecting Elastic Runtime to LDAP

This topic describes connecting [Pivotal Cloud Foundry](#)  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.

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**.
4. Enter the following LDAP endpoint configuration information:
 - **LDAP 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`

Note: We recommend that you provide LDAP credentials that grant read-only permissions on the LDAP Search Base and the LDAP Group Search Base.

- **LDAP Search Base:** The location in the LDAP directory tree from which any LDAP User search begins. The typical LDAP Search Base matches your domain name.

For example, a domain named “cloud.example.com” typically uses the following LDAP User Search Base:
`ou=Users,dc=example,dc=com`

- **LDAP 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`.

- **LDAP Group Search Base:** The location in the LDAP directory tree from which the LDAP Group search begins.

For example, a domain named “cloud.example.com” typically uses the following LDAP Group Search Base:
`ou=Groups,dc=example,dc=com`

- **LDAP Group Search Filter:** A string that defines LDAP Group search criteria. The standard value is
`member={0}`.

Configure an LDAP endpoint for the UAA

LDAP Server URL

LDAP Credentials

Username

Password

LDAP Search Base

LDAP Search Filter

LDAP Group Search Base

LDAP Group Search Filter

I LDAP Server SSL Cert

5. Click **Save**.
6. Click the **Installation Dashboard** link to return to the Installation Dashboard.
7. On the Installation Dashboard, click **Apply Changes**.

Pending Changes

Revert

UPDATE Pivotal Elastic Runtime

Apply changes

UAA Clients

In addition to any users or client applications that LDAP defines, UAA creates and authenticates default users defined in the `uaa.yml` file on the UAA VM.

UAA installs the following clients:

- admin
- autoscaling_service
- cc_service_broker_client
- cf
- cloud_controller

- login
- portal
- system_passwords

Use the UAA Command Line Interface (UAAC) to edit which default users the `uaa.yml` file defines. For information about the UAAC, see [Creating and Managing Users with the UAA CLI \(UAAC\)](#).

Understanding Availability Zones and Regions in Pivotal Cloud Foundry

The Availability Zone (AZ) functionality in [Pivotal Cloud Foundry](#) (PCF) enables applications to be highly available and tolerant to infrastructure failure by distributing app instances and redundant VMs across various infrastructure segments.

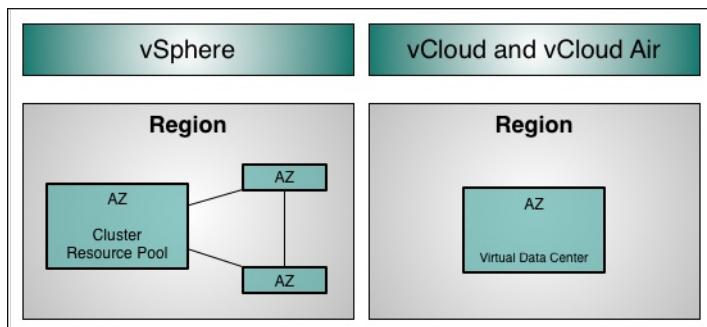
PCF uses the concepts of an AZ and *region* similar to how Amazon Web Services and OpenStack use these constructs, and defines these terms as follows:

- **AZ:** An AZ is a unique set of hardware components that is separated logically and generally also physically from any other AZ. A PCF operator assigns a VMware vSphere cluster and resource pool to an AZ.

Note: At present, the architecture of VMware vCloud and vCloud Air supports the use of only one AZ. For these products, the AZ is the virtual data center that an operator accesses when configuring the [vShield Edge Gateway Services interface](#). This means that if your Ops Manager deployment integrates with either of these products, PCF must place all software in the single AZ, the virtual data center.

- **Region:** A region correlates to either a single data center or to multiple data centers that are in close proximity and have very low latency network connections. A region includes a single PCF installation that an operator deploys to one or more Availability Zones.

The image shows the difference between how PCF handles Availability Zones and regions in vSphere, vCloud, and vCloud Air.



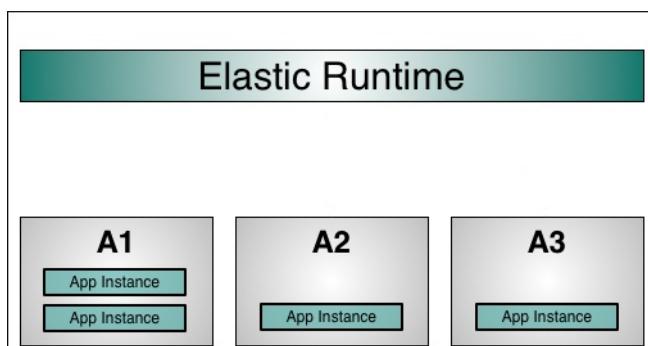
You configure an AZ with the Pivotal Ops Manager Director for VMware vSphere and Elastic Runtime products. For more information, refer to the following topics:

- Steps 5-6 of [Configuring Ops Manager Director for VMware vSphere](#)
- Step 3 of the Configure Elastic Runtime section in [Getting Started with Pivotal Cloud Foundry](#)

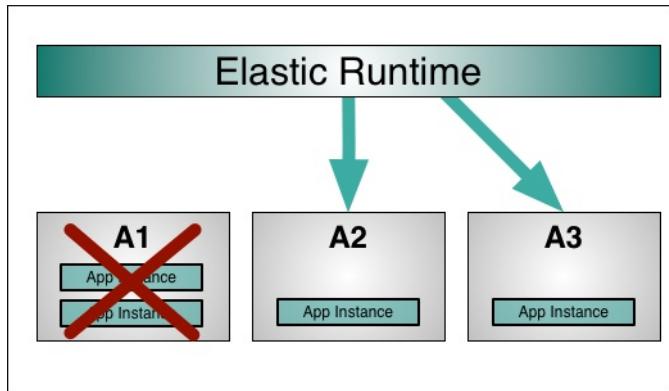
Pivotal Elastic Runtime manages the application instances that run in an AZ. When you run more than one instance of an application, Elastic Runtime balances the instances across all of the zones that you assign to the application.

Example Scenario

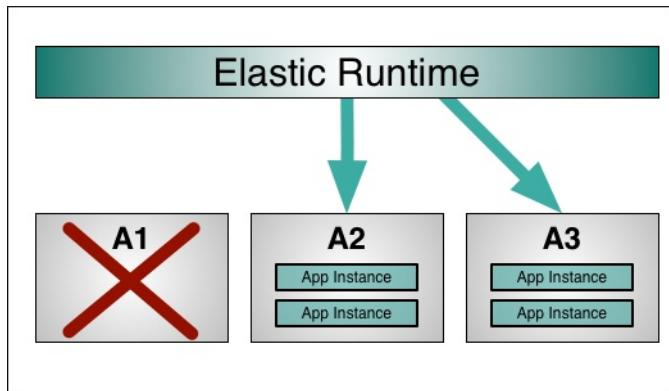
An operator assigns Availability Zones A1, A2, and A3 to an application. The following image shows that Elastic Runtime places two instances in A1, one in A2, and one in A3 when a developer runs four instances of the application.



If A1 experiences a power outage or hardware failure, the two application instances running in A1 terminate. Because A2 and A3 are physically separated from A1, the application instances in these zones continue to run, as the following image shows.



If A1 remains unavailable, Elastic Runtime balances new instances of the application across the remaining Availability Zones, as the following image shows.



Configuring Network Isolation Options in Ops Manager

This topic describes configuring different network isolation options in [Pivotal Cloud Foundry](#) Operations Manager.

Prerequisites

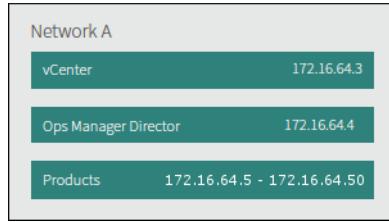
This topic assumes that you have deployed vCenter and created multiple networks.

Option 1: Installing PCF Products to a Common Network

Deploy Ops Manager, Ops Manager Director, and all Pivotal Cloud Foundry (PCF) products to a common network. The vCenter IP must be reachable on or from this network.

 **Note:** Production environments typically require higher security than this option provides.

The following image shows all PCF products and vCenter deployed to a common network.



To configure your PCF deployment for the common network option:

1. Log into the Ops Manager web interface.
2. On the Installation Dashboard, select **Ops Manager Director**.
3. In the left navigation menu, select **Create Networks**.
4. Click **Add**. Add a network that maps to a vCenter network.
The image shows an example network named A.

Installation Dashboard

Ops Manager Director

- Settings
- Status
- Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Create Networks

Networks
One or many IP ranges upon which your products will be deployed

A

Name*

vSphere Network Name*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

- In the left navigation menu, click **Assign Networks**.
- Click the **Infrastructure Network** drop-down menu and select the network that you created in step 4.
- Click the **Deployment Network** drop-down menu and select the network that you created in step 4.

Installation Dashboard

Ops Manager Director

- Settings
- Status
- Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Assign Networks

The Ops Manager director can be configured to have an IP on two networks. Typically one is routable to the IaaS API (*Infrastructure*) and the other is routable to deployed products (*Deployment*).

Infrastructure Network

Deployment Network

Save

- Return to the Installation Dashboard.
- Perform the following steps for each PCF product, including Pivotal Cloud Foundry Elastic Runtime:
 - On the Installation Dashboard, select the PCF product.
 - In the left navigation menu, click **Assign Networks**. Assign the product to the network that you created in step 4.

4.

The image shows an example network named A.

Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks

Assign Availability Zones

Root Filesystem

System Database Config

A

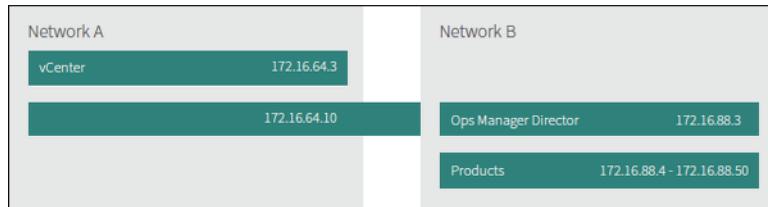
Save

c. Return to the Installation Dashboard.

Option 2: Installing All PCF Products to an Isolated Network

Deploy all PCF products to a Deployment Network. Ensure that you isolate this network from the Infrastructure Network that can reach your vCenter. This configuration provides a higher level of security than [Option 1](#) by reducing the risk of a malicious attack at the IaaS level.

You must configure Ops Manager Director to communicate on two networks, either through routing or by multihoming. For example, the following image shows vCenter in Network A, all PCF products in Network B, and Ops Manager Director multihomed in both networks.



To configure your PCF deployment for an isolated network:

1. Log into the Ops Manager web interface.
 2. On the Installation Dashboard, select **Ops Manager Director**.
 3. In the left navigation menu, select **Create Networks**.
 4. Click **Add**. Add a network that maps to a vCenter network.
- The image shows an example network named A.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Create Networks

Networks
One or many IP ranges upon which your products will be deployed

▼ A

Name*

vSphere Network Name*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

5. Add another network that maps to a different vCenter network.

The image shows an example network named B.

- In the Ops Manager Director configuration interface, configure the Director to receive an IP address on both networks, as follows:
 - In the left navigation menu, click **Assign Networks**. Click the **Infrastructure Network** drop-down menu and select the network that you created in step 4.
 - Click the **Deployment Network** drop-down menu and select the network that you created in step 5.

- Return to the Installation Dashboard.
- Perform the following steps for each PCF product, including Elastic Runtime:

- a. On the Installation Dashboard, select the PCF product.
- b. In the left navigation menu, click **Assign Networks**. Assign the product to the Deployment Network. The image shows an example network named B.

Pivotal Elastic Runtime

Assign Networks

B

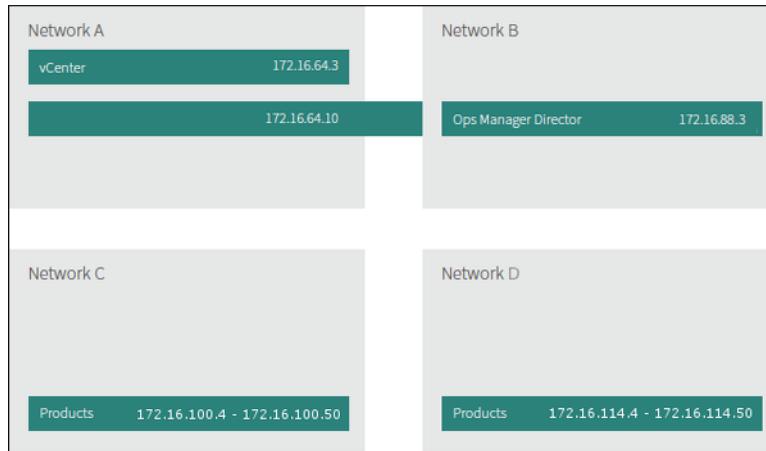
Save

- c. Return to the Installation Dashboard.

Option 3: Installing PCF Products to Multiple Isolated Networks

Deploy PCF products to multiple networks. Ensure that you isolate these networks from each other and from the Infrastructure Network that can reach your vCenter. This configuration provides a higher level of security than [Option 1](#) and [Option 2](#) by reducing the risk of a malicious attack at both the IaaS and product levels.

You must configure Ops Manager Director to communicate on two networks, either through routing or by multihoming. All other PCF products receive IP addresses on one or more separate networks. For example, the following image shows vCenter in Network A, Ops Manager Director multihomed in Network A and Network B, and all other PCF products in Network C and Network D.



To configure your PCF deployment for multiple isolated networks:

1. Log into the Ops Manager web interface.
 2. On the Installation Dashboard, select **Ops Manager Director**.
 3. In the left navigation menu, select **Create Networks**.
 4. Click **Add**. Add a network that maps to a vCenter network.
- The image shows an example network named A.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

vCenter Config Director Config Create Availability Zones Assign Availability Zones Create Networks Assign Networks Resource Config

Create Networks

Networks

One or many IP ranges upon which your products will be deployed

▼ A

Name*

vSphere Network Name*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

5. Add another network for the Ops Manager Director that maps to a different vCenter network.
The image shows an example network named B.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

✓ vCenter Config

✓ Director Config

✓ Create Availability Zones

✓ Assign Availability Zones

✓ Create Networks

✓ Assign Networks

✓ Resource Config

Create Networks

Networks
One or many IP ranges upon which your products will be deployed

► A

▼ B

Name*

vSphere Network Name*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

6. Add a third network specifically for Elastic Runtime that maps to a different vCenter network. Ensure that this network can route to the Ops Manager Director IP address that you created in step 3. The image shows an example network named C.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Create Networks

Networks

One or many IP ranges upon which your products will be deployed

▶ A

▶ B

▼ C

Name*

vSphere Network Name*

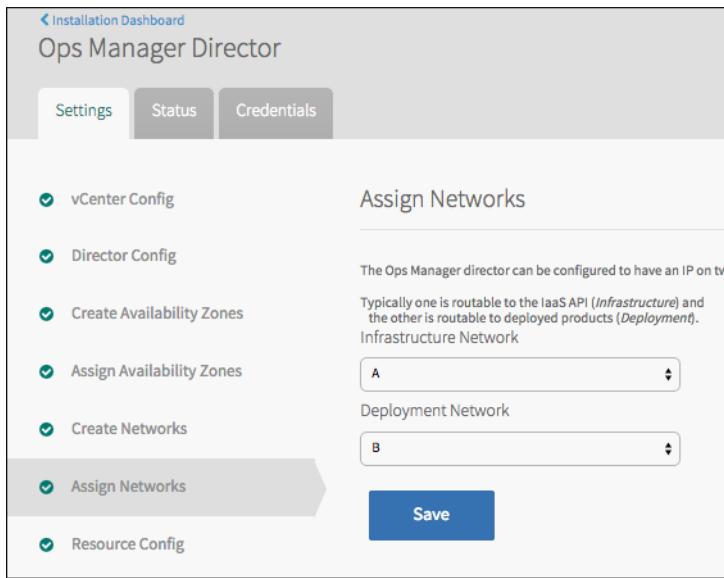
Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

7. **(Optional)** Add additional networks for other products if you want to isolate them from the applications running in Elastic Runtime and from each other. Ensure that each network can route to the Ops Manager Director IP address.
8. In the Ops Manager Director configuration interface, configure the Director to receive an IP address on the Infrastructure and Deployment networks, as follows:
 - In the left navigation menu, click **Assign Networks**. Click the **Infrastructure Network** drop-down menu and select the network that you created in step 4.
 - Click the **Deployment Network** drop-down menu and select the network that you created in step 5.



Installation Dashboard

Ops Manager Director

Settings Status Credentials

✓ vCenter Config

✓ Director Config

✓ Create Availability Zones

✓ Assign Availability Zones

✓ Create Networks

✓ Assign Networks

✓ Resource Config

Assign Networks

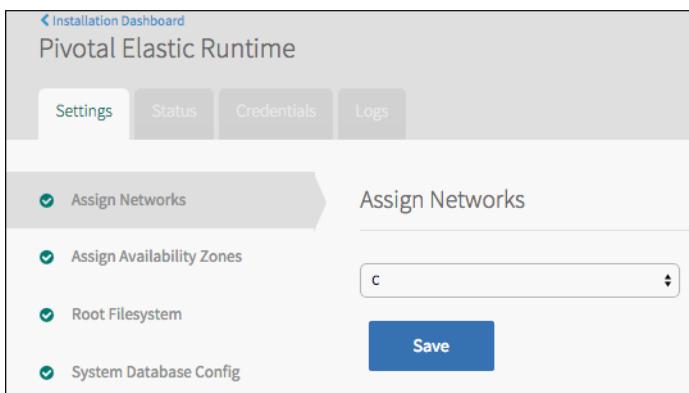
The Ops Manager director can be configured to have an IP on two networks. Typically one is routable to the IaaS API (*Infrastructure*) and the other is routable to deployed products (*Deployment*). Infrastructure Network: A Deployment Network: B

Save

9. Return to the Installation Dashboard.

10. Perform the following steps for each PCF product, including Elastic Runtime:

- On the Installation Dashboard, select the PCF product.
 - In the left navigation menu, click **Assign Networks**. Assign the product to a network other than Infrastructure or Deployment.
- The image shows an example network named C.



Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

✓ Assign Networks

✓ Assign Availability Zones

✓ Root Filesystem

✓ System Database Config

Assign Networks

c

Save

- Return to the Installation Dashboard.

Configuring AWS for PCF

This topic describes how to configure the AWS components that you need to run [Pivotal Cloud Foundry](#) (PCF) on Amazon Web Services (AWS).

After you complete this procedure, complete all steps in the [Configuring Ops Manager Director for AWS](#) and [Configuring Elastic Runtime for AWS](#) topics.

Note: PCF for AWS functionality works only in the us-east-1 region. You cannot deploy PCF to any other region.

Note: PCF for AWS functionality currently does not support the Amazon EC2 Auto Recovery feature. Do not enable this feature for any of your instances.

Step 1: Ensure Your AWS Account Can Launch More Than 20 Instances

The default instance limit for an AWS account is 20 instances. Deploying Ops Manager and a few products uses more than 20 instances. Increasing the AWS instance limit prevents errors and allows you to install more tiles. You can check the limits on your account by visiting the [limits section](#) of the EC2 site on the AWS console. AWS has a global instance limit, which is listed at the top of the limits page, as well as a limit per instance type. If you exceed these limits, Ops Manager displays an error. You can file an AWS support ticket to increase any relevant limits before retrying the Ops Manager installation.

File a support ticket with AWS, if you haven't already done so:

1. Login to your AWS account.
2. Select **Support Center** from the Support menu.
3. Click **Create case** and select **Service Limit Increase** in the “regarding” section.
4. Select **EC2 Instances** and ask for a limit of **50 t2.micro** instances in the **US East (Northern Virginia)** region.
5. **Add another request** for a limit of **20 c4.large** instances in the **US East (Northern Virginia)** region.
6. Enter a reason, such as “We'd like to run Pivotal Cloud Foundry and additional services, requiring more than 20 t2.micro and c4.large instances. Please increase the global instance limit to 50 as well.”

Regarding* Account and Billing Support Service Limit Increase Technical Support

Limit Type* EC2 Instances

Request 1

Region*	US East (Northern Virginia)
Primary Instance Type*	t2.micro
Limit*	Instance Limit
New limit value*	50

Request 2

Region*	US East (Northern Virginia)
Primary Instance Type*	c4.large
Limit*	Instance Limit
New limit value*	20

[Add another request](#)

Use Case Description* We'd like to run Pivotal Cloud Foundry and additional services, requiring more than 20 t2.micro and c4.large instances. Please increase the global instance limit to 50 as well.

7. Fill out the rest of the form and submit.
8. It is safe to continue these instructions, but you might get errors installing tiles until the limit is raised.
9. These tickets are usually processed within a business day. In many cases, processing time might be as little as 15 to 30 minutes. You can check your [instances limits](#) at any time.

If you attempt to start too many VMs using OpsManager, such as by installing several products, Ops Manager might display an error about exceeding your instance limits. You can retry the installation after the limit has been raised.

Step 2: Create S3 Buckets for Ops Manager and Elastic Runtime

Note: You must have two empty S3 blobstore buckets when you install or reinstall PCF for AWS.

1. Navigate to the S3 Dashboard.
2. Use the wizard to create two S3 buckets: one for Ops Manager (OPS_MANAGER_S3_BUCKET) and one for Elastic Runtime (ELASTIC_RUNTIME_S3_BUCKET).
3. Record the bucket names. You need this information for the IAM policy in the next step and for configuring your

product tiles.

Step 3: Create an IAM User for PCF

You must create an Amazon Identity and Access Management user (IAM) with the minimal permissions necessary to run and install PCF.

1. Login to your AWS account.
2. Select **Identity & Access Management** to access the IAM Dashboard.
3. Select **Users>Create New Users**.
4. Enter a user name, such as `pcf-user`.

Enter User Names:

1. `pcf-user`
2.
3.
4.
5.

Maximum 64 characters each

Generate an access key for each user

Users need access keys to make secure REST or Query protocol requests to AWS service APIs.

For users who need access to the AWS Management Console, create a password in the Users panel after completing this wizard.

5. Ensure that the **Generate an access key for each user** checkbox is selected.
6. Click **Create**.
7. Click **Download Credentials** to download the user security credentials.

Note: The `credentials.csv` contains the IDs for your user security access key and secret access key. Be sure to keep the `credentials.csv` file for your currently active key pairs in a secure directory.

8. Click **Close**.
9. On the User dashboard, click the user name to access the user details page.

	User Name	Groups
<input type="checkbox"/>	pcf-user	0

User Actions ▾

10. In the **Inline Policies** region, click the down arrow to display the inline policies. Click the **click here** link to create a new policy.

▼ Permissions

Managed Policies

There are no managed policies attached to this user.

[Attach Policy](#)

Inline Policies

There are no inline policies to show. To create one, [click here](#).

11. On the Set Permissions page, click **Custom Policy** and click **Select**.

Manage User Permissions

Set Permissions

Select a policy template, generate a policy, or create a custom policy. A policy is a document that formally states one or more permissions. You can edit the policy on the following screen, or at a later time using the user, group, or role detail pages.

Policy Generator

Custom Policy

Use the policy editor to customize your own set of permissions.

[Select](#)

12. On the Review Policy page, enter `pcf-iam-policy` in **Policy Name**.

13. Copy and paste the following text in the **Policy Document** field.

Note: Ensure that you edit the `S3_BUCKET_NAME` placeholder text with the bucket names that you define in the [Create S3 Buckets](#) section.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "iam:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "OpsMgrInfrastructureConfiguration",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeKeypairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeAccountAttributes"
      ],
      "Resource": "*"
    },
    {
      "Sid": "OpsMgrInfrastructureDirectorConfiguration",
      "Effect": "Allow",
      "Action": [
        "s3:)"
      ],
      "Resource": [
        "arn:aws:s3:::OPS_MANAGER_S3_BUCKET",
        "arn:aws:s3:::OPS_MANAGER_S3_BUCKET/*",
        "arn:aws:s3:::ELASTIC_RUNTIME_S3_BUCKET",
        "arn:aws:s3:::ELASTIC_RUNTIME_S3_BUCKET/*"
      ]
    },
    {
      "Sid": "OpsMgrInfrastructureAvailabilityZones",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeAvailabilityZones"
      ],
      "Resource": "*"
    }
  ]
}
```

```
},
{
  "Sid": "OpsMgrInfrastructureNetworks",
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeSubnets"
  ],
  "Resource": "*"
},
{
  "Sid": "DeployMicroBosh",
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeImages",
    "ec2:RunInstances",
    "ec2:DescribeInstances",
    "ec2:TerminateInstances",
    "ec2:RebootInstances",
    "elasticloadbalancing:DescribeLoadBalancers",
    "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
    "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
    "ec2:DescribeAddresses",
    "ec2:DisassociateAddress",
    "ec2:AssociateAddress",
    "ec2:CreateTags",
    "ec2:DescribeVolumes",
    "ec2:CreateVolume",
    "ec2:AttachVolume",
    "ec2:DeleteVolume",
    "ec2:DetachVolume"
  ],
  "Resource": "*"
}
]
```

14. Ensure that the **Use autoformatting for policy editing** checkbox is selected.
 15. Click **Apply Policy**.

The Inline Policies region now displays a list of available policies and actions, as the image shows.

Inline Policies

This view shows all inline policies that apply to this user, including policies that are embedded in this user and policies that are embedded in groups that this user is in.

[Create User Policy](#)

Policy Name	Actions
pcf-iam-policy	Show Policy Edit Policy Remove Policy Simulate Policy

Step 4: Create Key Pair

Create one key pair to access your AWS instances, the AWS EC2 MicroBOSH instance that manages your deployment, and your Ops Manager VM.

1. Ensure that you have select **US East (Northern Virginia)** as your region.
2. Select **EC2** to access the EC2 Dashboard.
3. Select **Key Pairs**, click **Create Key Pair**, and name it `pcf`.

The private key `pcf.pem` downloads automatically. Ensure that you add the private key to your local SSH keys directory. Refer to the [AWS Documentation](#).

Step 5: Create a VPC

1. Navigate to the VPC Dashboard.
2. Click **Start VPC Wizard**.

VPC Dashboard

Resources

Filter by VPC: None

[Start VPC Wizard](#) [Launch EC2 Instances](#)

Note: Your Instances will launch in the US East (N. Virginia) region.

You are using the following Amazon VPC resources in the US East (N. Virginia) region:

2 VPCs	2 Internet Gateways
4 Subnets	4 Route Tables
2 Network ACLs	3 Elastic IPs
6 Security Groups	18 Running Instances
0 VPC Peering Connections	0 Customer Gateways
0 VPN Connections	0 Virtual Private Gateways

Virtual Private Cloud

Your VPCs

Subnets

Route Tables

Internet Gateways

DHCP Options Sets

Elastic IPs

3. Select **VPC with Public and Private Subnets** and click **Select**.

Step 1: Select a VPC Configuration

VPC with a Single Public Subnet

In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation (NAT).

VPC with Public and Private Subnets

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via a Network Address Translation (NAT) instance in the public subnet. (Hourly charges for NAT instances apply.)

VPC with Public and Private Subnets and Hardware VPN Access

VPC with a Private Subnet Only and Hardware VPN Access

[Select](#)

4. Specify the following details for your VPC:
 - IP CIDR block: Enter `10.0.0.0/16`.
 - Enter a VPC name.
 - Public subnet: Enter `10.0.0.0/24`.
 - Set both **Availability Zone** fields to an available **us-east-1** region value. You must set both subnets to the

- same AZ, such as **us-east-1a**.
- **Public subnet name:** Enter `public-az1`.
 - **Private subnet:** Enter `10.0.16.0/20`.
 - **Private subnet name:** Enter `pcf-private-az1`.
 - **Instance type:** Select **m1.small**.
 - Select the `pcf` SSH key you created for **Key Pair name**.
 - **Enable DNS hostnames:** Click `Yes`.
 - **Hardware tenancy:** Select `Default`.
 - Click **Create VPC**.

Step 2: VPC with Public and Private Subnets

IP CIDR block*	<code>10.0.0.0/16</code>	(65531 IP addresses available)
VPC name:	<code>pcf-vpc</code>	
Public subnet*	<code>10.0.0.0/24</code>	(251 IP addresses available)
Availability Zone*	<code>us-east-1a</code>	
Public subnet name:	<code>public-az1</code>	
Private subnet*	<code>10.0.16.0/20</code>	(251 IP addresses available)
Availability Zone*	<code>us-east-1a</code>	
Private subnet name:	<code>pcf-private-az1</code>	
You can add more subnets after AWS creates the VPC.		
Specify the details of your NAT instance.		
Instance type*	<code>m1.small</code>	
Key pair name:	<code>pcf</code>	
Note: Instance rates apply. View Rates .		
Enable DNS hostnames*	<input checked="" type="radio"/> Yes <input type="radio"/> No	
Hardware tenancy*	<code>Default</code>	
Cancel and Exit Back Create VPC		

Step 6: Configure a Security Group for Ops Manager

1. Return to the EC2 Dashboard.
2. Select **Security Groups>Create Security Group**.
3. Enter a security group name and description: `OpsManager`.
4. Select the VPC to which to deploy Ops Manager.
5. Click the **Inbound** tab and add rules for HTTP, HTTPS and SSH for your IP only, as the table and image show.

Note: You may relax the IP restrictions after setting the password for OpsManager. Pivotal recommends limiting access to Ops Manager to IP ranges within your organization.

Type	Protocol	Port Range	Source
HTTP	TCP	80	My IP
HTTPS	TCP	443	My IP
SSH	TCP	22	My IP

6. Click **Create**.

Create Security Group

Security group name: OpsManager

Description: OpsManager

VPC: vpc-8ec593eb (10.0.0.0/16) |pcf-vpc|

Security group rules:

Inbound **Outbound**

Type	Protocol	Port Range	Source
HTTP	TCP	80	My IP: 207.114.222.38/
HTTPS	TCP	443	My IP: 207.114.222.38/
SSH	TCP	22	My IP: 207.114.222.38/

Add Rule

Create

Step 7: Configure a Security Group for PCF VMs

1. On the EC2 Dashboard, select **Security Groups>Create Security Group**.
2. Enter a security group name and description: `pcfVMs`.
3. Select the VPC to which to deploy the PCF VMs.
4. Click the **Inbound** tab and add rules for all traffic from your public and private subnets to your private subnet, as the table and image show. This rule configuration does the following:
 - Enables BOSH to deploy ERS and other services.
 - Enables application VMs to communicate through the router.
 - Allows the load balancer to send traffic to Elastic Runtime.

Type	Protocol	Port Range	Source
All traffic	All	0 - 65535	Custom IP 10.0.0.0/16

5. Click **Create**.

Security group name: pcfVMs

Description: pcfVMs

VPC: vpc-1ed01c7b (10.0.0.0/16) |pcf-vpc|

Security group rules:

Inbound **Outbound**

Type	Protocol	Port Range	Source
All traffic	All	0 - 65535	Custom IP 10.0.0.0/16

Step 8: Configure a Security Group for the Elastic Load Balancer

1. On the EC2 Dashboard, select **Security Groups>Create Security Group**.
2. Enter a security group name and description: `PCF_ELB_SecurityGroup`.
3. Select the VPC to which to deploy the ELB.
4. Click the **Inbound** tab and add rules to allow traffic to :80, :443, and :4443 from 0.0.0.0/0, as the table and image show.

Note: You can change the 0.0.0.0/0 to be more restrictive if you want finer control over what can reach the Elastic Runtime. This security group governs external access to the Elastic Runtime from applications such as the cf CLI and application URLs.

Type	Protocol	Port Range	Source	
Custom TCP rule	TCP	4443	Anywhere	0.0.0.0/0
HTTP	TCP	80	Anywhere	0.0.0.0/0
HTTPS	TCP	443	Anywhere	0.0.0.0/0

5. Click **Create**.

Security group name: PCF_ELB_SecurityGroup
 Description: PCF_ELB_SecurityGroup
 VPC: vpc-8ec593eb (10.0.0.0/16) | pcf-vpc

Security group rules:

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	4443	Anywhere
HTTP	TCP	80	Anywhere
HTTPS	TCP	443	Anywhere

Step 9: Configure a Security Group for the Outbound NAT

1. On the EC2 Dashboard, select **Security Groups>Create Security Group**.
2. Enter a security group name and description: **OutboundNAT**.
3. Select the VPC to which to deploy the Outbound NAT.
4. Click the **Inbound** tab and add a rule to allow all traffic from your VPCs, as the table and image show.

Type	Protocol	Port Range	Source
All traffic	All	All	Custom IP 10.0.0.0/16

5. Click **Create**.

Security group name: OutboundNAT
 Description: OutboundNAT
 VPC: vpc-8ec593eb (10.0.0.0/16) | pcf-vpc

Security group rules:

Type	Protocol	Port Range	Source
All traffic	All	0 - 65535	Custom IP 10.0.0.0/16

Step 10: Configure a Security Group for MySQL

Note: This step is optional. If you plan to use an internal database you can skip this step. If you are using RDS you must configure a security group that enables the Ops Manager VM and Ops Manager Director VM to access the database.

1. On the EC2 Dashboard, select **Security Groups>Create Security Group**.
2. Enter a security group name and description: **MySQL**.
3. Select the VPC to which to deploy MySQL.
4. Click the **Inbound** tab. Add a rule of type MYSQL and specify the subnet of your VPC in **Source**, as the table and image show.

Type	Protocol	Port Range	Source
MySQL	TCP	3306	Custom IP 10.0.0.0/16

5. Click **Create**.

Step 11: Configure the Pivotal Ops Manager AMI

1. Return to the EC2 Dashboard.
2. Click **AMIs**. Locate the Pivotal Ops Manager public AMI and click **Launch**. You can find the AMI ID in the *Ops Manager for AWS* PDF you downloaded from [Pivotal Network](#).

Note: If you cannot locate the AMI, ensure that you have set your AWS Management Console to the **US East (N. Virginia)** region. If you still cannot locate the AMI, log in to the [Pivotal Network](#) and file a support ticket.

3. Choose **m3.large** for your instance type and click **Next: Configure Instance Details**.

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)
Micro instances	t1.micro Free tier eligible	1	0.613	EBS only
General purpose	t2.micro Free tier eligible	1	1	EBS only
General purpose	t2.small	1	2	EBS only
General purpose	t2.medium	2	4	EBS only
General purpose	m3.medium	1	3.75	1 x 4 (SSD)
General purpose	m3.large	2	7.5	1 x 32 (SSD)

4. Configure the following for your instance:
 - Network:** Select the VPC you created.
 - Subnet:** Select your public subnet instance.
 - Auto-assign for Public IP:** Select **Enable**.
 - For all other fields, accept the default values.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of Instances	<input type="text" value="1"/>
Purchasing option	<input type="checkbox"/> Request Spot Instances
Network	vpc-8ec593eb (10.0.0.0/16) pcf-vpc <input type="button" value="Create new VPC"/>
Subnet	subnet-fe8129d5(10.0.0.0/24) public-az1 <input type="button" value="Create new subnet"/> 250 IP Addresses available
Auto-assign Public IP	<input type="button" value="Enable"/>
IAM role	<input type="button" value="None"/> <input type="button" value="Create new IAM role"/>
Shutdown behavior	<input type="button" value="Stop"/>
Enable termination protection	<input type="checkbox"/> Protect against accidental termination
Monitoring	<input type="checkbox"/> Enable CloudWatch detailed monitoring <small>Additional charges apply.</small>
Tenancy	<input type="button" value="Shared tenancy (multi-tenant hardware)"/> <small>Additional charges will apply for dedicated tenancy.</small>

5. Click **Next: Add Storage** and adjust the **Size (GiB)** value. The default persistent disk value is 50 GB. We recommend increasing this value to a minimum of 100 GB.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-f1ce6d7e	100	General Purpose (SSD)	150 / 3000	<input checked="" type="checkbox"/>	Not Encrypted
Instance Store 0	/dev/sdb	N/A	N/A	N/A	N/A	N/A	Not Encrypted
Add New Volume							

6. Click **Next: Tag Instance** and **Next: Configure Security Group**.
7. Select the Ops Manager security group that you created in [Configure a Security Group for Ops Manager](#).
8. Click **Review and Launch** and confirm the instance launch details.
9. Click **Launch**.
10. Select the **pcf** key pair, confirm that you have access to the private key file, and click **Launch Instances**. You use this key pair only to access the Ops Manager VM.

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

<input type="button" value="Choose an existing key pair"/>
Select a key pair
pcf
<input checked="" type="checkbox"/> I acknowledge that I have access to the selected private key file (pcf.pem), and that without this file, I won't be able to log into my instance.
Cancel Launch Instances

11. Click **View Instances** to access the **Instances** page on the EC2 Dashboard.

Step 12: Prepare a Load Balancer and a Wildcard DNS Record

To ensure that AWS caches the correct IP address during DNS resolution, you must create the following:

- A load balancer with your domain, a HTTP listener, and other basic information.
- A CNAME wildcard DNS record.

You finish setting up the load balancer with a HTTPS listener and SSL certification information when you configure Elastic Runtime. For more information, see the [Generate a Self-Signed Certificate](#) and [Finalize the Load Balancer Setup](#) sections of the *Configure Elastic Runtime for AWS* topic.

1. On the EC2 Dashboard, click **Load Balancers**.
2. Click **Create Load Balancer** and configure a load balancer with the following information:
 - Enter a load balancer name.
 - **Create LB Inside:** Select the pcf-vpc VPC you created in [Create a VPC](#).
 - Ensure that the **Internal Load Balancer** checkbox is not selected. In a later step you specify a security group to limit inbound traffic to the load balancer.
 - Click **Add** to define a HTTP listener. Set both protocol fields to **HTTP** and set both port fields to **80**.
 - Click **Continue**.

Create Load Balancer

1. Define Load Balancer 2. Configure Health Check 3. Add EC2 Instances 4. Add Tags 5. Review

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name:	pcf-aws-lb		
Create LB Inside:	vpc-8ec593eb (10.0.0.0/16) pcf-vpc		
Create an internal load balancer:	<input type="checkbox"/> (what's this?)		
Enable advanced VPC configuration:	<input checked="" type="checkbox"/>		
Listener Configuration:			
Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
HTTP	80	HTTP	80

3. Select **TCP** in **Ping Protocol** on the **Configure Health Check** page. Ensure that the **Ping Port** value is **80** and set the **Health Check Interval** to 10 seconds. Click **Continue**.

1. Define Load Balancer 2. Configure Health Check 3. Select Subnets 4. Assign Security Groups 5. Add EC2 Instances 6. Add Tags 7. Review

Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Protocol	TCP
Ping Port	80
Advanced Details	
Response Timeout	5 seconds
Health Check Interval	10 seconds
Unhealthy Threshold	2
Healthy Threshold	10

4. Click **Select Subnet**, select the public subnet you configured in [Create a VPC](#), and click **Continue**.
5. Click **Assign Security Groups**, select the security group you configured in [Configure a Security Group for the Elastic Load Balancer](#), and click **Continue**.

1. Define Load Balancer 2. Configure Health Check 3. Select Subnets 4. Assign Security Groups 5. Add EC2 Instances 6. Add Tags 7. Review

Assign Security Groups

Assign a security group: Create a new security group Select an existing security group

Security Group ID	Name	Description	Actions
sg-04bdda60	default	default VPC security group	Copy to new
sg-6d85e209	pcfVMs	pcfVMs	Copy to new
sg-7e80e71a	MySQL	MySQL	Copy to new
sg-37b8df53	OpsManager	OpsManager	Copy to new
sg-1482e570	OutboundNAT	OutboundNAT	Copy to new
sg-0283e466	PCF_ELB_SecurityGroup	PCF_ELB_SecurityGroup	Copy to new

6. Accept the defaults on the **Add EC2 Instances** page and click **Continue**.

Note: Do not add instances to the Load Balancer in this step. You add them on the [Resource Config page](#) of the Ops Manager Director tile.

7. Accept the defaults on the **Add Tags** page and click **Continue**.
8. Review and confirm the load balancer details, and click **Create**.
9. [Create a CNAME wildcard record](#) with your DNS provider to point to the load balancer domain.
10. Optionally, create an additional CNAME record with your DNS provider to map a host name such as 'pcf' or 'opsmng' to the public DNS of the Ops Manager AMI.

Step 13: Configure the NAT Instance

1. On the EC2 Dashboard, click **Instances**. Select the NAT instance, which has an instance type of **m1.small**.
2. From the **Actions** menu, select **Networking>Change Security Groups**.
3. Change the NAT security group from the default group to the outbound NAT security group you created in [Configure a Security Group for the Outbound NAT](#), and click **Assign Security Groups**.

Change Security Groups

Instance ID: i-be6cbb69
Interface ID: eni-9e3bdffb2

Select Security Group(s) to associate with your instance

Security Group ID	Name	Description
sg-04bdda60	default	default VPC security group
sg-6d85e209	ElasticRuntime	ElasticRuntime
sg-7e80e71a	MySQL	MySQL
sg-37b8df53	OpsManager	OpsManager
sg-1482e570	OutboundNAT	OutboundNAT
sg-0283e466	PCF_ELB_SecurityGroup	PCF_ELB_SecurityGroup

Cancel **Assign Security Groups**

Step 14: Record the Ops Manager Instance Public DNS Address

Locate and record the public DNS address of your Ops Manager instance. You need this DNS address anytime you want to access Ops Manager using a browser.

1. On the **Instances** page of the EC2 Dashboard, sort the **Instance Type** column to find the **m3.large** instance that

you created in [Configure the Pivotal Ops Manager AMI](#).

2. Record the public DNS address for the next part of this procedure, which is launching Ops Manager in a browser and continuing the configuration process.

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
<input checked="" type="checkbox"/>	i-89db0d5e	m3.large	us-east-1a	running	2/2 checks ...	None	ec2-...	compute-1.amazonaws.com

Step 15: Set up Subnets for RDS

You must create two new subnets to enable the RDS MySQL database. You create this database in the next step.

1. Navigate to the VPC Dashboard and click **Subnets**.
2. Click **Create Subnet** and enter the following information for the first subnet:
 - **Name tag:**
 - **VPC:** Select pcf-vpc.
 - **Availability Zone:** For performance reasons, set this AZ value to the same AZ value that you defined for the private subnet.
 - **CIDR block:** Define a unique network range, such as .

Create Subnet

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC.

Name tag	<input type="text" value="rds-1"/>
VPC	<input type="text" value="vpc-8ec593eb (10.0.0.0/16) pcf-vpc"/>
Availability Zone	<input type="text" value="us-east-1a"/>
CIDR block	<input type="text" value="10.0.2.0/24"/>

Cancel **Yes, Create**

3. Click **Create Subnet** and enter the following information for the second subnet:
 - **Name tag:**
 - **VPC:** Select pcf-vpc.
 - **Availability Zone:** Select a different AZ than what you specified in rds-1.
 - **CIDR block:** Define a unique network range, such as .

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC.

Name tag	<input type="text" value="rds-2"/>
VPC	<input type="text" value="vpc-8ec593eb (10.0.0.0/16) pcf-vpc"/>
Availability Zone	<input type="text" value="us-east-1b"/>
CIDR block	<input type="text" value="10.0.3.0/24"/>

4. Navigate to the the RDS Dashboard.
5. Create a RDS Subnet Group for the two RDS subnets.
 - Navigate to the RDS Dashboard, click **Subnet Groups>Create DB Subnet Group**.
 - Enter a name and description.
 - **VPC ID:** Select pcf-vpc.
 - **Availability Zone and Subnet ID:** Choose the AZ and subnet for rds-1 and click **Add**. Repeat this process to add rds-2 to the group.
 - Click **Create**.

The image shows a completed subnet group.

Create DB Subnet Group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Name	PCF_RDSGroup	
Description	PCF_RDSGroup	
VPC ID	pcf-vpc (vpc-8ec593eb)	

Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or [add all the subnets](#) related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Availability Zone	us-east-1b	
Subnet ID	subnet-970765e0 (10.0.3.0/24)	
Add		

Availability Zone	Subnet ID	CIDR Block	Action
us-east-1b	subnet-970765e0	10.0.3.0/24	Remove
us-east-1a	subnet-119c343a	10.0.2.0/24	Remove

[Cancel](#) [Create](#)

Note: On the Subnet Group dashboard page, you might need to refresh the page to view the new group.

Step 16: Create a MySQL Database using AWS RDS

Create a MySQL database. You can use the AWS RDS functionality available on the RDS Dashboard to create this database.

Note: You must have an empty MySQL database when you install or reinstall PCF for AWS.

This step provides the information for the Host, Port, Username, Password, and Database fields on the **Director Config** page of the Ops Manager Director tile.

1. Navigate to the RDS Dashboard. Click **Instances>Launch DB Instance** to launch the wizard.
2. Select **MySQL**.
3. Select **Yes** to create a database for production environments. Click **Next Step**.
4. Specify the following database details:
 - **DB Instance Class:** Select **db.m3.large - 2 vCPU, 7.5 GiB RAM**.
 - **Multi-AZ Deployment:** Select **Yes**.
 - **Allocated Storage:** Enter **100 GB**.
 - **DB Instance Identifier:** Enter **pcf-bosh**.
 - Define a secure **Master Username** and **Master Password**.
 - Click **Next Step**.

Note: Record the username and password that you entered. You need this later when configuring the **Director Config** page in the Ops Manager Director tile.

5. Configure advanced settings:
 - **VPC:** Select **pcf-vpc**.
 - **Subnet Group:** Select the security group you created in [Set up Subnets for RDS](#).
 - **Publicly Accessible:** Select **No**.
 - **VPC Security Groups:** Select the security group you created in [Configure a Security Group for MySQL](#).
 - **Database Name:** Enter **bosh**.
 - Accept the default values for the remaining fields.
6. Click **Launch DB Instance**. The instance generation process might take several minutes.

[Return to the Getting Started Guide](#)

Deploying Operations Manager to vSphere

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

1. Refer to the [Known Issues](#) topic before getting started.
2. Download the [Pivotal Cloud Foundry](#) Ops Manager .ova file at [Pivotal Network](#) . Click the **Pivotal Cloud Foundry** region to access the PCF product page. Use the dropdown menu to select an Ops Manager release.

Name	File Version	Size
Pivotal CF Operations Manager for vSphere	1.3.4.0	1.63 GB

3. Log into vCenter.
4. Select the **VM and Templates** view.

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

6. Name the folder "pivotal_cf" and select it.

7. Select **File > Deploy OVF Template**.

8. Select the .ova file and click **Next**.

Deploy from a file or URL

Enter a URL to download and install the OVF package from the Internet, or specify a location accessible from your computer, such as a local hard drive, a network share, or a CD/DVD drive.

9. Review the product details and click **Next**.

10. Accept the license agreement and click **Next**.

11. Name the virtual machine and click **Next**.

Name:

The name can contain up to 80 characters and it must be unique within the inventory folder.

Inventory Location:

anchovy-dc
 birchwood_templates
 birchwood_vms

Note: The selected folder is the one you created.

12. Select a vSphere cluster and click **Next**.

anchovy-dc
 anchovy-cluster

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

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

Note: Hardware virtualization must be off if your vSphere host does not support VT-X/EPT. Refer to the [Prerequisites](#) topic for more information.

Choose a specific host within the cluster.

On clusters that are configured with vSphere HA or Manual mode vSphere DRS, each virtual machine must be assigned to a specific host, even when powered off.

Select a host from the list below:

Host Name

15. Select a storage destination and click **Next**.

Select a destination storage for the virtual machine files:

VM Storage Profile:

Name	Drive Type	Capacity	Provisioned	Free	Type	Thin Pro
anchovy-ds	Non-SSD	5.41 TB	1.62 TB	3.98 TB	VMFS5	Supports
anchovy-ds	SSD	447.00 GB	345.66 GB	341.50 GB	VMFS5	Supports

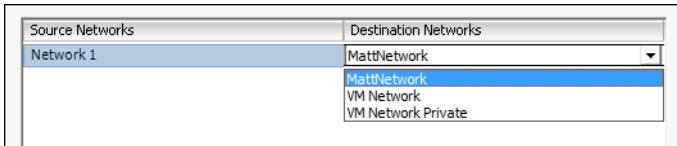
16. Select a disk format and click **Next**. For information on disk formats, see [Provisioning a Virtual Disk](#).

Datastore:

Available space (GB):

Thick Provision Lazy Zeroed
 Thick Provision Eager Zeroed
 Thin Provision

17. Select a network from the drop down list and click **Next**.



18. Enter network information and passwords for the VM admin user and click **Next**.

Note: The IP Address you enter will be the location of the PCF Operations Manager interface.

IP Address The IP address for the Pivotal CF Installer. Leave blank if DHCP is desired.	<input type="text"/>
Netmask The netmask for the Pivotal CF Installer's network. Leave blank if DHCP is desired.	<input type="text"/>
Default Gateway The default gateway address for the Pivotal CF Installer's network. Leave blank if DHCP is desired.	<input type="text"/>
DNS The domain name servers for the Pivotal CF Installer (comma separated). Leave blank if DHCP is desired.	<input type="text"/>
NTP Servers Comma-delimited list of NTP servers	<input type="text"/>
Admin Password This password is used to SSH into the Pivotal CF Installer. The username is 'tempest'.	<input type="password"/> Enter password <input type="password"/> Confirm password

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

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

[Return to the Getting Started Guide](#)

Provisioning a Virtual Disk in vSphere

When you create a virtual machine in VMware vSphere, vSphere creates a new virtual hard drive for that virtual machine. The virtual hard drive is contained in a virtual machine disk (VMDK). The disk format you choose for the new virtual hard drive can have a significant impact on performance.

You can choose one of three formats when creating a virtual hard drive:

- Thin Provisioned
- Thick Provisioned Lazy Zeroed
- Thick Provisioned Eager Zeroed

Thin Provisioned

Advantages:

- Fastest to provision
- Allows disk space to be overcommitted to VMs

Disadvantages:

- Slowest performance due to metadata allocation overhead and additional overhead during initial write operations
- Overcommitment of storage can lead to application disruption or downtime if resources are actually used
- Does not support clustering features

When vSphere creates a thin provisioned disk, it only writes a small amount of metadata to the datastore. It does not allocate or zero out any disk space. At write time, vSphere first updates the allocation metadata for the VMDK, then zeroes out the the block or blocks, then finally writes the data. Because of this overhead, thin provisioned VMDKs have the lowest performance of the three disk formats.

Thin provisioning allows you to overcommit disk spaces to VMs on a datastore. For example, you could put 10 VMs, each with a 50 GB VMDK attached to it, on a single 100 GB datastore, as long as the sum total of all data written by the VMs never exceeded 100 GB. Thin provisioning allows administrators to use space on datastores that would otherwise be unavailable if using thick provisioning, possibly reducing costs and administrative overhead.

Thick Provisioned Lazy Zeroed

Advantages:

- Faster to provision than Thick Provisioned Eager Zeroed
- Better performance than Thin Provisioned

Disadvantages:

- Slightly slower to provision than Thin Provisioned
- Slower performance than Thick Provisioned Eager Zero
- Does not support clustering features

When vSphere creates a thick provisioned lazy zeroed disk, it allocates the maximum size of the disk to the VMDK, but does nothing else. At the initial access to each block, vSphere first zeroes out the block, then writes the data.

Performance of a thick provisioned lazy zeroed disk is not as good a thick provisioned eager zero disk because of this added overhead.

Thick Provisioned Eager Zeroed

Advantages:

- Best performance
- Overwriting allocated disk space with zeroes reduces possible security risks

- Supports clustering features such as Microsoft Cluster Server (MSCS) and VMware Fault Tolerance

Disadvantages:

- Longest time to provision

When vSphere creates a thick provisioned eager zeroed disk, it allocates the maximum size of the disk to the VMDK, then zeros out all of that space.

Example: If you create an 80 GB thick provisioned eager zeroed VMDK, vSphere allocates 80 GB and writes 80 GB of zeroes.

By overwriting all data in the allocated space with zeros, thick provisioned eager zeroed eliminates the possibility of reading any residual data from the disk, thereby reducing possible security risks.

Thick provisioned eager zeroed VMDKs have the best performance. When a write operation occurs to a thick provisioned eager zeroed disk, vSphere writes to the disk, with none of the additional overhead required by thin provisioned or thick provisioned lazy zeroed formats.

Deploying Operations Manager to vCloud Air and vCloud

This topic is a prerequisite to [Configuring Ops Manager Director for vCloud Air and vCloud](#).

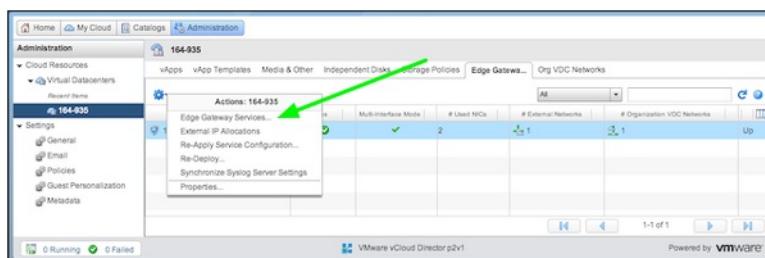
This topic describes how to configure the vCloud or vCloud Air Edge Gateways Configure Services screen and install Ops Manager for your Elastic Runtime environment.

 **Note:** Pivotal Cloud Foundry does not currently support vCloud Air On Demand.

Accessing the vShield Edge Gateway Services Interface

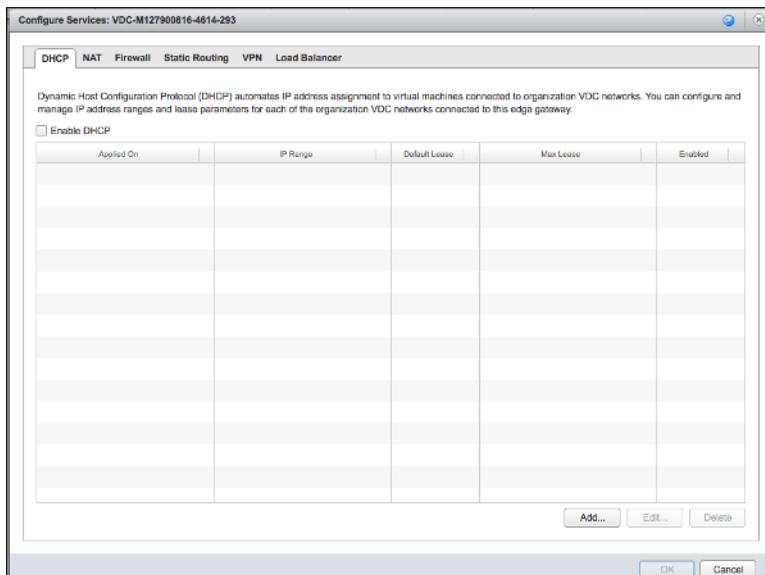
Follow these steps to access the vCloud or vCloud Air Edge Gateways Configure Services screen. For more information about edge gateway services, see the [VMware vCloud Director](#) documentation.

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



The **Configure Services** screen for your virtual datacenter displays with the following tabs:

- **DHCP**
- **NAT**
- **Firewall**
- **Static Routing**
- **VPN**
- **Load Balancer**



Note: The following sections describe how to perform the minimum configuration steps: setting up NAT rules, firewalls, static routing, and a load balancer. Ensure that you configure the Edge Gateways **Configure Services** screen with any additional settings that your environment requires.

Configuring NAT Rules

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

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

- Elastic Runtime accesses the Internet using an SNAT rule.
- Elastic Runtime's API endpoint, which is fronted by HAProxy, requires a DNAT rule to forward traffic from a public IP.
- Ops Manager also requires a DNAT rule to connect to it from a public IP.

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

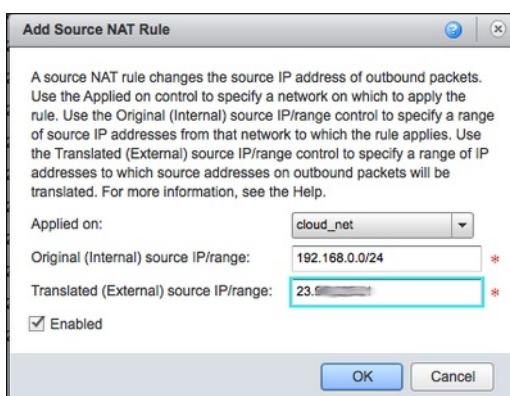
Network Address Translation (NAT) modifies the source/destination IP addresses of packets arriving to and leaving from this Edge Gateway. Source NAT (SNAT) translates the source address of a packet before leaving this gateway, whereas Destination NAT (DNAT) translates the destination IP address/port of a packet received by this gateway.							
Applied On	Type	Original IP	Original Port	Translated IP	Translated Port	Protocol	Enabled
d2p3-ext	SNAT	192.168.0.1-100	any	23.91.128.100	any	ANY	✓
d2p3-ext	DNAT	23.91.128.100	80	192.168.0.100	80	TCP & UDP	✓
d2p3-ext	DNAT	23.91.128.100	443	192.168.0.100	443	TCP & UDP	✓
d2p3-ext	DNAT	23.91.128.100	22	192.168.0.100	22	TCP & UDP	✓

Create SNAT and DNAT Rules

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

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

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

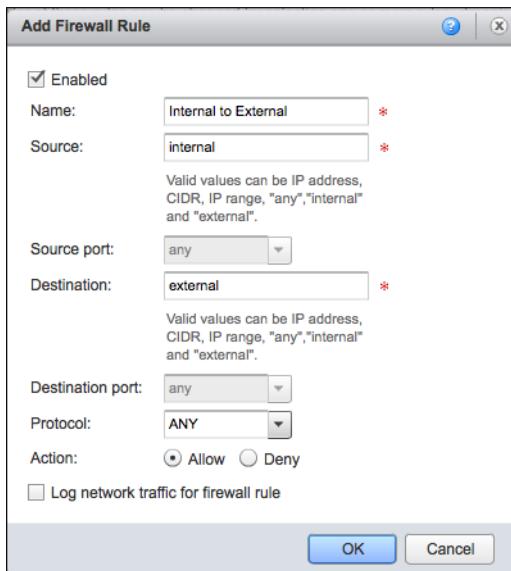


3. Create a destination NAT (DNAT) rule by following the same procedure, using the following configuration:
 - Applied on: Select your external network
 - Original (External) IP/range: Enter the public IP address for Ops Manager
 - Protocol: Select **TCP & UDP**

- Original Port: Select **22**
- Translated (Internal) IP/range: Enter the private IP address of your Ops Manager
- Translated port: **22**

Create Firewall Rules for SNAT and DNAT

1. In the Edge Gateways **Configure Services** screen, select the **Firewall** tab.
2. Create a SNAT firewall rule allowing outbound traffic from all internal IP addresses to all IP external addresses.



3. Create a DNAT firewall rule allowing inbound traffic from the public IP to the private IP address of your Ops Manager.

Allow Inbound Web Traffic for Ops Manager

Repeat the steps above for ports 80 and 443 for the same public address.

Allow Inbound Web Traffic for Elastic Runtime

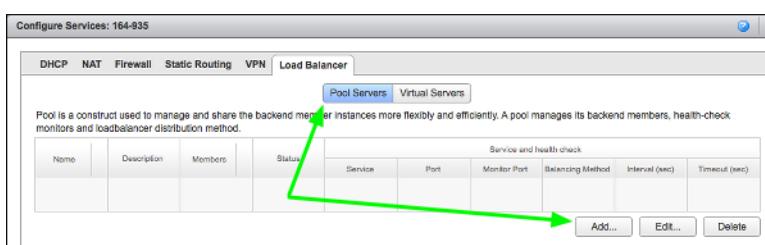
Repeat the steps above for ports 80 and 443 for the Elastic Runtime public IP address.

Setting up Static Routing

Select the **Enable static routing** checkbox.

Setting up Network Rules for Elastic Runtime DNS Resolution

1. In the Edge Gateways **Configure Services** screen, select the **Load Balancer** tab.
2. Click **Pool Servers**, then click **Add**.



The **Add Load Balancer Member Pool** wizard appears.

3. Name the pool **Load Balancer to Elastic Runtime**.

4. In the **Configure Service** step, enable the pool to support HTTP port **80** and HTTPS port **443**. We recommend using the default balancing method, **Round Robin**.

Enable	Service	Balancing Method	Port
<input checked="" type="checkbox"/>	HTTP	Round Robin	80
<input checked="" type="checkbox"/>	HTTPS	Round Robin	443
<input type="checkbox"/>	TCP	Round Robin	

5. In the **Configure Health-Check** step, enter Monitor Port **80** for HTTP and **443** for HTTPS. For both HTTP and HTTPS, change the Mode to **TCP**.

Service	Port	Monitor Port	Mode	Interval (sec)	Timeout (sec)	Health Threshold	Unhealth Threshold
HTTP	80	80	TCP	5	15	2	3
HTTPS	443	443	TCP	5	15	2	3
TCP			TCP	5	15	2	3

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

IP Address:	192.168.109.105	*
Ratio weight:	1	*
Specify how requests will be proportionately routed to an instance. Setting ratio weight to 0 will disable the member.		
Services & Monitoring:		
Service	Port	Monitor Port
HTTP	80	80
HTTPS	443	443
TCP		

7. Click **Finish**.

8. Click **Virtual Servers**.

Pool is a construct used to manage and share the backend member instances more flexibly and efficiently. A pool manages its backend members, health-check monitors and loadbalancer distribution method.									
Name	Description	Members	Status	Service and health check					
Load Balancer		1	✗	HTTP	80	80	Round Robin	5	15
				HTTPS	443	443	Round Robin	5	15

9. Click **Add**.

10. Complete the new virtual server form with the following information:

- Name: Load Balancer
- Applied On: Select your external network
- IP Address: Enter the public IP address of your Elastic Runtime instance
- Pool: Select the **Load Balancer to Elastic Runtime** pool
- Services: Enable HTTP on port **80** with a Persistence Method of **None**, and HTTPS on port **443** with a Persistence Method of **Session Id**
- Enabled: Select this checkbox

Edit Virtual Server

Name:	Load Balancer	*			
Description:					
Applied on:	d2p3-ext				
IP address:	[REDACTED] *				
Pool:	Load Balancer to Elastic Runtime	Supports (HTTP,HTTPS)			
Services:					
Enabled	Name	Port	Persistence Method	Cookie name	Cookie mode
<input checked="" type="checkbox"/>	HTTP	80	None		
<input checked="" type="checkbox"/>	HTTPS	443	Session Id		
<input type="checkbox"/>	TCP		None		
<input checked="" type="checkbox"/> Enabled <input type="checkbox"/> Log network traffic for virtual server					

11. Click **OK** to complete.

Deploying Ops Manager to vCloud or vCloud Air

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

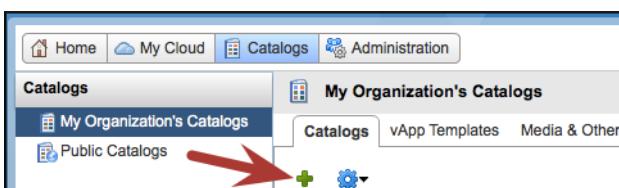
Note: vCloud and vCloud Air use the vCloud Director Web Console, which only supports 32-bit browsers like Firefox. It does not support Chrome. Refer to [Article 2034554](#) in the VMware Knowledge Base for more information about browser versions that the vCloud Director supports.

Upload Ops Manager

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

Note: The first time you upload software to vCloud Director, you must install the **Client Integration Plug-in** and restart all browsers. If the plug-in does not work and you continue to receive a message prompting you to download it, check the plug-in permissions for your browsers.

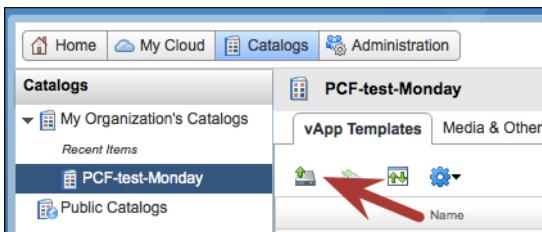
1. Download Pivotal Cloud Foundry Operations Manager for vCloud Air and vCloud Director from [Pivotal Network](#).
2. Log into vCloud Director.
3. Navigate to **Catalogs > My Organization's Catalogs** and select a catalog or click **Add** to create a new catalog.



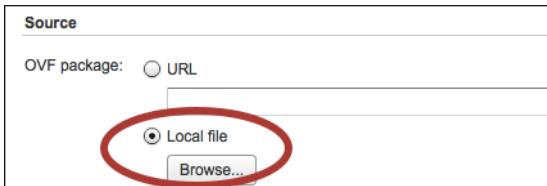
If you are creating a new catalog:

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

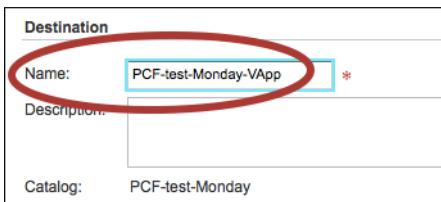
4. Navigate to the **vApp Templates** tab for your catalog and click **Upload**.



5. Select **Local file** and browse to your **.ovf** file.



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

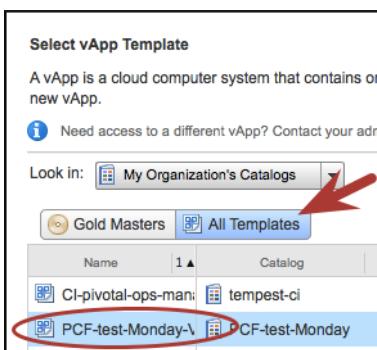


vCloud Director transfers the OVF package to a staging environment, then uploads it to your catalog.

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



8. Select your Ops Manager vApp and click **Next**.



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

Complete the vApp Wizard and Deploy Ops Manager

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

1. Check the **I agree** checkbox to accept licenses and click **Next**.
2. Enter the name of your Ops Manager vApp, select the virtual data center where the vApp should run, and click **Next**.

Select Name and Location

A vApp is a cloud computer system that contains one or more virtual machines. Describe this vApp and select its Virtual Datacenter.

Name:

Description:

Virtual Datacenter

Select the Virtual Datacenter (VDC) in which this vApp is stored and runs when it is started.

3. Choose a storage policy and click **Next**.

Configure Resources

Select what Storage Policies this vApp's virtual machines will use when deployed.

Virtual Machine	Computer Name	Storage Policy
PCF-test-Monday-VApp *	PivotalOpsM-001 *	SSD-Accelerated

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

Network Mapping

Map networks used in the OVF template to networks in your inventory.

Source	Destination	IP allocation
Network 1	164-935-default-routed	Static - Manual

Source: Network 1 - Description
Logical network used by this appliance.

Destination: 164-935-default-routed - Protocol settings

Gateway: 192.168.109.1
Netmask: 255.255.255.0
DNS: ,
DNS suffix:

5. Enter the desired networking information, set an admin password for the Ops Manager vApp, and click **Next**.

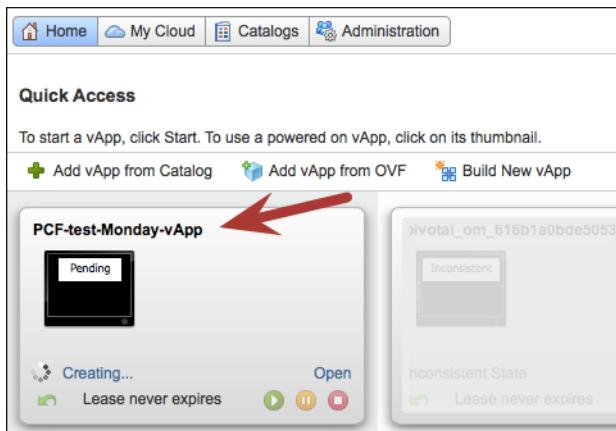
Uncategorized	
DNS:	<input type="text"/>
The domain name servers for the Pivotal Ops Manager (comma separated). Leave blank if DHCP is used.	
Admin Password:	<input type="password"/> Enter password <input type="password"/> Confirm password
This password is used to SSH into the Pivotal Ops Manager. The username is 'tempst'.	
Netmask:	<input type="text"/>
The netmask for the Pivotal Ops Manager's network. Leave blank if DHCP is desired.	
Default Gateway:	<input type="text"/>
The default gateway address for the Pivotal Ops Manager's network. Leave blank if DHCP is desired.	
IP Address:	<input type="text"/>
The IP address for the Pivotal Ops Manager. Leave blank if DHCP is desired.	
NTP Servers:	<input type="text"/>
Comma-delimited list of NTP servers	

 **Note:** The order of the items on your screen may vary from the order shown in this image.

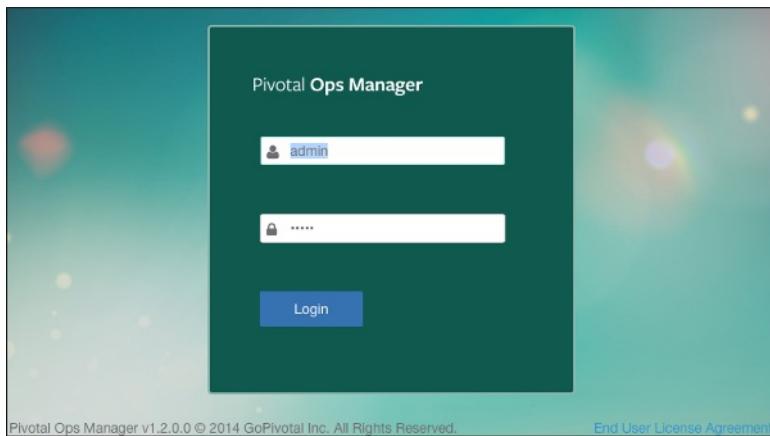
The following list contains tips on entering specific networking information:

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

6. Review the hardware specifications of the virtual machine and click **Next**.
7. In the **Ready to Complete** dialog, check the **Power on vApp After This Wizard is Finished** checkbox and click **Finish**.
8. Navigate to the **Home** view to verify that your Ops Manager vApp is being created.



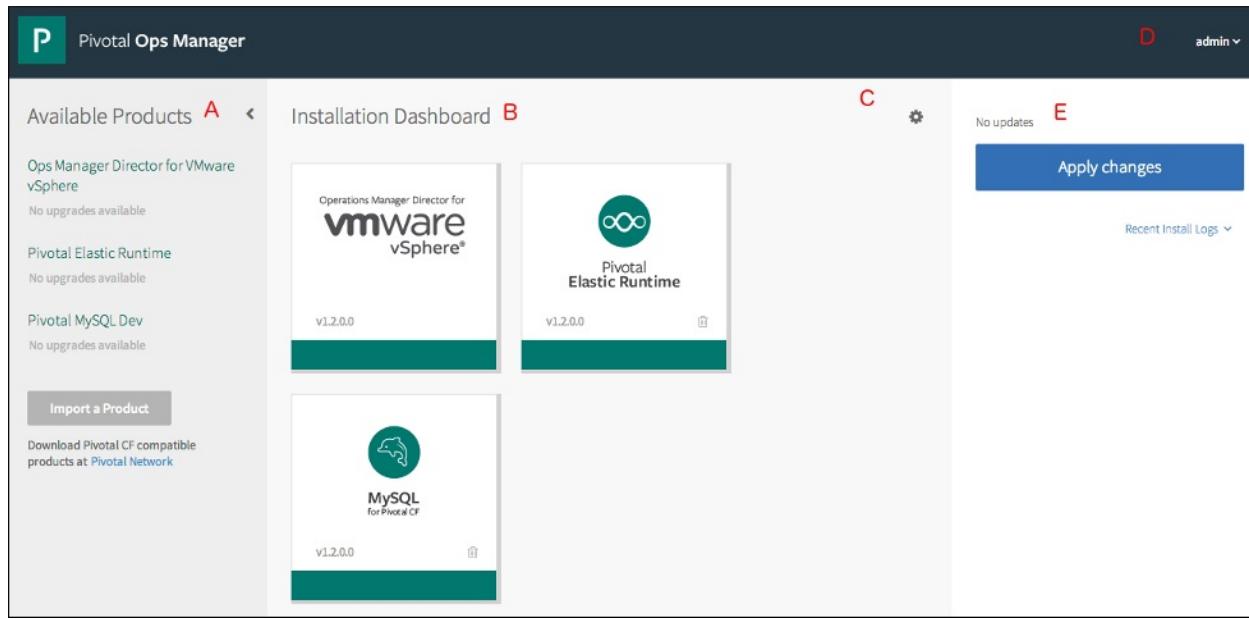
9. Once the VM boots, the Ops Manager login screen displays at the IP address you specified.



[Return to the Getting Started Guide](#)

Understanding the Ops Manager Interface

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



- **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.
- **E—Pending Changes view:** Displays queued installations and updates that will install during the next deploy.

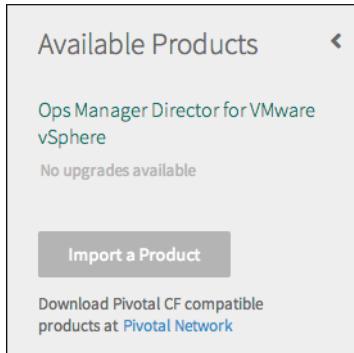
Note: When an update depends on prerequisites, the prerequisites automatically install first.

Adding and Deleting Products

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



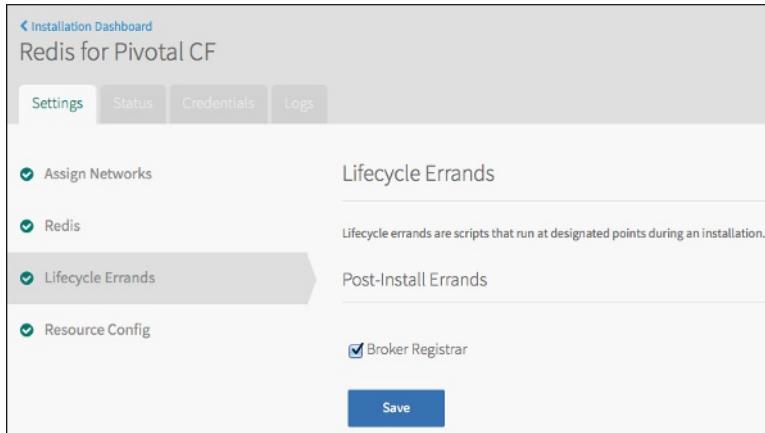
5. The product tile appears in the Installation Dashboard. If the product requires configuration, the tile appears orange.



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.



The screenshot shows the 'Redis for Pivotal CF' product settings page. The 'Lifecycle Errands' section is active. It contains checkboxes for 'Assign Networks', 'Redis', 'Lifecycle Errands' (which is checked), and 'Resource Config'. Below the checkboxes are two sections: 'Lifecycle Errands' and 'Post-Install Errands'. Under 'Lifecycle Errands', the 'Broker Registrar' checkbox is checked. At the bottom of the page is a 'Save' button.

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.

Configuring Ops Manager Director for AWS

This topic describes how to configure the Pivotal Cloud Foundry Operations Manager components that you need to run [Pivotal Cloud Foundry](#) (PCF) on Amazon Web Services (AWS).

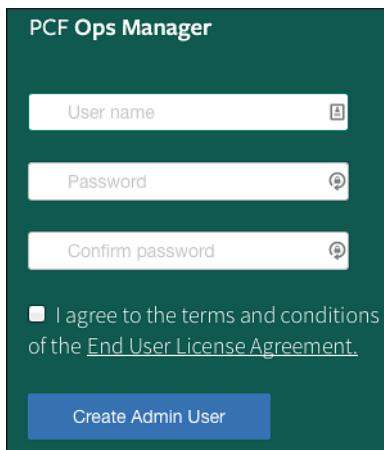
Ensure that you have successfully completed all steps in the [Configuring AWS for PCF](#) topic before beginning this procedure. After you complete this procedure, follow the instructions in the [Configuring Elastic Runtime for AWS](#) topic.

Note: PCF for AWS functionality works only in the us-east-1 region. You cannot deploy PCF to any other region.

Step 1: Access Ops Manager

1. Use the public DNS address of your Ops Manager instance to launch Ops Manager in a browser. See [Record the Ops Manager Instance Public DNS Address](#).
2. Enter a username and password and accept the EULA to create an admin user.

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.



Step 2: AWS Config Page

1. Click the **Ops Manager Director powered by AWS** tile.



2. Select **AWS Config**.
3. Enter your **Access Key ID** and **AWS Secret Key** information. To retrieve your AWS key information, use the IAM credentials that you generated. Refer to the [Obtain AWS Credentials](#) section of the [Configuring AWS for PCF](#) topic.
4. **VPC ID:** Enter your PCF VPC ID. You can find the VPC ID on the AWS VPC Dashboard page next to the VPC name.
5. **Security Group Name:** Enter `pcfVMs`.
For more information, refer to the [Configure a Security Group for PCF VMs](#).

6. Choose **pcf** as the key pair name.
7. Paste the contents of the generated key file `pcf.pem` into the **SSH Private Key** field.
8. Click **Save**.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

AWS Config Director Config Create Availability Zones Assign Availability Zones Create Networks Assign Networks Resource Config

AWS Management Console Config

Access Key ID* AKIAIBGY7CQ6MSSWOUUA

AWS Secret Key*

VPC ID* vpc-8ec593eb

Security Group Name* pcfMNs

Key Pair Name* pcf

SSH Private Key*
-----BEGIN RSA PRIVATE KEY-----
p0g+1mSj8qrAKV4GiDQqcKIH06HH6Z2FwGNM
n/RTA4RgPz+wXfhBTG9BdAtMzOqJfQEyupsf
/XA2J1GudQ8mLej+Hf8kpyp3BJdL/mXM=
-----END RSA PRIVATE KEY-----

Save

Step 3: Director Config Page

1. Select **Director Config**.
2. Enter at least two of the following NTP servers in the **NTP Servers (comma delimited)** field, separated by a comma:
 - 0.amazon.pool.ntp.org
 - 1.amazon.pool.ntp.org
 - 2.amazon.pool.ntp.org
 - 3.amazon.pool.ntp.org
3. Click **Enable VM Resurrector Plugin** to enable the Ops Manager Resurrector functionality and increase Elastic Runtime availability.
For more information, see the introduction and Limitation sections of the [Using Ops Manager Resurrector on VMware vSphere](#) topic.
4. Select **Amazon S3 for Blobstore Location** and complete the associated fields:
 - **Bucket Name:** Enter the Ops Manager bucket name you defined in the [Create a S3 Bucket](#) section of the [Configuring PCF for AWS](#) topic.
 - **Access Key and Secret Key:** Enter your **Access Key ID** and **AWS Secret Key** information. To retrieve your AWS key information, use the IAM credentials that you generated. Refer to the [Obtain AWS Credentials](#) section of the [Configuring AWS for PCF](#) topic.
5. Select **External MySQL Database** for **Database Location** and complete the associated fields.

The details page for your RDS instance contains values required for these fields. On the RDS Dashboard, click **Instances**, then click the arrow to the left of your instance to display this page.

The table lists the database instance field name with its equivalent field on the Director Config page, and also provides the values for the **Port** and **Database** fields.

RDS Instance Field	Ops Manager Director Field
Endpoint	Host
Port	Port, which is <code>3306</code> .
DB Name	Database, which is <code>bosh</code> .
Username	Username

For **Password**, enter the password that you defined for your MySQL database. For more information, refer to the [Create a MySQL Database using AWS RDS](#) section of the *Configuring PCF for AWS* topic .

6. **Max Threads** sets the maximum number of threads that the Ops Manager Director can run simultaneously. For AWS, the default value is 6. Leave the Max Threads field blank to use this default value.
Pivotal recommends that you use the default value unless doing so results in rate limiting or errors on your IaaS.
7. Click **Save**.

- AWS Config
- Director Config
- Create Availability Zones
- Assign Availability Zones
- Create Networks
- Assign Networks
- Resource Config

Director Config

NTP Servers (comma delimited)*

Metrics IP Address

Enable VM Resurrector Plugin

Blobstore Location

Internal

Amazon S3

Bucket Name*

Access Key*

Secret Key*

Database Location

Internal

External MySQL Database

Host*

Port*

Username*

Password*

Database*

Max Threads

Step 4: Availability Zones Pages

1. Click **Create Availability Zones**.
 2. **Amazon Availability Zone**: Enter the name of the AWS AZ that contains the private subnet CIDR (10.0.16.0/20) for your VPC, such as us-east-1b.
- The image shows where to find the **Availability Zone** name on the **Summary** tab of the VPC Subnets page. For more information, refer to the [Create a VPC](#) section of the *Configuring PCF for AWS* topic.

	Name	Subnet ID	State	VPC	CIDR	Available IPs	Availability Zone	Route Table
	rds-1	subnet-a2973489	available	vpc-76feb013 (10.0.0.0/16) PC...	10.0.20.0/25	122	us-east-1a	rtb-7a78231f
<input checked="" type="checkbox"/>	pcf-private-az1	subnet-ff8129d4	available	vpc-8ec593eb (10.0.0.0/16) pc...	10.0.16.0/20	4091	us-east-1a	rtb-5a2c013f
	rds-2	subnet-b20065c5	available	vpc-76feb013 (10.0.0.0/16) PC...	10.0.30.0/25	122	us-east-1b	rtb-7a78231f
	rds-2	subnet-970765e0	available	vpc-8ec593eb (10.0.0.0/16) pc...	10.0.3.0/24	250	us-east-1b	rtb-5a2c013f
	public-az1	subnet-fe8129d5	available	vpc-8ec593eb (10.0.0.0/16) pc...	10.0.0.0/24	248	us-east-1a	rtb-242c0141
	rds-1	subnet-119c343a	available	vpc-8ec593eb (10.0.0.0/16) pc...	10.0.2.0/24	250	us-east-1a	rtb-5a2c013f

subnet-ff8129d4 (10.0.16.0/20) | pcf-private-az1

Summary Route Table Network ACL Tags

Subnet ID: subnet-ff8129d4 | pcf-private-az1 Availability Zone: us-east-1a
 CIDR: 10.0.16.0/20 Route table: rtb-5a2c013f
 State: available Network ACL: acl-54672f31
 VPC: vpc-8ec593eb (10.0.0.0/16) | pcf-vpc Default subnet: no
 Available IPs: 4091 Auto-assign Public IP: no

- Click **Save**.
- Click **Assign Availability Zones**.
- Singleton Availability Zone:** Select the same AZ value that you specified for **Amazon Availability Zone**.

AWS Config Assign Availability Zones

Director Config The Ops Manager Director is a single instance.

Create Availability Zones Choose the availability zone in which to place that instance. It is highly recommended that you backup this VM on a regular basis to preserve settings.

Assign Availability Zones Singleton Availability Zone
 us-east-1a

Create Networks

Assign Networks

Resource Config

Save

- Click **Save**.

Step 5: Networks Pages

- Click **Create Networks**.
- Enter a unique name for this network.
- Enter your private subnet ID in **VPC Subnet ID**. This ID displays on the **Subnets** page of the **VPC Dashboard**.
- Subnet (CIDR Range):** Enter .
- Excluded IP Ranges:** Exclude Amazon IPs used at the bottom of the private range. Enter .
- DNS:** Enter .
- Gateway:** Enter .

- AWS Config
- Director Config
- Create Availability Zones
- Assign Availability Zones
- Create Networks
- Assign Networks
- Resource Config

Create Network

Name*

VPC Subnet ID*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

Save

8. Click **Save**.

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

PCF Ops Manager

⚠ Please review the errors below

- Cannot reach gateway with IP 10.0.16.1 (ignorable if ICMP is disabled)
- Cannot reach DNS with IP 10.0.0.2 (ignorable if ICMP is disabled)

All errors will be reverified before installation.

9. Select **Assign Networks** and assign the Director to the network you created in the previous step.

- AWS Config
- Director Config
- Create Availability Zones
- Assign Availability Zones
- Create Networks
- Assign Networks
- Resource Config

Assign Networks

The Ops Manager director can be configured to have an IP on a network.

Network

Save

Step 6: Resource Config Page

1. Click **Resource Config**.
2. Adjust any values as necessary for your deployment, such as increasing the persistent disk size.
3. Click **Save**.

AWS Config
Director Config
Create Availability Zones
Assign Availability Zones
Create Networks
Assign Networks
Resource Config

Resource Config

JOB	INSTANCES	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)	INSTANCE TYPE	ELB NAME
Ops Manager Director	1	16384	20480	c4.large	

Save

Step 7: Complete the Ops Manager Director Installation

1. Return to the **Installation Dashboard**.
2. Click **Apply Changes**. An ICMP error message appears.

⚠ Please review the errors below

- Cannot reach gateway with IP 10.0.16.1 (ignorable if ICMP is disabled)
- Cannot reach DNS with IP 10.0.0.2 (ignorable if ICMP is disabled)

Ignore errors and start the install

Stop and fix errors

3. Click **Ignore errors and start the install**.
Ops Manager Director begins to install. The image shows the **Changes Applied** window that displays when the installation process successfully completes.

Changes Applied

Ops Manager Director was successfully installed.
We recommend that you export a backup of this installation from the actions menu.

Close

Return to Installation Dashboard

[Return to the Getting Started Guide](#)

Configuring Ops Manager Director for VMware vSphere

Refer to the following procedure for help configuring the Ops Manager Director for VMware vSphere product tile.

1. Log in to [Pivotal Cloud Foundry](#) Operations Manager.
2. Click the **Ops Manager Director for VMware vSphere** tile.



3. Select **vCenter Config** and enter the following information:

Installation Dashboard

Ops Manager Director

Settings Status Credentials Logs

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

vCenter Config

vCenter IP Address:

vCenter Username:

vCenter Password:

Datacenter Name:

Datastore Names (comma delimited):

NOTE: Removing a Datastore after an initial deploy can result in a system outage and/or data loss.

Save

Note: The vCenter credentials must grant create and delete privileges for VMs and folders.

4. Select **Director Config**.
- a. Enter a comma-delimited list of time server addresses.
- b. If you have installed and configured the Ops Metrics product, enter the Metrics IP address.

- c. Click **Enable VM Resurrector Plugin** to enable the Ops Manager Resurrector functionality and increase Elastic Runtime availability.
For more information, see the introduction and Limitation sections of the [Using Ops Manager Resurrector on VMware vSphere](#) topic.
- d. Pivotal recommends that you select **Internal** for your **Blobstore Location**. However, if you choose **Amazon S3** for your blobstore location, complete the associated fields with information from your AWS installation. To retrieve your AWS key information, refer to the [Obtain AWS Credentials](#) section of the Deploying MicroBOSH to AWS topic.
- e. Select a **Database Location**. The default option is to have Pivotal Cloud Foundry deploy and manage a database for you. If you choose to use an **External MySQL Database**, complete the associated fields with information obtained from your external MySQL Database provider.
- f. **Max Threads** sets the maximum number of threads that the Ops Manager Director can run simultaneously. For vSphere, the default value is 32. Leave the Max Threads field blank to use this default value. Pivotal recommends that you use the default value unless doing so results in rate limiting or errors on your IaaS.
- g. Click **Save**.

Director Config

NTP Servers (comma delimited)*

Metrics IP Address

Enable VM Resurrector Plugin

Blobstore Location

Internal

Amazon S3

Bucket Name*

Access Key*

Secret Key*

Database Location

Internal

External MySQL Database

Host*

Port*

Username*

Password*

Database*

Max Threads

Save

5. Select **Create Availability Zones**. Ops Manager Availability Zones correspond to your vCenter clusters and resource pools.

Having multiple Availability Zones allows you to provide high-availability and load balancing to your applications. When an application runs more than one instance of a job, Ops Manager deterministically balances the instances across all of the Availability Zones you assign to the application.

Use the following steps to create one or more Availability Zones for your products to use:

- Click **Add**.
- Enter a unique name for the Availability Zone.
- Enter the name of an existing vCenter cluster to use as an Availability Zone.

- d. **(Optional)** Enter the name of a resource pool in the vCenter cluster you that specify. The jobs running in this Availability Zone share the CPU and memory resources defined by the pool.
e. Click **Save**.

Ops Manager Director for VMware vSphere

Settings Status Credentials Logs

Create Availability Zones

Availability Zones
Clusters and resource pools to which you will deploy Pivotal products

az-1

Name* az-1

Cluster* drinks-cl

Resource Pool classic

Save

6. Select **Assign Availability Zones**. Use the drop-down menu to select an Availability Zone. The Ops Manager Director installs in this Availability Zone.

Ops Manager Director for VMware vSphere

Settings Status Credentials Logs

Assign Availability Zones

The Ops Manager Director is a single instance.

Choose the availability zone in which to place that instance.
It is highly recommended that you snapshot this VM on a regular basis to preserve settings.

Singleton Availability Zone

az-1

Save

Create Availability Zones

Assign Networks

Resource Config

7. Select **Create Networks**. Having multiple networks allows you to place vCenter on a private network and the rest of your deployment on a public network. Isolating vCenter in this manner denies access to it from outside sources and reduces possible security vulnerabilities.

Use the following steps to create one or more Ops Manager networks:

- Click **Add**.
- Enter a unique name for the network.
- Enter the vSphere network name as it displays in vCenter.
- For **Subnet**, enter a valid CIDR block.
- For **Excluded IP Ranges**, enter any IP addresses from the **Subnet** that you want to blacklist from the installation.
- Enter DNS and Gateway IP addresses.
- Click **Save**.

[Installation Dashboard](#)

Ops Manager Director

Settings Status Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Create Networks

Networks
One or many IP ranges upon which your products will be deployed

classic

Name*

vSphere Network Name*

Subnet (CIDR Range)*

Excluded IP Ranges

DNS*

Gateway*

Save

8. Select **Assign Networks**. You can configure the Ops Manager Director to have an IP address on two networks to give the Director access to both networks. Use the drop-down menu to select an **Infrastructure** network and a **Deployment** network for Ops Manager.

When using multiple networks to isolate vCenter, install vCenter on the **Infrastructure** network and the rest of your deployment on the **Deployment** network. This gives the Director access to both vCenter and the rest of your deployment.

[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

Assign Networks

The Ops Manager director can be configured to have an IP on two networks. Typically one is routable to the IaaS API (**Infrastructure**) and the other is routable to deployed products (**Deployment**).

Infrastructure Network

Deployment Network

Save

9. Select **Resource Config**, accept the defaults or make necessary changes, and click **Save**.

Installation Dashboard

Ops Manager Director for VMware vSphere

Settings Status Credentials Logs

vCenter credentials vSphere configuration Network configuration

JOB	INSTANCES	CPU	RAM (MB)	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)
Ops Manager Director	1	4	3072	16384	20480

Save

10. Click the **Installation Dashboard** link to return to the Installation Dashboard.

[Return to the Getting Started Guide](#)

Configuring Ops Manager Director for vCloud Air and vCloud

Before following these instructions you must [deploy Ops Manager and configure NAT and Firewall rules](#).

Refer to the following procedure for help configuring the Ops Manager Director for VMware vCloud Air and vCloud product tile.

1. Log in to [Pivotal Cloud Foundry](#)  Operations Manager.
2. Click the **Ops Manager Director for VMware vCloud** tile.



3. Select **vCloud Config** and enter the following information:

- a. The URL of the vCloud Director.
- b. The **Organization name**.

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

- c. Your credentials.

 **Note:** This user must have create and delete privileges for VMs and folders.

- d. The **Virtual Datacenter name** and **Storage Profile name**. The name that you supply in the Ops Manager **Virtual Datacenter name** field must be the name of the virtual datacenter as it appears in vCloud Director. This name is an alphanumeric string that begins with the prefix "VDC." See the example in the image below.
- e. Click **Save**.

Installation Dashboard

Ops Manager Director

Settings Status Credentials Logs

vCloud Config

Director Config

Create Networks

Assign Networks

Resource Config

vCloud Director Config

vCloud API URL*

Organization name*

Username*

Password*

Virtual Datacenter name*

Storage Profile name*

Save

4. Select **Director Config**.

- Enter a comma-delimited list of time server addresses.
- If you have installed and configured the Ops Metrics product, enter the **Metrics IP Address**.
- Check or uncheck **Enable VM resurrector plugin**.
- Pivotal recommends that you select **Internal** for your **Blobstore Location** and **Database Location**.
- Max Threads** sets the maximum number of threads that the Ops Manager Director can run simultaneously. For vCloud, the default value is 4. Leave the Max Threads field blank to use this default value. Pivotal recommends that you use the default value unless doing so results in rate limiting or errors on your IaaS.
- Click **Save**.

Director Config

NTP Servers (comma delimited)*

Metrics IP Address

Enable VM Resurrector Plugin

Blobstore Location

Internal

Amazon S3

Bucket Name*

Access Key*

Secret Key*

Database Location

Internal

External MySQL Database

Host*

Port*

Username*

Password*

Database*

Max Threads

Save

5. Select **Create Networks**. Use the following steps to create one or more Ops Manager networks:

- Define a unique name for the network.
- Enter the **vCloud Network Name** as it appears in vCloud Director.
- Enter a valid CIDR block in **Subnet**.
- Enter any IP addresses from the **Subnet** that you want to blacklist from the installation in **Excluded IP Ranges**.
- Enter the **DNS** and **Gateway** IP addresses.
- Click **Save**.

Installation Dashboard

Ops Manager Director

- Settings
- Status
- Credentials
- Logs

vCloud Config

Director Config

Create Networks

Assign Networks

Resource Config

Create Networks

Networks
One or many IP ranges upon which your products will be deployed

▼ my network

Name*

vCloud Network Name*

Subnet*

Excluded IP Ranges

DNS*

Gateway*

Save

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

Installation Dashboard

Ops Manager Director

- Settings
- Status
- Credentials
- Logs

vCloud Config

Director Config

Create Networks

Assign Networks

Resource Config

Assign Networks

The Ops Manager director can be configured to have an IP on a network.

Network

Save

7. Select **Resource Config**, accept the defaults or make necessary changes, and click **Save**.

Installation Dashboard

Ops Manager Director

- Settings
- Status
- Credentials
- Logs

vCloud Config

Director Config

Create Networks

Assign Networks

Resource Config

Resource Config

Job	Instances	CPU	RAM (MB)	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)
Ops Manager Director	1	2	3072	15364	20480

Save

8. Click the **Installation Dashboard** link to return to the Installation Dashboard.

[Return to the Getting Started Guide](#)

Configuring Elastic Runtime for AWS

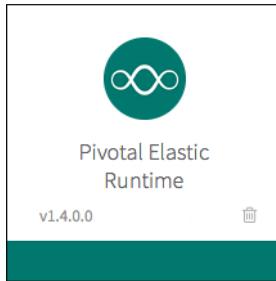
This topic describes how to configure the Pivotal Elastic Runtime components that you need to run [Pivotal Cloud Foundry](#) (PCF) on Amazon Web Services (AWS).

Before following this procedure, complete all steps in the [Configuring AWS for PCF](#) and [Configuring Ops Manager Director for AWS](#) topics.

Note: PCF for AWS functionality works only in the us-east-1 region. You cannot deploy PCF to any other region.

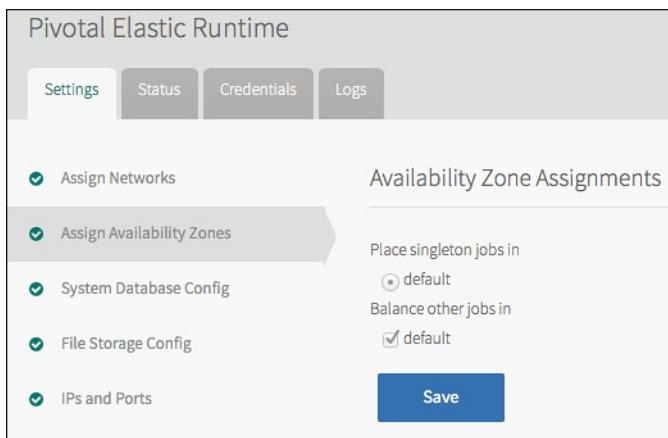
Step 1: Add Elastic Runtime to Ops Manager

1. Navigate to the Pivotal CF Ops Manager Installation Dashboard.
2. Click the Pivotal Network link on the left to add Elastic Runtime to Ops Manager. For more information, refer to the [Adding and Deleting Products](#) topic.
3. Click the Elastic Runtime tile in the Installation Dashboard.



Step 2: Assign Networks and Availability Zones

1. Click **Assign Network**.
2. Select the network you created in the [Networks Pages](#) section of the [Configuring Ops Manager Director for AWS](#) topic, and click **Save**.
3. Click **Assign Availability Zones**.
4. Select the option button and checkbox to define the availability zone for singleton jobs and job balancing.



5. Click **Save**.

Step 3: Configure System Databases

Complete the following procedure if you want to use the external RDS instance for your Elastic Runtime databases.

First, create the databases that Elastic Runtime requires. These instructions create those databases on the same RDS instance you created in [Configuring AWS for PCF](#).

1. Add your key pair to your ssh profile so that you can access the machine:

```
ssh-add pcf.pem
```
2. SSH into your Ops Manager VM using its IP address and the username `ubuntu`: `ssh ubuntu@OPS_MANAGER_IP`
 You can find the IP address for the Ops Manager VM on the AWS EC2 console.
3. Login to your RDS instance through the mysql client, using the hostname from the RDS console and the username you created the RDS instance with. This command prompts you to enter the password for your RDS user:

```
mysql -h RDS-HOSTNAME -u RDS-USERNAME -p
```
4. Create databases for each of the six CF components that require a database. Run `CREATE database DB-NAME` for each of the following: `uaa`, `ccdb`, `console`, `notifications`, `autoscale`, `app_usage_service`
5. You have now created the databases on the RDS instance. Exit the mysql client and close your connection to the Ops Manager VM.
6. Navigate to the **System Database Config** page of the Elastic Runtime product tile.
7. Select **External Databases**.
8. Enter the RDS hostname, port (`3306`), username, and password.

System Database Config

System Database Config*

Internal Databases

External Databases

Hostname DNS Name *

RDS-HOSTNAME

TCP Port *

3306

Username *

RDS-USERNAME

Password *

Save

Step 4: Configure File Storage

Complete the following procedure if you want to use Amazon S3 for your Elastic Runtime file storage.

1. Navigate to the **File Storage Config** page of the Elastic Runtime product tile.
2. Select **Amazon S3**.
3. Enter the Elastic Runtime bucket name you chose in [Create S3 Buckets](#).
4. Enter your Access Key and Secret Key. To retrieve your AWS key information, use the IAM credentials that you generated in [Create an IAM User for PCF](#).

- Root Filesystem
- System Database Config
- File Storage Config
- IPs and Ports
- MySQL Proxy Config
- Cloud Controller
- External Endpoints
- SSO Config
- LDAP Config

File Storage Config*

Internal

Amazon S3

Bucket Name *

Access Key *

Secret Key *

[Save](#)

Step 5: Generate a Self-Signed Certificate

1. Select **IPs and Ports**.
2. Leave the **Router IPs** and **HAProxy IPs** fields blank.
3. Ensure that the **Loggregator Port** field is set to **4443**.
4. Click the **Generate Self-Signed RSA Certificate** link to launch the RSA certificate wizard.
5. On the **Generate Self-Signed RSA Certificate** window, select a wildcard domain that you own and click **Generate**. The example in the image uses *.example.com.

Generate Self-Signed RSA Certificate

Example: *.app.domain.com, *.system.domain.com, my.webapp.com, *.domain.com*

[Cancel](#) [Generate](#)

6. Elastic Runtime populates the **SSL Certificate** fields with RSA certificate and private key information.
7. Click **Trust Self-Signed Certificates**.

Assign Networks

Assign Availability Zones

IPs and Ports

MySQL Proxy Config

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Lifecycle Errands

Resource Config

Stemcell

HAProxy is the default load balancer for SSL termination. Alternatively, you can use your own load balancer and forward traffic to the Cloud Foundry router IP. Self-signed certificates will function properly.

Router IPs

HAProxy IPs

Loggregator Port

4443

SSL Certificate *

-----BEGIN CERTIFICATE-----
MIIDEC...
-----BEGIN RSA PRIVATE KEY-----
Ea7Vjh7Op5RREsgWabHM8ShpIS
-----END RSA PRIVATE KEY-----

Generate Self-Signed RSA Certificate

Trust Self-Signed Certificates

8. Click **Save**.

Step 6: Finalize the Load Balancer Setup

In this step, you complete your load balancer configuration that you began when setting up AWS components for PCF. For more information, see the [Prepare a Load Balancer and Wildcard DNS Record](#) section in the *Configure AWS for PCF* topic.

1. On the EC2 Dashboard, click **Load Balancers**.
2. Select the load balancer you created in the [Prepare a Load Balancer and Wildcard DNS Record](#) section of the *Configuring AWS for PCF* topic. The **Load balancer** detail tabs display at the bottom of the page.
3. Select **Listeners**.

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Cipher	SSL Certificate
HTTP	80	HTTP	80	N/A	N/A

4. Click **Edit** to define a second listener with the following values:
 - Load Balancer Protocol: **HTTPS (Secure HTTPS)**

- **Load Balancer Port:** 443
- **Instance Protocol:** HTTP
- **Instance Port:** 80

5. Click **Edit** to define a third listener with the following values for use with websockets:

- **Load Balancer Protocol: SSL (Secure TCP)**
- **Load Balancer Port:** 4443
- **Instance Protocol:** TCP
- **Instance Port:** 80

When you click **Save**, AWS prompts you to enter SSL certificates for these listeners.

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Cipher	SSL Certificate
HTTP	80	HTTP	80	N/A	N/A
HTTPS (Secure HTTP)	443	HTTP	80	Change	pcfawscert Change
SSL (Secure TCP)	4443	TCP	80	Change	pcfawscert Change

6. Click **Change** in the **SSL Certificate** column of the second listener.

7. On the Select Certificate page, click **Upload a new SSL Certificate** and complete the following information:

- Enter a **Certificate Name**.
- **Private Key and Public Key Certificate:** Paste the RSA certificate and private key information from the **SSL Certificate** fields of the IP and Ports page to the appropriate field.
- Click **Save**.

Select Certificate

An SSL Certificate allows you to configure the HTTPS/SSL listeners of your load balancer. You may select a previously uploaded certificate below, or define a new SSL Certificate. [Learn more](#) about setting up HTTPS load balancers and certificate management.

Certificate Type: Choose an existing SSL Certificate Upload a new SSL Certificate

Certificate Name: pcfawscert

Private Key:
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAQEAwQDQWVH12W7QAxLCAwDQ
/73QAH3EMHvZZBFKnfra3JcFD2NPQ6oPH
ZhjaEW1tgzB1YqJYz8TGe2r5xeK14vyEDtZnEyR7cMIG08L1QD3Q==
-----END RSA PRIVATE KEY-----
(pem encoded)

Public Key Certificate:
-----BEGIN CERTIFICATE-----
MIIBIjANBgkqhkiG9w0BAwIwEAEwDwYVpnu0Brw79JyIaOcRbWpFnDV85V
-----END CERTIFICATE-----
(pem encoded)

Certificate Chain: Optional
(pem encoded)

8. Click **Change** in the **SSL Certificate** column of the third listener.

9. Select **Choose an existing SSL certificate** and select the certificate that you previously uploaded for the second listener.

Note: There might be a brief delay before the certificate you uploaded for the second listener appears. If you do not see the certificate you uploaded, try refreshing the page.

The image shows the completed listener information.

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Cipher	SSL Certificate
HTTP	80	HTTP	80	N/A	N/A
HTTPS (Secure HTTP)	443	HTTP	80	Change	pcfawscert Change
SSL (Secure TCP)	4443	TCP	80	Change	pcfawscert Change

10. Click **Save** to complete the load balancer configuration.

Step 7: Configure the Cloud Controller

1. Select **Cloud Controller** and enter your system and application domains.

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

Pivotal recommends that you use the same domain name, but different subdomain names, for your system domain and your app domain. This allows you to use a single wildcard certificate for the domain while preventing apps from creating routes that overlap with system routes.

For example, name your system domain `system.MYDOMAIN.COM` and your apps domain `apps.MYDOMAIN.COM`.

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

Stemcell

Coordinates Pivotal CF Elastic Runtime application lifecycles

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

2. Click **Save**.

Step 8: Configure Resources

Because you are using RDS, S3, and ELBs, you can turn off certain VMs that you do not need.

1. Click **Resource Config** and edit the following fields:

- **NFS Server**: Enter `0` in **Instances**.
- **Cloud Controller Database**: Enter `0` in **Instances**.
- **UAA Database**: Enter `0` in **Instances**.
- **Console Database**: Enter `0` in **Instances**.
- **HAProxy**: Enter `0` in **Instances**.
- **MySQL Proxy**: Enter `0` in **Instances**.
- **MySQL Server**: Enter `0` in **Instances**.

2. On the same page:

- **Router**: Enter the load balancer name in **ELB Name** that you created in the [Prepare a Load Balancer and Wildcard DNS Record](#) section of the [Configuring AWS for PCF](#) topic.

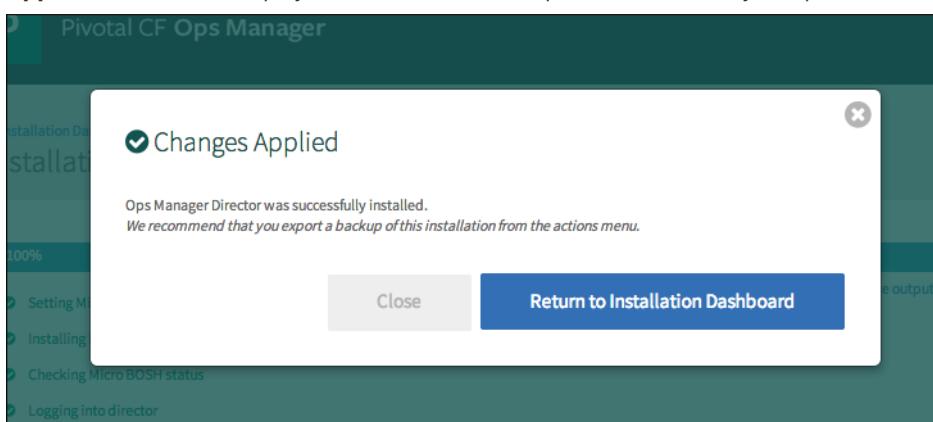
The images show the edited fields.

JOB	INSTANCES	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)	INSTANCE TYPE	ELB NAME
NATS	1	2048	0	t2.micro	
etcd	1	2048	1024	t2.micro	
NFS Server	0	2048	102400	t2.micro	
Cloud Controller Database	0	2048	2048	t2.micro	
UAA Database	0	2048	8192	t2.micro	
Console Database	0	2048	1024	t2.micro	
Cloud Controller	1	20480	0	m3.large	
HAProxy	0	2048	0	t2.micro	
Router	1	2048	0	t2.micro	pcf-aws-lb
Health Manager	1	2048	0	t2.micro	
Clock Global	1	2048	0	t2.micro	
Cloud Controller Worker	1	2048	0	t2.micro	
Collector	0	2048	0	t2.micro	
UAA	1	2048	0	t2.micro	
Login	1	2048	0	t2.micro	
MySQL Proxy	0	2048	0	t2.micro	
MySQL Server	0	30000	100000	r3.large	
DEA	1	32768	0	r3.xlarge	
Doppler Server	1	2048	0	t2.micro	

3. Click **Save**.

Step 7: Complete the Elastic Runtime Installation

1. Click the **Installation Dashboard** link to return to the Installation Dashboard.
2. Click **Apply Changes** to begin your installation of Elastic Runtime. The install process generally requires a minimum of 90 minutes to complete. The image shows the **Changes Applied** window that displays when the installation process successfully completes.

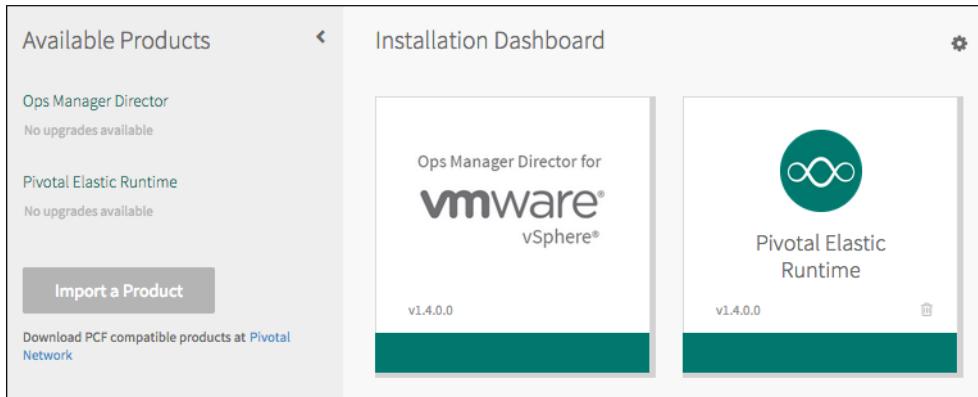


[Return to the Getting Started Guide](#)

Configuring Elastic Runtime for vSphere and vCloud

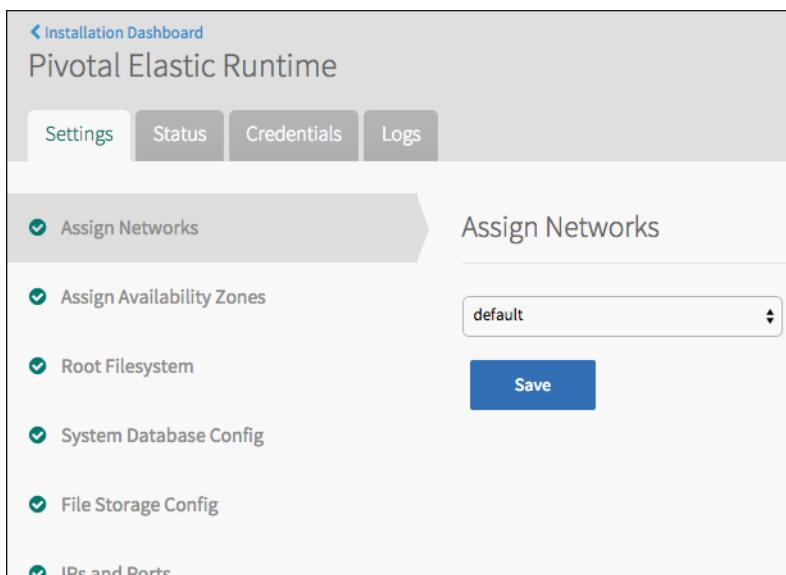
Refer to the following procedure for help configuring the [Pivotal Cloud Foundry](#)  Elastic Runtime product tile for VMware vSphere, vCloud, or vCloud Air.

1. Click the **Elastic Runtime** tile.



The screenshot shows the 'Available Products' section on the left with 'Ops Manager Director' and 'Pivotal Elastic Runtime' listed. The 'Pivotal Elastic Runtime' tile on the right is selected, showing the 'vmware vSphere' logo, version 'v1.4.0.0', and a trash icon. The overall interface is a light gray dashboard with a dark header.

2. Select **Assign Networks**. Elastic Runtime runs on the network you select.



The screenshot shows the 'Pivotal Elastic Runtime' configuration page. The 'Assign Networks' section is selected in the sidebar. The main area shows a dropdown menu set to 'default' and a 'Save' button. Other configuration options like 'Assign Availability Zones', 'Root Filesystem', 'System Database Config', 'File Storage Config', and 'IPs and Ports' are also listed in the sidebar.

3. **(vSphere 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 any job with a single instance in this Availability Zone.
 - o Select one or more Availability Zones under **Balance other jobs**. Ops Manager balances instances of jobs with more than one instance across the Availability Zones that you specify.

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- System Database Config
- File Storage Config
- IPs and Ports
- MySQL Proxy Config

Availability Zone Assignments

Place singleton jobs in default

Balance other jobs in default

Save

4. Select **Root Filesystem**. This page specifies the default root filesystem, or stack, for apps pushed to your PCF instance.

IMPORTANT: Support for the Lucid64 filesystem will be deprecated in a future Pivotal Cloud Foundry release. Developers or operators will need to take action to ensure existing applications migrate to using the new cflinuxfs2 stack. For more information, refer to the [release notes](#) or the [Knowledge Base](#).

Developers can override the default you specify using the cf CLI when they push their apps.

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- System Database Config
- File Storage Config
- IPs and Ports
- MySQL Proxy Config
- Cloud Controller

This configures the default stack used when pushing a new app.

Default Root Filesystem*

cflinuxfs2 - derived from Ubuntu 14.04 (Trusty Tahr)

lucid64 - derived from Ubuntu 10.04 (Lucid Lynx)

Save

5. Select **System Database Config**. By default, this page specifies that PCF uses an internal database for UAADB, ConsoleDB, CCDB, App Usage Events DB, Autoscaling and Notifications. Alternatively, you can configure an external AWS RDS database. If you choose to use an external database, complete the associated fields with information obtained from AWS.

Note: Pivotal recommends that you accept the default internal database unless you require the functionality of AWS RDS.

Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks
Assign Availability Zones
Root Filesystem
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

Select internal if you want to keep using the system-provided databases. Otherwise, select external if you want to use your own database service, such as the Amazon Relational Database Service (RDS), to host the system-required databases, in which case you will need to create the database schemas. Consult Pivotal Cloud Foundry documentation for schema requirements.

System Database Config*

Internal Databases
 External Databases

Hostname DNS Name *
TCP Port *
Username *
Password *

Save

6. Select **File Storage Config**. This page specifies whether to use an internal (NFS) or external (AWS S3) filestore.

Assign Networks
Assign Availability Zones
Root Filesystem
System Database Config
File Storage Config
IPs and Ports
MySQL Proxy Config
Cloud Controller
External Endpoints
SSO Config
LDAP Config
SMTP Config
Errands

Select internal if you want to keep using the system-provided filestore. Otherwise select Amazon S3 if you want to use your own S3 filestore.

File Storage Config*

Internal
 Amazon S3

Bucket Name *
Access Key *
Secret Key *

Save

7. In the left column, select **IPs and Ports**.
8. The value you enter in the **Router IPs** field depends on whether you are using your own load balancer or the HAProxy load balancer.
 - **Your own load balancer:** Enter the IP address(es) for PCF that you registered with your load balancer. Refer to the [Using Your Own Load Balancer](#) topic for help using your own load balancer with PCF.
 - **HAProxy load balancer:** Leave this field blank.
9. The value you enter in the **HAProxy IPs** field depends on whether you are using your own load balancer or the HAProxy load balancer.
 - **Your own load balancer:** Leave this field blank.
 - **HAProxy load balancer:** Enter at least one HAProxy IP address. Point your DNS to this address.

For more information, refer to the [Configuring PCF SSL Termination](#) topic. For help understanding the Elastic Runtime architecture, refer to the [Architecture](#) topic.

Assign Networks

Assign Availability Zones

Root Filesystem

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

Router IPs

HAProxy IPs

Loggregator Port

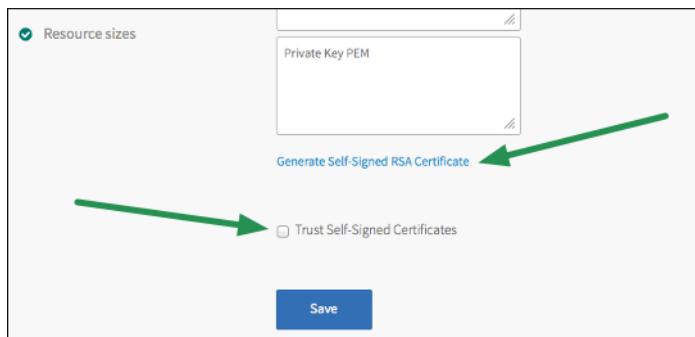
SSL Certificate *

Generate Self-Signed RSA Certificate

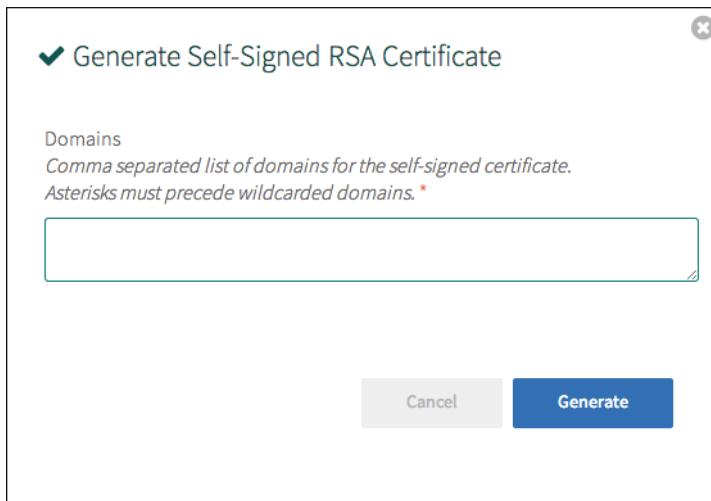
Trust Self-Signed Certificates

Save

10. The **Loggregator Port** field specifies an alternate port for Loggregator. If you leave this field blank, Elastic Runtime uses the default port 443. Pivotal recommends that you accept the default value unless you require Loggregator to use a different port.
11. If you are using a signed **SSL Certificate** from a known certificate authority (CA), copy and paste its values for **Certificate PEM** and **Private Key PEM** into the appropriate text fields. If you have an **Intermediate CA** paste it in the same box below the Certificate PEM. Alternatively, complete the following two steps to generate self-signed RSA certificates.
 - Check the **Trust Self-Signed Certificates** checkbox, then click **Generate Self-Signed RSA Certificate**.



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



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 `my.apps.domain.com` or `my.system.domain.com`.

12. The **Certificate Key PEM** and **Private Key PEM** fields now contain certificate keys. Click **Save**.

<input checked="" type="checkbox"/> Assign Networks <input checked="" type="checkbox"/> Assign Availability Zones <input checked="" type="checkbox"/> Root Filesystem <input checked="" type="checkbox"/> System Database Config <input checked="" type="checkbox"/> File Storage Config <input checked="" type="checkbox"/> IPs and Ports <input checked="" type="checkbox"/> MySQL Proxy Config <input checked="" type="checkbox"/> Cloud Controller <input checked="" type="checkbox"/> External Endpoints <input checked="" type="checkbox"/> SSO Config <input checked="" type="checkbox"/> LDAP Config <input checked="" type="checkbox"/> SMTP Config <input checked="" type="checkbox"/> Errands <input checked="" type="checkbox"/> Resource Config <input checked="" type="checkbox"/> Stemcell	<p>HAProxy is the default load balancer for SSL termination. Alternatively, you can use your own load balancer and forward traffic to the Cloud Foundry router IP. Self-signed certificates will function properly.</p> <p>Router IPs</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px;"></div> <p>Enter the IP address(es) for the Pivotal CF Elastic Runtime Router. This IP must be in your subnet.</p> <p>HAProxy IPs</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px; margin-top: 10px;"></div> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px; margin-top: 10px;"></div> <p>Loggregator Port</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px; margin-top: 10px;"></div> <p>SSL Certificate *</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 100px; margin-top: 10px;"></div> <p>-----BEGIN CERTIFICATE----- MIIDNzCCAhIgAwIBAgVAlzInuoEPKImEd017i5H/ghFPw3kMA0GCSqGSIb3DQEBrQUAMEECAjBgNVBAArA1VTMRAwDgYDVQQKDA dQaXZvDGFsMSAwHgDVQZQDBcQ LmNyW51LndpbGUy2YlXbwlNmNvbTaeFw0xNTA OMDlJuMTE4MTIzEw0xNiA0MDQw</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 100px; margin-top: 10px;"></div> <p>-----BEGIN RSA PRIVATE KEY----- MIIEowIBAAKCAQEAx0BBIERpygiZ7K1pYNE0zoyV uasMo1SPjhUPIi82AQ-B0B x52GlyM4svNHR6+eMmyrQ/3SsGwddxJ4tyE+fkdQos Bf9+uAWLPVLDScv6HQMSm ZSnDFxZwliquJ+kg2YoXX+4uXlb09DZUriinByx7e7R 79nwJQiwDeeVcLwLhf1D</p> <p>Generate Self-Signed RSA Certificate</p> <p><input checked="" type="checkbox"/> Trust Self-Signed Certificates</p> <p style="text-align: center;">Save</p>
---	--

13. Select **MySQL Proxy Config**. Use this page to configure the MySQL proxy connections required to run the internal MySQL database cluster for high-availability Elastic Runtime.

<input checked="" type="checkbox"/> Assign Networks <input checked="" type="checkbox"/> Assign Availability Zones <input checked="" type="checkbox"/> Root Filesystem <input checked="" type="checkbox"/> System Database Config <input checked="" type="checkbox"/> File Storage Config <input checked="" type="checkbox"/> IPs and Ports <input checked="" type="checkbox"/> MySQL Proxy Config <input checked="" type="checkbox"/> Cloud Controller <input type="checkbox"/> External Endpoints	<p>A proxy tier routes MySQL connections from internal components to healthy cluster nodes. Configure DNS and/or your own load balancer to point to multiple proxy instances for increased availability. TCP healthchecks can be configured against port 1936.</p> <p>MySQL Proxy IPs</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px; margin-top: 10px;"></div> <p>Enter the IP address(es) for the MySQL proxy instances configured on your external load balancer. The external load balancer IP will need to have been defined in the Router tab.</p> <p>MySQL Service Hostname</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px; margin-top: 10px;"></div> <p style="text-align: center;">Save</p>
--	---

14. Select **Cloud Controller** and enter the system and application domains.

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

Pivotal recommends that you use the same domain name, but different subdomain names, for your system domain and your app domain. This allows you to use a single wildcard certificate for the domain while preventing apps from creating routes that overlap with system routes.

For example, name your system domain `system.MYDOMAIN.COM` and your apps domain `apps.MYDOMAIN.COM`.

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- 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

Coordinates Pivotal CF Elastic Runtime application lifecycles

System Domain *

This domain is used to target and push apps to Pivotal CF Elastic Runtime (ex: api.my-cf.com). You must set up a wildcard DNS record for this domain that points to the router static IP(s) or a load balancer.

Apps Domain *

Cloud Controller DB Encryption Key

Maximum File Upload Size (MB) (min: 1024, max: 2048) *

Disable Custom Buildpacks

Default Quota App Memory (MB) (min: 10240, max: 102400) *

Default Quota Service Instances (min: 0, max: 1000) *

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

15. Leave the **Cloud Controller DB encryption key** field blank unless:
 - You deployed Elastic Runtime earlier
 - You then stopped Elastic Runtime, or it crashed
 - You are re-deploying Elastic Runtime with a backup of your Cloud Controller database
 Enter your Cloud Controller database encryption key only if these conditions apply. See [Backing Up Pivotal Cloud Foundry](#) for more information.
16. Enter your intended maximum file upload size.
17. **(Optional)** Check the **Disable Custom Buildpacks** checkbox. By default, the cf command line tool 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.
18. Enter your default app memory and service instances quotas.
19. Click **Save**.
20. **(Optional - Advanced)** If you are forwarding syslog messages via TCP to a RELP syslog server, complete this step. Select **External Endpoints**, enter the IP address and port of the syslog aggregator host, and click **Save**.

Copyright Pivotal Software Inc, 2013-2015

100 of 480

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

<input checked="" type="checkbox"/> Assign Networks	Forwards syslog messages to an external server. Leave these options blank unless you are providing a syslogd server.
<input checked="" type="checkbox"/> Assign Availability Zones	
<input checked="" type="checkbox"/> Root Filesystem	IP Address of Syslog Aggregator Host <input type="text"/>
<input checked="" type="checkbox"/> System Database Config	The host must be reachable from the Elastic Runtime network, accept TCP, UDP or RELP connections, and use the RELP protocol (e.g. rsyslogd).
<input checked="" type="checkbox"/> File Storage Config	Syslog Aggregator Port <input type="text"/>
<input checked="" type="checkbox"/> IPs and Ports	Networking Protocol <input type="text"/>
<input checked="" type="checkbox"/> MySQL Proxy Config	
<input checked="" type="checkbox"/> Cloud Controller	
<input checked="" type="checkbox"/> External Endpoints	Save

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

21. **(Optional - Advanced)** If you are using the VMware SSO appliance for integration with Active Directory, complete this step. Follow the instructions for [Configuring Single Sign-On](#) to configure this page.

<input checked="" type="checkbox"/> Assign Networks	Configure Identity Provider
<input checked="" type="checkbox"/> Assign Availability Zones	Provider Name <input type="text"/>
<input checked="" type="checkbox"/> Root Filesystem	Provider Metadata <input type="text"/>
<input checked="" type="checkbox"/> System Database Config	(OR) Provider Metadata URL <input type="text"/>
<input checked="" type="checkbox"/> File Storage Config	
<input checked="" type="checkbox"/> IPs and Ports	
<input checked="" type="checkbox"/> MySQL Proxy Config	
<input checked="" type="checkbox"/> Cloud Controller	
<input checked="" type="checkbox"/> External Endpoints	Save
<input checked="" type="checkbox"/> SSO Config	

22. **(Optional - Advanced)** If you are using the LDAP endpoint for UAA, you must configure Elastic Runtime with your LDAP endpoint information. See [Connecting Elastic Runtime to LDAP](#) for more information.

23. **(Optional)** Select **SMTP Config**, enter your reply-to and SMTP email information, and click **Save**. Apps Manager uses these settings to send invitations and confirmations to console users. These SMTP settings are required if you want to enable end-user self-registration.

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

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- 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

Configuration for Simple Mail Transfer Protocol

From Email

Address of SMTP Server

Port of SMTP Server

SMTP Server Credentials

SMTP Enable Automatic STARTTLS

SMTP Authentication Mechanism*

Plain

▼
▲

SMTP CRAMMD5 secret

24. Select **Lifecycle Errands**. By default, Ops Manager runs the **Run Smoke Tests** and **Push Console** errands after installing Elastic Runtime. See [Understanding Lifecycle Errands](#) for more information.

Note: The Push Console errand also deploys the usage service, which provides data on the count of instances and amount of memory that apps consume. When the usage service starts for the first time, it executes an API call to purge all application usage events from the Cloud Controller Database. Subsequent Apps Manager and usage service deploys do not trigger this API call, and do not purge application usage events.

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- System Database Config
- File Storage Config
- IPs and Ports
- MySQL Proxy Config
- Cloud Controller
- External Endpoints
- SSO Config
- LDAP Config
- SMTP Config
- Errands

Errands

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

<p>Post Install</p>	<p><input checked="" type="checkbox"/> Push Console</p> <p>Pushes the Pivotal Dev Console application to your Elastic Runtime installation</p>
<p><input checked="" type="checkbox"/> Run Smoke Tests</p>	<p>Runs Smoke Tests against your Elastic Runtime installation</p>
<p><input checked="" type="checkbox"/> Push App Usage Service</p>	<p>Monitors usage of an application pushed to your Elastic Runtime installation</p>
<p><input checked="" type="checkbox"/> Notifications With UI</p>	<p>Notifications release for PCF</p>
<p><input checked="" type="checkbox"/> Deploy CF Autoscaling App</p>	<p>Deploys the CF Autoscaling AP</p>
<p><input checked="" type="checkbox"/> Register autoscaling service broker</p>	<p>Register autoscaling service broker</p>

There are no pre-delete errands for this product.

25. Select **Resource Config**, accept the defaults or make necessary changes, and click **Save**.
26. Select **Stemcell**. This page displays the stemcell version that shipped with Ops Manager. You can also use this page to import a new stemcell version.

Installation Dashboard

Ops Manager Director

Settings Status Credentials Logs

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

Stemcell

Stemcell

A stemcell is a template from which Ops Manager creates the VMs needed for a wide variety of components and products.

microbosh requires BOSH stemcell version 2859 ubuntu-trusty

Using bosh-stemcell-2859-vsphere-esxi-ubuntu-trusty-go_agent.tgz

Import Stemcell

27. Click the **Installation Dashboard** link to return to the Installation Dashboard.

[Return to the Getting Started Guide](#)

Creating New Elastic Runtime User Accounts

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 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 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 Command Line Interface (CLI). See [Creating and Managing Users with the cf CLI](#).

[Return to the Getting Started Guide](#)

Logging into the Apps Manager

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

This topic describes two environment variables you can use to manage users' interactions with the Apps Manager, and how to set these variables using the cf CLI.

Understanding the Apps Manager Environment Variables

You can control which users can create orgs and perform user management actions on the Apps Manager using the following environment variables.

ENABLE_NON_ADMIN_ORG_CREATION

If set to `true`, a user of any role can create an org. On the Org Dashboard, the links to create a new org appear as follows:

- The **Create a New Org** link in the drop-down menu in the left navigation panel
- The **Create Org** link on the right side of the My Account page, which you access from the user name drop-down menu

The image shows the link locations.



If set to `false`, only an Admin can create an org. This is the default value.

ENABLE_NON_ADMIN_USER_MANAGEMENT

If set to `true`, Org Managers and Space Managers can invite users and manage roles, and a user of any role can leave an org, provided at least one Org Manager exists in the org.

If set to `false`, only an Admin can invite users and manage roles. This is the default value. Users cannot remove themselves from an org, and on the Org Dashboard, the Members tab appears as read-only for all users except Admins.

Changing an Environment Variable Value

Note: To run the commands discussed in this section, you must be an administrator for your org and log into 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. From a terminal window, run `cf api URL` and target your Apps Manager URL. For example:
`cf api api.YOUR-SYSTEM-DOMAIN`.
2. Run `cf login` and provide your UAA Administrator user credentials.
3. Select the `system` org and the `console` space.
4. Run `cf set-env APP NAME VALUE`. For example: `cf set-env console ENABLE_NON_ADMIN_ORG_CREATION true`
5. Run `cf restart console` to reinitialize the Apps Manager with the new environment variable value.

Configuring Your App Autoscaling Instance

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.

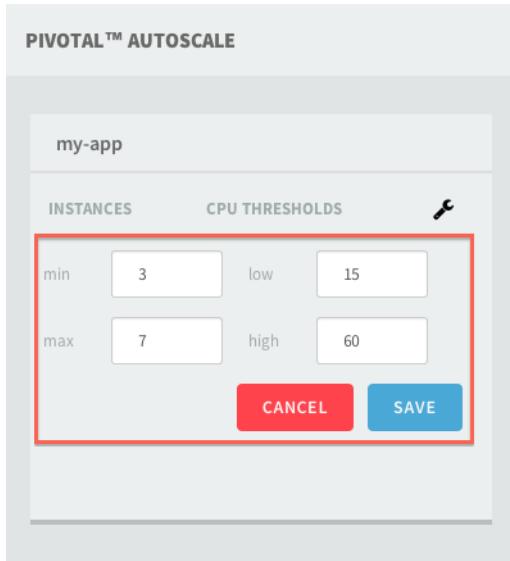
1. Log in to the Developer Console: [How to log in.](#)
2. Select a space containing an App Autoscaling service instance from the org dashboard or from the left navigation bar.
3. In the Services section of the space dashboard, under your App Autoscaling service instance name, click **Manage**.

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

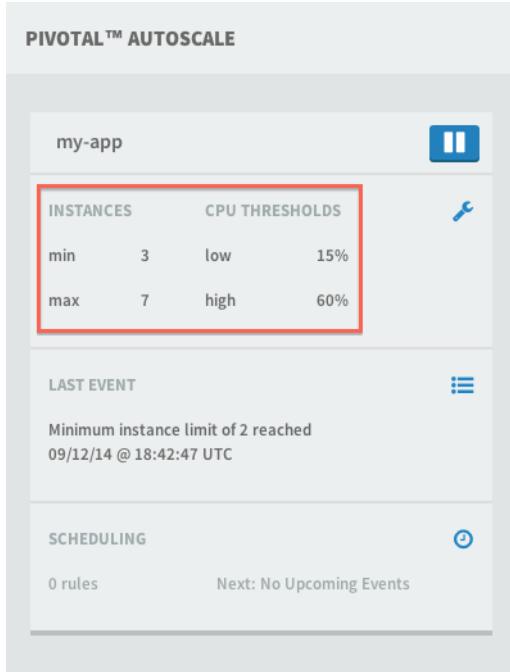
4. By default, new autoscaling instances are paused. If paused, click **Turn On** to enable monitoring of your application and start the autoscaling process.

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.



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: . 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: . 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 or greater than . **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.

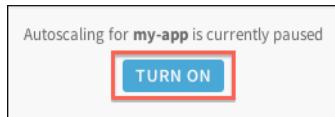
The autoscaling service calculates CPU utilization as a moving average across the CPUs of all currently running instances of an application.

- **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.



Managing Scheduled Scaling in the App Autoscaling Service

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

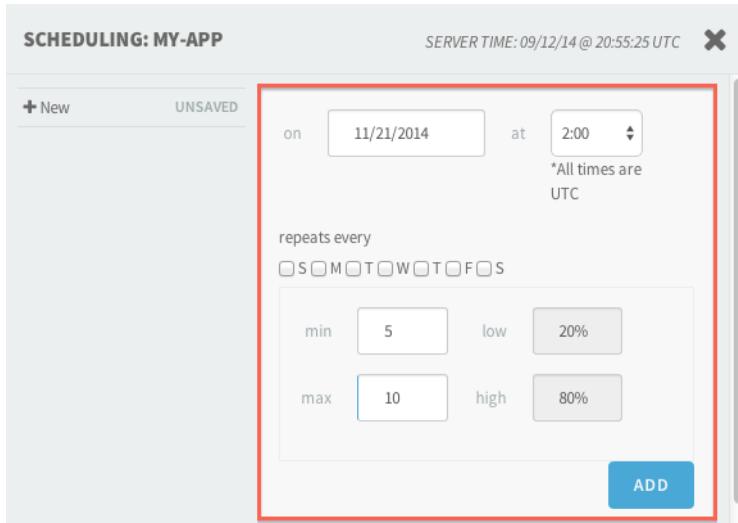
1. Log in to the Developer Console: [How to log in.](#)
2. Select a space containing an App Autoscaling service instance from the org dashboard or from the left navigation bar.
3. In the Services section of the space dashboard, under your App Autoscaling service instance name, click **Manage**.

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

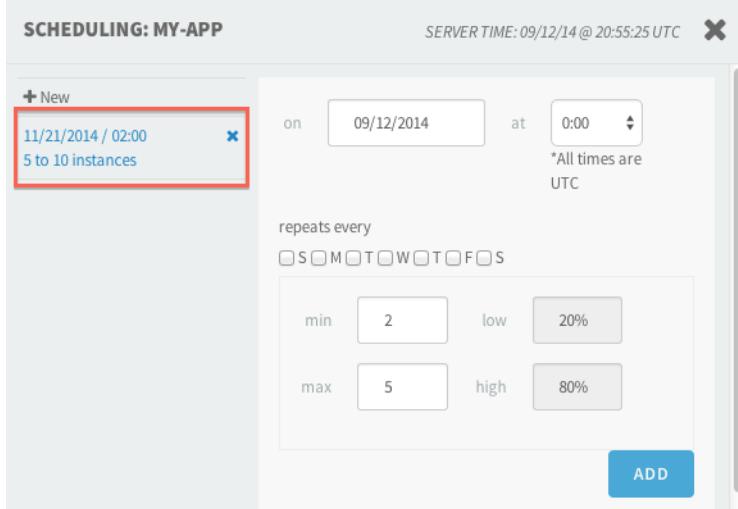
4. By default, new autoscaling instances are paused. If paused, click **Turn On** to enable monitoring of your application and start the autoscaling process.

5. Click the clock icon on your Autoscaling dashboard.

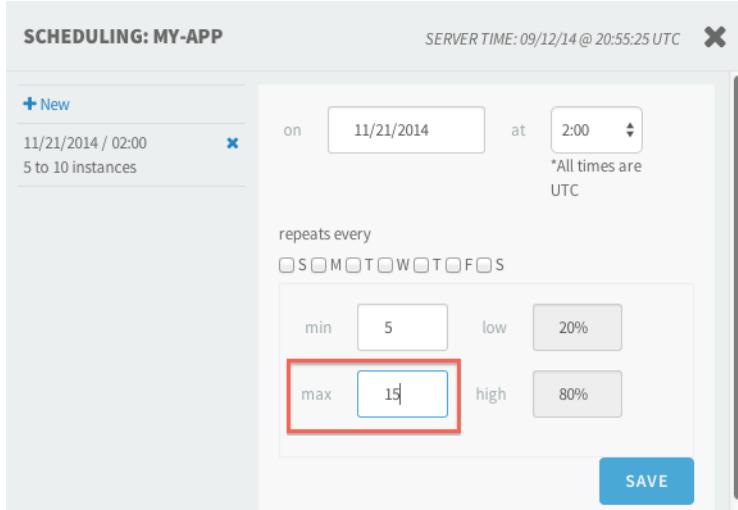
6. In the Scheduling interface, create a new rule by editing the date and time fields and choosing values for the number of minimum and maximum instances. When finished, click **Save**. See the [Rule Types](#) section of this topic for more information.



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



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



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

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

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**

11/28/2014 / 02:00 **5 to 10 instances**

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

repeats every **S M T W 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**

11/28/2014 / 02:00 **5 to 10 instances**

Mon, Fri / 04:00 **10 to 20 instances**

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

repeats every **S M T W 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.

Backing Up Pivotal Cloud Foundry

Refer to this topic for help backing up the contents of critical databases and backing up and restoring your [Pivotal Cloud Foundry](#) (PCF) installation settings.

Note: Contact Pivotal Support for help restoring your PCF installation from the backups that you create.

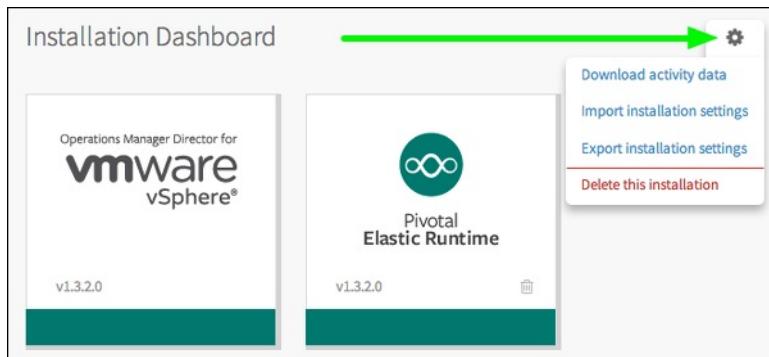
Backing Up and Restoring Installation Settings

Pivotal recommends that you back up your installation settings by exporting frequently. Always export an installation before importing a new one. Import an installation to restore your settings or to share your settings with another user.

Note: Exporting your installation only backs up your installation settings. It does not back up your VMs or any external MySQL databases that you have configured on the Ops Manager [Director Config](#) page.

Exporting an Installation

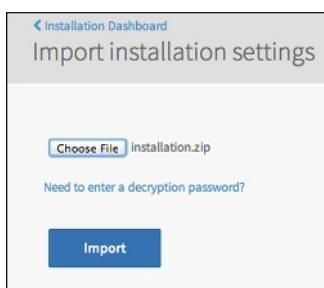
From the Product Installation Dashboard in the Ops Manager interface, click the gear icon and select **Export installation settings**. This option is only available after you have deployed at least one time.



Export installation settings exports the current PCF installation settings and assets. When you export an installation, the exported file contains the base VM images, necessary packages, and references to the installation IP addresses. As a result, an export can be 5 GB or more.

Importing an Installation

1. From the Product Installation Dashboard, click the gear icon and select **Import installation settings**.
2. Browse to and select a PCF installation file.
3. 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.



4. Click **Import**.

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

overwrites any existing installation.

Backing Up the Cloud Controller DB 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	

Download the BOSH Deployment Manifest

1. Install Ruby and the [BOSH CLI Ruby gem](#) on a machine outside your PCF system 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	CLUSTER
Ops Manager Director	0	10.0.0.3	director

3. 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]

4. From a command line, target 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`.

5. Run `bosh deployments` to identify the name of the current BOSH deployment:

```
$ bosh deployments
+-----+-----+-----+
| Name      | Release(s)      | Stemcell(s)
+-----+-----+-----+
| cf-5678dcba5678 | cf-mysql/10      | bosh-vsphere-esxi-ubuntu-trusty-go_agent/2690.3 |
|                   | cf/183.2          |                                 |
|                   | push-console-release/207 |                                 |
+-----+-----+-----+
```

6. 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.

Example:

```
$ bosh download manifest cf-5678dcba5678 cf.yml
Deployment manifest saved to `cf.yml'
```

Backing Up Critical Databases

Refer to this section for help backing up databases associated with your PCF installation.

Your Elastic Runtime deployment contains several critical data stores that must be present for a complete restore. You must back up each of the following:

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

Back Up the Cloud Controller Database

1. Download the BOSH manifest as instructed in the [Download the BOSH Manifest](#) section.
2. Run `bosh deployment DEPLOYMENT-MANIFEST` to set your deployment.

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

3. Run `bosh vms DEPLOYMENT-NAME` to view all the VMs in your selected deployment. Notice that all the Cloud Controller VMs begin with `api`.

```
$ bosh vms cf
Deployment `cf'

Director task 1430

Task 1430 done

+-----+-----+-----+
| Job/index | State | Resource Pool | IPs |
+-----+-----+-----+
| api_worker_z1/0 | running | small_z1 | 10.10.xx.xx |
| api_worker_z2/0 | running | small_z2 | 10.10.xx.xx |
| api_z1/0 | running | large_z1 | 10.10.xx.xx |
| api_z1/1 | running | large_z1 | 10.10.xx.xx |
| api_z2/0 | running | large_z2 | 10.10.xx.xx |
| api_z2/1 | running | large_z2 | 10.10.xx.xx |
| clock_global/0 | running | medium_z1 | 10.10.xx.xx |
| etcd_z1/0 | running | medium_z1 | 10.10.xx.xx |
| hm9000_z1/0 | running | medium_z1 | 10.10.xx.xx |
| loggregator_trafficcontroller_z1/0 | running | small_z1 | 10.10.xx.xx |
| nats_z2/0 | running | medium_z2 | 10.10.xx.xx |
| router_z1/0 | running | router_z1 | 10.10.xx.xx |
| uaa_z2/0 | running | medium_z2 | 10.10.xx.xx |
+-----+-----+-----+
VMs total: 13
```

4. Run `bosh -d DEPLOYMENT-MANIFEST stop SELECTED-VM` for each Cloud Controller VM. Example:

```
$ bosh -d cf.yml stop api_z2/1

Processing deployment manifest
-----
Processing deployment manifest
-----
You are about to stop api_z2/1

Processing deployment manifest
-----
Detecting deployment changes
-----
Stop api_z2/1? (type 'yes' to continue): yes

Performing `stop api_z2/1'...

Director task 1424
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 (00:00:00)

Started preparing dns > Binding DNS. Done (00:00:00)

Started preparing configuration > Binding configuration. Done (00:00:13)

Task 1424 done

Started 2015-02-25 17:52:23 UTC
Finished 2015-02-25 17:52:37 UTC
Duration 00:00:14

api_z2/1 has been stopped
```

5. In `cf.yml`, locate the `ccdb` component and record the IP address:

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

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

Cloud Controller	Vm Credentials	vcap / [REDACTED]
------------------	----------------	-------------------

7. SSH into the CCDB VM as the admin using the IP address and password recorded in the previous steps.

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

8. Run `find /var/vcap | grep 'bin/psql'` to find the locally installed psql client on the CCDB VM.
Example:

```
$ root@10.85.52.96:~# find /var/vcap | grep 'bin/psql'
/var/vcap/data/packages/postgres/5.1/bin/psql
```

9. Run `pg_dump` from the locally installed psql client to export the database:
Example:

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql/pg_dump -h 10.85.52.96 -U admin -p 2544 ccdb > ccdb.sql
```

10. Exit from the CCDB VM.

11. Run `scp` to copy the exported database to your local machine.
Example:

```
$ scp vcap@10.85.52.96:ccdb.sql
```

Back Up the UAA Database

1. In the BOSH deployment manifest, 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 / [REDACTED]
--------------	----------------	-------------------

3. SSH into the UAA DB VM as the admin using the IP address and password recorded in the previous steps.

4. Run `find /var/vcap | grep 'bin/psql'` to find the locally installed psql client on the UAA DB VM.
Example:

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

5. Run `pg_dump` from the locally installed psql client to export the database:
Example:

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql/pg_dump -h 10.85.52.101 -U root -p 2544 uaa > uaa.sql
```

6. Exit from the UAA DB VM.

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

```
$ scp vcap@10.85.52.101:uaa.sql
```

Back Up the Apps Manager Database

1. In the BOSH deployment manifest, locate the `consolddb` component and record the IP address and password:

```
consolddb:  
  address: 10.85.52.104  
  port: 2544  
  db_scheme: postgresql
```

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



3. SSH into the Apps Manager DB VM as the admin using the IP address and password recorded in the previous steps.

4. Run `find /var/vcap | grep 'bin/psql'` to find the locally installed psql client on the Apps Manager DB VM.

Example:

```
$ root@10.85.52.104:~# find /var/vcap | grep 'bin/psql'  
/var/vcap/data/packages/postgres/5.1/bin/psql
```

5. Run `pg_dump` from the locally installed psql client to export the database:

Example:

```
$ /var/vcap/data/packages/postgres/5.1/bin/psql/pg_dump -h 10.85.52.104 -U root -p 2544 console > console.sql
```

6. Exit from the Apps Manager DB.

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

Example: `$ scp vcap@10.85.52.104:console.sql`

Back Up the NFS Server

1. In the BOSH deployment manifest, locate the `nfs_server` component and note 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.



3. SSH into the NFS server VM using the NFS Server 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 the NFS server might be large. To estimate the size of the TAR file before you create it, run the following command: `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.

1. From the Installation Dashboard in Ops Manager, select **Pivotal Elastic Runtime**. Click **Status** and record the CID of **MySQL Server**.

Pivotal Elastic Runtime			
Settings	Status	Credentials	Logs
JOB	INDEX	IPS	CID
Login	0	10.85.52.97	vm-868c1e43-ab54-4ae3-84cc-aa5e1deb07d0
Console Database	0	10.85.52.104	vm-fa08079c-3baa-4a11-b643-f198451fd085
MySQL Server	0	10.85.52.105	vm-f7ec423b-675d-4b14-bf8f-2034d27d61ef

2. Download and save the BOSH deployment manifest for MySQL Server. You need this manifest to locate information about your databases.

```
$ bosh download manifest p-mysql-1e136692e37b3ec1f942 mysql.yml
Deployment manifest saved to `mysql.yml`
```

3. In mysql.yml, locate the IP address and password for the MySQL node:

```
mysql_node:
  host: 10.85.52.105
  admin_password: c0eeec42e465428312fd
  services:
    - name: p-mysql
```

4. Dump the data from the p-mysql DB and save it:

```
$ mysqldump -u root -p -h 10.85.52.105 --all-databases > user_databases.sql
```

Note: Contact Pivotal Support for help restoring your PCF installation from the backups that you create.

Start Cloud Controller

Run `bosh start SELECTED-VM` for each stopped VM to restart the Cloud Controller. For information on how to review stopped VMs, refer to the [Back Up the Cloud Controller Database](#) section.

The following example demonstrates starting a cloud controller VM:

```

$ bosh start api_z2 1

Processing deployment manifest
-----
Processing deployment manifest
-----
You are about to start api_z2/1

Processing deployment manifest
-----
Detecting deployment changes
-----
Start api_z2/1? (type 'yes' to continue): yes

Performing `start api_z2/1'...

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 api_z2 > api_z2/1 (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

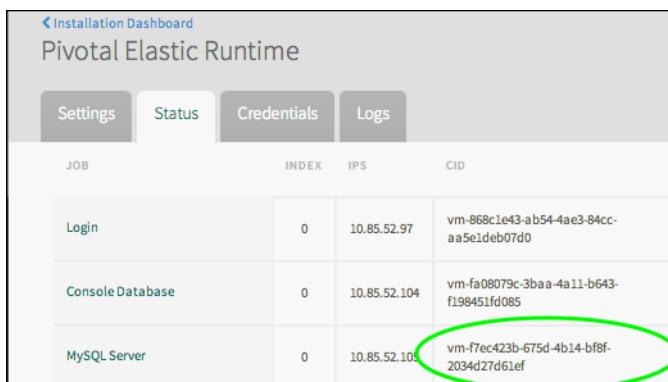
api_z2/1 has been started

```

Back Up Pivotal MySQL Server

Note: The Elastic Runtime deploy contains an embedded MySQL Server that serves as the data store for the `app_usage_events` service.

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



JOB	INDEX	IPS	CID
Login	0	10.85.52.97	vm-868c1e43-ab54-4ae3-84cc-aa5e1deb07d0
Console Database	0	10.85.52.104	vm-fa08079c.3baa-4a11-b643-f198451fd085
MySQL Server	0	10.85.52.105	vm-f7tec423b-675d-4b14-bf8f-2034d27d61ef

- Download and save the BOSH deployment manifest for MySQL Server. You need this manifest to locate information about your databases.

```
$ bosh download manifest p-mysql-1e136692e37b3ec1f942 mysql.yml
Deployment manifest saved to `mysql.yml'
```

3. In mysql.yml, locate the IP address and password for the MySQL node:

```
mysql_node:
  host: 10.85.52.105
  admin_password: c0eec42e465428312fd
  services:
    - name: p-mysql
```

4. Dump the data from the p-mysql DB and save it:

```
$ mysqldump -u root -p -h 10.85.52.105 --all-databases > user_databases.sql
```

 **Note:** Contact Pivotal Support for help restoring your PCF installation from the backups that you create.

Upgrading Operations Manager

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

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

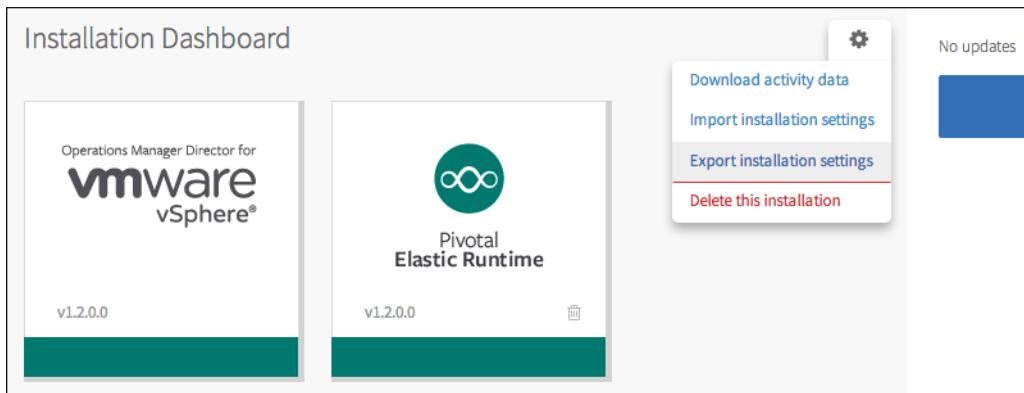
Note: To upgrade your [Pivotal Cloud Foundry](#) (PCF) installation to a target release, you must install all releases from your currently deployed version to the target version in sequential order. For example, if your deployment uses Ops Manager release 1.1 and you are upgrading to 1.3, you must sequentially install 1.2 and 1.3.

Upgrading with Installed Products

Follow these steps 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.

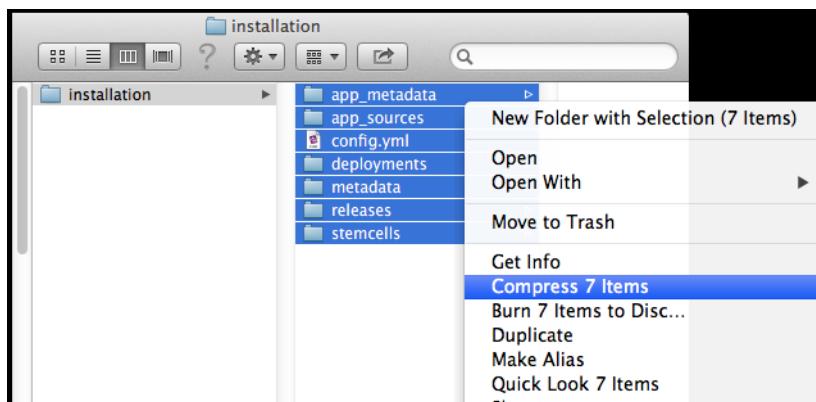
1. From the Product Dashboard, select **Actions > Export installation settings**.



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.

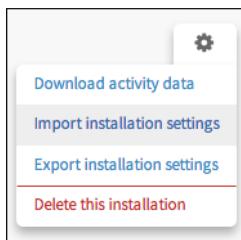
- Because of the size of the exported file, exporting can take tens of minutes.
- Some browsers do not provide feedback on the status of the export process, and may appear to hang.
- Some operating systems may automatically unzip the exported installation. If this occurs, create a zip file of the unzipped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

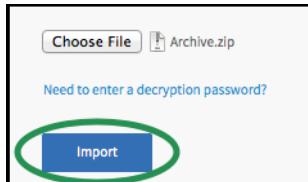


2. Download the latest Ops Manager VM Template from the [Pivotal Network](#) site.

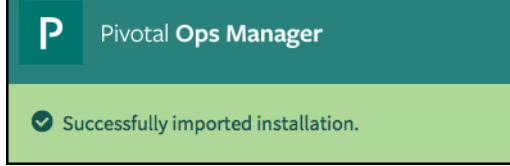
3. Record the IP address of the existing Ops Manager VM.
4. To avoid IP conflicts, power off the existing Ops Manager VM.
5. Deploy the new Ops Manager VM. See [Deploying Operations Manager to vSphere](#) or [Deploying Operations Manager to vCloud Air and vCloud](#).
6. From the Product Dashboard, select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



7. Click **Choose File**, browse to the installation zip file exported in Step 1, and click **Choose**.
8. 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.



9. 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.
10. When complete, a "Successfully imported installation" message should appear.



11. Verify the new installation functions correctly.
12. Remove the original Ops Manager VM 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 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		✓	

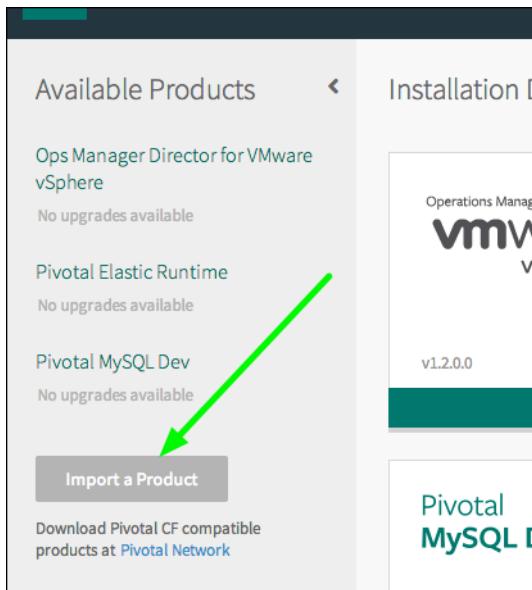
Upgrading Products in a PCF Deployment

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

This topic 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.
3. Browse to the Pivotal Cloud Foundry Operations Manager web interface and click **Import a Product**.



4. Upload the new version of a product you want to upgrade.
5. Under **Available Products**, click **Upgrade** for the uploaded product.
6. Repeat the import, upload, and **Upgrade** steps for each product you downloaded.
7. 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**.

8. Click **Apply changes**.

Monitoring VMs in PCF

This topic covers strategies for monitoring VM status and performance in [Pivotal Cloud Foundry](#).

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	Download					
NATS	0	10.0.0.5	vm-dee49615-aea8-4f4f-bf0f-b1060083ddef	0.12%	0.1%	9.7%	0.0%	41%	21%	N/A	Download					
			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

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 two virtual machines: the JMX Provider and a VM that governs compilation.

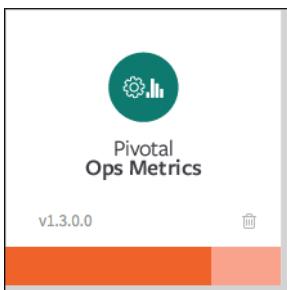
Deploy Pivotal Ops Metrics using the [Pivotal Cloud Foundry](#) Operations Manager as follows:

Adding the Pivotal Ops Metrics Product

See [Adding and Importing Products](#).

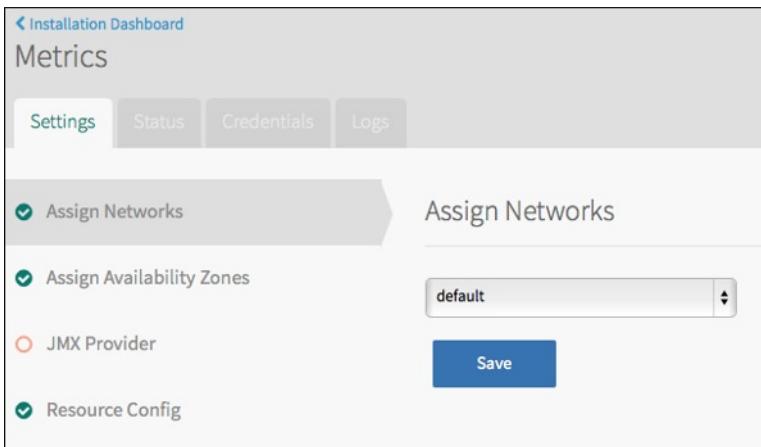
Configuring Pivotal Ops Metrics

1. On the Installation Dashboard, click the **Pivotal Ops Metrics** tile.



Note the orange bar on the **Pivotal Ops Metrics** tile. This indicates the product requires configuration.

2. Select **Assign Networks Zones**. Use the drop-down menu to select a Network. Ops Manager Metrics will use the default Assigned Network if this field is not altered.



3. **(vSphere Only)** Select **Assign Availability Zones**. These are the Availability Zones that you [create](#) when configuring Ops Manager Director.
 - Select an Availability Zone under **Place singleton jobs**. Ops Manager runs Metrics jobs with a single instance in this Availability Zone.
 - 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.

Metrics

Settings Status Credentials Logs

Assign Networks

Assign Availability Zones

JMX Provider

Resource Config

Availability Zone Assignments

Place singleton jobs in

default

Balance other jobs in

default

Save

4. Create a username and password. JMX clients use these credentials to connect to the JMX Provider.
5. **(Optional)** Check **Enable SSL**. 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

Resource Config

JMX Provider credentials *

admin

Enable SSL

SSL Certificate

Certificate PEM

Private Key PEM

Generate Self-Signed RSA Certificate

Save

If you check **Enable SSL**, 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** field.
- If you want to use SSL but do not want to use a signed certificate, you must:
 - Generate a self-signed certificate on the server
 - Import the self-signed certificate to a trust store on the client
 - Start jConsole (or your 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

Assign Availability Zones

JMX Provider

Resource Config

Credentials to connect to JMX Provider

JMX Provider credentials *

Enable SSL

SSL Certificate

```
-----BEGIN CERTIFICATE-----  
MIIDJCCAg2gAwIBAgIVAP2UDTsMs56HI  
UstM4RoYYPMruhuMA0GCSqGSIb3DQE  
B  
BQUAMDsxCzAJBgNVBAYTANTRAwDgY  
-----BEGIN RSA PRIVATE KEY-----  
MIIEpgIBAAKCAQEazINFM7S/rpn1rLyKM  
aK11yYFygii/NB926E51SwXfCapS5  
HSof6zER1r1SijwtgZ7nHobjJmp3UnWV5q  
lxSseKpgSJaQkM+u5/zbwZj4gAg+F5O
```

[Generate Self-Signed RSA Certificate](#)

Save

6. In the Pending Changes view, click **Apply Changes** to install Pivotal Ops Metrics.

Installation Dashboard

Pending Changes

Recent Install Logs

Apply changes

Operations Manager Director for vmware vSphere® v1.3.0.0	Pivotal Elastic Runtime v1.3.0.0.alpha.323.9052878
Pivotal Ops Metrics v1.3.0.0	

7. When complete, a “Changes Applied” message appears. Click **Return to Product Dashboard**.

8. Click the **Pivotal Ops Metrics** tile and select the **Status** tab.

9. 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 you supplied.

Metrics

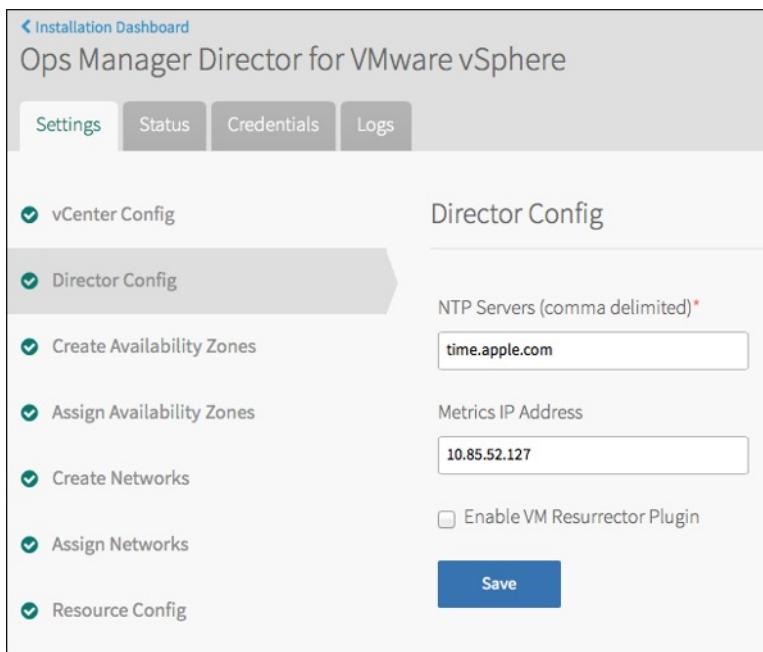
Settings Status Credentials Logs

Jobs on Availability Zone "default"

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

10. Return to the **Installation Dashboard**. Click the **Ops Manager Director** tile and select **Director Config**.

11. In the **Metrics IP address** field, enter the IP address of the JMX Provider. Click **Save**.



Ops Manager Director for VMware vSphere

Director Config

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

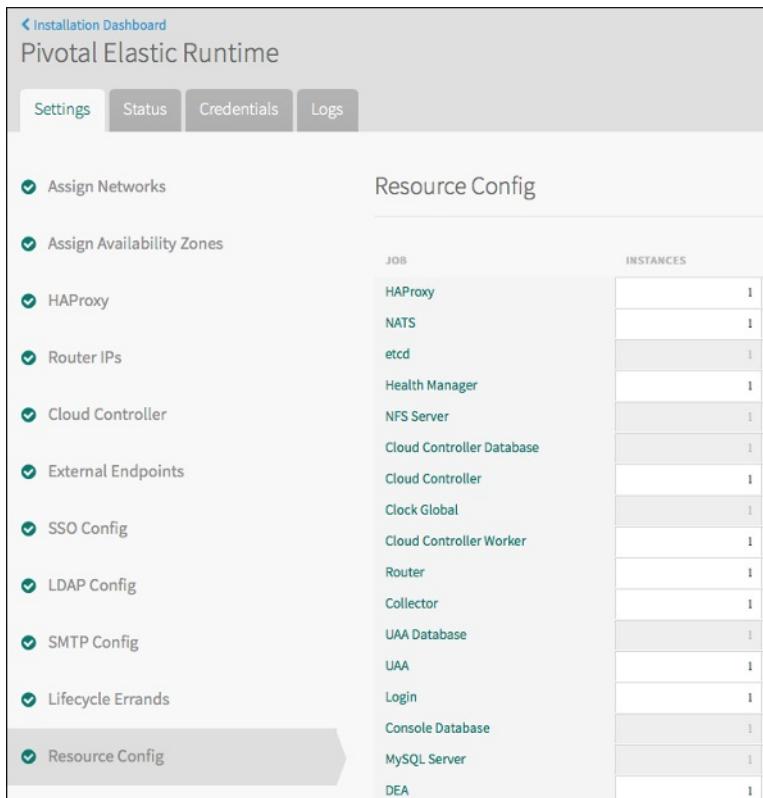
Metrics IP Address
10.85.52.127

Enable VM Resurrector Plugin

Save

12. Return to the **Installation Dashboard**. Click the **Pivotal Elastic Runtime** tile and select **Resource Config**.

13. Change the number of **Instances** for the **Collector** job from 0 to 1. Click **Save**.

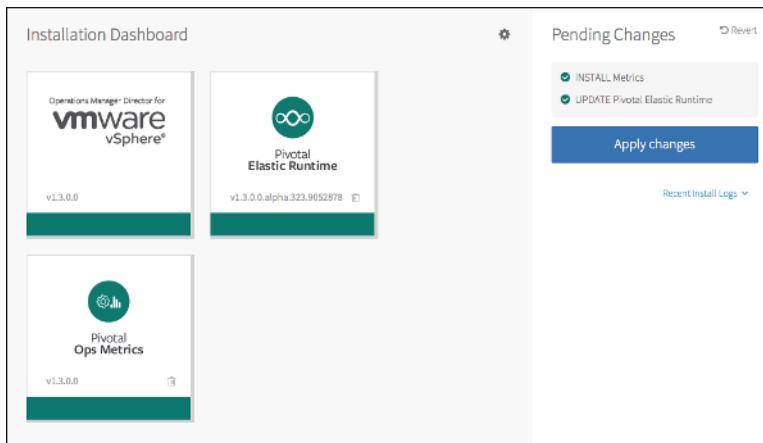


Pivotal Elastic Runtime

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

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



15. 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.

Using SSL with a Self-Signed Certificate in Pivotal Ops Metrics

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.

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.
- **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.

Clients keep all trusted certificates in a kind of keystore called a truststore. To configure a client to trust a self-signed certificate, import the self-signed certificate to a truststore on the client.

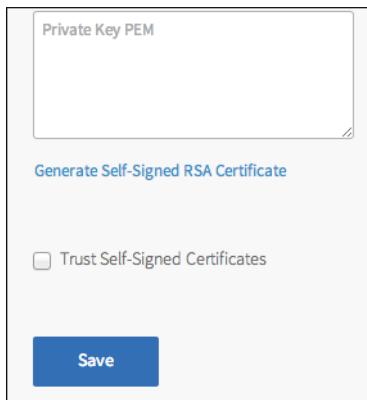
Follow these steps to use SSL with a self-signed certificate:

1. [Generate](#) the self-signed certificate on the server.
2. [Import](#) the self-signed certificate to a truststore on the client.
3. [Start](#) jConsole or another monitoring tool on the client with the truststore.

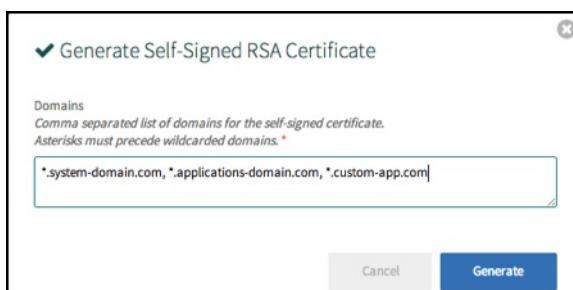
Generating a Self-Signed Certificate

To generate a self-signed certificate on your server:

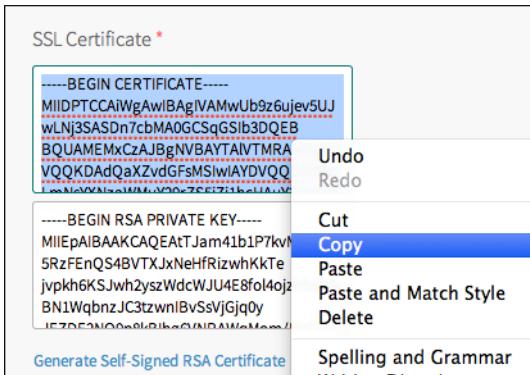
1. In Pivotal Ops Manager, click the **Pivotal Ops Metrics** tile.
2. Check **Enable SSL**.
3. Click **Generate Self-Signed RSA Certificate** and check the **Trust Self-Signed Certificates** box.



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 `ops-metrics.cer`.

Importing the Self-signed Certificate to a Truststore

To import the self-signed certificate to your client:

1. Copy `ops-metrics.cer` from your server to your client.
2. Navigate to the client directory where you copied the saved certificate.
3. Run the keytool command:

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

This command imports the certificate with an alias of `ops-metrics-ssl` to the truststore `localhost.truststore`. If `localhost.truststore` already exists, a prompt for the password appears. If `localhost.truststore` does not exist, you must create a password for it.

4. Verify the details of the imported certificate.

Starting a Monitoring Tool with the Truststore

Once you have imported the self-signed certificate to the `localhost.truststore` truststore on the client, you must instruct your monitoring tool to use the truststore.

On a command line, start your monitoring tool with the location and password of the truststore:

- Pass the location of `localhost.truststore` to the monitoring tool using the `javax.net.ssl.trustStore` property.
- Pass the truststore password using the `javax.net.ssl.trustStorePassword` property.

Example starting jConsole:

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

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

Using Pivotal Ops Metrics

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 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 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 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)

PCF Troubleshooting Guide

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 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, it's a good idea to click the `Install` or `Apply Changes` button. Even doing so repeatedly does no harm.

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 CF. 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.

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	0.08%	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	
			vm-6d43e59e-								

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

Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

FILENAME	UPDATED AT
Downloaded:	
/var/tempest/workspaces/default/jobs_logs/dea-0-792ba29b2bc8.zip	2014-02-20 18:27:22 UTC
Pending:	
/var/tempest/workspaces/default/jobs_logs/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>@<Pivotal_CF_VM_IP_address>` to connect to the 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 console` to target the `system` org and the `console` space.

```
$ cf target -o system -s console
```

3. Run `cf logs console` to tail the Apps Manager logs.

```
$ cf logs console
Connected, tailing logs for app console in org system / space console 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 console LOG_LEVEL` with the desired severity level.

```
$ set-env console 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 1.

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-4d80ede4-b0a5-4992-ae6a0386e18e", "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": "console.deploy"}
```

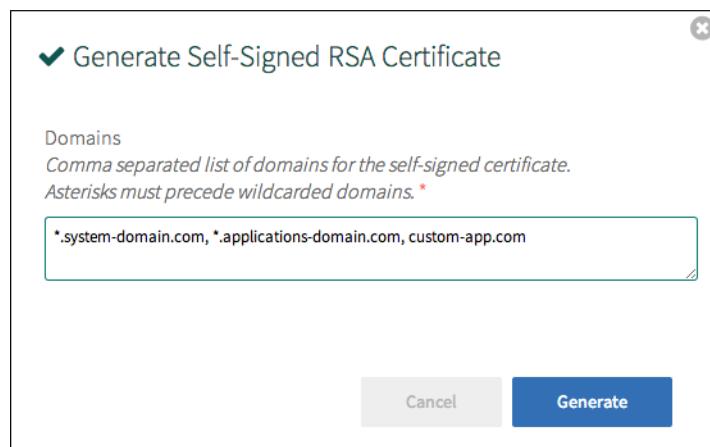
```
/home/tempest-web/tempest/web/vendor/bundle/ruby/1.9.1/gems/mechanize-2.7.2/lib/mechanize/http/agent.rb:306:in
`fetch': 403 => Net::HTTPForbidden for https://login.api.x.y/oauth/authorizeresponse_type=code&client_id=portal&redirect_uri=https%3...
-- unhandled response (Mechanize::ResponseCodeError)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

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 XXXXXXXX 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

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 [PCF Troubleshooting Guide](#).

Common Issues

The following sections list common issues you might encounter and possible resolutions.

PCF Installation Fails

If you modify the vCenter Statistics Interval Duration setting from its default setting of 5 minutes, the PCF installation might fail at the MicroBOSH deployment stage, and the logs might contain the following error message:

`The specified parameter is not correct, interval`. This failure happens because Ops Manager expects a default value of 5 minutes, and the call to this method fails when the retrieved value does not match the expected default value.

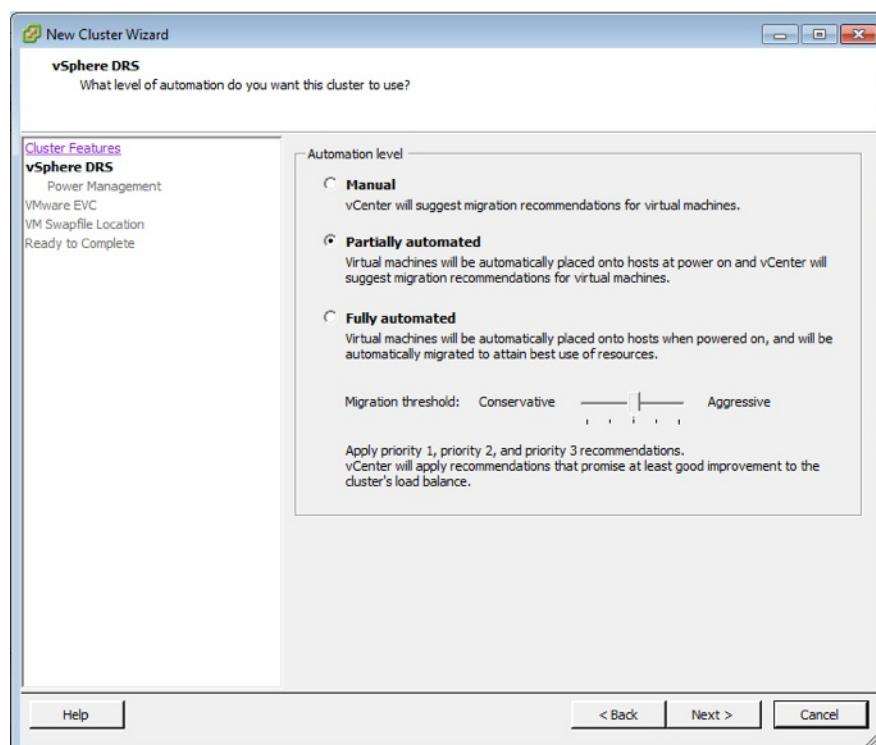
To resolve this issue, launch vCenter, navigate to **Administration > vCenter Server Settings > Statistics**, and reset the vCenter Statistics Interval Duration setting to 5 minutes.

BOSH Automated Installation Fails

Before starting an Elastic Runtime deployment, you must set up and configure a vSphere cluster.

If you enable vSphere DRS (Distributed Resource Scheduler) for the cluster, you must set the Automation level to **Partially automated** or **Fully automated**.

If you set the Automation level to **Manual**, the BOSH automated installation will fail with a `power_on_vm` error when BOSH attempts to create virtual VMs.



Ops Manager Loses Its IP Address After HA or Reboot

Ops Manager can lose its IP address and use DHCP due to an issue in the open source version of VMware Tools. 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 static

# 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

```

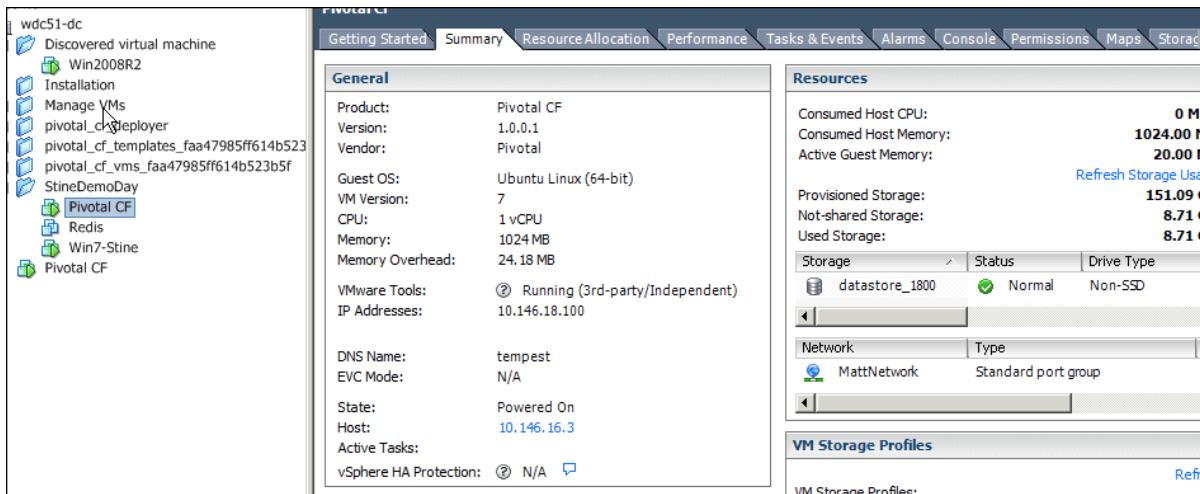
After editing the file run `sudo /etc/init.d/networking restart`. The interfaces files will be overwritten on reboot.

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 [VCNS](#). All communication between PCF VMs and vCenter or ESXi hosts route through the VCNS 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.

Advanced Troubleshooting with the BOSH CLI

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 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.

Installation Dashboard Ops Manager Director for VMware vSphere																
Settings	Status	Credentials	Logs													
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.

JOB	NAME		CREDENTIALS	
	Ops Manager Director	VM credentials	vcap /	██████████
		BOSH agent credentials	vcap /	██████████
		Director credentials	director /	██████████
		NATS credentials	nats /	██████████

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 tempest@OPS-MANAGER-IP-ADDRESS`.
2. When prompted, enter the password you set during the the .ova deployment into vCenter:

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

AWS:

1. Locate the Ops Manager public IP on the AWS EC2 instances 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.

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:

```
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, do the following:
 - a. Replace the `director_uuids` value with the UUID value.
 - b. Run `bosh deployment DEPLOYMENT-FILENAME.yml` to reset the file for your deployment.
4. Run `bosh deployment DEPLOYMENT-FILENAME.yml` to instruct the BOSH Director to apply CLI commands against the deployment described by the YAML file you identified:

```
$ bosh deployment /var/tempest/workspaces/default/deployments/cf-cca18e39287264623408.yml
```

Use the BOSH CLI for Troubleshooting

Many of the BOSH CLI commands are useful in troubleshooting. This section describes three BOSH CLI commands that are particularly useful:

- **VMS:** Lists all VMs in a deployment
- **Cloudcheck:** Cloud consistency check and interactive repair
- **SSH:** Start an interactive session or execute commands with a VM

BOSH VMS

`bosh vms` provides an overview of the virtual machines BOSH is managing 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-console | 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 VMs in an **unknown** state to have BOSH attempt to diagnose the problem.

You can also use `bosh vms` to identify VMs in your deployment, then use `bosh ssh` 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

`bosh cloudcheck` attempts 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` offers 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 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 this reference is in error. Once you delete the VM reference, BOSH can no longer control the VM.

Example Scenarios

Unresponsive Agent

```
$ bosh cloudcheck
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 cloudcheck
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 cloudcheck
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 cloudcheck
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.

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 `bosh ssh` 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-console/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
```

PCF Security Overview and Policy

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

Pivotal reports the severity of vulnerabilities using the following severity classes:

Critical

Critical vulnerabilities are those that can be exploited by an unauthenticated attacker from the Internet or those that break the guest/host Operating System isolation. The exploitation results 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.

Important

Important vulnerabilities are those that are not rated critical but whose exploitation results in the complete compromise of confidentiality and/or integrity of user data and/or processing resources through user assistance or by authenticated attackers. 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.

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.

Table of Contents

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

Cloud Foundry Overview

What is an open PaaS?

Each generation of computing ushers in a new application platform. In the cloud era, the application platform will be delivered as a service, often described as Platform as a Service (PaaS). PaaS makes it much easier to deploy, run, and scale applications.

Not all PaaS offerings are created equal. Some have limited language and framework support, do not deliver key application services needed for cloud applications, or restrict deployment to a single cloud. By offering an open PaaS, you get a choice of clouds for deployment, frameworks for development, and application services for running your application. And as an open source project, there is a community both contributing and supporting Cloud Foundry.

What is Cloud Foundry?

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.

Cloud Foundry takes an open approach to Platform as a Service. Most PaaS offerings restrict developer choices of frameworks, application services, and deployment clouds. The open and extensible nature of Cloud Foundry means developers will not be locked into a single framework, single set of application services, or a single cloud.

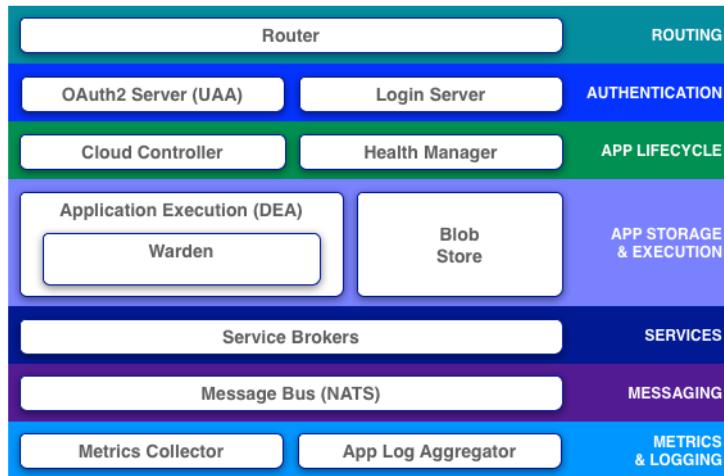
Who should use Cloud Foundry?

Cloud Foundry is ideal for any developer interested in removing the cost and complexity of configuring infrastructure for their applications. Cloud Foundry allows developers to deploy and scale their applications without being locked in to a single cloud. Developers can deploy their applications to Cloud Foundry using their existing tools and with zero modification to their code.

Cloud Foundry Components

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.



Router

The [router](#) routes incoming traffic to the appropriate component, usually the Cloud Controller or a running application on a DEA node.

Authentication

The OAuth2 server (the [UAA](#)) and Login Server work together to provide identity management.

Cloud Controller

The [Cloud Controller](#) is responsible for managing the lifecycle of applications. When a developer pushes an application to Cloud Foundry, she is targeting the Cloud Controller. The Cloud Controller then stores the raw application bits, creates a record to track the application metadata, and directs a DEA node to stage and run the application. The Cloud Controller also maintains records of orgs, spaces, services, service instances, user roles, and more.

HM9000

HM9000 has 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 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 is essential to ensuring that apps running on Cloud Foundry remain available. HM9000 restarts applications whenever the DEA running an app shuts down for any reason, when Warden kills the app because it violated a quota, or

when the application process exits with a non-zero exit code.

Refer to the [HM9000 readme](#) for more information about the HM9000 architecture.

Application Execution (DEA)

The [Droplet Execution Agent](#) manages application instances, tracks started instances, and broadcasts state messages.

Application instances live inside [Warden](#) containers. Containerization ensures that application instances run in isolation, get their fair share of resources, and are protected from noisy neighbors.

Blob Store

The blob store holds:

- Application code
- Buildpacks
- Droplets

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.

Message Bus

Cloud Foundry uses [NATS](#), a lightweight publish-subscribe and distributed queueing messaging system, for internal communication between components.

Logging and Statistics

The metrics collector gather metrics from the components. Operators can use this information to monitor an instance of Cloud Foundry.

The application log aggregator streams application logs to developers.

(Go)Router

The router routes traffic coming into Cloud Foundry to the appropriate component: usually, [Cloud Controller](#) or a running application on a [DEA](#) node. 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, via HTTP CONNECT). A single process contains all routing logic, removing unnecessary latency.

Getting started

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 c29tZXVzZXI6c29tZXhC3M=  
> 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

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

The simplest way to get started is to fetch the UAA from Git, and 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 all be running 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:

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

Local Command Line Usage

First run the UAA server as described above:

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

Then start another terminal and, from the project base directory, ask the login endpoint to tell you 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"]}
}
```

Then you can try logging in with the UAA Ruby gem. Make sure you have Ruby 1.9, then:

```
$ 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 `cf` CLI. The token is stored in `~/.uaac.yml`. Open that file 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.

Then you can login 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:

```
jti: 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. Ask the external login endpoint to tell you about the system:

```
$ curl -H "Accept: application/json" uaa.run.pivotal.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. Make sure you have Ruby 1.9, then:

```
$ gem install cf-uaac  # If you have not already installed cf-uaac
$ uaac target uaa.run.pivotal.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

You can run the integration tests with the following command:

```
$ ./gradlew integrationTest
```

This will start a uaa server running in a local Apache Tomcat instance, so for example, 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 (see below for detail).

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.mozyccloud
  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:${CLOUD_FOUNDRY_CONFIG_PATH}/uaa.yml
file:${UA_CONFIG_FILE}
${UA_CONFIG_URL}
```

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 actually several projects here—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 actual 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 (the end user).
- `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 HSQL:

```
$ 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.

Cloud Controller

The Cloud Controller is written in Ruby and provides REST API endpoints for clients to access the system. The Cloud Controller maintains a database with tables for orgs, spaces, apps, services, service instances, user roles, and more.

Database (CC_DB)

The Cloud Controller database has been tested with Postgres.

Blob Store

The Cloud Controller manages a blob store for:

- 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 - the result of taking an app package and staging it (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, it performs the following using NATS:

- Instructs a DEA to stage an application (processes a buildpack for the app) to prepare it to run
- Instructs a DEA to start or stop an application
- Receives information from the Health Manager about applications
- Subscribes to Service Gateways that advertise available services
- Instructs Service Gateways to handle provisioning, unprovision, bind and unbind operations for services

Testing

By default `rspec` will run a test suite with sqlite3 in-memory database. However, you can specify a connection string using the `DB_CONNECTION` environment variable to test against postgres and mysql. Examples:

```
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 uses [Steno](#) to manage its logs. 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](#), Cloud Foundry's OAuth 2.0 server.

Droplet Execution Agent

The key functions of a Droplet Execution Agent (DEA) are:

- 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 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 `dea.yml`.

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
## <a id='run-standalone'></a>Running the DEA Standalone ##
```

When you contribute to the DEA, it is useful to run it as a standalone component. This section has instructions for doing so on Vagrant 1.1x.

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

Follow these steps to set up DEA to run locally on your computer:

```

$ 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
```

VM creation is likely to take a while.

If the `rake test_vm` step fails and you see an error similar to “undefined method ‘configure’ for Vagrant” or “found character that cannot start any token while scanning for the next token,” you are using a unsupported version of

Vagrant. Install 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](#) page for a description of the application staging flow the DEA uses.

These are the staging messages that a DEA publishes on NATS:

- `staging.advertise` — Stagers (now DEAs) 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 (old protocol)

Logs

[Steno](#) handles the DEA logging. You can configure the DEA to log to a file, a syslog server, or both. If neither is provided, the DEA logs to its stdout. The logging level specifies the verbosity of the logs (e.g. `warn`, `info`, `debug`).

Warden

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.

Components

This repository 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 short 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 the right 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
```

Setup Warden

Run the setup routine, which 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, try `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
```

Implementation for Linux

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.

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 gets 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. In fact, early versions of Warden even **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. It 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.

Messaging (NATS)

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

Getting Started

```
$ 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, queuing 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_recv = true }

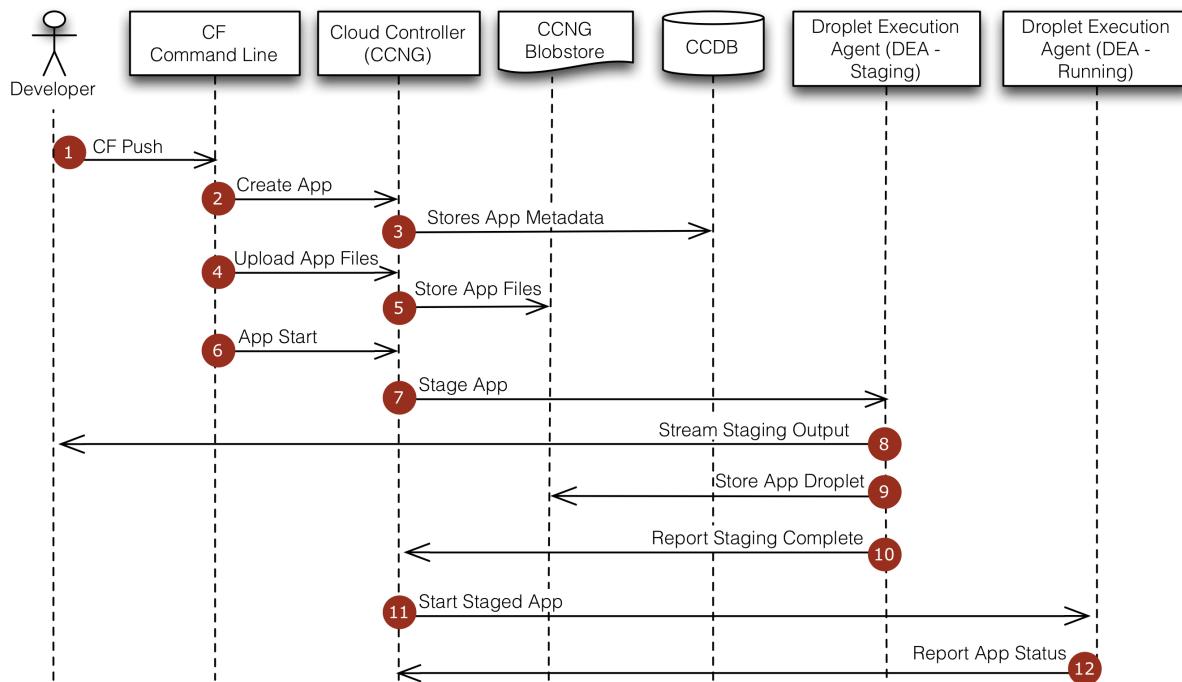
# Timeout unless a certain number of messages have been received
NATS.timeout(sid, TIMEOUT_IN_SECS, :expected => 2) { timeout_recv = true }

# 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!') }
```

How Applications Are Staged



1. At the command line, the developer enters the directory containing her application and uses the cf command line tool to issue a push command.
2. The cf command line tool tells the Cloud Controller to create a record for the application.
3. The Cloud Controller stores the application metadata (e.g. the app name, number of instances the user specified, and the buildpack).
4. The cf command line tool uploads the application files.
5. The Cloud Controller stores the raw application files in the blobstore.
6. The cf command line tool issues an app start command.
7. Because the app has not already been staged, the Cloud Controller chooses a DEA instance from the DEA pool to stage the application. The staging DEA uses the instructions in the buildpack to stage the application.
8. The staging DEA streams the output of the staging process so the developer can troubleshoot application staging problems.
9. The staging DEA packages the resulting staged application into a tarball called a "droplet" and stores it in the blobstore. The results are cached and used next time the application is staged.
10. The staging DEA reports to the Cloud Controller that staging is complete.
11. The Cloud Controller chooses one or more DEAs from the pool to run the staged application.
12. The running DEAs report the status of the application to the Cloud Controller.

Zero Downtime Deployment and Scaling in CF

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.

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

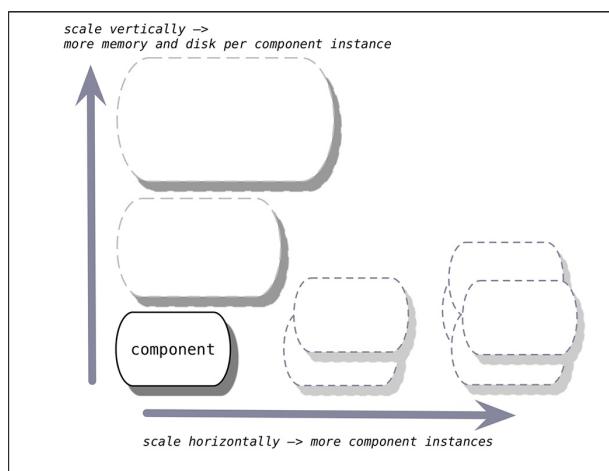
Configure your [resource pools](#) according to the requirements of your deployment.

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 DEAs.

- 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.

Component	Number	Notes
Load Balancer	1	
NATS Server	≥ 2	If you lack the network bandwidth, CPU utilization, or other resources to deploy two stable NATS servers, Pivotal recommends 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.
Doppler Server (formerly Loggregator Server)	≥ 2	Deploying additional Doppler servers splits traffic across them.
Loggregator Traffic Controller	≥ 2	Deploying additional Loggregator Traffic Controllers allows you to direct traffic to them in a round-robin manner.

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 Pivotal PSO if you need help.

SAML Login Server:

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

Secondly, there are components whose availability does not affect that of the platform as a whole. For these, recovery by normal IT procedures should be sufficient even in a high availability Cloud Foundry deployment.

Syslog:

An event that degrades availability of the Syslog VM causes a gap in logging, but otherwise Cloud Foundry continues to operate normally.

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

Collector:

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

Compilation:

This component is active only during platform installation and upgrades.

Etcd:

Etcd is a highly-available key value store used for shared configuration and service discovery. More information on running etcd on single node is available [here](#)

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 Pivotal PSO if you require replicated databases or any assistance.

Orgs, Spaces, Roles, and Permissions

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.

Spaces

Every application and service is scoped to a space. Each org contains at least one space. A space provides a set of users 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:

- 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

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:

- View users and org roles
- View the org quota

Space Roles and Permissions

Space Manager

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

A Space Manager can:

- 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:

- 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:

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

Understanding Cloud Foundry Security

Pivotal protects customers from security threats by applying security controls and by isolating customer applications and data.

Cloud Foundry:

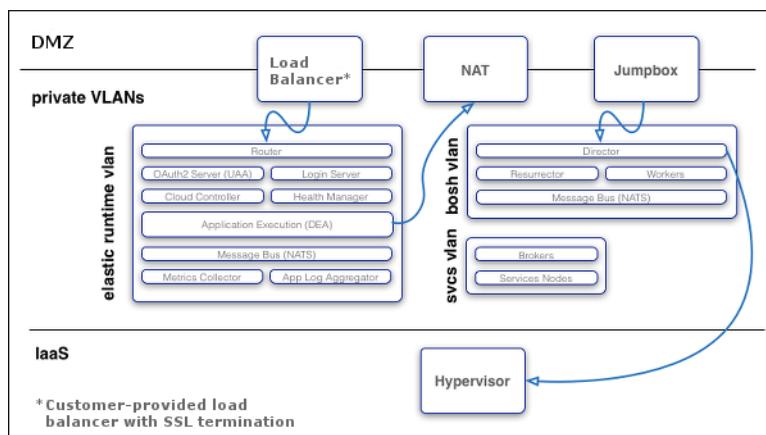
- Implements role-based access controls, applying and enforcing roles and permissions to ensure that users can only view and affect the spaces for which they have been granted access.
- Ensures security of application bits in a multi-tenant environment.
- Prevents possible denial of service attacks through resource starvation.

Before you read this document you might want to become familiar with 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.

Application Traffic

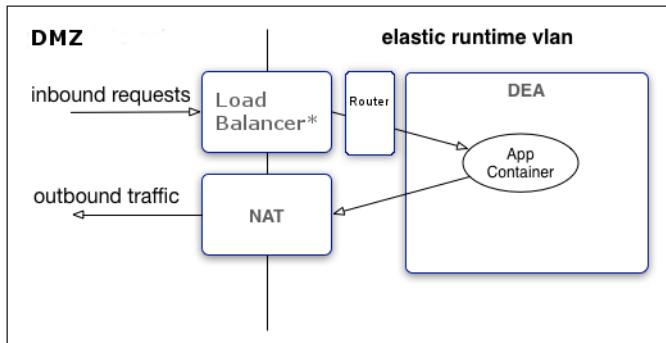
When an app instance starts, the DEA allocates an IP address and also assigns an arbitrary port to the Application Container. The application uses the PORT environment variable provided in the container environment to determine which port to listen on. Because the DEA assigns a random value to the PORT environment variable, the value is generally unique for each application instance.

The router handles all inbound traffic to applications, routing traffic to one of the application instances.

A DEA 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:

- **Inbound:** From the load balancer through the router to the DEA, then from the DEA to the App Container.
- **Outbound:** From the App Container to the DEA, then to the gateway on the DEA virtual network interface. This gateway might be a NAT to external networks depending on your IaaS.

The image below shows traffic flow through the recommended deployment configuration.



Network Traffic Rules

Cloud Foundry implements network traffic rules using Linux iptables on the component VMs. Operators can configure rules to prevent system access from external networks and between internal components, and to restrict applications from establishing connections over the DEA network interface.

Operators can use either of the following to configure these rules:

- Cloud Foundry [Application Security Groups](#): An Application Security Group (ASG) consists of a list of access rules to control application outbound traffic. Operators can define Cloud Foundry ASGs both in a manifest and on the cf Command Line Interface (CLI), however, Cloud Foundry ASGs defined in a manifest are valid only until operators update them on the CLI. For this reason, Pivotal recommends that operators use the CLI to create and bind Cloud Foundry ASGs to the running and staging deployment phases for greater flexibility.
- [DEA Network Properties](#): Operators can configure the `allow_networks` and `deny_networks` parameters for DEAs to prohibit communication between system components and applications. Any subsequent Cloud Foundry ASG configurations will overwrite these configurations.

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.

Spoofing

If an IP, MAC, or ARP spoofing attack bypasses the physical firewall for your deployment, Cloud Foundry network traffic rules help prevent the attack from accessing application containers. Cloud Foundry uses application isolation, operating system restrictions, and encrypted connections to further mitigate risk.

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 on 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](#).

Security for Service Broker Integration

The Cloud Controller [authenticates](#) every request with the Service Broker API using HTTP or HTTPS, depending on which protocol 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.

Application Isolation with Containers

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 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 cannot detect each other. This means that each container is given its own Process ID (PID), namespace, network namespace, and mount namespace.

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.

Networking

The [DEA](#) for each Warden container assigns the container a dedicated virtual network interface. This interface is one side of a virtual ethernet pair created on the host VM. 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 includes 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.

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.

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

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.

Building Stacks

DEAs can support multiple stacks. The scripts for building the available Cloud Foundry stacks (and later for building other stacks) reside in the [stacks](#) project.

Available Stacks

The cflinuxfs2 Stack

The cflinuxfs2 stack is derived from Ubuntu Trusty 14.04, and will be the new default stack for newly pushed applications.

The lucid64 stack

The lucid64 stack is derived from Ubuntu Lucid 10.04. This stack will reach end of support for security fixes on April 29th, 2015.

CLI Commands

Use the `cf stacks` command to list all the stacks available in a deployment.

```
$ cf stacks
Getting stacks in org My-0rg / space development as developer@example.com...
OK

name      description
lucid64   Ubuntu 10.04 on x86-64
cflinuxfs2 Cloud Foundry Linux-based filesystem
```

Use the `cf push APPNAME -s STACKNAME` to change your stack and restage your application.

```
$ cf push testappmx1 -s cflinuxfs2
Using stack cflinuxfs2...
OK
Creating app testappmx1 in org pivotal-pubtools / space test-acceptance as developer@sample-app.com...
OK

Creating route testappmx1.cfapps.io...
OK

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

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

Starting app testappmx1 in org pivotal-application-tools / space test-acceptance as developer@sample-app.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)
  https://devcenter.heroku.com/articles/ruby-default-web-server

----> 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 pivotal-application-tools / space test-acceptance as developer@sample-app.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](#).

Glossary

Term	Definition
API	Application Programming Interface
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>example.com</code> . Domains can also be multi-level and contain sub-domains like the “myapp” in <code>myapp.example.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.
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.
Ops	
Manager or Operations Manager	Operations Manager is a web application that you use to deploy and manage a Pivotal CF PaaS.
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>example.com</code> is the domain when using the route <code>myapp.example.com</code> . It is possible to have a route that represents <code>example.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.
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.

Operator's Guide

This guide covers networking and user management for [Pivotal Cloud Foundry](#) operators.

Table of Contents

- [Isolating a PCF Deployment with a Private Network](#)
- [Understanding the Elastic Runtime Network Architecture](#)
- [Configuring PCF SSL Termination](#)
- [Changing the Quota Plan of an Organization with cf CLI](#)
- [Identifying Elastic Runtime Jobs Using vCenter](#)
- [Understanding the Effects of Single Components on a PCF Upgrade](#)
- [Configuring Single Sign-On](#)
- [Getting Started with the Notifications Service](#)

Isolating a PCF Deployment with a Private Network

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 subnetting the private network.

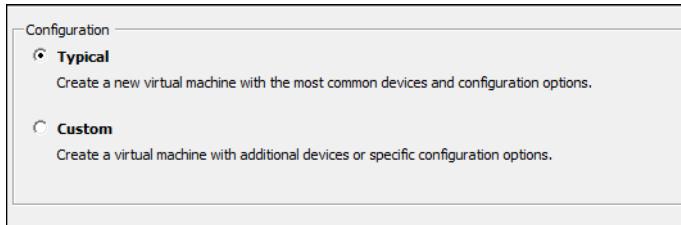
One way to accomplish this is to configure Network Address Translation (NAT) on a VM. The following example explains how to set up a virtual CentOS NAT box in vSphere, and use that to isolate a PCF deployment.

Step 1: Deploy a CentOS image to vSphere

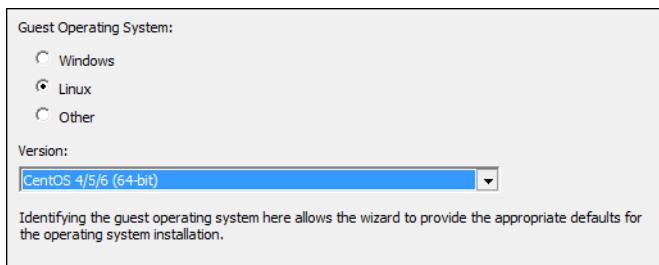
Connect to vCenter using the vSphere client. You will assign two network adapters to the VM. This results in the VM having two IP addresses:

- 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>`.

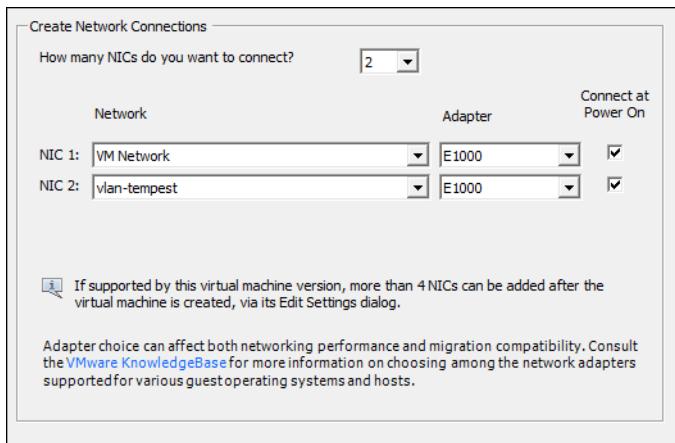
1. Download the latest CentOS image from <http://wiki.centos.org/Download>.
2. In vSphere, select **File > New > Virtual Machine**.
3. On the **Configuration** page, select the **Typical** configuration and click **Next**.



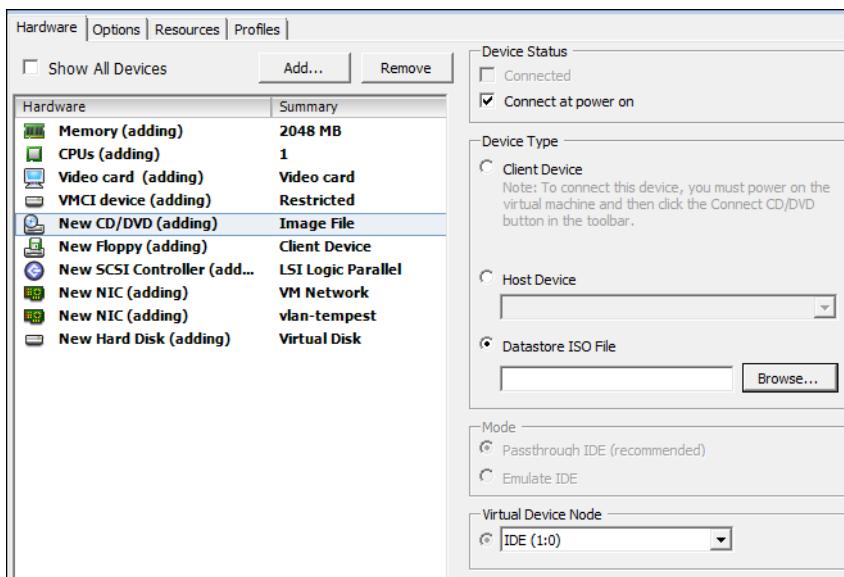
4. On the **Name and Location** page, name the virtual machine and click **Next**.
5. On the **Resource Pool** page, select a resource pool and click **Next**.
6. On the **Storage** page, select a destination storage for the VM and click **Next**.
7. On the **Guest Operating System** page, select **Linux** as the Guest Operating System and **CentOS 4/5/6 (64-bit)** as the version, then click **Next**.



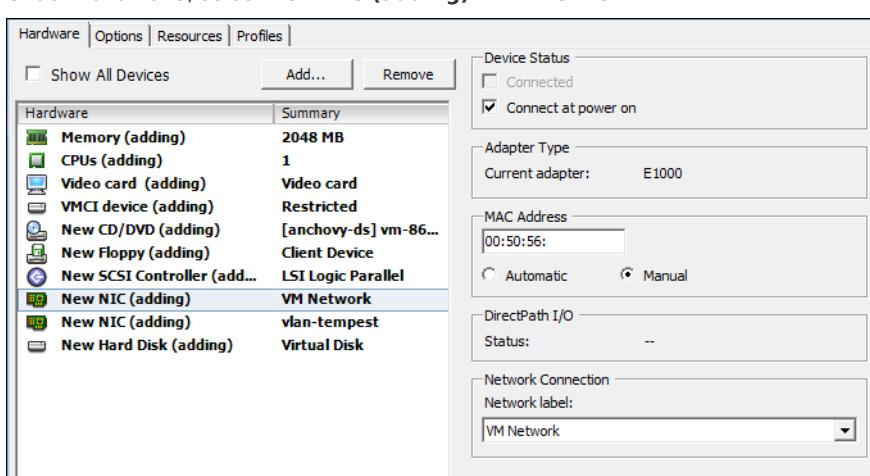
8. On the **Network** page, select **2** from the drop down menu to specify how many NICs you want to connect. Under **Network**, select **VM Network** for NIC 1 and the name of your private network for NIC 2. Then, click **Next**.



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

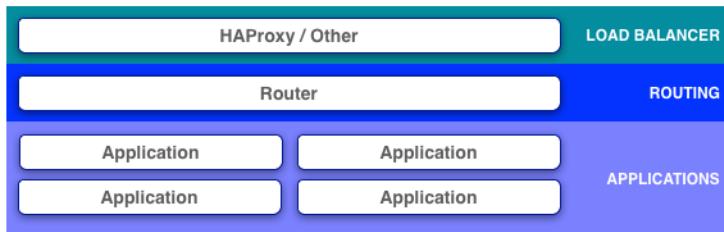
service iptables save
shutdown -r now
```

Step 3: Deploy the Ops Manager VM Template in the private network using the internal IP address

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 according to the instructions in the [Getting Started Guide](#). Access the deployed VM at port 8443 using `https://<GATEWAY_EXTERNAL_IP>:8443`.

Understanding the Elastic Runtime Network Architecture

The diagram below shows the key [Pivotal Cloud Foundry](#) Elastic Runtime network components.



Load Balancer

The load balancer is primarily used for SSL termination. If you do not want to manage your own SSL certificates or you do not have multiple certificates you want to use with Elastic Runtime, use the HAProxy component to provide SSL access to your PaaS. You also might want to omit HAProxy in favor of dedicated load balancing hardware that facilitates SSL termination.

To omit HAProxy, set the number of instances to zero in Ops Manager.

Refer to the [Configuring PCF SSL Termination](#) topic for more information.

Router

Routers manage HTTP and HTTPS traffic between web clients and application instances. In conjunction with the Cloud Controller, routers map application URLs to the corresponding application instances, distributing load between multiple instances as required.

Routers are designed to be horizontally scalable. If you have multiple routers, the load balancer distributes incoming traffic across these routers.

Refer to the Cloud Foundry [Architecture](#) topic for more information about Cloud Foundry components.

Configuring PCF SSL Termination

To use SSL termination in [Pivotal Cloud Foundry](#) (PCF), you must configure the Pivotal-deployed HAProxy load balancer or your own load balancer.

We recommend using 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.

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 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>176.16.1.1</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. Configure two fields on the HAProxy page in Elastic Runtime:

- **HAProxy IPs**: Enter the IP address for HAProxy. The HAProxy IP address must be in your subnet.
- **SSL Certificate**: If you are using a signed certificate from a well-known certificate authority, you can import it. Alternatively, you can [generate a self-signed RSA certificate](#).

Note: The domains that you use when you generate RSA public and private keys must match the wildcard A record that you create for HAProxy.

4. Configure one field on the Router IPs page in Elastic Runtime:

- **Router IPs**: Leave this field blank. HAProxy assigns the router IPs internally.

[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 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. Configure two fields on the HAProxy page in Elastic Runtime:
 - **HAProxy IPs:** Leave blank.
 - **SSL Certificate:** If you are using a signed certificate from a well-known certificate authority, you can import it. Alternatively, you can [generate a self-signed RSA certificate](#).
4. Configure one field on the Router IPs page in Elastic Runtime:
 - **Router IPs:** Enter the static IP address for PCF that you have registered with your load balancer.

 **Note:** When adding or removing PCF routers, you must update your load balancing solution configuration with the appropriate IP addresses.

[Return to the Getting Started Guide](#)

Changing the Quota Plan of an Organization with cf CLI

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`.

Note: Only an **account** administrator can run `cf curl`.

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/aaaaa1234-1111 -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, GUID *cccc0000-3333*.

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",
```

Identifying Elastic Runtime Jobs Using vCenter

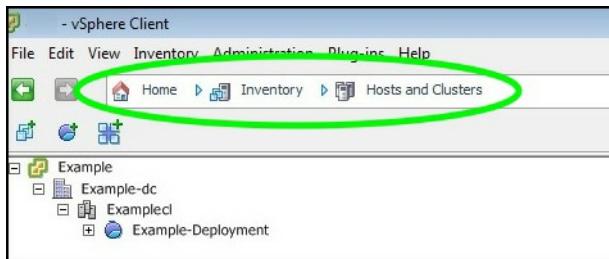
To effectively monitor, control, and manage the virtual machines making up your Elastic Runtime deployment, you may need to identify which VM corresponds to a particular job in Elastic Runtime. You can find the CID of a particular VM from [Pivotal Cloud Foundry](#)  Operations Manager by navigating to **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.

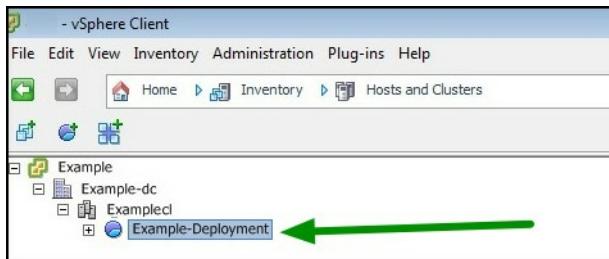
 **Note:** The CID shown in Ops Manager is the name of the machine in vCenter.

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**.

The screenshot shows the 'Example-Deployment' section of the Pivotal Ops Manager interface. On the left, a tree view lists various virtual machines and their status. A dropdown menu is open under 'Status' with several options: 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, and a search bar for 'Name, Target or Status contains:'. Below this is a table with columns: Name, Provisioned Space, Host Mem - MB, Guest, Requested Start Time, and Start Time. The table lists several VMs with their respective resource usage and start times. A green arrow points from the 'job' option in the dropdown menu to the 'job' column in the table.

6. The job column displays the Elastic Runtime job associated with each virtual machine.

The screenshot shows the 'Example-Deployment' section of the Pivotal Ops Manager interface. On the left, a tree view lists various virtual machines. A dropdown menu is open with the 'job' option selected. Below this is a table with columns: Name and job. The table lists several VMs with their associated jobs. A green arrow points from the 'job' column in the table to the 'job' option in the dropdown menu.

Understanding the Effects of Single Components on a PCF Upgrade

The [Resource Config page](#) of Pivotal Elastic Runtime 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. Though 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 on 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	✓	✓	
Console 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 Console App and App Usage Service components for the Console Database run in a single DEA instance. You cannot use the Console or the CLI during the upgrade of the DEA.
Loggregator 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 Console and Push App Usage Service errands			These errands run scripts that connect the Console App and the App Usage Service components to the Console Database. The Console App and App Usage Service components runs in a single DEA instance and the Console Database is a single component. If there is an upgrade issue with either Console Database instance or the DEA instance, the upgrade fails and Ops Manager will not run this errand.

Configuring Single Sign-On

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 [PCF Apps Manager Console](#) 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

Root Filesystem

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 This domain is used to target and push apps to Pivotal CF Elastic Runtime (ex: api.my-cf.com). You must set up a wildcard DNS record for this domain that points to the router static IP(s) or a load balancer.

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**.

- Assign Networks
- Assign Availability Zones
- Root Filesystem
- System Database Config
- File Storage Config
- IPs and Ports
- MySQL Proxy Config
- Cloud Controller
- External Endpoints
- SSO Config

Configure Identity Provider

Provider Name

Provider Metadata

(OR) Provider Metadata URL

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.com>. 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.

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:

- 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>
```

- 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. Click **Save**.
7. 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.com}/saml/metadata>. Consult the documentation for your Identity Provider for configuration instructions.

 **Note:** Some Identity Providers allow uploads of Service Provider Metadata. Other providers require you to manually enter the Service Provider Metadata into a form.

Getting Started with the Notifications Service

This guide is intended to walk you through using the Notifications Service. For more information about the Notifications Service, see the Notifications [API docs](#).

We'll show you how to set up the service through the following example: Darla is a cloud operator who needs to take the system down for maintenance and wants to notify everybody about her intentions.

Prerequisites

We're assuming that Darla has:

- Cloud Foundry set up somewhere. Darla uses the [Bosh Lite](#) installation as her development environment.
- An `admin` account on her Cloud Foundry instance
- The Notifications service installed and running. She can use the [BOSH release](#) to simplify deployment to her environment.
- The `cf` and `uaac` command line tools

Create Your Client and Get a Token

To interact with the Notifications service, Darla needs certain [UAA](#) scopes (authorities). Rather than use her `admin` user account directly, she creates 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
```

It's worth noting that she doesn't need all of these scopes just to send a notification. `notifications.manage` is used to update notifications and assign templates for that notification. `notification_templates.write` allows Darla to custom-make her own template for a notification, and `notification_templates.read` allows her to check which templates are saved in the database. Finally, `notifications.write` is the scope necessary to send a notification to a user, space, everyone in the system, and more.

Now, Darla logs in using her newly created client:

```
uaac token client get notifications-admin
```

Stay logged in with this client for the rest of the examples in this guide.

Registering Notifications

Darla cannot send a notification unless she has registered it first. Registering notifications requires the `notifications.manage` scope on her client. For example:

```
uaac curl https://notifications.darla.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" }
  }
}'
```

Darla has registered two different notifications: `system-going-down` and `system-up`. In addition, she gives the `notifications-admin` client the human-friendly description "Cloud Ops Team." Notice that she has made the `system-going-down` notification `critical`. This means that no users can unsubscribe from that notification. Setting notifications as critical requires the `critical_notifications.write` scope.

Create a Custom Template

The system provides a default template for all notifications, but Darla has decided to forgo this luxury. Darla wants to

include her own branding and has opted to create her own custom template using the curl below (this action requires the `notification_templates.write` scope):

```
uaac curl https://notifications.darla.example.com/templates -X POST --data \
'{"name":"site-maintenance","subject":"Maintenance: {{.Subject}}","text":"The site has gone down for maintenance. More information to follo
```

A template is made up of a human readable name, a subject, a text representation of the the template you are sending (for mail clients that do not support HTML), and an HTML version of the template.

Special attention and care should be paid to the variables that take this form `{{.}}`. These variables interpolate data provided in the send step below into the template before a notification is sent. Data that you can insert into a template during the send step includes `{{.Text}}`, `{{.HTML}}`, and `{{.Subject}}`.

This curl 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"}
```

Darla can check all of the saved templates by curling:

```
uaac curl https://notifications.darla.example.com/templates -X GET
```

To view a list of all templates, you must have the `notifications_templates.read` scope.

Associate a Custom Template with Your Notification

Darla now wants to associate her custom template with the `system-going-down` notification. Any notification that does not have a custom template applied, like her `system-up` notification, defaults to a system-provided template.

```
uaac curl https://notifications.darla.example.com/clients/notifications-admin/notifications/system-going-down/template \
-X PUT --data '{"template": "E3710280-954B-4147-B7E2-AF5BF62772B5"}'
```

Here, Darla has associated the `system-going-down` notification belonging to the `notifications-admin` client with the template ID `E3710280-954B-4147-B7E2-AF5BF62772B5`. This is the template id of the template we created in the previous step.

This action requires the `notifications.manage` scope.

Send Your Notification to All Users

Darla is ready to send her `system-going-down` notification to all users of the system. She performs this curl and includes some other pertinent information that gets directly inserted into the template:

```
uaac curl https://notifications.darla.example.com/everyone -X POST --data \
'{"kind_id":"system-going-down","text":"The system is going down while we upgrade our storage","html":"<h1>THE SYSTEM IS DOWN</h1><p>The sy
```

The data included in the post body above gets interpolated into the variables we previously inserted into our created template (they had the special syntax similar to `{{.Text}}`).

Sending a critical notification requires the scope `critical_notifications.write`, whereas sending a non-critical notification requires the scope of `notifications_write`.

Darla could have also chosen to send the above notification to one specific user, an email address, or a particular space. For more information on targeting your notification at particular audiences, see the [Notifications API docs](#).

Administering Cloud Foundry

This section presents a guide to managing Elastic Runtime at the administrator level.

Table of Contents

- [Adding Buildpacks to Cloud Foundry](#)
- [Creating and Managing Users with the cf CLI](#)
- [Managing Users with the UAA CLI \(UAAC\)](#)
- [Application Security Groups](#)

Adding Buildpacks to Cloud Foundry

If your application uses a language or framework that Cloud Foundry system buildpacks do not support, you can [write your own buildpack](#), customize an existing buildpack, or use a [Cloud Foundry Community Buildpack](#) or a [Heroku Third-Party Buildpack](#). You can add the new buildpack to Cloud Foundry, making it available alongside the system buildpacks.

The cf Admin-Only Buildpack Commands

 **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` do the following:

- **buildpack** specifies what you want to call the buildpack.
- **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.

You can also update and delete buildpacks. For more information, run:

```
cf update-buildpack -h
```

```
cf delete-buildpack -h
```

Confirming that a Buildpack was Added

To confirm that you have successfully added a buildpack, view the available buildpacks by running `cf buildpacks`.

The example below shows the `cf buildpacks` output 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
```

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 Buildpacks** checkbox, as shown in the image below.

Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks Coordinates Pivotal CF Elastic Runtime application lifecycles

Assign Availability Zones

IPs and Ports

MySQL Proxy Config

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Lifecycle Errands

Resource Config

Stemcell

System Domain *

Apps Domain *

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

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.

Creating and Managing Users with the cf CLI

This page assumes you are using cf CLI v6.

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	cf create-user Alice pa55w0rd
Delete a user	cf delete-user USERNAME	cf delete-user Alice

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	cf orgs
View all users in an organization by role	cf org-users ORGANIZATION_NAME	cf org-users my-example-org
Assign an org role to a user	cf set-org-role USERNAME ORGANIZATION_NAME ROLE	cf set-org-role Alice my-example-org OrgManager
Remove an org role from a user	cf unset-org-role USERNAME ORGANIZATION_NAME ROLE	cf unset-org-role Alice myexample-org OrgManager

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)

Using the UAA Command Line Interface (UAAC), an administrator can create users and manage organization and space roles.

For additional details and information, refer to the following topics:

- [UAA Overview](#)
- [UAA Sysadmin Guide](#)
- [Other UAA Documentation](#)

Creating an Admin User

1. Install the UAA CLI, `uaac`.

```
$ gem install cf-uaac
```

2. Use `uaac target uaa.YOUR-DOMAIN` to target your UAA server.

```
$ uaac target uaa.example.com
```

3. Record the **uaa:admin:client_secret** from your deployment manifest.

4. Use `uaac token client get admin -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
```

5. Use `uaac contexts` to display the users and applications authorized by the UAA server, and the permissions granted to each user and application.

```
$ uaac contexts

[1]*[admin]
  client_id: admin
  access_token: yJhbGciOiJIUzI1NiJ9.e
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin clients.secret scim.read
  jti: 91b3-abcd1233
```

6. In the output from `uaac contexts`, search in the `scope` section of the `client_id: admin` user for **scim.write**. The value **scim.write** represents sufficient permissions to create accounts.

7. If the admin user lacks permissions to create accounts:

- Use `uaac client update admin --authorities "EXISTING-PERMISSIONS scim.write"` to add the necessary permissions to the admin user account on the UAA server. Replace EXISTING-PERMISSIONS with the current contents of the `scope` section from `uaac contexts`.
- Use `uaac token delete` to delete the local token.
- Use `uaac token client get admin` to obtain an updated access token from the UAA server.

```
$ 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
```

Granting 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. Use `uaac group map cloud_controller.admin GROUP-DISTINGUISHED-NAME` to grant all user under the mapped LDAP Group admin permissions.

Creating 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
```

Changing 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: yJhbGciOiJIUzI1NiJ9.e
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin clients.secret password.read
  jti: 91b3-abcd1233
```

4. In the output from `uaac contexts`, search in the `scope` section of the `client_id: admin` user for **password.write**. The value **password.write** represents sufficient permissions to change passwords.

5. If the admin user lacks permissions to change passwords:

- Use `uaac client update admin --authorities "EXISTING-PERMISSIONS password.write"` to add the necessary permissions to the admin user account on the UAA server. Replace EXISTING-PERMISSIONS with the current contents of the `scope` section from `uaac contexts`.
- Use `uaac token delete` to delete the local token.
- Use `uaac token client get admin` to obtain an updated access token from the UAA server.

```
$ uaac contexts

[1]*[admin]
  client_id: admin
  ...
  scope: uaa.admin.clients.secret password.read
  ...

$ uaac client update admin --authorities "`uaac client get admin | \
  awk '/:/{e=0}/authorities:/{{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...
```

Retrieving 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: yJhbGciOiJIUzI1NiJ9.e
  token_type: bearer
  expires_in: 43200
  scope: uaa.admin clients.secret
  jti: 91b3-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:
 - Use `uaac client update admin --authorities "EXISTING-PERMISSIONS scim.write"` to add the necessary permissions to the admin user account on the UAA server. Replace EXISTING-PERMISSIONS with the current contents of the `scope` section from `uaac contexts`.
 - Use `uaac token delete` to delete the local token.
 - Use `uaac token client get admin` to obtain an updated access token from the UAA server.

```
$ uaac contexts

[1]*[admin]
  client_id: admin
  ...
  scope: uaa.admin clients.secret
  ...

$ uaac client update admin --authorities "uaa.admin clients.secret scim.read"

$ uaac token delete
$ uaac token client get admin
```

7. Use `uaac users` to list your Cloud Foundry instance users. By default, the `uaac users` command returns information about each user account including GUID, name, permission groups, activity status, and metadata. Use the `--attributes emails` or `-a emails` flag to limit the output of `uaac users` to email addresses.

```
$ uaac users --attributes emails

resources:
  emails:
    value: user1@example.com
    emails:
      value: user2@example.com
      emails:
        value: user3@example.com
```

8. Use `uaac users "id eq GUID" --attributes emails` with the GUID of a specific user to retrieve that user's email address.

```
$ uaac users "id eq 'aabbcc11-22a5-87-8056-beaf84'" --attributes emails

resources:
  emails:
    value: user1@example.com
```

Application Security Groups

This page assumes you are using cf CLI v6.4 or higher.

Application security groups act as virtual firewalls to control outbound traffic from the applications in your deployment. A security group consists of a list of network egress access rules.

An administrator can assign one or more security groups to a Cloud Foundry deployment or to a specific [space](#) in an [org](#) within a deployment.

Note: A security group assigned to a Cloud Foundry deployment affects all spaces in all orgs within the deployment.

Within a space, Cloud Foundry runs each instance of an application inside a separate application container. When you launch an application for the first time, Cloud Foundry creates a new container for each application instance, then applies any space-specific and deployment-wide security groups to the container. Cloud Foundry determines whether to allow or deny outbound traffic from the container by evaluating the rules defined in these security groups.

Creating Security Groups

A security group consists of a list of operator-defined network egress [allow](#) rules. These rules define the outgoing traffic allowed to application containers. Each rule contains the following three parts:

- **Protocol:** TCP, UDP, or ICMP
- **Open Port / Port Range:**
 - For TCP and UDP: Either a single port or a range of ports
 - For ICMP: An ICMP type and code
- **Destination:** Destination of the traffic allowed by this rule as an IP address or CIDR block

Run `cf create-security-group SECURITY-GROUP PATH-TO-RULES-FILE` from a command line to create a security group named `SECURITY-GROUP`. `PATH-TO-RULES-FILE` can be an absolute or relative path to a rules file. The rules file must be a JSON-formatted single array containing objects that describe the rules.

Example JSON-formatted rules file:

```
[{"protocol": "tcp", "destination": "10.0.11.0/24", "ports": "1-65535"}, {"protocol": "udp", "destination": "10.0.11.0/24", "ports": "1-65535"}]
```

Binding Security Groups

A security group consists of a list of rules. You must bind this list to either your entire deployment or to a space in an org for Cloud Foundry to apply the rules to outgoing traffic.

To apply the rules in a security group to your entire Cloud Foundry deployment, bind the security group to either the **Default Staging** or the **Default Running** security group set. Cloud Foundry applies the rules in every security group in the Default Staging and Default Running sets to all applications in your CF deployment.

Note: If you do not bind a security group to either the Default Staging or the Default Running set, Cloud Foundry only applies the rules to application running in the specific space where you bound the security group.

Binding to your CF Deployment

To create a rule to be applied to every space in every org of your Cloud Foundry deployment, bind the security group to either the **Default Staging** or the **Default Running** security group set.

Cloud Foundry applies the rules in every security group in these sets as follows:

- **Default Staging:** Cloud Foundry applies rules in this set to every application staged anywhere in your CF

deployment. To bind a security group to the Default Staging set, run `cf bind-staging-security-group SECURITY-GROUP`.

- **Default Running:** Cloud Foundry applies rules in this set to every application running anywhere in your CF deployment. To bind a security group to the Default Running set, run `cf bind-running-security-group SECURITY-GROUP`.

Binding to a Space

Run `cf bind-security-group SECURITY-GROUP ORG SPACE` to bind a security group to a specific space. Cloud Foundry applies the rules that this security group defines to all application containers in the space. A space may belong to more than one security group.

Network Traffic Rules Evaluation Sequence

Cloud Foundry evaluates security groups and other network traffic rules in a strict priority order. Cloud Foundry returns an `allow`, `deny`, or `reject` result for the first rule that matches the outbound traffic request parameters, and does not evaluate any lower-priority rules.

Cloud Foundry evaluates the network traffic rules for an application in the following order:

1. **Security Groups:** The rules described by the 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.

Viewing Security Groups

Run the following commands to view information about existing security groups:

- `cf security-groups`: Displays all security groups in an org.
- `cf staging-security-groups`: Displays all security groups in the Default Staging set.
- `cf running-security-groups`: Displays all security groups in the Default Running set.
- `cf security-group SECURITY-GROUP`: Displays details about the specified security group.

Using the Apps Manager

This section provides help with using the web-based console application for managing users, organizations, spaces, and applications.

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](#)

Getting Started with the Apps Manager

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 push` to update the application.

Understanding Apps Manager Permissions

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 the Admin specifies for the [console 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.

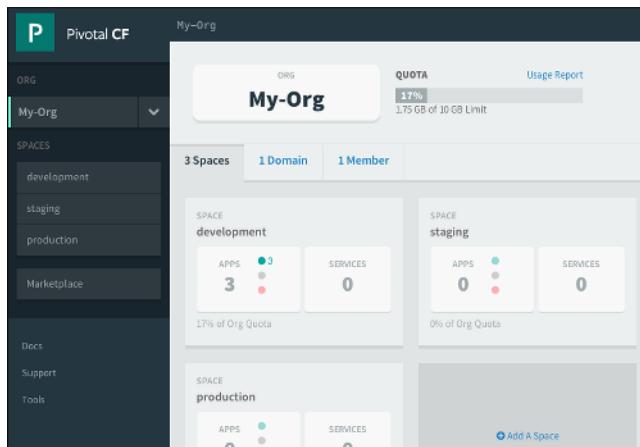
	CLI	Apps Manager		
Command	Admin	Admin or Org Manager	Space Manager	Org Auditor or Space Developer or Space Auditor
create-org	X	X	X	X
delete-org	X	X		
rename-org	X	X		
org-users	X	X	X	X
set-org-role	X	X		
unset-org-role	X	X		
space-users	X	X	X	X
set-space-role	X	X	X	
unset-space-role	X	X	X	

Managing Spaces Using the Apps Manager

To manage a space in an org, you must have org manager or space manager permissions in that space.

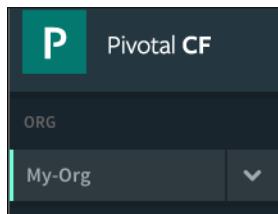
Org managers have space manager permissions by default and can manage user accounts in all spaces associated with the org.

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



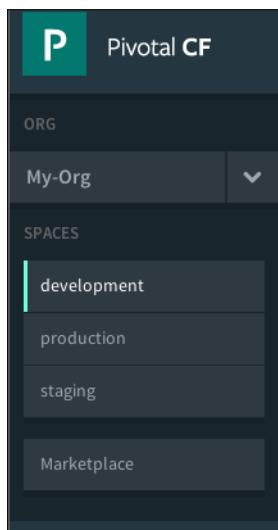
Orgs

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.



Spaces

The Apps Manager navigation also shows the spaces in the current org. The current space is highlighted.



- **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	
	hello-world hello-world.ch...	1	128MB	
	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

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

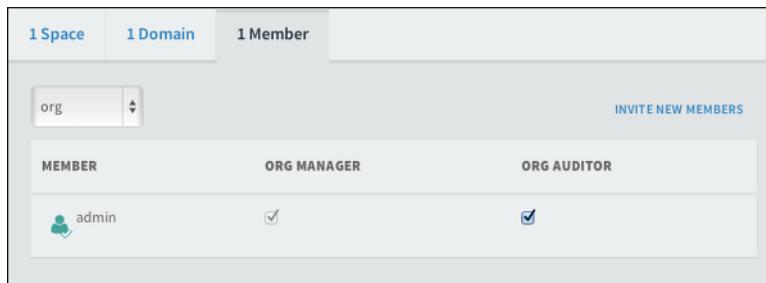
Space Developers can create, delete, and manage applications and services, and have full access to all usage reports and logs. Space Developers can also edit applications, including the number of instances and memory footprint. Assign this role to app developers or other users who need to interact with 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 Org dashboard with the 'Members' tab selected. 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 table has three columns: 'MEMBER', 'ORG MANAGER', and 'ORG AUDITOR'. The 'admin' row has a checked checkbox in the 'ORG MANAGER' column and another checked checkbox in the 'ORG AUDITOR' column.

2. Click **Invite New Members**. The **Invite New Team Member(s)** form appears.

4 Spaces | 1 Domain | 2 Members | [Delete Org](#)

INVITE NEW TEAM MEMBER(S)

 Add Email Address(es)

ASSIGN TEAM ROLES

ORG	ORG MANAGER	ORG AUDITOR	ORGANIZATION ROLES
org	<input type="checkbox"/>	<input type="checkbox"/>	ORGANIZATION MANAGER Can invite/manage users, select/change the plan, establish spending limits
<input type="checkbox"/> Select All			ORGANIZATION AUDITOR Read-only access to org info and reports

APP SPACES	SPACE MANAGER	SPACE DEVELOPER	SPACE AUDITOR	APP SPACE ROLES
development	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SPACE MANAGER Can invite/manage users, enable features for a given space
production	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SPACE DEVELOPER Can create, delete, manage applications and services, full access to all usage reports and logs
staging	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SPACE AUDITOR Read-only access to all space information, settings, reports, logs
testing	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/> Select All				

[CANCEL](#) [SEND INVITE](#)

3. In the **Add Email Address(es)** text field, enter the email addresses of the users 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 you want to grant the invited users.
5. Click **Send Invite**. The Apps Manager sends an email containing an invitation link to each email address 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 each space you want to change.

1. On the Org dashboard, click the **Members** tab.

1 Space | 1 Domain | 1 Member

[INVITE NEW MEMBERS](#)

MEMBER	ORG MANAGER	ORG AUDITOR
 admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

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

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"/>

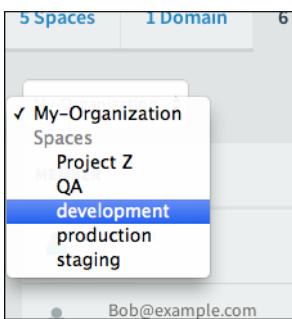
4. The **Members** panel displays all members of the org. Select a checkbox to grant an org role to a user, or deselect a checkbox to revoke a role from a user.

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

Controlling Apps Manager User Activity with Environment Variables

This topic describes two environment variables you can use to manage users' interactions with the Apps Manager, and how to set these variables using the cf CLI.

Understanding the Apps Manager Environment Variables

You can control which users can create orgs and perform user management actions on the Apps Manager using the following environment variables.

ENABLE_NON_ADMIN_ORG_CREATION

If set to `true`, a user of any role can create an org. On the Org Dashboard, the links to create a new org appear as follows:

- The **Create a New Org** link in the drop-down menu in the left navigation panel
- The **Create Org** link on the right side of the My Account page, which you access from the user name drop-down menu

The image shows the link locations.



If set to `false`, only an Admin can create an org. This is the default value.

ENABLE_NON_ADMIN_USER_MANAGEMENT

If set to `true`, Org Managers and Space Managers can invite users and manage roles, and a user of any role can leave an org, provided at least one Org Manager exists in the org.

If set to `false`, only an Admin can invite users and manage roles. This is the default value. Users cannot remove themselves from an org, and on the Org Dashboard, the Members tab appears as read-only for all users except Admins.

Changing an Environment Variable Value

Note: To run the commands discussed in this section, you must be an administrator for your org and log into 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. From a terminal window, run `cf api URL` and target your Apps Manager URL. For example:
`cf api api.YOUR-SYSTEM-DOMAIN`.
2. Run `cf login` and provide your UAA Administrator user credentials.
3. Select the `system` org and the `console` space.
4. Run `cf set-env APP NAME VALUE`. For example: `cf set-env console ENABLE_NON_ADMIN_ORG_CREATION true`
5. Run `cf restart console` to reinitialize the Apps Manager with the new environment variable value.

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

[Deploy an Application](#)

[Deploy a Large Application](#)

[Creating Domains and Routes](#)

[Domains and Shared Domains](#)

[Deploying with Application Manifests](#)

[Cloud Foundry Environment Variables](#)

[Using Blue-Green Deployment to Reduce Downtime and Risk](#)

[Application Logging in Cloud Foundry](#)

[Troubleshooting Application Deployment and Health](#)

cf Command Line Interface

[Installing the cf Command Line Interface](#)

[Getting Started with the cf CLI](#)

[Using the cf CLI with an HTTP Proxy Server](#)

[Using cf CLI Plugins](#)

[About Starting Applications](#)

[Scaling an Application Using cf scale](#)

Services

[Services Overview](#)

[Adding a Service](#)

[Binding a Service Instance](#)

[Managing Service Instances with the CLI](#)

[Third-Party Managed Service Instances](#)

[User-Provided Service Instances](#)

[Configuring Play Framework Service Connections](#)

[Migrating a Database in Cloud Foundry](#)

[Using Third Party Log Management Services](#)

[Configuring Selected Third-Party Log Management Services](#)

[Integrating Cloud Foundry with Splunk](#)

Considerations for Designing and Running an Application in the Cloud

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. Rather we recommend using a shared data service like a database or blobstore for this purpose.

For example, rather than using the local file system, you can use a Cloud Foundry service such as the MongoDB document database or a relational database (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.

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.

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.

Some production load balancers require separate ports for HTTP and TCP/WebSocket traffic. Applications deployed to Elastic Runtime use port 443 by default for TCP/WebSocket communications. As of PCF 1.4, you can configure a different port from the Elastic Runtime product tile in Ops Manager. This configuration enables you to access logs from your app with the `cf logs APP_NAME` command.

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.

Deploy an Application

This page assumes 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 you run `push` and the time that the application is available, Cloud Foundry:

- 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.

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](#)

 **Note:** Using a Content Delivery Network (CDN) such as [Akamai](#) or [Amazon Cloudfront](#) for static assets makes web pages load faster, partly by enabling browsers to make multiple requests in parallel. For best performance on a large scale, either use a CDN or compensate by scaling your app up to a higher number of instances.

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.
- 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.

For more information about domains, see [Creating Domains and Routes](#).

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` is running. Stopping all instances is necessary to prevent two versions of your code 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.

Step 5: Push the Application

The minimal command for deploying an application without a manifest is:

```
cf push <appname>
```

If you provide the application name in a manifest, you can reduce the command even further, 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, generated when `joeclouduser` deployed the “my-app” Sinatra application, shows how Cloud Foundry assigns default values to application when given a minimal push command.

 **Note:** When deploying your own apps, avoid generic names like `my-app`, because a route containing a generic host like `my-app` is unlikely to be unique. Deployment fails unless the application has a globally unique route.

```
$ cf push my-app
Creating app my-app in org joeclouduser-org / space development as joeclouduser@example.com...
OK

Creating route my-app.example.com...
OK

Binding my-app.example.com to my-app...
OK

Uploading my-app...
Uploading app: 560.1K, 9 files
OK

Starting app my-app in org joeclouduser-org / space development as joeclouduser@example.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 joeclouduser-org / space development as joeclouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: my-app.example.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 you deployed, it may be necessary 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 check 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

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. Pivotal recommends 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 on 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 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 cf has successfully uploaded and packaged 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`: Controls the maximum time that the cf CLI waits for an app to start. When you use this flag, cf 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 on using the cf CLI to deploy apps, refer to the [Push section](#) of the Getting Started with the cf CLI topic.

 **Note:** Your app might successfully start even though the CLI reports `App failed` because of the differences between the phase timeout values you specify in the CLI or manifest and those set by your administrator. For example, your admin sets a start value of 180 seconds. You set a start value of 120 seconds in the manifest, but override this on the CLI with a start value of 60 seconds and deploy the app. In this scenario, your app might successfully start even though the CLI reports that the app fails because of the longer start timeout value specified by the administrator. 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_STAGING_TIMEOUT	CLI environment variable Default: 15 minutes
CF_STARTUP_TIMEOUT	CLI environment variable Default: 5 minutes
cf push -t TIMEOUT	App start timeout maximum Default: 60 seconds
Disk Space Allocation	Default: 1024 MB
Internet Connection Speed	Recommended Minimum: 874 KB/s

Creating Domains and Routes

This page assumes you are using cf CLI v6.

This topic describes how to create and use domains and routes in Cloud Foundry.

Note: The term **domain** in this topic differs from its common use and is specific to Cloud Foundry.

In Cloud Foundry, you create a domain according to the guidelines in this topic. Cloud Foundry uses these domains in routes to direct requests to applications. A **custom domain** refers to a third party registered domain that you own and can use with your applications in Cloud Foundry. For more information on using custom domains shared across all orgs in your Cloud Foundry account, refer to the [Domains and Shared Domains](#) topic.

Domains are associated with orgs and are not directly bound to applications. Domains can be **shared** or **private**. Shared domains are registered to multiple orgs while private domains, or **owned** domains, are registered to one org. A Cloud Foundry instance defines a default shared domain that your application uses unless you specify a different domain.

A **route** is a path used to access an application online. Each route is directly bound to one or more applications in Cloud Foundry. A route is a URL composed of a domain and an optional host as a prefix. For example, “myappname” is the host and “example.com” is the domain in the route “myappname.example.com.” Here “example.com” could either be a custom domain of your own, or a domain that you have created through Cloud Foundry.

Using a Root Domain

A **root domain** or a **zone apex domain** such as “example.com” consists of a domain without a host. To access an application deployed to a root domain, you must use either custom record types or **subdomain redirection**.

The following section contains instructions for mapping a root domain to your application by configuring ALIAS or ANAME custom record types or by using subdomain redirection.

Configuring an ALIAS or ANAME Record with Your DNS Provider

If your DNS provider supports using an ALIAS or ANAME record, configure your root domain with your DNS provider.

ALIAS and ANAME records follow the pattern shown below:

Record	Name	Target	Note
ALIAS or ANAME	empty or @	foo.example.com.	Refer to your DNS provider documentation to determine whether to use an empty or @ value for the Name entry.

If your DNS provider does not support ANAME or ALIAS records, use subdomain redirection, a technique also known as **domain forwarding**, to add a root domain. If you use this method, note that SSL requests to the root domain result in an error because the SSL certificate generally only matches the subdomain.

Configure the redirect from the root domain to the target “www” subdomain, and configure the “www” subdomain as a CNAME record reference to the target app URL. The table below illustrates this pattern.

Record	Name	Target	Note
URL or Forward	example.org	www.example.org	This method results in a 301 permanent redirect to the subdomain you configure.
CNAME	www	foo.cfapps.io	

Configuring Your Application to Use a Root Domain

The following section contains instructions for creating a domain and using that domain as the route for your application. Since the route is a root domain and does not have a host, both the domain and route in this section are represented with “example.org.”

1. Create the domain in Cloud Foundry and associate it with an organization.
The command below defines the domain “example.org” in the “test-org” organization:

```
$ cf create-domain test-org example.org
```

2. Map the route to your app. The command you use depends on whether the app uses a shared domain.

If your route is not using a shared domain, use the `cf map-route` command as the example shows:

```
$ cf map-route myapp example.org
```

If your route is using a shared domain, use the `-n HOSTNAME` parameter with the `cf map-route` command to specify a unique hostname for the route. The command below maps the route “myapp.example.org” with the hostname “app” to the application named “myapp”:

```
$ cf map-route myapp example.org -n app
```

Creating a Parent Domain

In the multi-level domain “myapp.example.com,” “example.com” is the parent domain of subdomain “myapp.”

Configuring a CNAME Record with Your DNS Provider

Follow the [Configuring a CNAME Record with Your DNS Provider](#) procedure in the Using a Private or Shared Subdomain section.

Configuring Your Application to Use a Private or Shared Parent Domain

Parent domains must meet the following requirements:

- **Private Parent Domains:** You can only create a private domain that is parent to a private subdomain.
- **Shared Parent Domains:** You can create a shared domain that is parent to either a shared or a private subdomain.

Using a Private or Shared Subdomain

Domains in Cloud Foundry can be multi-level and contain subdomains like the “myapp” in the domain “myapp.example.com.” In this case, “example.com” is the parent domain of the subdomain “myapp.” In the domain “test.myapp.example.org,” “test” is a subdomain with the parent domain “myapp.example.org.”

Configuring a CNAME Record with Your DNS Provider

Configure your domain using a CNAME record.

CNAME records follow the pattern illustrated below:

Record	Name	Target	Note
CNAME	test	foo.example.com.	Refer to your DNS provider documentation to determine whether the trailing <code>.</code> is required.

Configuring Your Application to Use a Private or Shared Subdomain

You can create a new domain by mapping subdomains to an existing domain. Each added subdomain is the child to its following parent domain. The subdomains must meet the following requirements:

- **Private Subdomains:**

- You can map a private subdomain to a private parent domain only if the domains belong to the same org.
- You can map a private subdomain to a shared parent domain.

- **Shared Subdomains:**

- You can only map a shared subdomain to a shared parent domain.
- You cannot map a shared subdomain to a private parent domain.

To configure your application to use a domain with a subdomain, create the domain and map the route to your

application as follows:

1. Create the domain in Cloud Foundry and associate it with an org.

The command below defines the domain “myapp.example.org” in the “test-org” organization:

```
$ cf create-domain test-org myapp.example.org
```

2. Map the route to your app with an optional hostname, as the example shows:

```
$ cf map-route myapp myapp.example.org -n test
```

Managing Your Domains

View Domains for an Org

You can view available domains for the targeted organization using the `cf domains` command.

In this example, there are two available domains: a system-wide default “example.com” domain and the custom “example.org” domain.

```
$ cf domains
Getting domains in org console as user@example.org... OK

name      status
example.com  shared
example.org   owned
```

Assign Domains and Hosts

You can assign a domain and host to your application using either the command line or the application manifest.

Assign Using the cf CLI

When you run `cf push`, you can optionally assign a domain and host to the application.

- **Domain:** Use the `-d` flag to specify one of the domains available to the targeted org.
- **Host:** Use the `-n` flag to provide a string for the host.

The route Cloud Foundry creates for the application as a result of the following command is `myapp.example.org`.

```
$ cf push myapp -d example.org -n myapp
```

Assign Using the Manifest

When you create or edit the manifest for an application, you can use the `host` and `domain` attributes to define the host and domain components of the application route, respectively. For more information, see [Application Manifests](#).

Delete a Domain

You can delete a domain from Cloud Foundry with the `cf delete-domain` command:

```
$ cf delete-domain example.org
```

Managing Your Routes

List Routes

You can list routes for the current space with `cf routes` command. Note that the host is separate from the domain

segment.

For example:

```
$ cf routes
Getting routes as user@example.org ...

host           domain     apps
myapp          example.org myapp1
myapp2
1test
sinatra-hello-world  example.com 1test
sinatra-to-do    example.com sinatra-hello-world
sinatra-to-do    example.com sinatra-to-do
```

Create a Route

Create a route and associate it with a space for later use with the `cf create-route` command. You can use the optional `-n HOSTNAME` parameter to specify a unique hostname for each route that uses the same domain.

For example, this command creates the “myapp.example.org” route in the “development” space:

```
$ cf create-route development example.org -n myapp
```

If you are an administrator, you can create a **wildcard route** by specifying a host as `*`. The star operator `*` signals a wildcard route to match any URL that uses your domain, regardless of the host.

Note: You must surround the `*` with quotation marks when referencing it using the CLI.

For example, the following command creates the wildcard route “*.example.org” in the “development” space:

```
$ cf create-route development example.org -n "*"
```

Assign or Change a Route

Assign or change the route for a particular application with the `cf map-route` command. Specifying the host is optional. If the route does not already exist, this command creates it and then maps it.

For example, the following command maps the route “myapp.example.org” to the “myapp” application:

```
$ cf map-route myapp example.org -n myapp
```

An application bound to a wildcard route is a “fallback” app that a user accesses if no other application routes match what the user typed. For example, if a user types “myap.example.org” in an attempt to access the application bound to “myapp.example.org”, the user accesses the application bound to “*.example.org.”

The command below maps the route “*.example.org” to the “myfallbackapp” application:

```
$ cf map-route myfallbackapp example.org -n *
```

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.

If the application is running when you map a route, restart the application. The new route is not active until the application is restarted.

Remove a Route

You can remove a route from an app using the `cf unmap-route` command. Unmapping a route leaves the route available in the targeted space for later use.

```
$ cf unmap-route myapp example.org -n myapp
```

You can remove a route from a space using the `cf delete-route` command:

```
$ cf delete-route example.org -n myapp
```

Note that for each of the above commands, using the `-n` parameter to specify the host is optional.

Domains and Shared Domains

This page assumes you are using cf CLI v6.

This page has information about domains and shared domains. For more information, see [Creating Domains and Routes](#).

A domain is an IP resource represented by a domain name like “example.com”. A domain is not directly bound to an application, but acts as part of the route to an application.

When you deploy an application, you supply a hostname. This hostname is prepended to the domain to create the full URL, or route, to the application.

Example: If I use the hostname “demo-time” for my application, and my domain is “example.com”, the route to the application is the full URL “demo-time.example.com”.

A Cloud Foundry instance defines a default shared domain. Unless you specify a different domain, routes to your application are created using this shared domain.

Cloud Foundry also supports custom domains.

Create a Custom Domain

If you want to use a registered domain of your own, you can:

- Create and associate it with a single organization in your account.
- Create and associate it with *all* organizations in your account. Note: This can only be done by an **account** administrator.

Associate a Custom Domain with a Single Organization

Use the `create-domain` command to create the custom domain `example.com` in the “test” organization:

```
$ cf create-domain test example.com
```

Associate a Custom Domain with All Organizations in an Account

Use the `create-shared-domain` command to create a shared custom domain available to all organizations in your account:

```
$ cf create-shared-domain example.com
```

View Domains

You can see available domains for the targeted organization using the `cf domains` command. In this example, there are two available domains: a system-wide default `example.com` domain and the custom `example.org` domain.

```
$ cf domains
Getting domains in org console as a.user@example.com... OK
  name      status
example.com  shared
example.org   owned
```

Delete a Domain

Delete a Custom Domain

Use the `cf delete-domain` command to delete a domain:

```
$ cf delete-domain example.com
```

Delete a Custom Shared Domain

Deleting a shared domain can only be done by an **account** administrator.

Caution: Deleting a shared domain will remove all associated routes, and will make any application with this domain unreachable.

To delete a shared domain:

```
$ cf delete-shared-domain example.com
```

Deploying with Application Manifests

This page assumes 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. Use the `-f` option to provide a non-standard manifest location or filename.

To use `cf push` without an `-f` option, you must provide a manifest named `manifest.yml` in the current working directory.

```
$ cf push
Using manifest file /path_to_working_directory/manifest.yml
```

With the `-f` option, a path with no filename means that 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
```

If the manifest is named something other than `manifest.yml`, use the `-f` option with both path *and* filename.

```
$ cf push -f ./some_directory/some_other_directory/alternative_manifest.yml
Using manifest file /path_to_working_directory/some_directory/some_other_directory/alternative_manifest.yml
```

An 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 begins 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.

Place your `manifest.yml` in the directory that contains the application files you want to push and run `cf push`. `cf push` searches the current directory for a manifest unless you specify another path with the `-p` option.

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 simple command `cf push` works when the `manifest.yml` file is in the current working directory.

If you do *not* use a manifest, the minimal push command looks like this:

```
cf push my-application-name
```

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; `cf` does not push the others. These behaviors are useful 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-application-name` 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.

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 instances attribute

Use the `instances` attribute to specify the number of app instances 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 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

Use the `timeout` attribute to give your application more time to start, up to 180 seconds. The default timeout is 60 seconds. For example:

```
...  
...  
timeout: 80
```

If the app has not started by the timeout limit you set, Cloud Foundry stops trying to start the app. You might want to increase the timeout length to ensure that very large apps have sufficient time to start.

The command line option that overrides this 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 [Bind a Service](#).

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 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 `cf` from fireplace, 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: example.com
path: ./spark/
services:
- mysql-flint-99
- name: flame
  memory: 1G
  instances: 2
  host: burnin-77
  domain: example.com
  path: ./flame/
  services:
- redis-burnin-77
```

Follow these general rules when using a multiple-application manifest:

- Name and completely describe your applications in the manifest.
- Use a `no-route` line in the description of any application that provides background services to another application.
- Do not provide an application name with `cf push`.
- Do not use any command line options with `cf push`.

There are only two narrow exceptions:

- If your manifest is not named `manifest.yml` or not in the current working directory, use the `-f` command line option.
- If you want to push a single application rather than all of the applications described in the manifest, provide the desired application name by running `cf push my-application_name`.

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: example.com
memory: 1G
instances: 1
services:
- clockwork-mysql
applications:
- name: springtock
  host: tock09876
  path: ./spring-music/build/libs/spring-music.war
- name: springtick
  host: tick09875
  path: ./spring-music/build/libs/spring-music.war
```

In the next section we carry this principle further by distributing content across multiple manifests.

Multiple Manifests with Inheritance

A single manifest can describe multiple applications. Another powerful technique is to create multiple manifests with inheritance. Here, manifests have parent-child relationships such that children inherit descriptions from a parent. Children can use inherited descriptions as-is, extend them, or override them.

Content in the child manifest overrides content in the parent manifest, if the two conflict.

This technique helps in these and other scenarios:

- An application has a set of different deployment modes, such as debug, local, and public. Each deployment mode is described in child manifests that extend the settings in a base parent manifest.
- An application is packaged with a basic configuration described by a parent manifest. Users can extend the basic configuration by creating child manifests that add new properties or override those in the parent manifest.

The benefits of multiple manifests with inheritance are similar to those of minimizing duplicated content within single manifests. With inheritance, though, we “promote” content by placing it in the parent manifest.

Every child manifest must contain an “inherit” line that points to the parent manifest. Place the inherit line immediately after the three dashes at the top of the child manifest. For example, every child of a parent manifest called `base-manifest.yml` begins like this:

```
---  
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: example.com  
memory: 256M  
instances: 1  
services:  
- singular-backend  
  
# app-specific configuration  
applications:  
- name: springtck  
host: 765shower  
path: ./april/build/libs/april-weather.war  
- name: wintertick  
host: 321flurry  
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: 321blizzard  
path: ./december/target/december-weather.war
```

Note: Inheritance can add an additional level of complexity to manifest creation and maintenance. Comments that precisely explain how the child manifest extends or overrides the descriptions in the parent manifest can alleviate this complexity.

Cloud Foundry Environment Variables

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 Droplet Execution Agents (DEAs) 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_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 my-space as admin...
OK

System-Provided:

{
  "VCAP_APPLICATION": {
    "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
```

Variables Available to Your Application

The subsections that follow describe the environment variables that Cloud Foundry makes available for your application container.

You can access environment variables programmatically, including variables defined by the buildpack. Refer to the buildpack documentation for [Java](#), [Node.js](#), and [Ruby](#).

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 the DEA for communication with the application. The DEA allocates a port to the application during staging. For this reason, code that obtains or uses the application port should reference it using `PORT`.

```
PORT=61857
```

PWD

Identifies the present working directory, where the buildpack that processed the application ran.

```
PWD=/home/vcap
```

TMPDIR

Directory location where temporary and staging files are stored.

```
TMPDIR=/home/vcap/tmp
```

USER

The user account under which the DEA runs.

```
USER=vcap
```

VCAP_APP_HOST

The IP address of the DEA host.

```
VCAP_APP_HOST=0.0.0.0
```

VCAP_APPLICATION

This variable contains useful information about a deployed application. Results are returned in JSON format. The table below lists the attributes that are returned.

Attribute	Description
<code>application_users</code> , <code>users</code>	
<code>instance_id</code>	GUID that identifies the application.
<code>instance_index</code>	Index number of the instance. You can access this value directly with the <code>CF_INSTANCE_INDEX</code> variable.
<code>application_version</code> , <code>version</code>	GUID that identifies a version of the application that was pushed. Each time an application is pushed, this value is updated.
<code>application_name</code> , <code>name</code>	The name assigned to the application when it was pushed.
<code>application_uris</code>	The URI(s) assigned to the application.
<code>started_at</code> , <code>start</code>	The last time the application was started.
<code>started_at_timestamp</code>	Timestamp for the last time the application was started.

host	IP address of the application instance.
port	Port of the application instance. You can access this value directly with the PORT variable.
limits	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.
state_timestamp	The timestamp for the time at which the application achieved its current state.

For example:

```
VCAP_APPLICATION={"instance_id":"451f045fd16427bb99c895a2649b7b2a",
"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":"c1063c1c-40b9-434e-a797-db240b587d32","application_name"
:"styx-james","application_uris":["styx-james.al.app.cf-app.com"],"version":"c10
63c1c-40b9-434e-a797-db240b587d32","name":"styx-james","uris":["styx-james.al.ap
p.cf-app.com"],"users":null}
```

VCAP_APP_PORT

Equivalent to the [PORT](#) variable, defined above.

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
name	The name assigned to the service instance by the user when it was created.
label	v1 broker API The service name and service version (if there is no version attribute, the string “n/a” is used), separated by a dash character, for example “cleardb-n/a”. v2 broker API The service name.
tags	An array of strings an app can use to identify a service.
plan	The service plan selected when the service was created.
credentials	A JSON object containing the service-specific set of credentials needed to access the service instance. For example, for the cleardb service, this will include name, hostname, port, username, password, uri, and jdbcUrl.

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 in the [v1 format](#) for bound instances of several services available in the [Pivotal Web Services Marketplace](#).

```
VCAP_SERVICES=
{
  "elephantsql-n/a": [
    {
      "name": "elephantsql-c6c60",
      "label": "elephantsql-n/a",
      "tags": [
        "postgres",
        "postgresql",
        "relational"
      ],
      "plan": "turtle",
      "credentials": {
        "uri": "postgres://seilbmbd:PHxTPJSbkcDakfK4cYwXHiIX9Q8p5Bxn@babar.elephantsql.com:5432/seilbmbd"
      }
    }
  ],
  "sendgrid-n/a": [
    {
      "name": "mysendgrid",
      "label": "sendgrid-n/a",
      "tags": [
        "smtp"
      ],
      "plan": "free",
      "credentials": {
        "hostname": "smtp.sendgrid.net",
        "username": "QvsXMbJ3rK",
        "password": "HCHMOYluTv"
      }
    }
  ]
}
```

The [v2 format](#) of the same services would look like this:

```
VCAP_SERVICES=
{
  "elephantsql": [
    {
      "name": "elephantsql-c6c60",
      "label": "elephantsql",
      "tags": [
        "postgres",
        "postgresql",
        "relational"
      ],
      "plan": "turtle",
      "credentials": {
        "uri": "postgres://seilbmbd:PHxTPJSbkcDakfK4cYwXHiIX9Q8p5Bxn@babar.elephantsql.com:5432/seilbmbd"
      }
    }
  ],
  "sendgrid": [
    {
      "name": "mysendgrid",
      "label": "sendgrid",
      "tags": [
        "smtp"
      ],
      "plan": "free",
      "credentials": {
        "hostname": "smtp.sendgrid.net",
        "username": "QvsXMbJ3rK",
        "password": "HCHMOYluTv"
      }
    }
  ]
}
```

Application Instance-Specific Variables

Each instance of an application can access a set of variables with the `CF_INSTANCE` prefix. These variables provide information for a given application instance, including identification of the host DEA by its IP address.

CF_INSTANCE_ADDR

The `CF_INSTANCE_IP` and `CF_INSTANCE_PORT` of the app instance in the format `particular-DEA-IP:particular-app-instance-port`.

```
CF_INSTANCE_ADDR=1.2.3.4:5678
```

CF_INSTANCE_INDEX

The index number of the app instance.

```
CF_INSTANCE_INDEX=0
```

CF_INSTANCE_IP

The external IP address of the DEA running the container with the app instance.

```
CF_INSTANCE_IP=1.2.3.4
```

CF_INSTANCE_PORT

The `PORT` of the app instance.

```
CF_INSTANCE_PORT=5678
```

CF_INSTANCE_PORTS

The external and internal ports allocated to the app instance.

```
CF_INSTANCE_PORTS=[{external:5678,internal:5678}]
```

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> , <code>revg</code>	Retrieves the contents of the running environment variable group.
<code>staging-environment-variable-group</code> , <code>sevg</code>	Retrieves the contents of the staging environment variable group.
<code>set-staging-environment-variable-group</code> , <code>ssevg</code>	Passes parameters as JSON to create a staging environment variable group.
<code>set-running-environment-variable-group</code> , <code>srevg</code>	Passes parameters as JSON to create a running environment variable group.

The following examples demonstrate how to get the environment variables:

```
$ cf revg
Retrieving the contents of the running environment variable group as sampledeveloper@pivotal.io...
OK
Variable Name    Assigned Value
HTTP Proxy      87.226.68.130

$ cf sevg
Retrieving the contents of the staging environment variable group as sampledeveloper@pivotal.io...
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@pivotal.io...
OK

name          requested state  instances  memory  disk  urls
APP-NAME      started        1/1        256M    1G    APP-NAME.cf-app.com, test-.

$ cf env APP-NAME
Getting env variables for app APP-NAME in org SAMPLE-ORG-NAME / space dev as sampledeveloper@pivotal.io...
OK

System-Provided:

{
  "VCAP_APPLICATION": {
    "application_name": "APP-NAME",
    "application_uris": [
      "APP-NAME.sample-app.com",
      "test-foo.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": [
      "APP-NAME.sample-app.com",
      "test-foo.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 the 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

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, let's say 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.

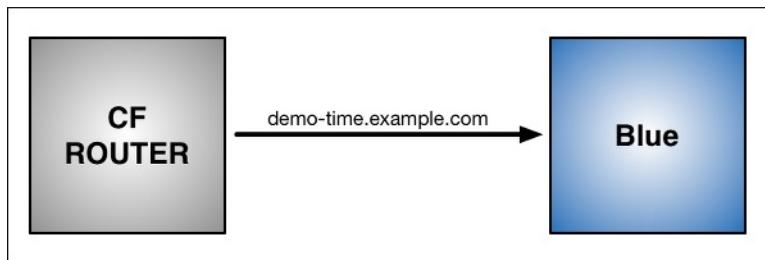
Step 1: Push an App

Use the `cf` command-line interface 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.



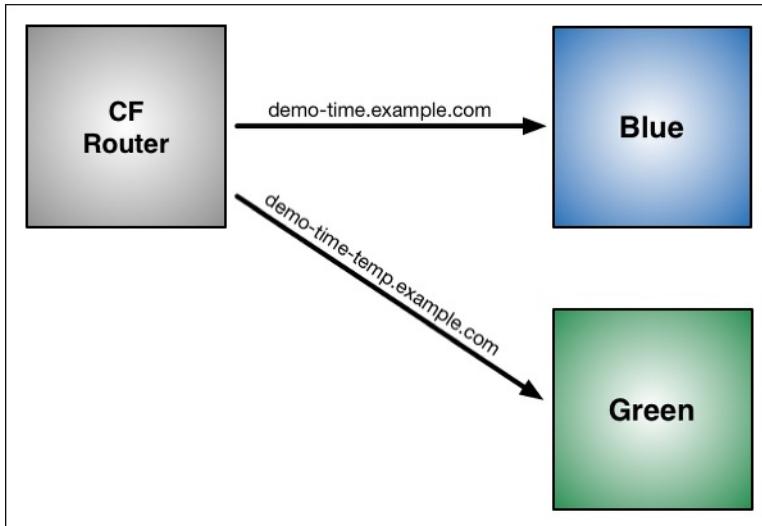
Step 2: Update App and Push

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 cf 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.



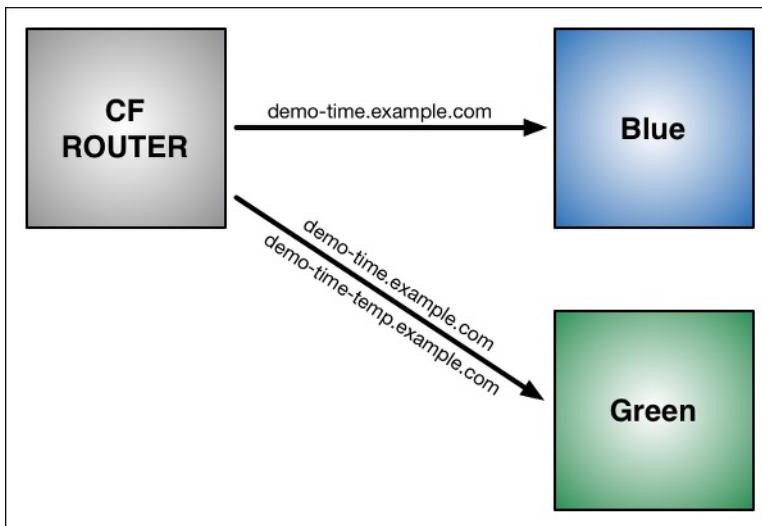
Step 3: Map Original Route to Green

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:

- The CF Router continues to send traffic for `demo-time-temp.com` to Green.
- The CF Router immediately begins to load balance traffic for `demo-time.example.com` between Blue and Green.



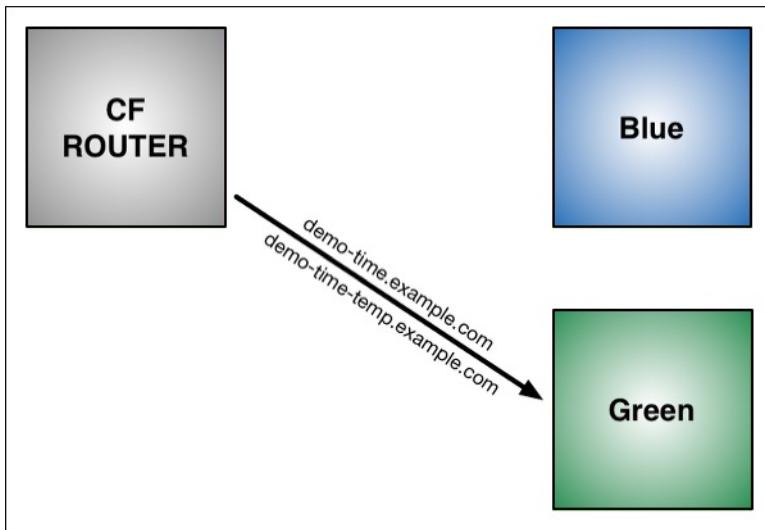
Step 4: Unmap Route to Blue

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
```

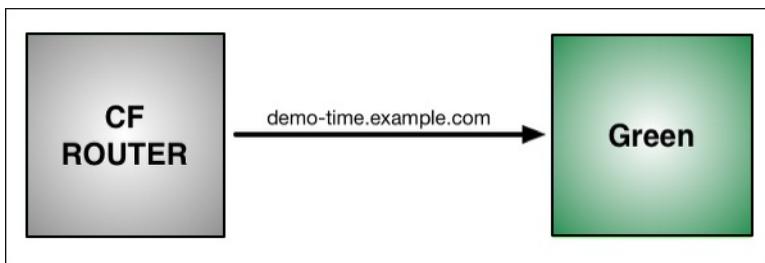
After cf unmap-route:

- 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

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 insures 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.example.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-d4a1156eacbf
    {"route":>"2ef5796b-475a-4615-9c71-75bbe277022e"}
2014-02-06T12:00:27.54-0800 [DEA]
    OUT Got staging request for app with id
    c8612fc2-85b1-464c-92f5-d4a1156eacbf
2014-02-06T12:00:28.51-0800 [API]
    OUT Updated app with guid c8612fc2-85b1-464c-92f5-d4a1156eacbf
    {"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` excludes all Router logs:

```
$ cf logs nifty-gui --recent | grep -v RTR
Connected, dumping recent logs for app nifty-gui in org jancloudspace-org / space development
as jancloudspace@example.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

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 apps 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. Recommended internet connection speed is 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:

- 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 `VCAP_APP_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 `VCAP_APP_PORT` environment variable.

For example, this Ruby snippet assigns the port value to the `listen_here` variable:

```
listen_here = ENV['VCAP_APP_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.

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://lrra:e6B-X@p-mysqlprovider.example.com:5432/lraa",
          "label": "p-mysql-n/a",
          "name": "p-mysql",
          "syslog_drain_url": "",
          "tags": ["postgres", "postgresql", "relational"]
        }
      }
    ]
  }
}

User-Provided:
my-env-var: 100
my-drain: http://drain.example.com
```

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 `cf` itself. 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` command line interface.

Some cf CLI commands may return connection credentials. Remove credentials and other sensitive information from command output before you post the output 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.

Identifying the API Endpoint for your Elastic Runtime Instance

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)
```

cf Command Line Interface (CLI)

When deploying and managing your applications on Cloud Foundry, you'll use the cf command line interface (CLI).

The current production version of the cf CLI is 6.x.

Contents in this section:

- [Installing cf CLI v6](#)
- [Getting Started with the cf CLI](#)
- [Using the cf CLI with an HTTP Proxy Server](#)
- [Using cf CLI Plugins](#)
- [Developing cf CLI Plugins](#)
- [Scaling an Application Using cf scale](#)

Installing the cf Command Line Interface

To install the cf CLI, download it from the Tools page of your Apps Manager instance and follow the instructions for your operating system.

Uninstall cf CLI v5

If you previously used the cf CLI v5 Ruby gem, you must uninstall this gem before installing cf CLI v6.

To uninstall, run `gem uninstall cf`.

 **Note:** To ensure that your Ruby environment manager registers the change, close and reopen your terminal.

Windows Installation

To install cf on Windows:

1. Unpack the zip file.
2. Double click the `cf` executable.
3. When prompted, click **Install**, then **Close**.

Mac OSX and Linux Installation

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**.

Next Steps

To verify your installation, open a terminal window and type `cf`. If your installation was successful, the cf help listing appears.

For information on how to use the cf CLI version 6, see [Getting Started with cf CLI v6](#).

Getting Started with the cf CLI

This guide introduces cf CLI v5 users to what is new in cf CLI v6.

The cf CLI v6 is simpler, faster and more powerful than v5. Consider these key differences:

- The cf CLI has been completely re-written in Go, and is more performant than previous versions, which were written in Ruby.
- There are now native installers for all major operating systems.
- Commands behave consistently and logically: required arguments never have flags; optional arguments always have flags.
- Command names are shorter, yet more communicative, and most have single-letter aliases.
- While many cf CLI v5 commands read manifests, only the push command in cf CLI v6 reads manifests.
- While cf CLI v5 used interactive prompts widely, cf CLI v6 uses interactive prompts only for login and optionally for creating user-provided services (you can also create user-provided services non-interactively).

In the guided tour of the cf CLI v6 which follows, we highlight new and improved features and how to use them.

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`

Installation

cf CLI v6 installs with a simple point-and-click, and you no longer need to install Ruby on your system first (or ever). You can use new binaries or new native installers. See [Install cf CLI Version 6](#).

We recommend that you watch our [tools page](#) to learn when updates are released, and download a new binary or a new installer when you want to update.

CF CLI Localization

The CF CLI i18n enabled the CLI localization so that all strings that are used to communicate with the end-user are translated in the user's native language or in a language that is close to it. For example, a French Canadian user can use the CLI in French (frFR) and Portuguese user can use the CLI in Brazilian Portuguese (ptBR).

`cf config --locale` can be used to set the locale. By default if the user does not sets any locale then en_US will be used as a default locale. User can set the locale as described below...

```
cf config --locale en_US
```

At present Cli supports 8 different locales which are listed below: `deDE` : **German** `en_US` : **English** `esES` : **Spanish** `fr_FR` : **French** `it_IT` : **Italian** `ja_JP` : **Japanese** `ptBR` : **Portuguese (Brazil)** `zh_Hans` : **Chinese (simplified)**

Note: Translations only affect messages generated by the CLI. ## Login ## The `login` command in the cf CLI v6 has expanded functionality. In addition to your username and password, you can provide a target API endpoint, organization, and space. If not specified on the command line, the cf CLI prompts for: * **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. Usage:

```
cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]
```

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 v6 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 new `CF_HOME` environment variable allows you to

locate the `config.json` file wherever you like. ## Push ## In cf CLI v6, `push` is simpler to use and faster. * `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. * Many command line options are now one character long. For example, `-n` is now the flag for hostname or subdomain, replacing “-host”. * There is no longer an interactive mode. * You no longer create manifests interactively. See [Deploying with Application Manifests](./deploy-apps/manifest.html). * You no longer create services with push interactively or in a manifest. See [User-Provided Services](#user-provided) to learn about new commands for creating services. * The `-m` (memory limit) option now requires a unit of measurement: `M`, `MB`, `G`, or `GB`, in upper case or lower case. The cf CLI v6 has expanded capabilities in the form of four new options. * `-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` is suitable for applications which process data while running in the background. These applications, sometimes called “workers” are bound only to services and should not have routes. Usage:

```
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]
```

Optional arguments include: * `-b` — Custom buildpack URL, for example, <https://github.com/heroku/heroku-buildpack-play.git> or <https://github.com/heroku/heroku-buildpack-play.git#stable> to select `stable` branch * `-c` — Start command for the application. * `-d` — Domain, for example, example.com. * `-f` — replaces `-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 (NEW). * `-no-manifest` — Ignore manifests if they exist. * `-no-route` — Do not map a route to this application (NEW). * `-no-start` — Do not start the application after pushing.

 **Note:** The `--no-route` option also removes existing routes from previous pushes of this app.

User-Provided Service Instances

The cf CLI v6 has new commands for creating and updating user-provided service instances. You have the choice of three ways to use these commands: interactively, non-interactively, and in conjunction with third-party log management software as described in [RFC 6587](#). When used with third-party logging, cf 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
```

The cf update-user-provided-service Command

The alias for `update-user-provided-service` is `uups`. You can 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
```

Domains, Routes, Organizations, and Spaces

The relationships between domains, routes, organizations, and spaces have been simplified in cf CLI v6.

- All domains are now mapped to an org.
- Routes are (still) scoped to spaces and apps.

As before, 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 v6 has improved separation between management of private domains and management of shared domains. Only administrators can manage shared domains.

In terms of domains, cf CLI v6 is designed to work with both new and older versions of `cf_release`.

 **Note:** The `map-domain` and `unmap-domain` commands no longer exist.

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. Use `cf create-domain` to create a domain in the desired organization, unless the domain already exists in (or has been shared with) the organization.
2. Use `cf map-route` to map the domain to an application. Use the `-n HOSTNAME` option to specify a unique hostname for each route that uses the same domain.

 **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.

Users and Roles

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.

Consistent Behavior to Help You Work Efficiently

We have focused on building clear principles into the cf CLI v6 to help operators and developers work efficiently and comfortably.

The cf CLI v6 commands behave consistently and logically:

- Required arguments never have flags.
- Optional arguments always have flags.
- When there is more than one required argument, their order matters.
- Optional arguments can be provided in any order.

For example, `cf create-service`, which has three required arguments and you must provide them in order: `SERVICE`, `PLAN`, and `SERVICE_INSTANCE`. On the other hand, `cf push` has one required argument and several optional arguments. You can provide the optional arguments in any order.

Command names have been made more specific and explicit to communicate what they do as clearly as possible. For example:

- `cf map-route` replaces `cf map`.
- `cf marketplace` replaces `cf m`.

One improvement mainly of interest to developers is that `cf curl` is easier to use cf CLI v6.

- `curl` automatically uses the identity with which you are logged in.
- `curl` usage has been simplified to closely resemble the familiar UNIX pattern.

New Aliases and Command Line Help Conventions

The cf CLI v6 command aliases and command line help feature improved consistency and convenience.

The cf CLI v6 introduces single-letter aliases for commonly used commands. For example:

- `cf p` is the alias for `cf push`.
- `cf t` is the alias for `cf target`.

The cf CLI v6 command line help introduces streamlined style conventions:

- User input is capitalized, as in `cf push APP`.
- Optional user input is designated with a flag and brackets, as in `cf create-route SPACE DOMAIN [-n HOSTNAME]`.

Run `cf help` to view a list of all cf CLI commands and a brief description of each. Run `cf <command-name> -h` to view detailed help (including alias) for any command. For example:

```
$ cf p -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/heroku/heroku-buildpack-play.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

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 `HTTP_PROXY` with the hostname or IP address of the proxy server.

The `HTTP_PROXY` environment variable holds the hostname or IP address of your proxy server.

`HTTP_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 `HTTP_PROXY`

`HTTP_PROXY` is set with hostname or IP address of the proxy server in URL format: `http_proxy=http://proxy.example.com`

If the proxy server requires a user name and password, include the credentials:

```
http_proxy=http://username:password@proxy.example.com
```

If the proxy server uses a port other than 80, include the port number:

```
http_proxy=http://username:password@proxy.example.com:8080
```

Setting `HTTP_PROXY` in Mac OS or Linux

Set the `HTTP_PROXY` environment variable using the command specific to your shell. For example, in bash, use the `export` command.

Example:

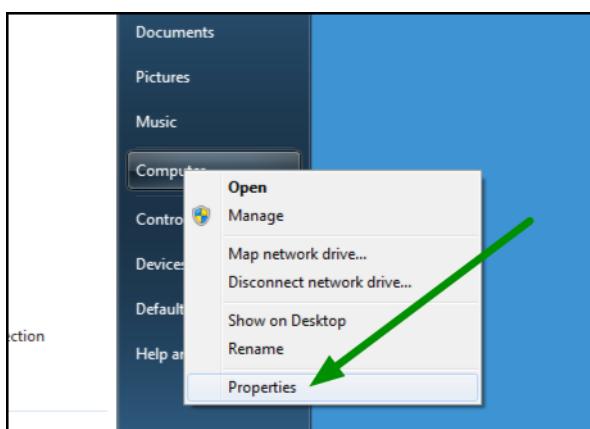
```
$ export HTTP_PROXY=http://my.proxyserver.com:8080
```

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:

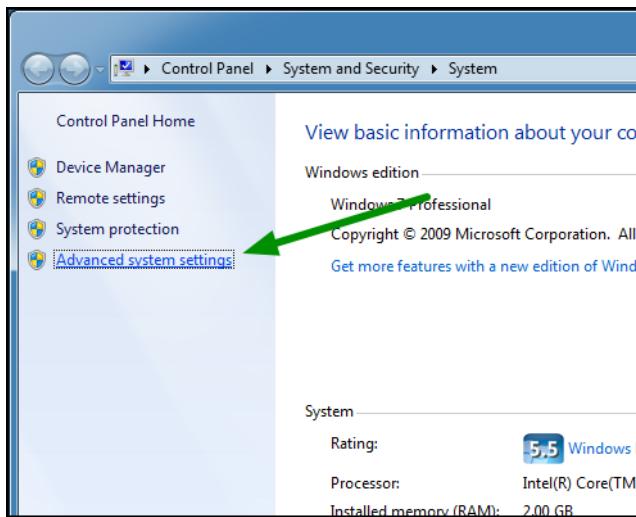
```
http_proxy=http://username:password@hostname:port;
export $HTTP_PROXY
```

Setting `HTTP_PROXY` in Windows

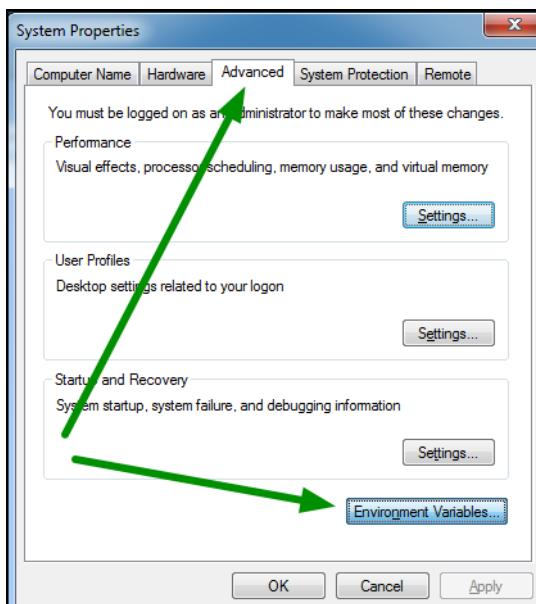
1. Open the Start menu. Right-click **Computer** and select **Properties**.



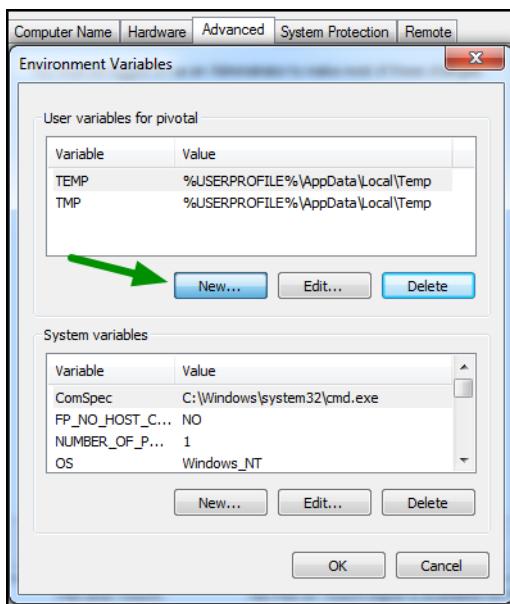
2. In the left pane of the System window, click **Advanced system settings**.



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 `HTTP_PROXY`. In the Variable value field, input your proxy server information.



6. Click **OK**.

Using cf CLI Plugins

The Cloud Foundry Command Line Interface (cf CLI) v.6.7 and higher 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.

The cf CLI identifies a plugin by its binary filename, its developer-defined plugin name, and the commands 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 on how to download, install, and uninstall 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.

 **Notes:**

- 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.
- 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.

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.

About Starting Applications

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](#).

Scaling an Application Using cf scale

Factors such as user load, or the number and nature of tasks performed by an application, can change the disk space and memory the application uses. For many applications, increasing the available disk space or memory can improve overall performance. Similarly, running additional instances of an application can allow the application to handle increases in user load and concurrent requests. These adjustments are called **scaling** an application.

Use `cf scale` to scale your application up or down to meet changes in traffic or demand.

Scaling Horizontally

Horizontally scaling an application creates or destroys instances of your application.

Incoming requests to your application are automatically load balanced across all instances of your application, and each instance handles tasks in parallel with every other instance. Adding more instances allows your application to handle increased traffic and demand.

Use `cf scale APP -i INSTANCES` to horizontally scale your application. Cloud Foundry will increase or decrease the number of instances of your application to match `INSTANCES`.

```
$ cf scale myApp -i 5
```

Scaling Vertically

Vertically scaling an application changes the disk space limit or memory limit that Cloud Foundry applies to all instances of the application.

Use `cf scale APP -k DISK` to change the disk space limit applied to all instances of your application. `DISK` must be an integer followed by either an **M**, for megabytes, or **G**, for gigabytes.

```
$ cf scale myApp -k 512M
```

Use `cf scale APP -m MEMORY` to change the memory limit applied to all instances of your application. `MEMORY` must be an integer followed by either an **M**, for megabytes, or **G**, for gigabytes.

```
$ cf scale myApp -m 1G
```

Services Overview

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 [Running Services](#).

Services and Service Instances

Cloud Foundry Services enable end users to provision a resource on demand. Examples of a resource a service might provide are databases on a multi-tenant server, or simply accounts on a SaaS application.

These resources are known as Service Instances. Think of a service as a factory which produces service instances.

There are two ways in which Cloud Foundry enables developers to provision services: [Managed Services](#) and [User-provided Service Instances](#).

- [Adding a Service](#)
- [Managing Services from the Command Line](#)

Managed Services

A Managed Service integrates with Cloud Foundry via a service broker that implements the Service Broker API.

A service broker advertises a catalog of service offerings and service plans to Cloud Foundry, and receive calls from Cloud Foundry for four functions: create, delete, bind, and unbind. The broker then passes these calls onto the service itself.

For more information on creating managed services see [Managed Service Instances](#).

User-Provided Services

Managed Services have been integrated with Cloud Foundry via APIs and enable end-users to provision new service instances and credentials on demand. User-provided Service Instances are a mechanism to deliver credentials to applications for service instances which have been pre-provisioned outside of Cloud Foundry.

Both operators and developers frequently ask how applications pushed to their Cloud Foundry instances can bind with external services, such as Oracle, which they operate themselves or are outside of Cloud Foundry's control. User-provided Service Instances enable developers to use external services with their applications using familiar workflows.

For more information on using User-Provided Services, see [User-Provided Service Instances](#).

Third-Party Log Management Services

To learn how your applications can drain their logs to a third-party log management solution, see [Third-Party Log Management Services](#). This is also known as setting up a syslog drain.

For more information about specific third-party log management services, see [Configuring Selected Third-Party Log Management Services](#)

About Service Binding

Some services support the concept of Binding. Applications pushed to Cloud Foundry can be bound with service instances that support this feature. When an application is bound to a service instance, information about the service instance (such as location and account credentials) is written to an environment variable in the application runtime called [VCAP_SERVICES](#). The application can use this information to integrate with the service instance.

- [Using Bound Service Instances with your Application](#)
- [VCAP_SERVICES Environment Variable](#)

The following topics describe the different types of services in Cloud Foundry, and have instructions for creating a service, and as necessary, binding it to an application, and configuring the application to connect to it.

For more information on binding services, see [Bind a Service Instance](#)

For information on using services with specific application development frameworks, refer to the [buildpacks](#) documentation.

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](#).

Adding a Service

This page assumes you are using cf CLI v6.

This guide walks you through adding, binding, and using services. It assumes you have pushed an application to your Cloud Foundry instance.

Intro to Services

Cloud Foundry Services are add-ons that can be provisioned alongside your application. Learn all about Services at [Using Services](#).

There are two types of Cloud Foundry services:

- Service brokers advertise catalogs of [managed services](#) such as databases, key-value stores, messaging, or other types of services.
- [User-provided services](#) allow you to represent external assets like databases, messaging services, and key-value stores in Cloud Foundry.

In order to use services with your application you will need to:

1. [Create](#) a service instance.
2. [Bind](#) a service instance to your application.
3. [Update](#) your application to use the service.

Services vs. Service Instances

Services provision services instances. For example, ExampleDB might be a service that provisions MySQL databases. Depending on the plan you select, you might get a database in a multi-tenant server, or a dedicated server.

Not all services provide databases; a service may simply provision an account on their system for you. Whatever is provisioned for you by a service is a service instance.

Creating Managed Service Instances

You can create a managed service instance with the command: `cf create-service SERVICE PLAN SERVICE_INSTANCE`

`cf create-service` takes the following required arguments:

- 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 alpha-numeric characters, hyphens (-), and underscores (_). You can rename the instance at any time.

Following this step, your managed service instance is provisioned:

```
$ cf create-service rabbitmq small-plan my_rabbitmq
Creating service my_rabbitmq in org console / space development as user@example.com... OK
```

 **Note:** For more information about creating a user-provided service instance, refer to [User-Provided Service Instances](#).

Choosing the right plan

Like all PaaSes, Cloud Foundry updates its VMs periodically. When updating DEAs, Cloud Foundry spins up a new DEA with a new copy of your app, then spins down the old one. The number of app instances temporarily increases during the

“rolling update.”

Choose a service plan with enough connections to cover the increase in application instances. During DEA updates, new connections start to fail if the number of connections reaches the service plan limit.

For a small number of app instances, look for a plan that offers twice that many connections. As you increase the number of instances, you can bring the ratio of service connections to instances closer to one-to-one.

Binding a Service Instance to your Application

Some services provide bindable service instances. For services that offer bindable service instances, binding a service to your application adds credentials for the service instance to the [VCAP_SERVICES](#) environment variable.

In most cases these credentials are unique to the binding; another application bound to the same service instance would receive different credentials.

How your application leverages the contents of environment variables may depend on the framework you employ. Refer to the [Deploy Applications](#) topics for more information.

You can bind either a managed or a user-provided service instance to an application with the command

```
cf bind-service APPLICATION SERVICE_INSTANCE
```

`cf bind-service` takes the following required arguments:

- **APPLICATION:** The name of the application to which you want to bind a service.
- **SERVICE_INSTANCE:** The name you provided when you created your service instance.

If the service supports binding, this command binds the service instance to your application.

```
$ cf bind-service rails-sample my_rabbitmq
Binding service my_rabbitmq to app rails-sample in org console / space development as user@example.com... OK
TIP: Use 'cf push' to ensure your env variable changes take effect
```

Use `cf push` to update the [VCAP_SERVICES](#) environment variable with your changes.

Using Bound Services

Once you have a service instance created and bound to your application, you will need to configure the application to dynamically fetch the credentials for your service. These credentials are stored in the [VCAP_SERVICES](#) environment variable. There are generally two methods for these consuming credentials.

- **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.
- **Manual:** [Parse the JSON yourself](#). Helper libraries are available for some frameworks.

See the [buildpacks documentation](#) to learn more about working with specific frameworks.

Binding a Service Instance

This page assumes you are using cf CLI v6.

Binding a Service Instance to your Application

Some services provide bindable service instances. For services that offer bindable service instances, binding a service to your application adds credentials for the service instance to the `VCAP_SERVICES` environment variable.

In most cases these credentials are unique to the binding; another application bound to the same service instance would receive different credentials.

How your application leverages the contents of environment variables may depend on the framework you employ. Refer to the [Deploy Applications](#) topics for more information.

You can bind a service to an application with the command `cf bind-service APPLICATION SERVICE_INSTANCE`. Example:

```
$ cf bind-service my_app example_service
```

Note: You can see which services are available in your current space with the `cf services` command.

Using Bound Services

Once you have a service instance created and bound to your app, you will need to configure your application to use the correct credentials for your service.

There are three ways of consuming service instance credentials within your application.

Binding Strategy	Description
Auto-configuration	Cloud Foundry creates a service connection for you.
cfruntime	Creates an object with the location and settings of your services. Set your service connections based on the values in that object.
Manual	Parse the JSON credentials object yourself from VCAP_SERVICES

Managing Service Instances with the CLI

This page assumes you are using cf CLI v6.

View Available Services

After targeting and logging into Cloud Foundry, you can view what services are available to your targeted organization. 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
```

Create a Service Instance

Creating a service instance provisions a reserved resource from the service. See [Services Overview](#) for more information.

```
$ cf create-service p-mysql 100mb mydb
Creating service mydb in org my-org / space test as me@example.com...
OK
```

Create a User-Provided Service Instance

User-provided service instances are resources which have been pre-provisioned outside of Cloud Foundry. For example, a DBA may provide a developer with credentials to an Oracle database managed outside of, and unknown to Cloud Foundry. Rather than hard coding credentials for these instances into your applications, you can create a mock service instance in Cloud Foundry to represent an external resource and configure it with the credentials provided by your DBA. Once this mock instance is created in the platform, the same CLI commands (documented on this page) can be used to manage user-provided instances as for instances provisioned by the platform.

Refer to the [User Provided Service Instances](#) topic for more information.

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

With v192 of [cf-release](#) and v6.7 of the [CLI](#), Cloud Foundry introduced support for users to update attributes of a service instance (besides the name) after provisioning. Currently the only attribute of an instance that can be modified is the service plan.

Upgrade/Downgrade Service Plan

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 supports this feature, services must expressly implement support for it so not all services will. Further, a service may support updating between some plans but not others (e.g. a service may 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.

You can update the plan for an existing service instance as follows:

```
$ cf update-service mydb -p new-plan
Updating service instance mydb as me@example.com...
OK
```

Bind a Service Instance

Binding a service to your application adds credentials for the service instance to the [VCAP_SERVICES](#) environment variable. In most cases these credentials are unique to the binding; another app bound to the same service instance would receive different credentials. How your app leverages the contents of environment variables may depend on the framework you employ. Refer to the [Deploying Apps](#) section for more information.

- You must restart or in some cases re-push your application for the application to recognize changes to environment variables.
- Not all services support application binding. Many services provide value to the software development process and are not directly used by an application running on Cloud Foundry.

You can bind an existing service to an existing application as follows:

```
$ 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
```

Unbind a Service Instance

Unbinding a service removes the credentials created for your application from the [VCAP_SERVICES](#) environment variable. You must restart or in some cases re-push your application for the application to recognize changes to environment variables.

```
$ cf unbind-service my-app mydb
Unbinding app my-app from service mydb in org my-org / space test as me@example.com...
OK
```

Delete a Service Instance

Deleting a service unprovisions the service instance and deletes *all* data along with the service instance.

```
$ cf delete-service mydb

Are you sure you want to delete the service mydb ? y
Deleting service mydb in org my-org / space test as me@example.com...
OK
```

Third-Party Managed Service Instances

This page assumes you are using cf CLI v6.

Service brokers advertise catalogs of *managed services* to Cloud Foundry. These may be databases, key-value stores, messaging, or other types of services.

The `cf services` command lists all the service instances (both managed and user-provided) in your target space.

In contrast to the managed kind, a user-provided service is managed outside of Cloud Foundry and is not advertised by a service broker. See [User-Provided Service Instances](#).

cf CLI Commands for Working with Managed Service Instances

You can create an instance of a managed service with the `cf create-service` command.

```
cf create-service SERVICE PLAN SERVICE_INSTANCE
```

You can obtain values for the first two arguments from a service broker catalog. The third argument is name of the service instance, and that is up to you.

For example:

```
$ cf create-service example-db basic-plan my_ex-db
```

Once you have created a managed service instance, you can bind it to an application with `cf bind-service`, unbind it with `cf unbind-service`, rename it with `cf rename-service`, and delete it with `cf delete-service`.

Binding Managed Service Instances to Applications

There are two ways to bind managed service instances to applications:

- With a manifest, *when* you push the application.
- With the `cf bind-service` command, *after* you push the application.

In either case, you must push your app to a space where the desired managed service instances already exist.

For example, if you want to bind a service instance called `test-mysql-01` to an app, the services block in the manifest should look like this:

```
services:
- test-mysql-01
```

This excerpt from the `cf push` command and response shows that cf 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-0rg / space development as Bob@example.com
OK
```

Once the binding has taken effect, it is described in the [VCAP_SERVICES](#) environment variable.

For more information about manifests, see [Deploying with Application Manifests](#).

Example: Create a Managed Service Instance and Bind it to an

App

Suppose we want to create an instance of a service called `example-db` and bind it to an app called `db-app`. A service broker catalog informs us that `example-db` offers a plan called “basic.” Our Cloud Foundry username is `Bob` and we plan to name the service instance `test-777`.

1. We begin with the `cf create-service` command.

```
$ cf create-service example-db basic test-777
Creating service test-777 in org My-0rg / space development as Bob@example.com...
OK
```

2. When we list available services, we see that the new service is not yet bound to any apps.

```
$ cf services
Getting services in org My-0rg / space development as Bob@example.com...
OK

name      service      plan      bound apps
test-777  example-db  basic
```

3. Now we bind the service to our app.

```
$ cf bind-service db-app test-777
Binding service test-777 to app db-app in org My-0rg / space development as Bob@example.com...
OK
TIP: Use 'cf push' to ensure your env variable changes take effect
```

4. For the binding to take effect, we must push our app a second time.

```
$ cf push db-app
Updating app db-app in org My-0rg / space development as Bob@example.com...
OK
...
```

5. To verify that the service instance is now recorded in `VCAP_SERVICES`, examine the output of `cf env`:

```
$ cf env db-app
Getting env variables for app db-app in org My-0rg / space development as Bob@example.com...
OK

System-Provided:
{
  "VCAP_SERVICES": {
    "example-db-n/a": [
      {
        "name": "test-777",
        "label": "example-db-n",
        "tags": ["mysql", "relational"],
        "plan": "basic",
        "credentials": {
          "jdbcUrl": "jdbc:mysql://aa11:2b@cdbr-05.example.net:3306/ad_01",
          "uri": "mysql://aa11:2b@cdbr-05.example.net:3306/ad_01?reconnect=true",
          "name": "ad_01",
          "hostname": "cdbr-05.example.net",
          "port": "1234",
          "username": "aa11",
          "password": "2b"
        }
      }
    ]
  }
}

No user-defined env variables have been set
```

User-Provided Service Instances

This page assumes you are using cf CLI v6.

User-provided service instances allow you to represent external assets like databases, messaging services, and key-value stores in Cloud Foundry.

For example, suppose a developer obtains credentials, a URL, and a port number for communicating with an Oracle database that is managed outside of Cloud Foundry. Then the developer creates a user-provided service instance to represent this external resource and provisions the service instance with all the parameters required for communicating with the real Oracle database. Finally, she binds the user-provided service instance to her application.

The user-provided service instance allows the developer to:

- Avoid hard-coding credentials for a service instance into her application.
- Treat the user-provide service instance exactly like any other service instance.

Although the external asset in this example was a real Oracle database, it could have been a simple placeholder used for testing.

User provided services are outside of Cloud Foundry, so you never create or otherwise manage the service itself through Cloud Foundry; you only create and manage service *instances*.

The `cf services` command lists all the service instances (both user-provided and managed) in your target space.

In contrast to the user-provided kind, a managed service is one that a service broker advertises in a catalog. See [Managed Service Instances](#).

cf CLI Commands for Working with User-Provided Service Instances

cf CLI commands for creating and updating user-provided service instances can be used in three ways: interactively, non-interactively, and in conjunction with third-party log management software as described in [RFC 6587](#). When used with third-party logging, cf 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
```

The cf update-user-provided-service Command

The alias for `update-user-provided-service` is `uups`. You can 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
```

Binding User-Provided Service Instances to Applications

There are two ways to bind user-provided service instances to applications:

- With a manifest, *when* you push the application.
- With the `cf bind-service` command, *after* you push the application.

In either case, you must push your app to a space where the desired user-provided instances already exist.

For example, if you want to bind a service instance called `test-mysql-01` to an app, the services block in the manifest should look like this:

```
services:
- test-mysql-01
```

This excerpt from the `cf push` command and response shows that Cloud Foundry reads the manifest and binds the service instance to the app, which is called `msgapp`.

```
$ cf push
Using manifest file /Users/a.user/test-apps/msgapp/manifest.yml

...
Binding service test-mysql-01 to msgapp in org my-org / space development as a.user@example.com
OK
```

Once the binding has taken effect, it is described in the [VCAP_SERVICES](#) environment variable.

For more information about manifests, see [Deploying with Application Manifests](#).

For an example of the `cf bind-service` command, see the next section.

Example: Creating a User-Provided Service Instance and Binding it to an App

Suppose we want to create and bind an instance of a publish-subscribe messaging service to an app called `msgapp`. Our Cloud Foundry username is `a.user`, and we plan to call the instance `pubsub`. We know the URL and port for the service.

1. We begin with the `cf create-user-provided-service` command in interactive mode, using its alias, `cups`.

```
$ cf cups pubsub -p "host, port, binary, username, password"
host> pubsub01.example.com
port> 1234
url> pubsub-example.com
username> pubsubuser
password> p@$$w0rd
Creating user provided service pubsub in org my-org / space development as a.user@example.com...
OK
```

2. When we list available services, we see that the new service is not yet bound to any apps.

```
$ cf services
Getting services in org my-org / space development as a.user@example.com...
OK

name          service      plan   bound apps
pubsub        user-provided
```

3. Now we bind the service to our app.

```
$ cf bind-service msgapp pubsub
Binding service pubsub to app msgapp in org my-org / space development as a.user@example.com...
OK
TIP: Use 'cf push' to ensure your env variable changes take effect
```

4. For the binding to take effect, we must push our app a second time.

Note: You can skip this step if you bind a syslog drain service instance to a deployed app. Syslog drain service instances bound to deployed apps automatically start after a short delay.

```
$ cf push msgapp
Updating app msgapp in org my-org / space development as a.user@example.com...
OK
...
```

5. To verify that the service instance is now recorded in `VCAP_SERVICES`, examine the output of `cf env`:

```
$ cf env msgapp
Getting env variables for app msgapp in org My-Org / space development as a.user@example.com...

System-Provided:
{
  "VCAP_SERVICES": {
    "user-provided": [
      {
        "name": "pubsub",
        "label": "user-provided",
        "tags": [],
        "credentials": {
          "binary": "pubsub.rb",
          "host": "pubsub01.example.com",
          "password": "pg29w0rd",
          "port": "1234",
          "username": "pubsubuser"
        },
        "syslog_drain_url": ""
      }
    ]
  }
}
```

Configuring Play Framework Service Connections

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

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@cdb-05.example.net:3306/ad_01",
          "uri": "mysql://aa11:2b@cdb-05.example.net:3306/ad_01?reconnect=true",
          "name": "ad_01",
          "hostname": "cdb-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](#).

Using Third-Party Log Management Services

This page assumes you are using cf CLI v6.

Logs tell you both what your app does and what the system does with your app. For any one application, logs come from each instance of the app and also from Cloud Foundry system components. For example, when a Cloud Foundry Router routes an HTTP request to an app, the Router records that event in the application log stream.

Cloud Foundry keeps a limited amount of logging information in memory. When you want to persist more log information than this, you must drain logs to a third-party log management service. You can then use the third-party service to analyze the logs. For example, you could produce metrics for application response times or error rates.

Providers whose log management products or services work with Cloud Foundry include Logentries, logstash, Papertrail, and Splunk. You will need to [setup an appropriate connection with these services for use with Cloud Foundry](#).

You can think of the relationship between Cloud Foundry and the third-party logging service in terms of source, sink, and drain:

- The application and the system components that interact with the app are an information **source**.
- The logging service is an information **sink**.
- A **drain** configured on the application helps the source communicate with the sink.

Loggregator

The Loggregator component of Cloud Foundry consolidates the logs of all actions that directly affect the app.

Loggregator keeps a limited amount of app log data in memory. Eventually the newest log lines replace the oldest. Since all log data is stored transiently in memory, restarting the loggregator service loses any log data that you have not exported.

When sending logs to third-party services, Loggregator formats the data according to the *Syslog Protocol* defined in [RFC 5424](#). Third-Party Log Management Software is defined in [RFC 6587](#).

In the Command Line Interface (CLI), you can stream an app log or view the most recent log data in memory. Logs viewed in the CLI are not in syslog format, and do not exactly match the output to third-party services.

Draining Logs to a Third-Party Log Management Service

To drain logs from your application and relevant Cloud Foundry components to a third-party log management service, complete this procedure:

1. Configure the log management service to treat your application as a source of data.
2. Create a user-provided service instance with a syslog drain.
3. Bind the service instance to the application.
4. Verify that logs are draining correctly.

Details about each of these steps follow.

Configuring the log management service to treat your application as a source of data

Typically, you must configure a third-party log management service to accept traffic from your Cloud Foundry environment. Depending on the service provider, you may need the following information:

- The external IP addresses your Cloud Foundry administrator assigns to outbound traffic.

Note: You must whitelist all of these IP addresses. Allowing traffic from a subset of the external IP addresses assigned to outbound traffic restricts the logs that are available to third-party services.
- 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.

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.

Creating a user-provided service instance with a syslog drain

Create a user-provided service instance using the `create-user-provided-service` command with the `-l` option and the syslog URL you obtained earlier. The `-l` option configures the syslog drain. Refer to the [User-Provided Services](#) topic for more information.

Binding the service instance to the application

You have two options:

- Use `cf push` with a manifest. The services block in the manifest must specify the service instance that you want to bind.
- Use `cf bind-service`. After a short delay, logs begin to flow automatically.

See [Binding User-Provided Service Instances to Applications](#).

Verifying that Logs are Draining Correctly

To verify that logs are draining correctly to a third-party log management service:

1. Take actions that you believe should produce log messages, such as making requests of your app.
2. Compare the logs you see in the CLI against those that the service shows you.

For example, assuming that 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. The third-party logging service should show you corresponding messages, although probably in a different format.

Note: For security reasons, Cloud Foundry applications do not respond to ping, so you cannot use ping to generate log entries.

Configuring Selected Third-Party Log Management Services

Instructions for configuring:

- [Papertrail](#)
- [Splunk Storm](#)
- [SumoLogic](#)
- [Logentries](#)

Once you have configured the service, refer to [Third-Party Log Management Services](#) to bind your application to the service.

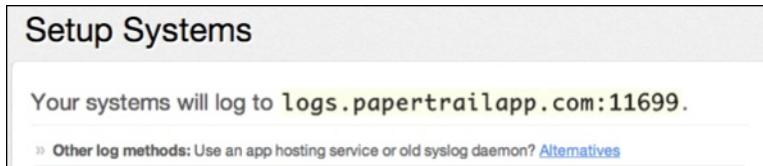
Papertrail

From your Papertrail account:

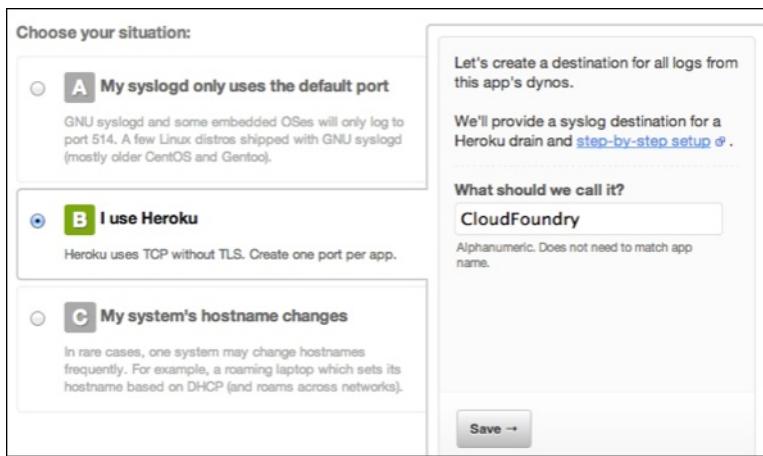
1. Click the **Add System** button.



2. Click the **Alternatives** link.



3. Select **I use Heroku**, enter a name, and click **Save**.



4. Note the URL with port that is displayed after creating the system.



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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
  org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
  Mar 05 14:57:36 CloudFoundry 40dc960-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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
  org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} INFO: No beans of type
  org.springframework.orm.amp.ConnectionFactory found
  in application context. Skipping auto-reconfiguration.
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
  org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
Mar 05 14:57:36 CloudFoundry 40dc960-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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.680 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/albums/{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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.681 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/albums","methods=[GET]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public void org.cloudfoundry.samples.music.domain.Album
  org.cloudfoundry.samples.music.web.controllers.AlbumController.deleteById(java.lang.String)
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.681 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/info","methods=[GET]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public
  {"/albums","methods=[PUT]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public org.cloudfoundry.samples.music.domain.Album
  org.cloudfoundry.samples.music.web.controllers.AlbumController.add(org.cloudfoundry.samples.music.domain.Album)
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.682 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/albums/{id}","methods=[PUT]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public org.cloudfoundry.samples.music.domain.Album
  org.cloudfoundry.samples.music.web.controllers.AlbumController.update(java.util.Map<java.lang.String, java.util.List<org.cloudfoundry.samples.music.domain.Album>>)
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.684 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/info","methods=[PUT]","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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.684 INFO RequestMappingHandlerMapping:181 - Mapped
  {"/info","methods=[PUT]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public
  java.util.List<org.springframework.cloud.service.ServiceInfo> org.cloudfoundry.samples.music.web.controllers.InfoController.infoServiceInfo
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.684 INFO RequestMappingHandlerMapping:181 - Mapped
  {"{/env}","methods=[PUT]","params=[],headers=[],consumes=[],produces=[],custom={}"} onto public
  java.util.Map<java.lang.String, java.lang.String> org.cloudfoundry.samples.music.web.controllers.InfoController.infoShowEnvironment
Mar 05 14:57:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.704 INFO SimpleUrlHandlerMapping:362 - Root mapping to handler of
  type [class org.springframework.web.servlet.mvc.ParameterizableViewController]
Mar 05 14:57:36 CloudFoundry 40dc960-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 40dc960-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.DispatcherServlet]
Mar 05 14:57:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:37.040 INFO DispatcherServlet:488 - FrameworkServlet 'appServlet':
  initialization completed in 989 ms
Mar 05 14:57:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:37 PM org.apache.coyote.AbstractProtocol start
  Mar 05 14:57:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:37.040 INFO: Starting ProtocolHandler ["http-bio-43390"]
Mar 05 14:57:37 CloudFoundry 40dc960-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 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} INFO: Server startup in 11278 ms
Mar 05 14:59:36 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Tried to stop app that never received a start event
Mar 05 14:59:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:37.040 INFO: Stopped ProtocolHandler ["http-bio-43390"]
Mar 05 14:59:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{App/1} Stopped app instance (Index 1) with guid 40dc960-c093-4bbb-8718-186f147f3e73
Mar 05 14:59:37 CloudFoundry 40dc960-c093-4bbb-8718-186f147f3e73/{DEA} Stopped app instance (Index 1) with guid 40dc960-c093-4bbb-8718-186f147f3e73

```

Example: "access denied" (1.2.3.4 OR redis) -sshd

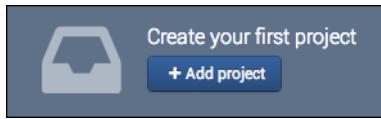
Search ⌂ Contrast ⌂ PAUSE

https://papertrailapp.com/groups/553803/events?centered_on_id=378003420845892611&q=program%3Aa%a&c060-c093-4bbb-8718-186f147f3e73%27%5BApp%27%5D

Splunk Storm

From your Splunk Storm account:

1. Click the **Add project** button.



2. Enter the project details.

Add project

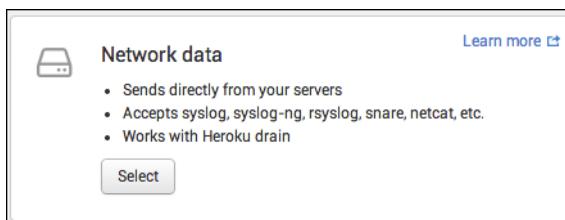
* Project name: Cloud Foundry

* Project time zone: (UTC-0600) America/Chicago

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.

Add network data

Storm can receive data from any network port. For example, Storm can accept remote data from [syslog](#), [rsyslog](#), [syslog-ng](#), [snare](#), [netcat](#) or any other application that transmits via TCP or UDP. [Learn more](#)

Authorize your IP address ▾

Automatically

Manually

5. Note the host and port provided for TCP input.

Authorized network inputs

Storm can receive data from any network port. For example, Storm can accept remote data from [syslog](#), [rsyslog](#), [syslog-ng](#), [snare](#), [netcat](#) or any other application that transmits via TCP or UDP.

1. Authorize your IP address

2. Send data to these ports for this project only

TCP	tcp.k22g-wt6r data.splunkstorm.com:15486
UDP	udp.k22g-wt6r data.splunkstorm.com:15486

Authorize automatically

Authorize all IP addresses sending data to this project within the next 15 min.

Authorize manually

Specify the IP address

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**.

What to Search

266 Events INDEXED a minute ago EARLIEST EVENT a few seconds ago LATEST EVENT

Data Summary

9. Click the **loggregator** link to view all incoming log entries from Cloud Foundry.

Data Summary

Hosts (1) Sources (1) Sourcetypes (1)									
<input type="text" value="Filter"/> <table border="1"> <thead> <tr> <th>Host</th> <th>Count</th> <th>Last Update</th> </tr> </thead> <tbody> <tr> <td>loggregator</td> <td>26</td> <td>3/7/14 3:26:01.000 PM</td> </tr> </tbody> </table>				Host	Count	Last Update	loggregator	26	3/7/14 3:26:01.000 PM
Host	Count	Last Update							
loggregator	26	3/7/14 3:26:01.000 PM							

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.

Manage Collectors and Sources

Upgrade Collectors Add Collector Access Keys

2. Choose **Hosted Collector** and fill in the details.



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						
Upgrade Collectors Add Collector Access Keys						
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 **HTTP**

Collects logs from an Amazon S3 bucket.

HTTP receiver that collects logs sent to a specific address.

Name * Maximum name length is 128 characters

Description

Source Host Host name for the system from which the log files are being collected, e.g. LDAP_Server

Source Category Log category metadata to use later for querying, e.g. OS_Security

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.

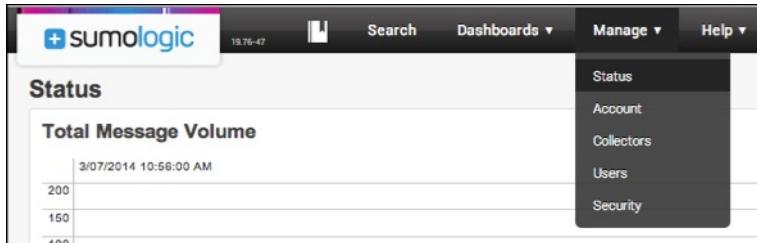
Manage Collectors and Sources						
Upgrade Collectors Add Collector Access Keys						
Show: All Collectors Running Collectors Stopped Collectors Expand: All None						
Name	Type	Status	Source Category	Sources	Last Hour	Messages
Cloud Foundry	Hosted	✓		1	None	Add Source Edit Delete
CloudFoundry	HTTP	✓				Show URL Edit Delete

6. Create the log drain service in Cloud Foundry using the displayed URL.

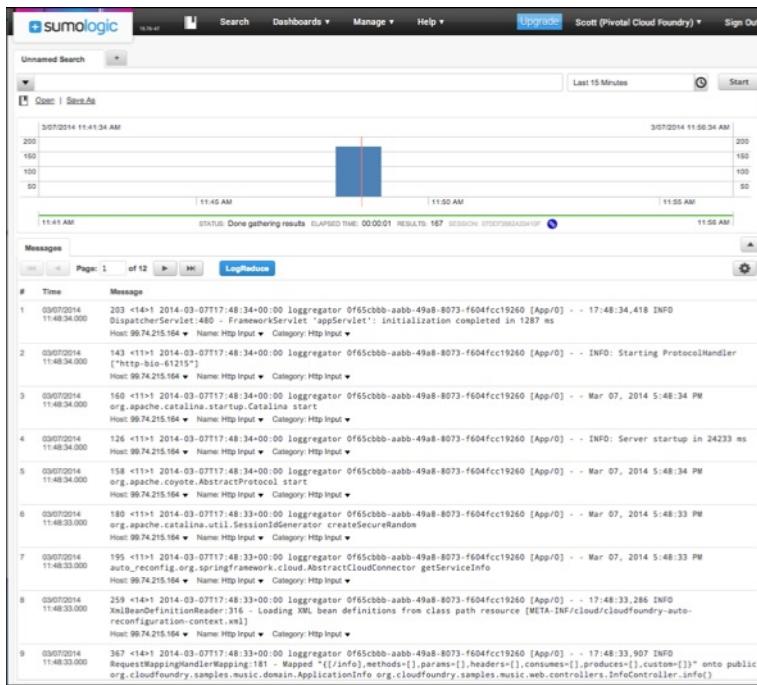
```
$ cf cups my-logs -l <HTTP SOURCE-URL>
```

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.



Logentries

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.

Integrating Cloud Foundry with Splunk

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 = privel::$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

Buildpacks

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.

Cloud Foundry deployments often have limited access to dependencies. This occurs when the deploy is behind a firewall, or when administrators want to use local mirrors and proxies. In these circumstances, Cloud Foundry provides a [Buildpack Packager](#) application.

System Buildpacks

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
Go	NA	Go source
Java	Grails, Play, Spring, or any other JVM-based language or framework	Java source
Node.js	Node or JavaScript	Node.js source
PHP	NA	PHP source
Python	NA	Python source
Ruby	Ruby, Rack, Rails, or Sinatra	Ruby 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

 **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](#).

- [Write](#) your own buildpack
- Use a [Cloud Foundry Community Buildpack](#)
- Use a [Heroku Third-Party Buildpack](#)

Buildpack Detection

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

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
```

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
```

Partially or completely disconnected environments

Cloud Foundry buildpacks work with limited or no internet connectivity. You can enable the same support in your custom buildpack by using the Buildpack Packager [application](#).

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` command 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, it is possible to 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

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

Java Buildpack

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](#)

The source for the buildpack is [here](#).

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.

The Java buildpack source documentation states:

- The Java buildpack logs all messages, regardless of severity to `<app dir>/java-buildpack.log`. It also logs messages to `$stderr`, filtered by a configured severity.
- If the buildpack fails with an exception, the exception message is logged with a log level of `ERROR` whereas the exception stack trace is logged with a log level of `DEBUG` to prevent users from seeing stack traces by default.

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](#).

Getting Started Deploying Grails Apps

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 [cf Command Line Interface \(CLI\)](#)
- 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.

```
dependencies {  
    // specify dependencies here under either 'build', 'compile', 'runtime', 'test' or 'provided' scopes e.g.  
    // runtime 'mysql:mysql-connector-java:5.1.29'  
    // runtime 'org.postgresql:postgresql:9.3-1101-jdbc41'  
    test "org.grails:grails-datastore-test-support:1.0-grails-2.4"  
    runtime 'mysql:mysql-connector-java:5.1.33'  
}
```

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.

```
dataSource {  
    pooled = true  
    jmxExport = true  
    driverClassName = "com.mysql.jdbc.Driver"  
    dialect = org.hibernate.dialect.MySQL5InnoDBDialect  
    uri = new URI(System.env.DATABASE_URL ?: "mysql://foo:bar@localhost")  
    username = uri.userInfo ? uri.userInfo.split(":")[0] : ""  
    password = uri.userInfo ? uri.userInfo.split(":")[1] : ""  
    url = "jdbc:mysql://" + uri.host + uri.path  
  
    properties {  
        dbProperties {  
            autoReconnect = true  
        }  
    }  
}
```

Step 4: (Optional) Configure a Procfile

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/[PROJECT_NAME]/bin/[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 on creating and using service instances, refer to the [Services Overview](#) topic.

Create a Service Instance

Run `cf marketplace` to view managed and user-provided services and plans available to you.

The example shows two of the available managed database-as-a-service providers and their offered plans: `cleardb`, database-as-a-service for MySQL-powered apps and `elephantsql` PostgreSQL as a Service.

```
$ cf marketplace
Getting services from marketplace in org Cloud-Apps / space development as clouduser@example.com...
OK

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 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:  
- 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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different `HOST` name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs [APP-NAME]`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step

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.

```
$ |master ✓| → 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
      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
      Modifying /WEB-INF/web.xml for Auto Reconfiguration
----> Downloading Tomcat Instance 8.0.14 from https://download.run.pivotal.io/tomcat/tomcat-8.0.14.tar.gz (0.4s)
      Expanding Tomcat to .java-buildpack/tomcat (0.1s)
----> Downloading Tomcat Lifecycle Support 2.4.0 RELEASE from https://download.run.pivotal.io/tomcat-lifecycle-support/tom
----> Downloading Tomcat Logging Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-logging-support/tomcat-
----> Downloading Tomcat Access Logging Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-access-logging-s

----> 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 Created 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

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.

```
dependencies {  
    // SpringLoaded enables runtime hot reloading.  
    // It is not part of the app runtime and is not shipped in the distribution.  
    springloaded "org.springframework:springloaded:1.2.0.RELEASE"  
  
    // Default SLF4J binding. Note that this is a blocking implementation.  
    // See here for a non blocking appender http://logging.apache.org/log4j/2.x/manual/async.html  
    runtime 'org.slf4j:slf4j-simple:1.7.7'  
  
    compile group: 'redis.clients', name: 'jedis', version: '2.5.2', transitive: true  
  
    testCompile "org.spockframework:spock-core:0.7-groovy-2.0"  
}
```

Step 2: Allocate Sufficient Memory

Use the `cf push -m` command to specify the amount of memory that should be allocated to the application. Memory allocated this way is done in preset amounts of `64M`, `128M`, `256M`, `512M`, `1G`, or `2G`. For example:

```
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/[PROJECT_NAME]/bin/[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 on 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   25mb, 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 25mb baby-redis`. This creates a service instance named `baby-redis` that uses the `rediscloud` service and the `25mb` plan, as the example below shows.

```
$ cf create-service rediscloud 25mb 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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different `HOST` name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs [APP-NAME]`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step

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.

```
$ |master x| → 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 clouduser@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 clouduser@example.com...
OK

Starting app pong_matcher_groovy in org Cloud-Apps / space development as clouduser@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 clouduser@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 Created` 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 Spring Apps

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.

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
  <dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

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 on 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 clouduser@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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app
- `--random-route` to create a URL that includes the app name and random words
- `cf help push` to view other options for this command

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs [APP-NAME]`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step

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 [HOST_NAME]` 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.

```
$ |master ~/| → 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 clouduser@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 clouduser@example.com...
OK

Starting app pong_matcher_spring in org Cloud-Apps / space development as clouduser@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
      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
----> Uploading droplet (63M)

0 of 1 instances running, 1 starting
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 clouduser@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 Tools 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 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 an Elastic Runtime administrator can run. If you are not an Elastic Runtime administrator, the following message displays for these types of commands:
`error code: 10003, message: You are not authorized to perform the requested action` For more information about specific Admin commands you can perform with the Apps Manager, depending on your user role, refer to the [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

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 <application-name> -p target/<application-name>-<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 <application-name> -p build/distributions/<application-name>.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 <application-name>
```

Java Main

Java applications with a `main()` method can be run provided that they are packaged as [self-executable JARs](#).

Maven

A Maven build can create a self-executable JAR. To build and deploy the JAR, run the following:

```
$ mvn package
$ cf push <application-name> -p target/<application-name>-<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 <application-name> -p build/libs/<application-name>-<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 <application-name> -p target/universal/<application-name>-<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 <application-name>
```

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 <application-name> -p target/<application-name>-<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 <application-name> -p build/libs/<application-name>-<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#0928916a2dd78e9faf9469c558046eef09f60e5d
--> 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 on 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>:<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 APPNAME` 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");
```

The environment variables you can access are those [defined by the DEA](#).

Configure Service Connections for Grails

Cloud Foundry provides extensive support for connecting a Grails application to services such as MySQL, Postgres, MongoDB, Redis, and RabbitMQ. In many cases, a Grails application running on Cloud Foundry can automatically detect and configure connections to services. For more advanced cases, you can control service connection parameters yourself.

Auto-Configuration

Grails provides plugins for accessing SQL (using [Hibernate](#)), [MongoDB](#), and [Redis](#) services. If you install any of these plugins and configure them in your `Config.groovy` or `DataSource.groovy` file, Cloud Foundry re-configures the plugin with connection information when your app starts.

If you were using all three types of services, your configuration might look like this:

```
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

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

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 reconfigures 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 on 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

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 Tools 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:

- Deploy applications from an Eclipse or STS workspace to a running Cloud Foundry instance. The 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 several 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 Tools Suite**) from the Eclipse (or **Spring Tools Suite**) menu and click **Installation Details**.
2. On the **Installation Details**, select the previous version of the plugin and click **Uninstall**.

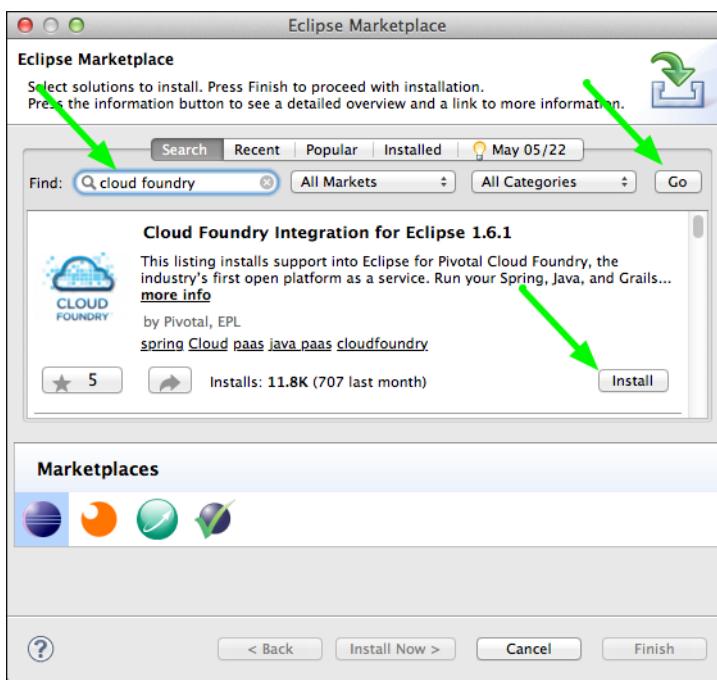
Follow the installation instructions appropriate for your environment:

- [Install to Eclipse from Marketplace](#)
- [Install to STS from Extensions Tab](#)
- [Install from a Local Repository](#)

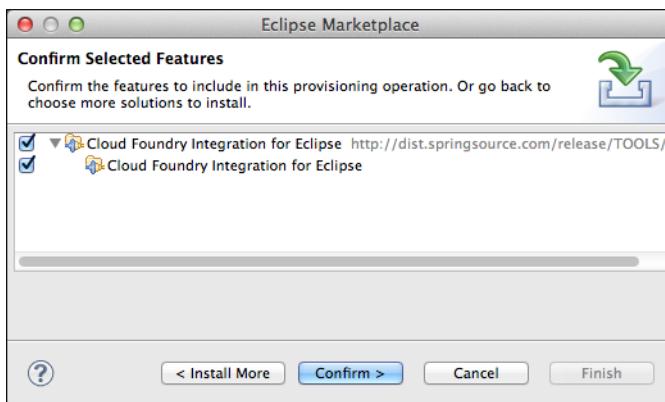
Install to Eclipse from Marketplace

Follow these instructions to install the Cloud Foundry Eclipse Plugin to Eclipse from the Eclipse Marketplace.

1. Start Eclipse.
2. Select **Eclipse Marketplace** from the Eclipse **Help** menu.
3. On the marketplace window, enter “Cloud Foundry” in the **Find** field, and click **Go**.
4. In the search results, click the **Install** control next to the listing for Cloud Foundry Integration.

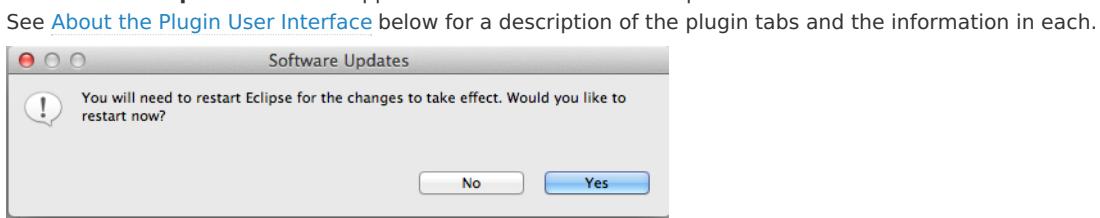


5. In the **Confirm Selected Features** window, click **Confirm**.



6. The **Review Licenses** window appears. Click "I accept the terms of the license agreement" and click **Finish**.

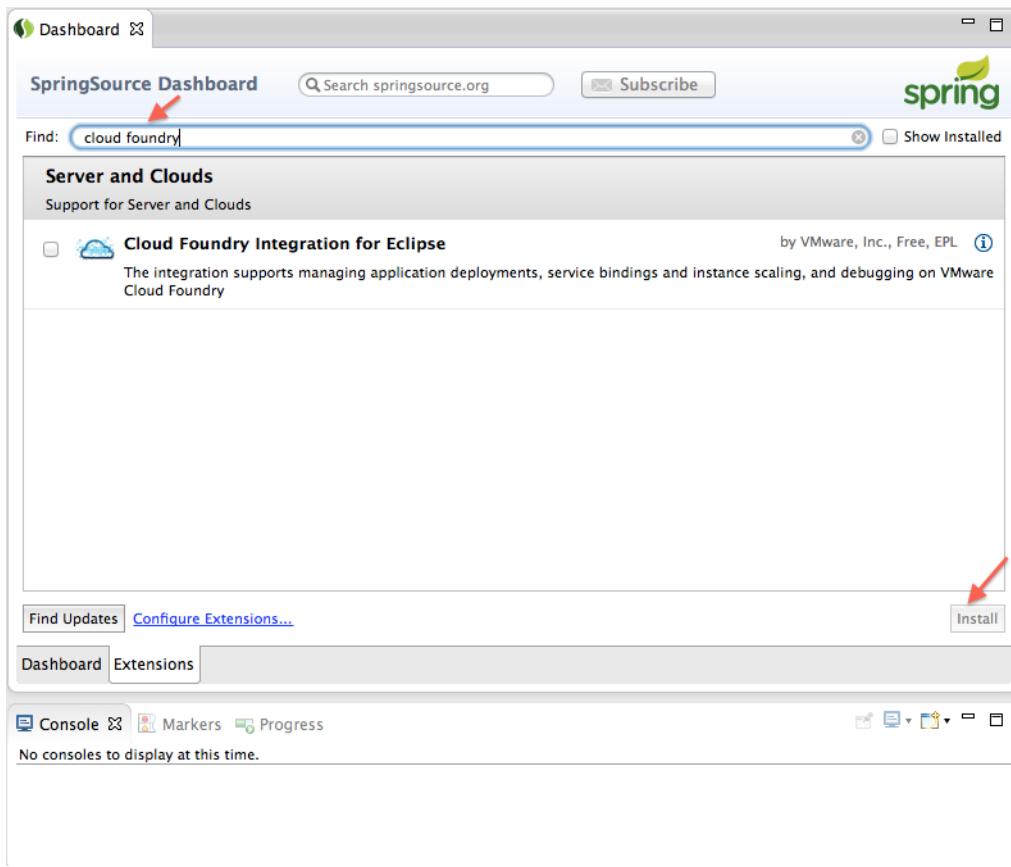
7. The **Software Updates** window appears. Click **Yes** to restart Eclipse.



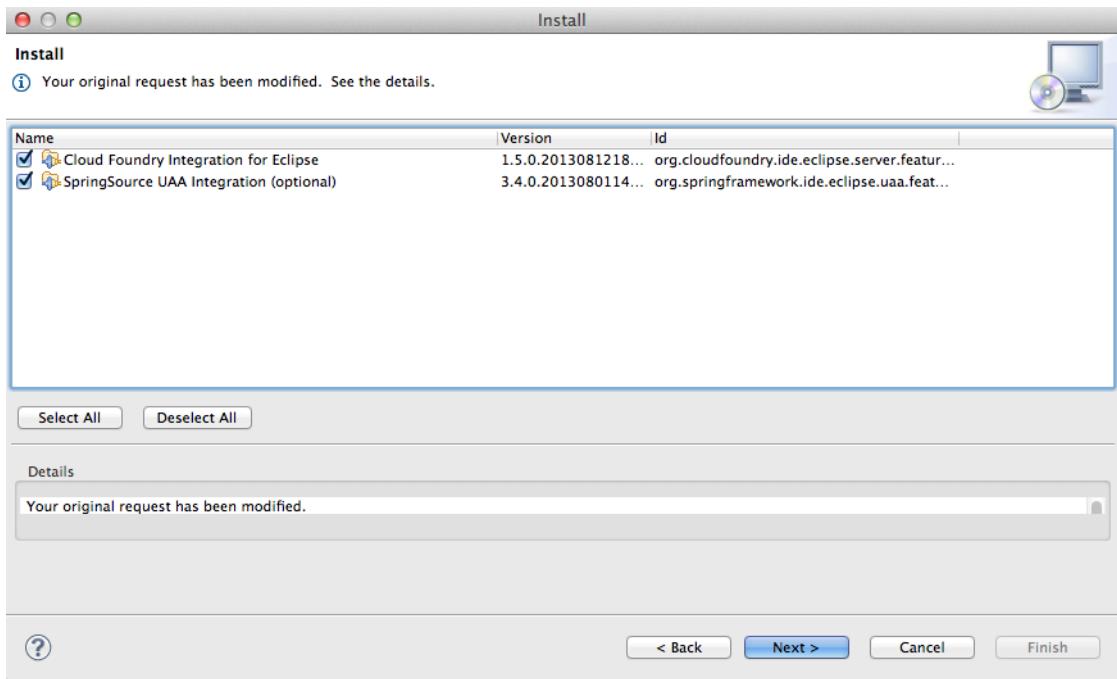
Install to STS from Extensions Tab

Follow these instructions to install the Cloud Foundry Eclipse Plugin to STS from the **Extensions** tab.

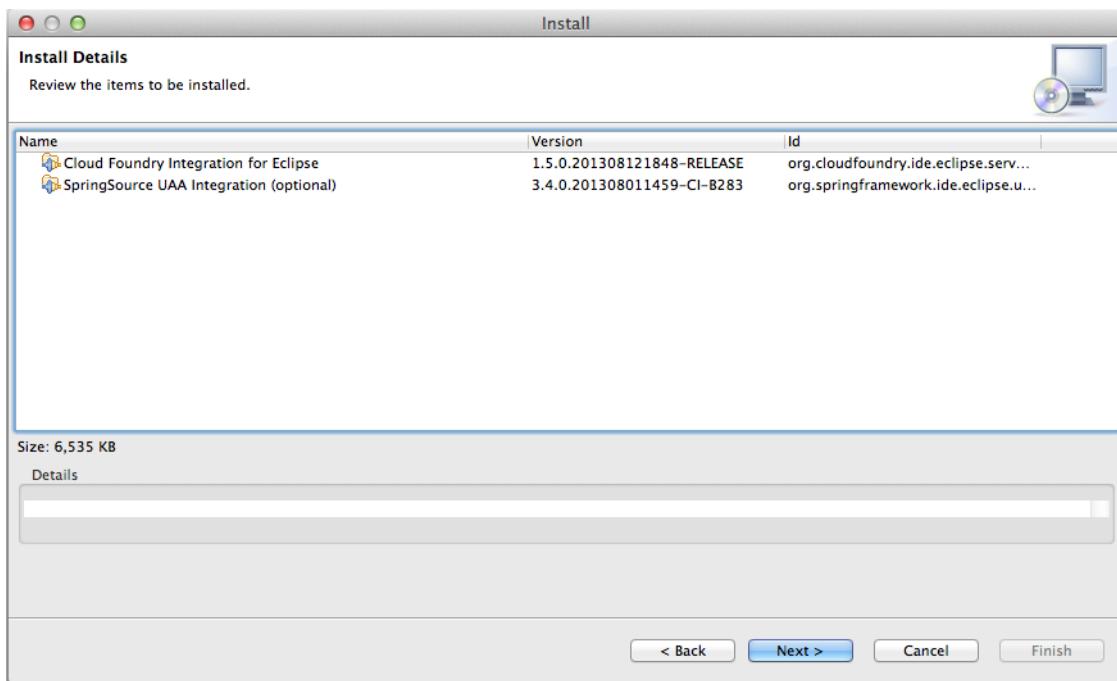
1. Start STS.
2. Select the **Extensions** tab from the STS Dashboard.
3. Enter "Cloud Foundry" in the **Find** field.
4. Select **Cloud Foundry Integration for Eclipse** and click **Install**.



5. In the **Install** window, click **Next**.



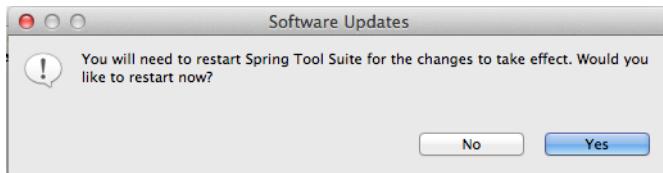
6. On the **Install Details** window, click **Next**.



7. The **Review Licenses** window appears. Click “I accept the terms of the license agreement” and click **Finish** to perform the installation.

8. The **Software Updates** window appears. Click **Yes** to restart STS.

After STS restarts, you can see new panes in the Editor portion of the screen. See [About the Plugin User Interface](#) below for a description of the plugin tabs and the information in each.



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.

Step 1: Download a release update site zip file.

- Select a release update site zip file from: <https://github.com/cloudfoundry/eclipse-integration-cloudfoundry#offline-installation>

Step 2: Install from local update site.

On the machine running Eclipse or Spring Tools Suite:

1. In Eclipse (or STS), select **Install New Software** from the **Help** menu.
2. On the **Available Software** window, click **Add** to the right of the **Work with** field.
3. On the **Add Repository** window, enter a name for the repository, for example “Cloud Foundry Integration”, and click **Archive**.
4. On the **Open** window, browse to the location of the update site zip file and click **Open**.
5. On the **Add Repository** window, click **OK**.
6. On the **Available Software** window, check “Core/Cloud Foundry Integration” and, optionally, “Resources/Cloud Foundry Integration” and click **Next**.

7. On the **Review Licenses** window, click “I accept the terms of the license agreement” and click **Finish**. See [About the Plugin User Interface](#) below for a description of the plugin tabs and the information in each.

Install from a Local Build

If you need to install the Cloud Foundry Eclipse Plugin from a local build, rather than 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.

Step 1: Obtain the plugin source from GitHub.

- You can download archived source code for released versions of the plugin from <https://github.com/SpringSource/eclipse-integration-cloudfoundry/releases>.
- If you want the very latest source code for the plugin, clone the project repository:

```
git clone https://github.com/SpringSource/eclipse-integration-cloudfoundry
```

Step 2: Build repository.

Unzip the downloaded archive, change directory to the root directory and run this command in the shell:

```
mvn -Pe37 package
```

Transfer `org.cloudfoundry.ide.eclipse.server.site/target/site` to the machine where you want to install the plugin. If you zip it up, be sure to unzip prior to installation.

Step 3: Install from local repository.

On the machine running Eclipse or Spring Tools Suite:

1. In Eclipse (or STS), select **Install New Software** from the **Help** menu.
2. On the **Available Software** window, click **Add** to the right of the **Work with** field.
3. On the **Add Repository** window, enter a name for the repository, for example “Cloud Foundry Integration”, and click **Local**.
4. On the **Open** window, browse to `org.cloudfoundry.ide.eclipse.server.site/target/site` and click **Open**.
5. On the **Add Repository** window, click **OK**.
6. On the **Available Software** window, check “Core/Cloud Foundry Integration” and, optionally, “Resources/Cloud Foundry Integration” and click **Next**.
7. On the **Review Licenses** window, click “I accept the terms of the license agreement” and click **Finish**. See [About the Plugin User Interface](#) below for a description of the plugin tabs and the information in each.

About the Plugin User Interface

The sections below describe the Cloud Foundry Eclipse plugin user interface. If you do not see the tabs described below, select the Pivotal Cloud Foundry server in the **Servers** view. To expose the Servers view, ensure that you are using the Java perspective, then select **Window > Show View > Other > Server > Servers**.

The Cloud Foundry editor, outlined in red in the screenshot below, is the primary plugin user interface. Some workflows involve interacting with standard elements of the Eclipse user interface, such as the **Project Explorer** and the **Console** and **Servers** views.

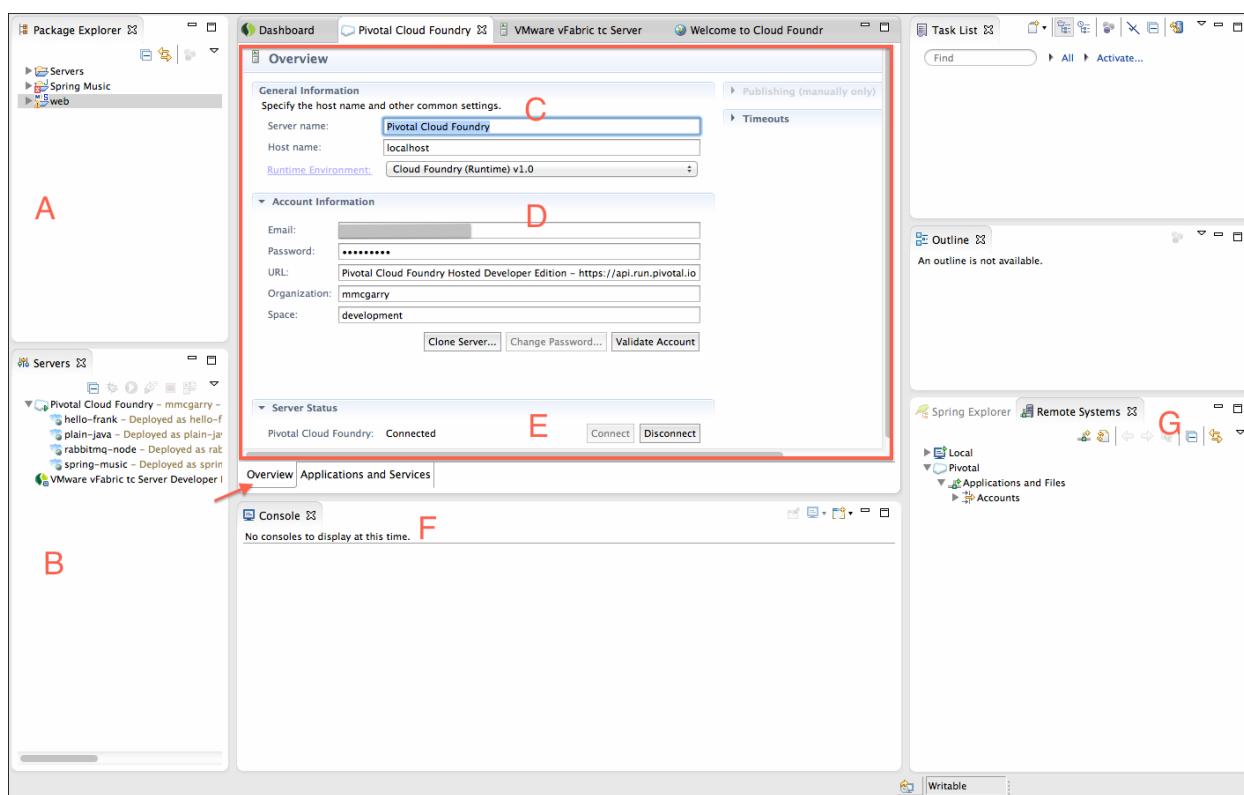
Note that the Cloud Foundry editor allows you to work with a single Cloud Foundry space; each space is represented by a distinct server instance in the **Servers** view (B). Multiple editors, each targeting a different space, can be open simultaneously. However, only one editor targeting a particular Cloud Foundry server instance can be open at a time.

Overview Tab

The follow panes and views are present when the **Overview** tab is selected:

- A — The **Package Explorer** view lists the projects in the current workspace.

- B – The **Servers** view lists server instances configured in the current workspace. A server of type “Pivotal Cloud Foundry” represents a targeted space in a Cloud Foundry instance.
- C – The **General Information** pane.
- D – The **Account Information** pane lists your Cloud Foundry credentials and the target organization and space. The pane includes these controls:
 - **Clone Server** — Use to create additional Pivotal Cloud Foundry server instances. You must configure a server instance for each Cloud Foundry space 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 the 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.

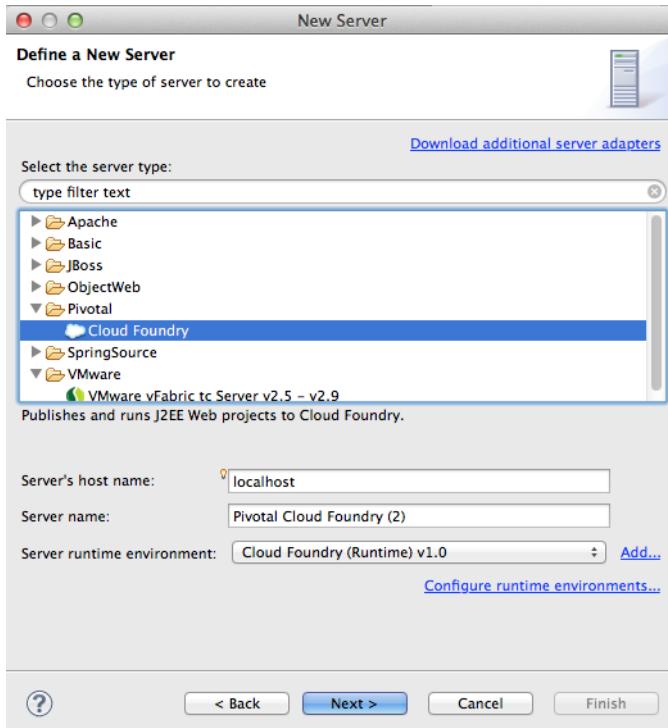
- K — The **Services** pane lists services that are bound to the application currently selected in the **Applications** pane. Note the icon in the upper right corner of the pane — it allows you to create a service, as described in [Create a Service](#).

Create a Cloud Foundry Server

This section has 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 unused and will be removed in a future version of the plugin.



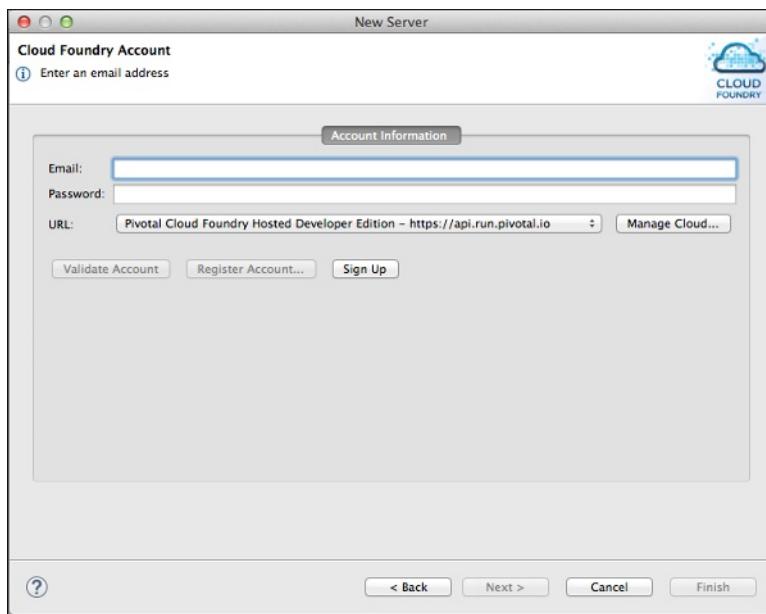
3. On the **Cloud Foundry Account** window, if you already have a Pivotal Cloud Foundry Hosted Developer Edition

account, enter your email account and password credentials and click **Validate Account**.

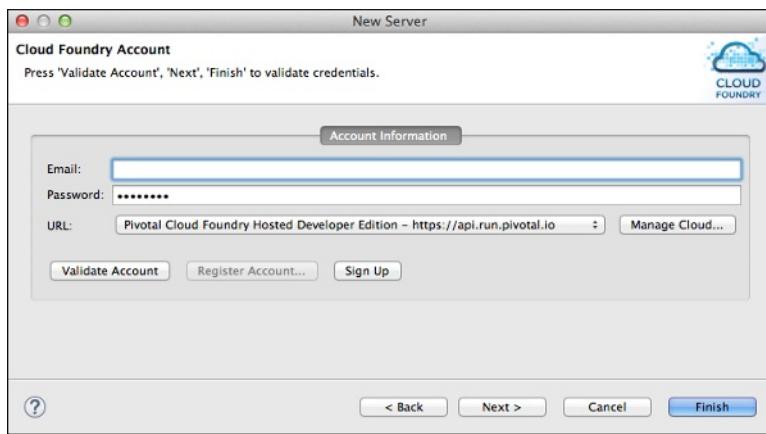
Note: By default, the **URL** field points to the Pivotal Cloud Foundry Hosted Developer Edition URL of <https://api.run.pivotal.io>. If you have a Pivotal Elastic Runtime account, refer to the [Logging into the Apps Manager](#) topic to determine your Pivotal Elastic Runtime URL. Click **Manage Cloud...** 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.

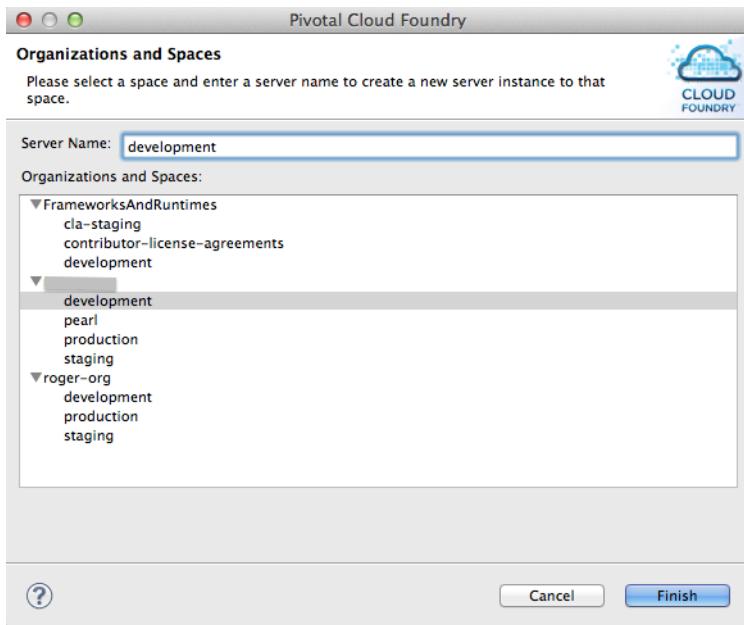


4. The **Cloud Foundry Account** window is refreshed and displays a message indicating whether or not your credentials were valid. Click **Next**.



5. On the **Organizations and Spaces** window, select the space 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.



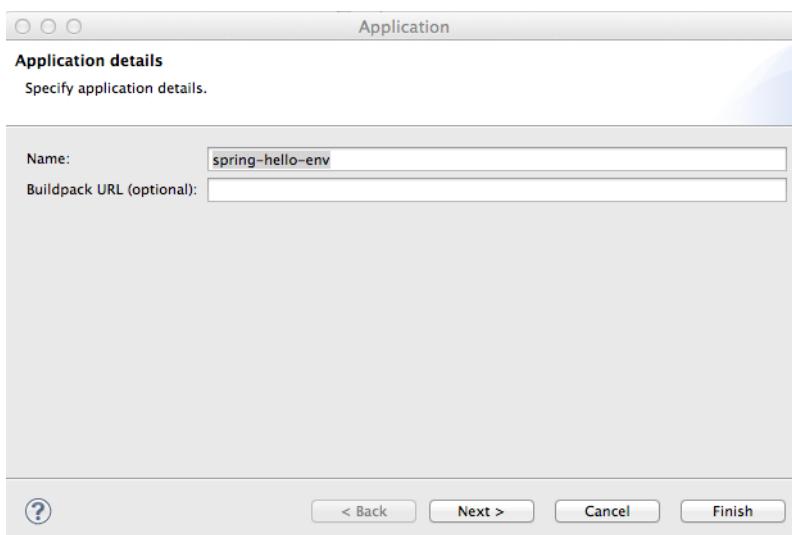
- Once you have successfully configured the Pivotal Cloud Foundry server, it will appear in the **Servers** view of the Eclipse or STS user interface. To familiarize yourself with the plugin user interface, see [About the Plugin User Interface](#). When you are ready, proceed to [Deploy an Application](#).

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.
- On the **Application Details** window:
 - By default, the *Name* field is populated with the application project name. You can enter a different name if desired — the name will be assigned to the deployed application, but will 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. On the **Launch Deployment** window:

Host — By default, contains the name of the application. You can enter a different value if desired. Note that if you push the same application to multiple spaces in the same organization, you must assign a unique **Host** to each.

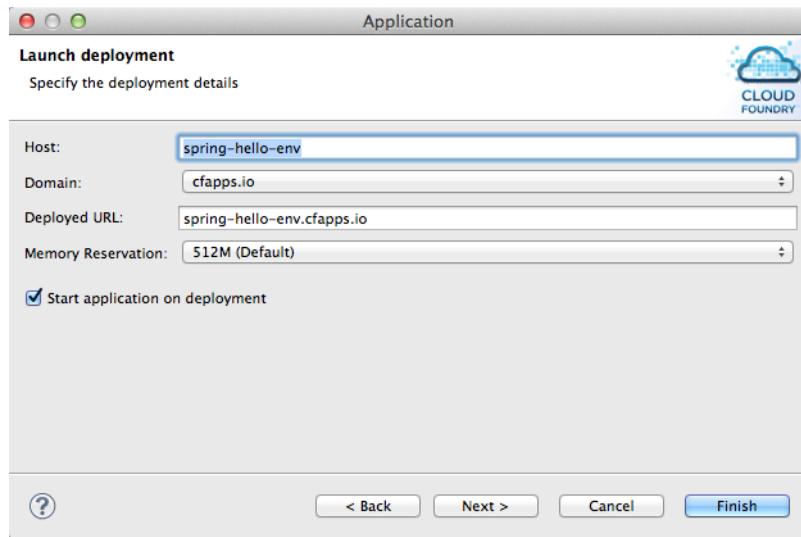
Domain — Contains the default domain. If you have mapped custom domains to the target space, they appear in the pull-down list.

Note: This version of the Cloud Foundry Eclipse plugin does not provide a mechanism for mapping a custom domain to a space. You must use the `cf map domain` command to do so.

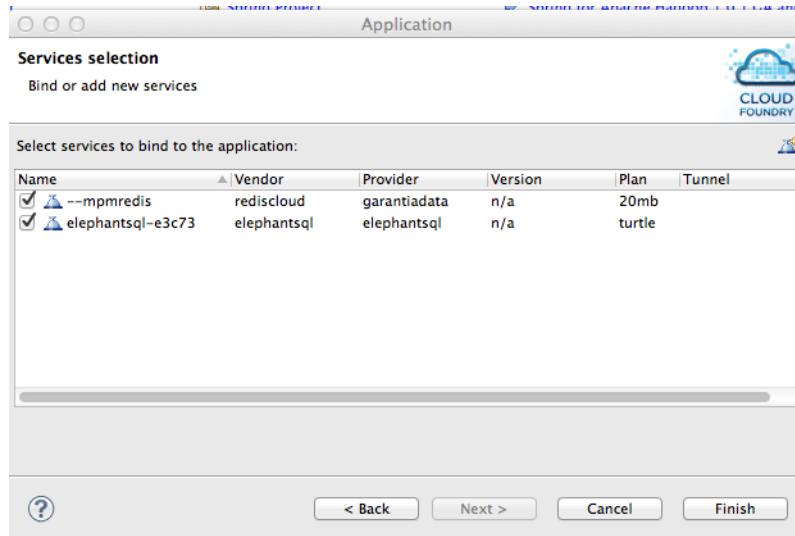
Deployed URL — By default, contains the value of the **Host** and **Domain** fields, separated by a period (.) character.

Memory Reservation — Select the amount of memory to allocate to the application from the pull-down list.

Start application on deployment — If you do not want the application to be started upon 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**. Note that you can bind services to the application after deployment, as described in [Bind and Unbind Services](#).



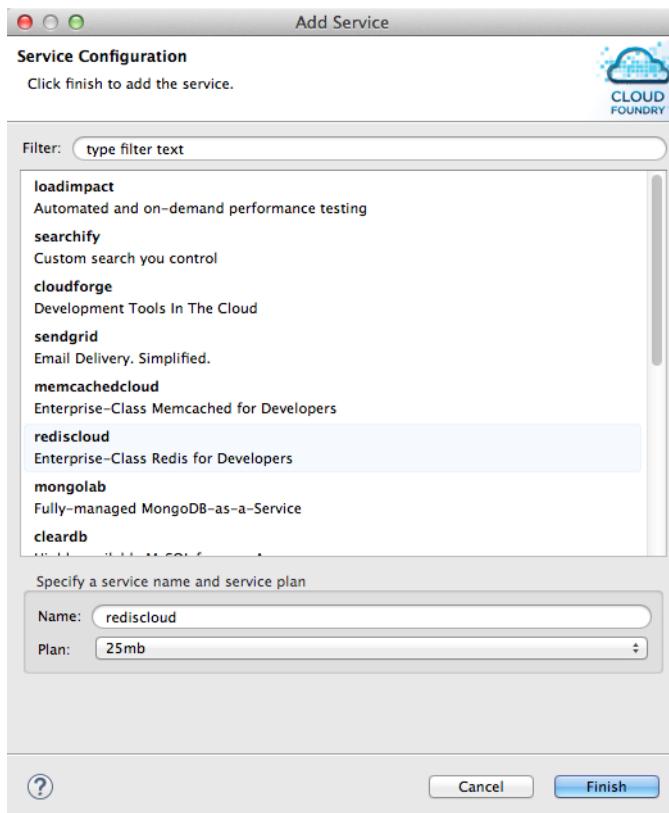
As the deployment proceeds, progress messages appear in the **Console** view. When deployment is complete, the application is listed in the **Applications** pane.

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

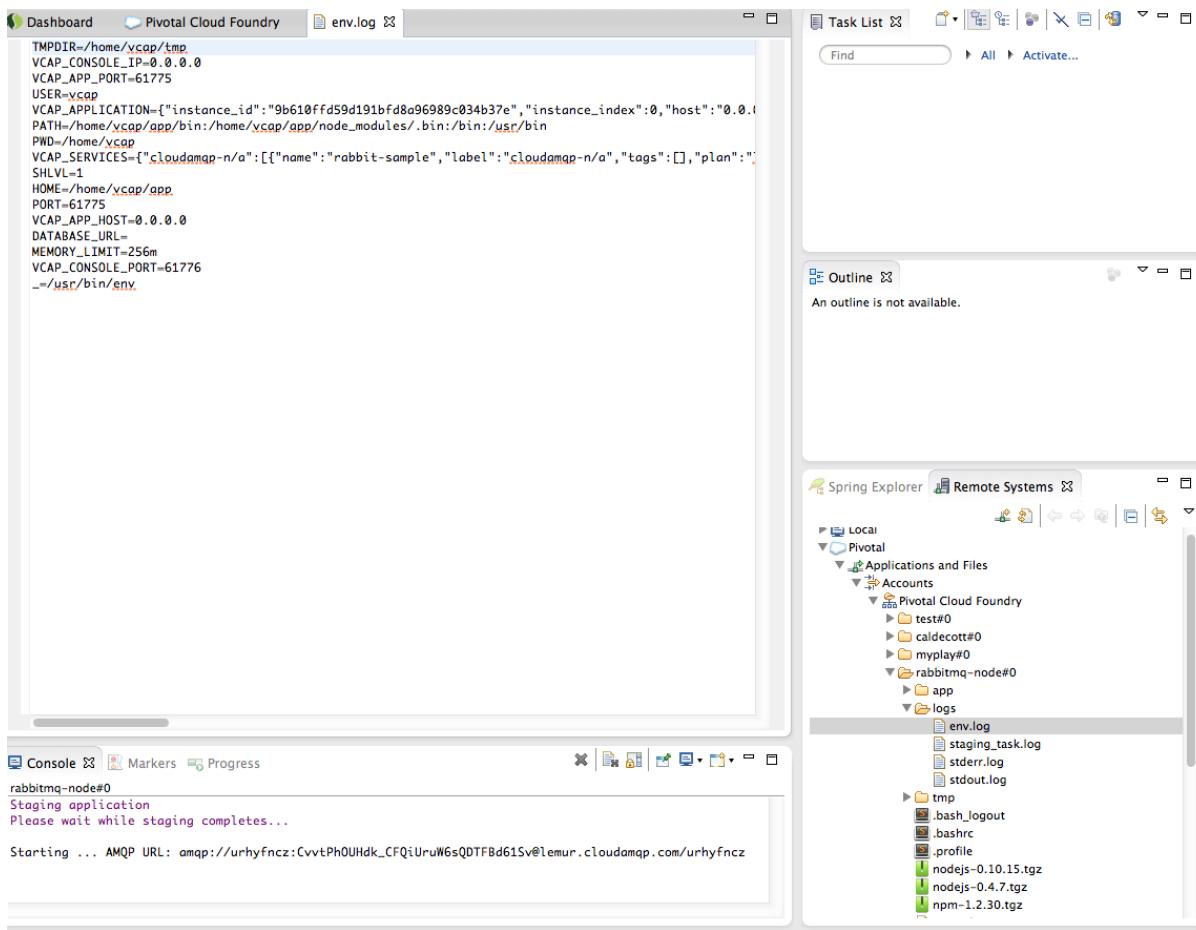
You can bind a service to an application when you deploy it. To bind a service to an application that is already deployed, drag the service from the **Services** pane to the **Application Services** pane. (See the area labelled "G" in the screenshot in the [Applications and Services](#) above.)

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. On 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](#).

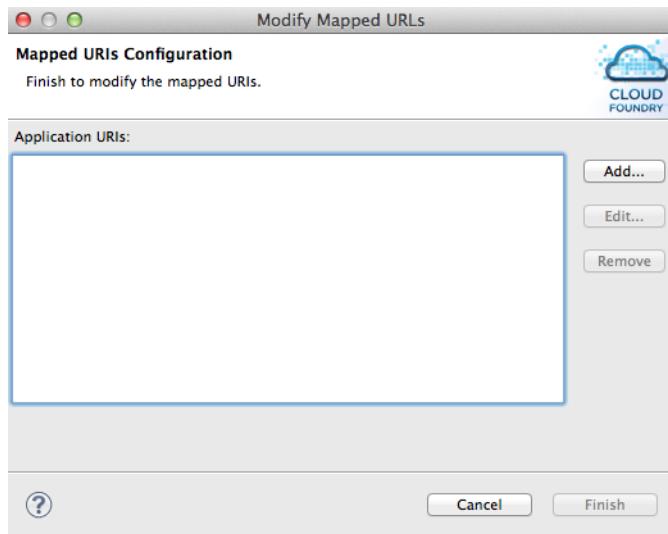
Push Application Changes

The Cloud Foundry editor supports these application operations:

1. **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.
2. **Restart** — When you **Restart** a deployed application, the plugin does not push any resources to the Cloud Foundry instance.
3. **Update and Restart** — When you run this command, the plugin pushes only the changes that were made to the application since last update, not the entire application. This is useful for performing incremental updates to large applications.

Manage Application URLs

You add, edit, and remove URLs mapped to the currently selected application in the **General** pane when the **Applications and Services** tab is selected. Click the pencil icon to display the **Mapped URIs Configuration** window.



Information in the Console View

When you start, restart, or update and restart an application, application output will generally be streamed to the **Console** view (labelled "F" in the screenshot in [Overview Tab](#)). The information shown in the **Console** view for a running application instance includes staging information, and the application's `std.out` and `std.error` logs.

If multiple instances of the application are running, only the output of the first instance appears in the **Console** view. To view the output of another running instance, or to refresh the output that is currently displayed:

1. In the **Applications and Services** tab, select the deployed application in the *Applications* pane.
2. Click the **Refresh** button 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.

To clone a server:

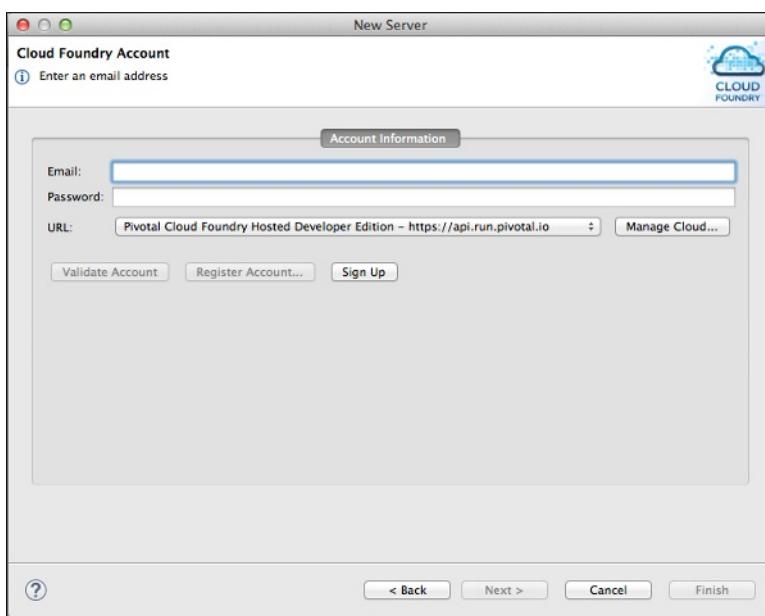
1. Either:
 - In the Cloud Foundry server instance editor "Overview" tab, click on the "Clone Server" button, or
 - Right-click a Cloud Foundry server instance in the **Servers** view, and select **Clone Server** from the context menu,
2. On the **Organizations and Spaces** window, select the space you want to target.
3. The name field will be filled with the name of the space you selected. If desired, edit the server name before clicking **Finish**.

Add a Cloud Foundry Instance URL

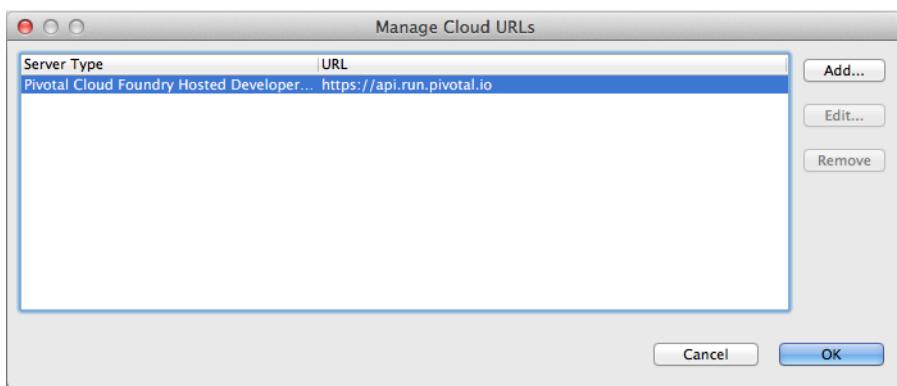
You can configure the plugin to work with any Cloud Foundry instances to which you have access. To do so:

1. Perform steps 1 and 2 of [Create a Cloud Foundry Server](#).

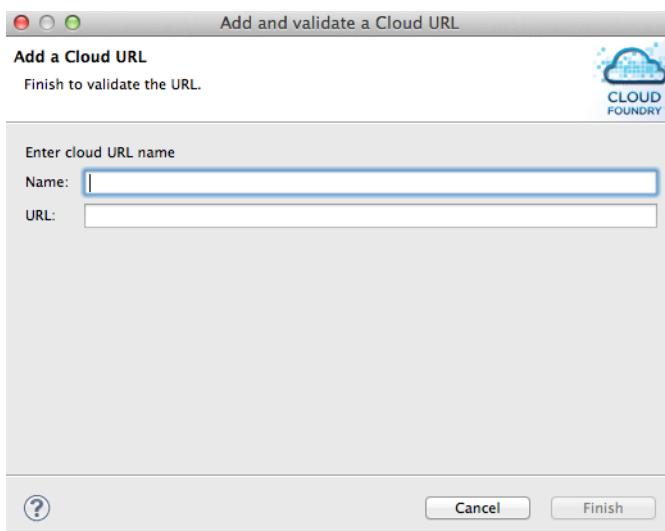
2. On the **Cloud Foundry Account** window, enter the email account and password you use to log on to the target instance, and then click **Manage Cloud URLs**



3. On the **Manage Cloud URLs** window, click **Add**.



4. On 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. On the **Cloud Foundry Account** window, click **Validate Account**.

7. Complete steps 4 through 6 of [Create a Cloud Foundry Server](#).

Cloud Foundry Java Client Library

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>1.0.2</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:1.0.2'
}
```

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.pivotl.io>) along with a valid user name and password as command-line parameters.

```
import org.cloudfoundry.client.lib.CloudCredentials;
import org.cloudfoundry.client.lib.CloudFoundryClient;
import org.cloudfoundry.client.lib.domain.CloudApplication;
import org.cloudfoundry.client.lib.domain.CloudService;
import org.cloudfoundry.client.lib.domain.CloudSpace;

import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;

public final class JavaSample {

    public static void main(String[] args) {
        String target = args[0];
        String user = args[1];
        String password = args[2];

        CloudCredentials credentials = new CloudCredentials(user, password);
        CloudFoundryClient client = new CloudFoundryClient(credentials, getTargetURL(target));
        client.login();

        System.out.printf("%nSpaces:%n");
        for (CloudSpace space : client.getSpaces()) {
            System.out.printf("  %s\t(%s)%n", space.getName(), space.getOrganization().getName());
        }

        System.out.printf("%nApplications:%n");
        for (CloudApplication application : client.getApplications()) {
            System.out.printf("  %s%n", application.getName());
        }

        System.out.printf("%nServices%n");
        for (CloudService service : client.getServices()) {
            System.out.printf("  %s\t(%s)%n", service.getName(), service.getLabel());
        }
    }

    private static URL getTargetURL(String target) {
        try {
            return URI.create(target).toURL();
        } catch (MalformedURLException e) {
            throw new RuntimeException("The target URL is not valid: " + e.getMessage());
        }
    }
}
```

For more details 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

This page assumes you are using cf CLI v6 and version 1.0.4 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>1.0.4</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>1.0.4</version>
    <configuration>
      <target>http://api.run.pivotal.io</target>
      <org>mycloudfoundry-org</org>
      <space>development</space>
      <appname>my-app</appname>
      <url>my-app.example.com</url>
      <memory>512</memory>
      <instances>2</instances>
      <env>
        <ENV-VAR-NAME>env-var-value</ENV-VAR-NAME>
      </env>
      <services>
        <service>
          <name>my-rabbitmq</name>
          <label>rabbitmq</label>
          <provider>rabbitmq</provider>
          <version>n/a</version>
          <plan>small_plan</plan>
        </service>
      </services>
    </configuration>
  </plugin>
</plugins>
```

After adding and configuring the plugin you can build and push the application to Cloud Foundry with the following command:

```
$ mvn clean package cf:push
```

Security Credentials

While you can include Cloud Foundry security credentials in the `pom.xml` file, a more secure method is to store the credentials in the Maven `settings.xml` file, using the server XML configuration element (<http://maven.apache.org/settings.html#Servers>). The default location for this configuration file is `~/.m2/settings.xml`.

To implement this:

1. Add a server to the servers section of the `settings.xml` file. Include the Cloud Foundry security credentials (username and password) and an `ID` tag. The `pom.xml` references this ID to access the security credentials.

```
<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>1.0.4</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:1.0.4'
        ...
    }
}

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.example.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

Node.js Buildpack

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](#)

The source for the buildpack is [here](#).

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](#).

Tips for Node.js Applications

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, 2013, Cloud Foundry uses 0.10.x as the default.

Example `package.json` file:

```
{  
  "name": "default-nodejs-app",  
  "version": "0.0.1",  
  "author": "Your Name",  
  "dependencies": {  
    "express": "3.4.8",  
    "consolidate": "0.10.0",  
    "express": "3.4.8",  
    "swig": "1.3.2",  
  },  
  "engines": {  
    "node": "0.10.x",  
    "npm": "1.3.x"  
  }  
}
```

Application Port

You need to use the `VCAP_APP_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.VCAP_APP_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 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/heroku-buildpack-nodejs.git  
... 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/heroku-buildpack-nodejs.git
```

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 <https://github.com/cloudfoundry/heroku-buildpack-nodejs>.

The table below lists:

- **Resource** — The software installed by the buildpack.
- **Available Versions** — The versions of each software resource that are available from the buildpack.
- **Installed by Default** — The version of each software resource that is installed by default.
- **To Install a Different Version** — How to change the buildpack to install a different version of a software resource.

This page was last updated on August 2, 2013.

Resource	Available Versions	Installed by Default	To Install a Different Version
Node.js	0.10.0 - 0.10.6	latest version of 0.10.x	To change the default version installed by the buildpack, see "hacking" on https://github.com/cloudfoundry/heroku-buildpack-nodejs .
	0.10.8 - 0.10.12		
	0.8.0 - 0.8.8		
	0.8.10 - 0.8.14		
	0.8.19		
	0.8.21 - 0.8.25		
	0.6.3		
	0.6.5 - 0.6.8		
	0.6.10 - 0.6.18		
	0.6.20		
	0.4.10		
	0.4.7		

Resource	Available Versions	Installed by Default	To Install a Different Version
npm	1.3.2		
	1.2.30		
	1.2.21 - 1.2.28		
	1.2.18		
	1.2.14 - 1.2.15		
	1.2.12		
	1.2.10		
	1.1.65		
	1.1.49	latest version of 1.2.x	as above
	1.1.40 - 1.1.41		
	1.1.39		
	1.1.35 - 1.1.36		
	1.1.32		
	1.1.9		
	1.1.4		
	1.1.1		
	1.0.10		

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 DEA](#) and those defined by the Node.js buildpack, 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:/bin:/usr/bin
```

Environment Variables Defined by the Node Buildpack

When you use the Node buildpack, you get three Node-specific environment variables in addition to the regular [Cloud Foundry environment variables](#).

- `BUILD_DIR` — The directory into which Node.js is copied each time a Node.js application is run.
- `CACHE_DIR` — The directory that Node.js uses for caching.
- `PATH` — The system path used by Node.js:
`PATH=/home/vcap/app/bin:/home/vcap/app/node_modules/.bin:/bin:/usr/bin`

Configure Service Connections for Node.js

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 Postgres, for example, you will need a `uri` to connect. You can assign the value of the `uri` to a variable as follows:

```
var uri = vcap_services.mypostgres[0].credentials.uri
```

Once assigned, you can use your credentials as you would normally in your program to connect to your database.

Connecting to a Service

You must include the appropriate package for the type of services your application uses. For example:

- Rabbit MQ via the [ampq](#) 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": "*",  
    "ampq": "*"  
  },  
  "engines": {  
    "node": "0.8.x"  
  }  
}
```

You must run `npm shrinkwrap` to regenerate your `npm-shrinkwrap.json` file after you edit `package.json`.

Ruby Buildpack

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

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 on 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 clouduser@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 25mb redis`. This creates a service instance named `redis` that uses the `rediscloud` service and the `25mb` plan, as the example below shows.

```
$ cf create-service rediscloud 25mb redis
Creating service redis in org Cloud-Apps / space development as clouduser@example.com....
OK
```

Bind a Service Instance

When you bind an app to a service instance, Elastic Runtime writes information about the service instance to the `VCAP_SERVICES` app environment variable. The app can use this information to integrate with the service instance.

Most services support bindable service instances. Refer to your service provider's documentation to confirm if they support this functionality.

You can bind a service to an application with the command `cf bind-service APPLICATION SERVICE_INSTANCE`.

Alternately, you can configure the deployment manifest file by adding a `services` block to the `applications` block and specifying the service instance. For more information and an example on service binding using a manifest, see the Sample App Step.

You can also bind a service using the [Apps Manager](#).

Sample App Step

You can skip this step. The manifest for the sample app contains a `services` sub-block in the `applications` block, as the example below shows. This binds the `redis` service instance that you created in the previous step.

```
services:
  - redis
```

Step 2: Configure Deployment Options

Configure the Deployment Manifest

You can specify app deployment options in a manifest that the `cf push` command uses. For more information about application manifests and supported attributes, refer to the [Deploying with Application Manifests](#) topic.

Configure a Production Server

Elastic Runtime uses the default standard Ruby web server library, WEBrick, for Ruby and RoR apps. However, Elastic Runtime can support a more robust production web server, such as Phusion Passenger, Puma, Thin, or Unicorn. If your app requires a more robust web server, refer to the [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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words.
- `cf help push` to view other options for this command.

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs APP_NAME`.

Once your app deploys, browse to your app URL. Search for the `urls` field in the `App started` block in the output of the `cf push` command. Use the URL to access your app online.

Sample App Step

Run `cf push pong_matcher_ruby -n HOST_NAME`.

Example: `cf push pong_matcher_ruby -n 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.example.com
      Binding pongmatch-ex12.example.com to pong_matcher_ruby...
OK

Uploading pong_matcher_ruby...
Uploading app files from: /Users/clouduser1/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

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 on 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 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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different HOST name for the app.
- `--random-route` to create a URL that includes the app name and random words.
- `cf help push` to view other options for this command.

To view log activity while the app deploys, launch a new terminal window and run `cf logs [APP-NAME]`.

In the terminal output of the `cf push` command, the `urls` field of the `App started` block contains the app URL. This is the `HOST_NAME` you specified with the `-n` flag plus the domain `example.com`. Once your app deploys, use this URL to access the app online.

Next Steps

You've deployed an app to Elastic Runtime! Consult the sections below for information about what to do next.

Test a Deployed App

Use the `cf` CLI or the [Apps Manager](#) to review information and administer your app and your Elastic Runtime account. For example, you could edit the `manifest.yml` to increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.

Manage Your Application with the cf CLI

Run `cf help` to view a complete list of commands, grouped by task categories, and run `cf help [COMMAND]` for detailed information about a specific command. For more information about using the `cf` CLI, refer to the 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 Application

This guide walks you through deploying a sample Ruby on Rails app to Elastic Runtime. Follow this guide to get started with Elastic Runtime and experience the deploy process from start to finish.

Step 1: Clone the App

Run `git clone https://github.com/cloudfoundry-samples/rails_sample_app` to clone the `rails_sample_app` from GitHub.

Step 2: Log in and Target the API Endpoint

Run `cf login -a [API-ENDPOINT]` to log in and target the 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 3: Create a Service Instance

Run `cf create-service elephantsql turtle rails-postgres`. This creates a service instance named `rails-postgres` that uses the `elephantsql` service and the `turtle` plan, as the example below shows.

```
$ cf create-service elephantsql turtle rails-postgres
Creating service rails-postgres in org Cloud-Apps / space development as clouduser@example.com....
OK
```

The manifest for the sample app contains a `services` sub-block in the `applications` block, as the example below shows. This binds the `rails-postgres` service instance you created in the previous step.

```
...
applications:
...
services:
- rails-postgres
```

Step 4: Deploy the App

From the root directory of your application, run `cf push rails_sample_app -n [HOST]` to deploy the app.

Example: `cf push rails_sample_app -n jc0825-rails`

`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 `example.com`. For example: `cf push my-app` creates the URL `my-app.example.com`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. Use the following options to help create a unique URL:

- `-n` to assign a different `HOST` name for the app.
- `--random-route` to create a URL that includes the app name and random words.
- `cf help push` to view other options for this command.

If you want to view log activity while the app deploys, launch a new terminal window and run `cf logs [APP-NAME]`.

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` you specified with the `-n` flag plus the domain `example.com`. Once your app deploys, use this URL to access the app online.

The example below shows the terminal output of 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 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.

```
$ cf push rails_sample_app -n jmt2014-rails
Using manifest file /Users/pivotal/workspace/rails_sample_app/manifest.yml

Updating app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...
OK

Using route jmt2014-rails.example.com
Uploading rails_sample_app...
Uploading app files from: /Users/pivotal/workspace/rails_sample_app
Uploading 445.7K, 217 files
OK
Binding service rails-postgres to app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com
OK

Starting app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...
OK

...
0 of 1 instances running, 1 starting
1 of 1 instances running

App started

Showing health and status for app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: jmt2014-rails.example.com

#0  state      since
     running   2014-08-25 03:32:10 PM
   cpu      memory      disk
     0.0%    68.4M of 256M  73.4M of 1G
```

Step 5: Verifying the app

You can verify the sample app you have pushed by browsing to the URL generated in the output of the previous step. In this example, the url is `urls: jmt2014-rails.example.com`.

You've now pushed an app to Elastic Runtime! For help deploying your own app, visit the **Deploy Your App: Guides by Framework** section of the help.

Configure a Production Server for Ruby Apps

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

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

This page assumes you are using cf CLI v6.

This page has information specific to deploying Rack, Rails, or Sinatra applications.

Application Bundling

You need to 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 need a `config.ru` file like this:

```
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 this 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 this 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 this 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`, with 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`.

Running Rails 3 Worker Tasks

Often when developing a Rails 3 application, you may want delay certain tasks so as not to consume resource that could be used for servicing requests from your user. This section shows you how to create and deploy an example Rails application that will make use of a worker library to defer a task, this task will then be executed by a separate application. The guide also shows how you can scale the resource available to the worker application.

Choosing a Worker Task Library

The first task is to decide which worker task library to use. Here is a summary of 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

Creating 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
```

```
<%= @things.inspect %>
```

Deploying Once, Deploying Twice

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 cf 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  
  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.

About the Ruby Buildpack

For information about using and extending the Ruby buildpack in Cloud Foundry, see <https://github.com/cloudfoundry/heroku-buildpack-ruby>.

The table below lists:

- **Resource** — The software installed by the Cloud Foundry Ruby buildpack, when appropriate.
- **Available Versions** — The versions of each software resource that are available from the buildpack.
- **Installed by Default** — The version of each software resource that is installed by default.
- **To Install a Different Version** — How to change the buildpack to install a different version of a software resource.

Resource	Available Versions	Installed by Default	To Install a Different Version
Ruby	1.8.7 patchlevel 374, Rubygems 1.8.24		
	1.9.2 patchlevel 320, Rubygems 1.3.7.1	The latest security patch release of 1.9.3	Specify desired version in application gem file.
	1.9.3 patchlevel 448, Rubygems 1.8.24		
	2.0.0 patchlevel 247, Rubygems 2.0.3 1.2.1		
Bundler	1.3.0.pre.5 1.3.2		Not supported.

This table was last updated on August 14, 2013.

Environment Variables

You can access environments variable programmatically.

For example, you can obtain `VCAP_SERVICES` like this:

```
ENV['VCAP_SERVICES']
```

Environment variables available to you include both those [defined by the DEA](#) and those defined by the Ruby buildpack, 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

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

This page assumes 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 of 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`.

Rails Applications Have Auto-Configured database.yml

During staging, the Ruby buildpack replaces your `database.yml` with one based on the `DATABASE_URL` variable.

 **Note:** The Ruby buildpack ignores the contents of any `database.yml` you provide and overwrites it during staging.

Configuring non-Rails Applications

Non-Rails applications can also see 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 Service Broker API](#), the hash key is the name of the service.
- For services that use the [v1 Service Broker API](#), the hash key is formed by combining the service provider and version, in the format PROVIDER-VERSION.

For example, given a service provider “p-mysql” with version “n/a”, the hash key is `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://lrra:e6B-X@p-mysqlprovider.example.com:5432/lraa"
        },
        "label": "p-mysql-n/a",
        "name": "p-mysql",
        "syslog_drain_url": "",
        "tags": ["postgres", "postgresql", "relational"]
      }
    ]
  }
}

User-Provided:
my-env-var: 100
my-drain: http://drain.example.com
```

View Logs

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-6]
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
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.TransactionF
1:27:16.96 [App/0]  OUT 16,963  INFO IninitiateTerminatorT1000Deployment: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` again.

```
$ gem update bundler
$ gem update --system
$ bundle install
```

Services

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 [Using Services](#).

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](#)
- [Binding Credentials](#)
- [Dashboard Single Sign-On](#)
- [Example Service Brokers](#)
- [Application Log Streaming](#)

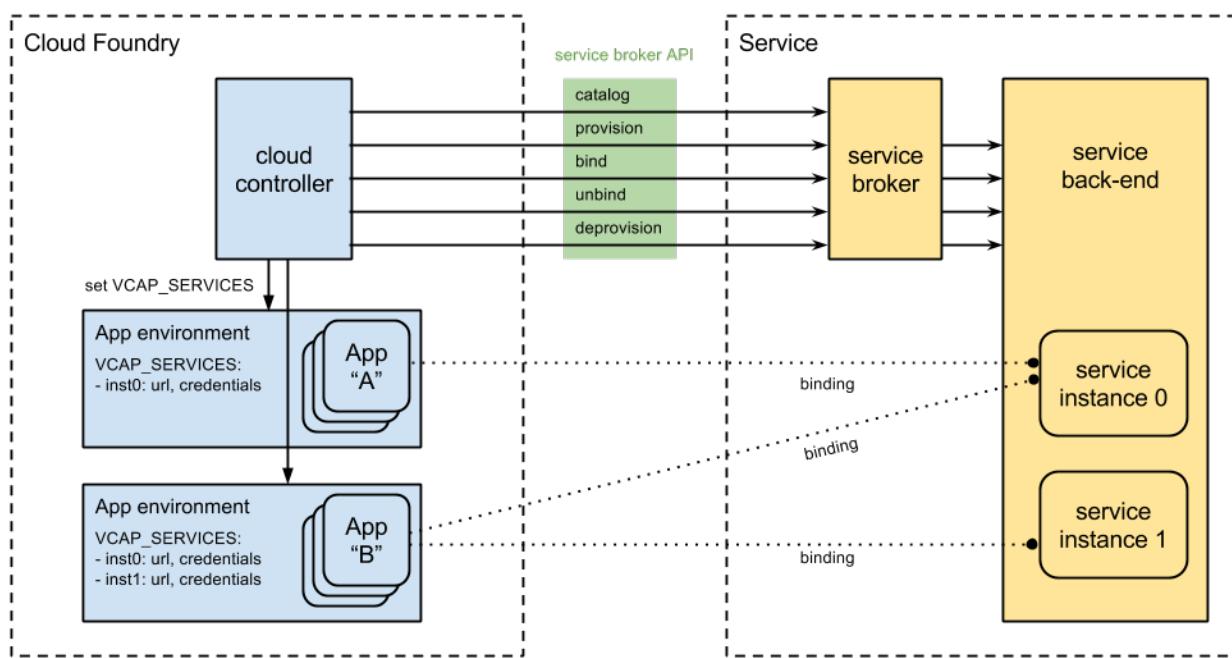
Overview

Architecture & Terminology

Services are integrated with Cloud Foundry by implementing a documented API for which the cloud controller is the client; we call this the Service Broker API. This should not be confused with the cloud controller API, often used to refer to the version of Cloud Foundry itself; when one refers to “Cloud Foundry v2” they are referring to the version of the cloud controller API. The services API is versioned independently of the cloud controller API.

Service Broker is the term we use to refer to a component of the service which implements the service broker API. This component was formerly referred to as a Service Gateway, however as traffic between applications and services does not flow through the broker we found the term gateway caused confusion. Although gateway still appears in old code, we use the term broker in conversation, in new code, and in documentation.

Service brokers advertise a catalog of service offerings and service plans, as well as interpreting calls for provision (create), bind, unbind, and deprovision (delete). What a broker does with each call can vary between services; in general, ‘provision’ reserves resources on a service and ‘bind’ delivers information to an application necessary for accessing the resource. We call the reserved resource a Service Instance. What a service instance represents can vary by service; it could be a single database on a multi-tenant server, a dedicated cluster, or even just an account on a web application.



Implementation & Deployment

How a service is implemented is up to the service provider/developer. Cloud Foundry only requires that the service provider implement the service broker API. A broker can be implemented as a separate application, or by adding the required http endpoints to an existing service.

Because Cloud Foundry only requires that a service implements the broker API in order to be available to Cloud Foundry end users, many deployment models are possible. The following are examples of valid deployment models.

- Entire service packaged and deployed by BOSH alongside Cloud Foundry
- Broker packaged and deployed by BOSH alongside Cloud Foundry, rest of the service deployed and maintained by other means
- Broker (and optionally service) pushed as an application to Cloud Foundry user space
- Entire service, including broker, deployed and maintained outside of Cloud Foundry by other means

Service Broker API

Document Changelog

[v2 API Change Log](#)

Versions

Two major versions of the Service Broker API are currently supported by Cloud Foundry, v1 and v2. As v1 is deprecated and support will be removed in the next major version of Cloud Foundry, it is recommended that all new brokers implement v2 and that current brokers are upgraded.

Current Version	Past Versions
2.4	2.3 2.2 2.1 2.0 v1 (unversioned)

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 v2.3

The key change from v2.3 to v2.4 is support for users to change the service plan for a specified service instance.

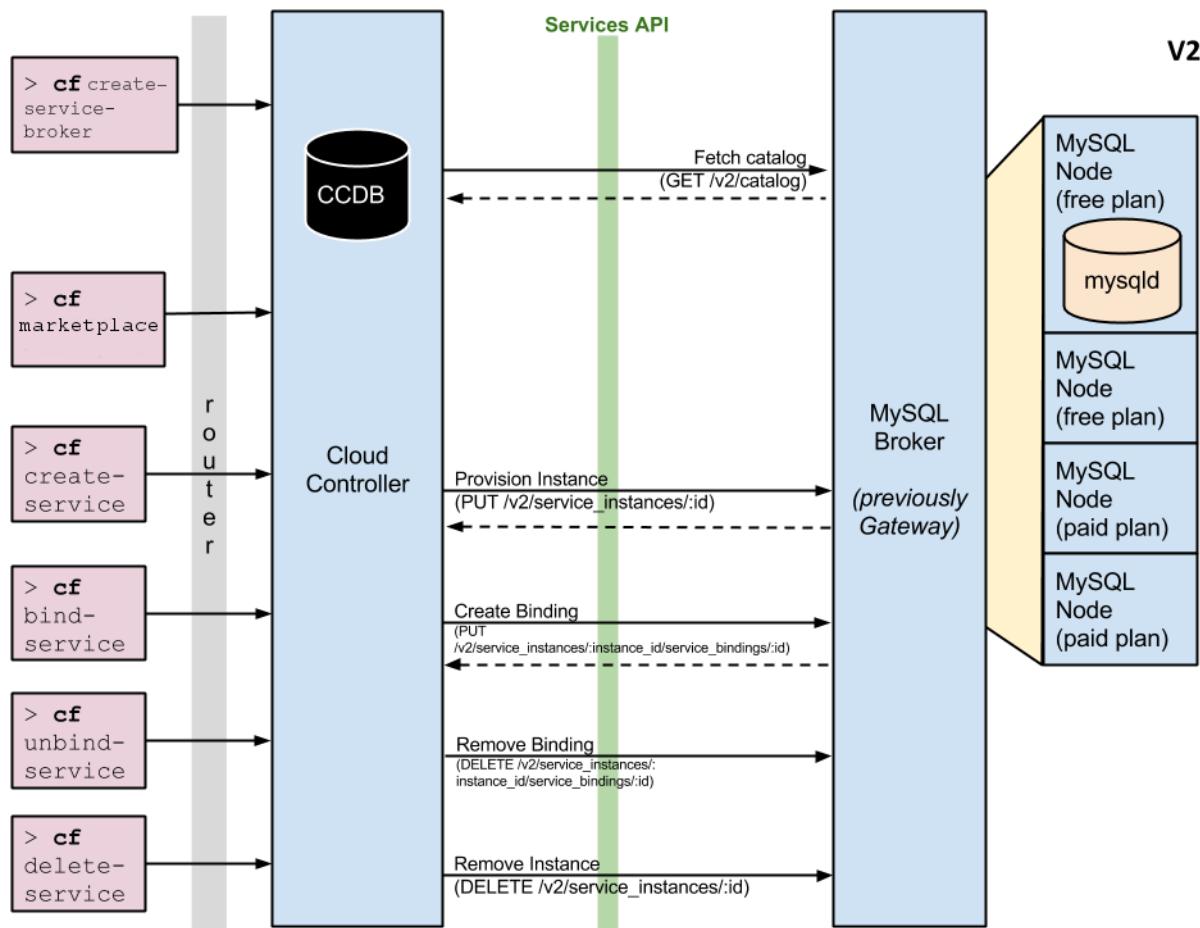
Dependencies

v2.4 of the services API has been supported since:

- [Final build 192](#) of [cf-release](#)
- v2.17 of the [Cloud Controller API](#)
- v6.7 of the Cloud Foundry [Command Line Interface \(CLI\)](#)

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 URL 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: 2.4
```

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: 2.4" 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 syslog_drain; for more info see [Application Log Streaming](app-log-streaming.html) .
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
id*	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.

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
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	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.
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.
space_guid*	string	Similar to organization_guid, but for the space.

```
{
  "service_id": "service-guid-here",
  "plan_id": "plan-guid-here",
  "organization_guid": "org-guid-here",
  "space_guid": "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",
  "organization_guid": "org-guid-here",
  "space_guid": "space-guid-here"
}' -X PUT -H "X-Broker-API-Version: 2.4" -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.
409 Conflict	Should be returned if the requested service instance already exists. The expected response body is "{}", though the description field can be used to return a user-facing error message, as described in Broker Errors .

Responses with any other status code will be interpreted as a failure. Brokers can include a user-facing message in the `description` field; for details see [Broker Errors](#).

Body

All response bodies must be a valid JSON Object ({}). This is for future compatibility; it will be easier to add fields in the future if JSON is expected rather than to support the cases when a JSON body may or may not be returned.

For success responses, the following fields are valid.

Response field	Type	Description
----------------	------	-------------

Response field	Type	Description
dashboard_url	string	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 ("9189kdsk0vfnku" in the example below). For information on how users can authenticate with service dashboards via SSO, see Dashboard Single Sign-On .

```
{
  "dashboard_url": "http://mongomgmthost/databases/9189kdsk0vfnku"
}
```

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
plan_id	string	ID of the new plan from the catalog.

```
{
  "plan_id": "plan-guid-here"
}
```

cURL

```
$ curl http://username:password@broker-url/v2/service_instances/:instance_id -d '{
  "plan_id": "plan-guid-here"
}' -X PATCH -H "X-Broker-API-Version: 2.4" -H "Content-Type: application/json"
```

Response

Status Code	Description
200 OK	New plan is effective. The expected response body is "{}".
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 .

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 derive their value just from being provisioned. Brokers that do not provide 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
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.
app_guid*	string	GUID of the application that you want to bind your service to.

```
{
  "plan_id":      "plan-guid-here",
  "service_id":    "service-guid-here",
  "app_guid":      "app-guid-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"
}' -X PUT -H "X-Broker-API-Version: 2.4" -H "Content-Type: application/json"
```

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.
409 Conflict	Should be returned if the requested binding already exists. The expected response body is “{}”, though the description field can be used to return a user-facing error message, as described in Broker Errors .

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 valid.

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 to for the bound application. 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.

Query-String Field	Type	Description
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: 2.4"
```

Response

Status Code	Description
200 OK	Binding was deleted. The expected response body is "{}"
410 Gone	Should be returned if the binding does not exist. The expected response body is "{}"

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.

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: 2.4"
```

Response

Status Code	Description
200 OK	Instance was deleted. The expected response body is "{}".

Status Code	Description
200 OK	Service instance was deleted. The expected response body is “{}”
410 Gone	Should be returned if the service instance does not exist. The expected response body is “{}”

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. If empty JSON 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
<code>description</code>	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, a broker may not return a response, the provision requests times out, but the broker eventually succeeds in provisioning an instance without Cloud Controller knowing about it. 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	

Status code	Result	Orphan mitigation
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

This page assumes you are using cf CLI v6.

In order to run any of the commands below, you must be authenticated with Cloud Foundry as an admin user.

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.

1. [Register a Broker](#)
2. [Make Plans Public](#)

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.

```
$ cf create-service-broker mybrokername someuser somethingsecure http://mybroker.example.com/
```

Make Plans Public

New service plans are private by default. To make plans available to end users, see [Make Plans Public](#). Instances of a private plan can not be provisioned until either the plan is made public or is made available to an organization.

Multiple Brokers, Services, Plans

Many service brokers may be added to a Cloud Foundry instance, each offering many services and plans. The following constraints should be kept in mind:

- It is not possible to have multiple brokers with the same name
- It is not possible to have multiple brokers with the same base URL
- The service ID and plan IDs of each service advertised by the broker must be unique across Cloud Foundry. GUIDs are recommended for these fields.

See [Possible Errors](#) below for error messages and what to do when you see them.

List Service Brokers

```
$ cf service-brokers
Getting service brokers as admin...Cloud Controller
OK

Name          URL
my-service-name http://mybroker.example.com
```

Update a Broker

Updating a broker is how to ingest changes a broker author has made into Cloud Foundry. Similar to adding a broker, update causes Cloud Controller to fetch the catalog from a broker, validate it, and update the Cloud Controller database with any changes found in the catalog.

Update also provides a means to change the basic auth credentials cloud controller uses to authenticate with a broker,

as well as the base URL of the broker's API endpoints.

```
$ cf update-service-broker mybrokername someuser somethingsecure http://mybroker.example.com/
```

Rename a Broker

A service broker can be renamed with the `rename-service-broker` command. This name is used only by the Cloud Foundry operator to identify brokers, and has no relation to configuration of the broker itself.

```
$ cf rename-service-broker mybrokername mynewbrokername
```

Remove a Broker

Removing a service broker will remove all services and plans in the broker's catalog from the Cloud Foundry Marketplace.

```
$ cf delete-service-broker mybrokername
```

Note: Attempting to remove a service broker will fail if there are service instances for any service plan in its catalog. When planning to shut down or delete a broker, make sure to remove all service instances first. Failure to do so will leave [orphaned service instances](#) in the Cloud Foundry database. If a service broker has been shut down without first deleting service instances, you can remove the instances with the CLI; see [Purge a Service](#).

Purge a Service

If a service broker has been shut down or removed without first deleting service instances from Cloud Foundry, you will be unable to remove the service broker or its services and plans from the Marketplace. In development environments, broker authors often destroy their broker deployments and need a way to clean up the Cloud Controller database.

The following command will delete a service offering, all of its plans, as well as all associated service instances and bindings from the Cloud Controller database, without making any API calls to a service broker. For services from v1 brokers, you must provide a provider with `-p PROVIDER`. Once all services for a broker have been purged, the broker can be removed normally.

```
$ cf purge-service-offering v1-test -p pivotal-software
Warning: This operation assumes that the service broker responsible for this
service offering is no longer available, and all service instances have been
deleted, leaving orphan records in Cloud Foundry's database. All knowledge of
the service will be removed from Cloud Foundry, including service instances and
service bindings. No attempt will be made to contact the service broker; running
this command without destroying the service broker will cause orphan service
instances. After running this command you may want to run either
delete-service-auth-token or delete-service-broker to complete the cleanup.

Really purge service offering v1-test from Cloud Foundry? y
OK
```

`purge-service-offering` requires cf-release v160 and edge of cf CLI 6.0.2.

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.al-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

By default, all new service plans are private. This means that when adding a new broker, or when adding a new plan to an existing broker's catalog, new service plans won't immediately be available to end users. This enables an admin to control which service plans are available to end users, and to manage limited availability.

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-riakcs
  service  plan      access    orgs
  p-riakcs  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": "1af5050-664e-4be2-9389-6bf0c967c0c6",
    "url": "/v2/service_plans/1af5050-664e-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/1af5050-664e-4be2-9389-6bf0c967c0c6/service_instances"
  }
}
```

In this example, the GUID of plan-name-1 is 1af5050-664e-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-4af2-1526-6bfacf61b111 -X 'PUT' -d '{"public":true}'

{
  "metadata": {
    "guid": "1113aa0-124e-4af2-1526-6bfacf61b111",
    "url": "/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111",
    "created_at": "2014-02-12T06:24:04+00:00",
    "updated_at": "2014-02-12T20:55:10+00:00"
  },
  "entity": {
    "name": "plan-name-1",
    "free": true,
    "description": "plan-desc-1",
    "service_guid": "d9011411-1463-477c-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-4af2-1526-6bfacf61b111/service_instances"
  }
}
```

Enable access to a private plan for a particular organization

Users have access to private plans that have been enabled for an organization only when targeting a space of that organization. See [Managing Services](#) for more information.

To make a service plan available to users of a specific organization, you need the GUID of both the organization and the service plan. To get the GUID of the service plan, run the same command described above for [enabling access to a plan for all organizations](#):

```
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-b56d0936cfdc",
    "url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc",
    "created_at": "2013-05-06T16:34:56+00:00",
    "updated_at": "2013-09-25T18:44:35+00:00"
  },
  "entity": {
    "name": "my-org",
    "billing_enabled": true,
    "quota_definition_guid": "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-b56d0936cfdc/spaces",
    "domains_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/domains",
    "private_domains_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/private_domains",
    "users_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/users",
    "managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/managers",
    "billing_managers_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/billing_managers",
    "auditors_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/auditors",
    "app_events_url": "/v2/organizations/c54bf317-d791-4d12-89f0-b56d0936cfdc/app_events"
  }
}
```

In this example, the GUID of my-org is c54bf317-d791-4d12-89f0-b56d0936cfdc.

To make a private plan available to a specific organization, run:

```
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-4af2-1526-6bfacf61b111","organization_guid":"aaaa1234-da91-4f12-8ffa-b51d0336aaaa","service_plan_url":"/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111","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-4af2-1526-6bfacf61b111 -X 'PUT' -d '{"public":false}'\n{\n  \"metadata\": {\n    \"guid\": \"1113aa0-124e-4af2-1526-6bfacf61b111\",\n    \"url\": \"/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111\",\n    \"created_at\": \"2014-02-12T06:24:04+00:00\",\n    \"updated_at\": \"2014-02-12T20:55:10+00:00\"\n  },\n  \"entity\": {\n    \"name\": \"plan-name-1\",\n    \"free\": true,\n    \"description\": \"plan-desc-1\",\n    \"service_guid\": \"d9011411-1463-477c-b223-82e04996b91f\",\n    \"extra\": \"{\\\"bullets\\\": [\\\"bullet1\\\",\\\"bullet2\\\"]}\",\n    \"unique_id\": \"plan-id-1\",\n    \"public\": false,\n    \"service_url\": \"/v2/services/d9011411-1463-477c-b223-82e04996b91f\",\n    \"service_instances_url\": \"/v2/service_plans/1113aa0-124e-4af2-1526-6bfacf61b111/service_instances\"\n  }\n}
```

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

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		
metadata.bullets	array-of-strings	Features of this plan, to be displayed in a bulleted-list	extra.bullets		X
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). Each object must provide the following keys:</p> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;"> amount: { usd: float }, unit: string </div> <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://d33na3ni6eqf5j.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",
                  "eur": "49.0"
                },
                "unit": "MONTHLY"
              },
              {
                "amount": {
                  "usd": "0.99",
                  "eur": "0.49"
                },
                "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,
                  "eur": 49.0
                },
                "unit": "MONTHLY"
              },
              {
                "amount": {
                  "usd": 0.99,
                  "eur": 0.49
                },
                "unit": "1GB of messages over 20GB"
              }
            ],
            "displayName": "Big Bunny"
          },
          "unique_id": "addonOffering_1889",
          "public": true
        }
      }
    ]
  }
}
```

Binding Credentials

If your service is bindable, it means that in response to the bind API call, you will return credentials which can be consumed by an application. Cloud Foundry writes these credentials to the `environment variable` `VCAP_SERVICES`. In some cases buildpacks will write a subset of these credentials to other environment variables that frameworks may expect.

Please choose from the following list of credential fields if possible. You can provide additional fields as needed, but if any of these fields meet your needs you should use them. This convention allows developers to provide buildpacks and libraries which either parse `VCAP_SERVICES` and deliver useful objects to applications, or which actually configure applications themselves with a service connection.

Important: If you provide a service which supports a connection string, you should provide at least the `uri` key; as mentioned you may also provide discrete credential fields. Buildpacks and application libraries use the `uri` key.

Credentials	Description
uri	Connection string of the form <code>dbtype://username:password@hostname:port/name</code> , where <code>dbtype</code> is mysql, postgres, mongodb, amqp, etc.
hostname	The FQDN of the server host
port	The port of the server host
name	Name of the service instance; database name
vhost	Name of the messaging server virtual host (replacement for <code>name</code> specific to AMQP providers)
username	Server user
password	Server password

Here is an example output of `ENV['VCAP_SERVICES']`. Note that ClearDB chooses to return both discrete credentials, a uri, as well as another field. CloudAMQP chooses to return just the uri, and RedisCloud returns only discrete 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:IwN6dCdZmeQD4O0ZPKpu1Y0aLx1he8wo@lemur.cloudamqp.com/ksvyjmiv"
      }
    }
  ],
  {
    name: "cloudamqp-9dbc6",
    label: "cloudamqp",
    plan: "lemur",
    credentials: {
      uri: "amqp://vhuklnxa:9INFxpTujsAdTts98vQIdKHW3MojyMyV@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"
  },
],
}
```

Dashboard Single Sign-On

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: 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:

- `id` is the unique identifier for the OAuth2 client that will be created for your service dashboard on the token server (UAA), and will be used by your dashboard to authenticate with the token server (UAA).
- `secret` is the shared secret your dashboard will use to authenticate with the token server (UAA).
- `redirect_uri` is used by the token server as an additional security precaution. UAA will not provide a token if the callback URL declared by the service dashboard doesn't match the domain name in `redirect_uri`. The token server matches on the domain name, so any paths will also match; e.g. a service dashboard requesting a token and declaring a callback URL of `http://p-mysql.example.com/manage/auth` would be approved if `redirect_uri` for its client is `http://p-mysql.example.com/`.

2. When a service broker which advertises the `dashboard_client` property for any of its services is [added or updated](#),

Cloud Controller will create or update UAA clients as necessary. This client will be used by the service dashboard to authenticate users.

Dashboard URL

A service broker should return a URL for the `dashboard_url` field in response to a [provision request](#). Cloud Controller clients should expose this URL to users. `dashboard_url` can be found in the response from Cloud Controller to create a service instance, enumerate service instances, space summary, and other endpoints.

Users can then navigate to the service dashboard at the URL provided by `dashboard_url`, initiating the OAuth2 login flow.

Service Dashboard Responsibilities

OAuth2 Flow

When a user navigates to the URL from `dashboard_url`, the service dashboard should initiate the OAuth2 login flow. A summary of the flow can be found in [section 1.2 of the OAuth2 RFC](#). OAuth2 expects the presence of an [Authorization Endpoint](#) and a [Token Endpoint](#). In Cloud Foundry, these endpoints are provided by the UAA. Clients can discover the location of UAA from Cloud Controller's info endpoint; in the response the location can be found in the `token_endpoint` field.

```
$ curl api.example-cf.com/info
{"name":"vcap","build":"2222","support":"http://support.cloudfoundry.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}
```

More specifically, a service dashboard should implement the OAuth2 Authorization Code Grant type ([UAA docs](#), [RFC docs](#)).

1. When a user visits the service dashboard at the value of `dashboard_url`, the dashboard should redirect the user's browser to the Authorization Endpoint and include its `client_id`, a `redirect_uri` (callback URL with domain matching the value of `dashboard_client.redirect_uri`), and list of requested scopes. Scopes are permissions included in the token a dashboard client will receive from UAA, and which Cloud Controller uses to enforce access. A client should request the minimum scopes it requires. The minimum scopes required for this workflow are `cloud_controller_service_permissions.read` and `openid`. For an explanation of the scopes available to dashboard clients, see [On Scopes](#).
2. UAA authenticates the user by redirecting the user to the Login Server, where the user then approves or denies the scopes requested by the service dashboard. The user is presented with human readable descriptions for permissions representing each scope. After authentication, the user's browser is redirected back to the Authorization endpoint on UAA with an authentication cookie for the UAA.
3. Assuming the user grants access, UAA redirects the user's browser back to the value of `redirect_uri` the dashboard provided in its request to the Authorization Endpoint. The `Location` header in the response includes an authorization code.

```
HTTP/1.1 302 Found
Location: https://p-mysql.example.com/manage/auth?code=F45jH
```

4. The dashboard UI should then request an access token from the Token Endpoint by including the authorization code received in the previous step. When making the request the dashboard must authenticate with UAA by passing the client `id` and `secret` in a basic auth header. UAA will verify that the client id matches the client it issued the code to. The dashboard should also include the `redirect_uri` used to obtain the authorization code for verification.
5. UAA authenticates the dashboard client, validates the authorization code, and ensures that the redirect URI received matches the URI used to redirect the client when the authorization code was 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' \
-H 'Authorization: bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoid' \
http://api.cloudfoundry.com/v2/service_instances/44b26033-1f54-4087-b7bc-da9652c2a539/permissions
```

Response:

```
{  
  "manage": true  
}
```

The response will indicate to the service whether this user is allowed to manage the given instance. A `true` value for the `manage` key indicates sufficient permissions; `false` would indicate insufficient permissions. Since administrators may change the permissions of users, the service should check this endpoint whenever a user uses the SSO flow to access the service's UI.

On Scopes

Scopes let you specify exactly what type of access you need. Scopes limit access for OAuth tokens. They do not grant any additional permission beyond that which the user already has.

Minimum Scopes

The following two scopes are necessary to implement the integration. Most dashboard shouldn't need more permissions than these scopes enabled.

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 [UAU OmniAuth Strategy](#). See this [OmniAuth wiki page](#) for instructions on how to create your own strategy.

The UAU OmniAuth strategy is used to first get an authorization code, as documented in [this section](#) of the UAU 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 UAU 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 UAU as seen in the [Auth Controller](#).

Restrictions

- UAU 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

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.

Application Log Streaming

By binding an application to an instance of an applicable service, Cloud Foundry will stream logs for the bound application to the service instance.

- Logs for all apps bound to a log-consuming service instance will be streamed to that instance
- Logs for an app bound to multiple log-consuming service instances will be streamed to all instances

To enable this functionality, a service broker must implement the following:

1. In the [catalog](#) endpoint, the broker must include `requires: syslog_drain`. This minor security measure validates that a service returning a `syslog_drain_url` in response to the [bind](#) operation has also declared that it expects log streaming. If the broker does not include `requires: syslog_drain`, and the bind request returns a value for `syslog_drain_url`, Cloud Foundry will return an error for the bind operation.
2. In response to a [bind](#) request, the broker should return a value for `syslog_drain_url`. The syslog URL has a scheme of syslog, syslog-tls, or https and can include a port number. For example:
`"syslog_drain_url": "syslog://logs.example.com:1234"`

How does it work?

- Broker returns a value for `syslog_drain_url` in response to bind
- When application is restarted, `VCAP_SERVICES` is updated with the key and value for `syslog_drain_url`
- DEAs continuously stream application logs to the Loggregator
- If `syslog_drain_url` is present in `VCAP_SERVICES`, the DEA tags the logs with this field
- Loggregator streams logs tagged with this key to the location specified as its value

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](#).

Packaging Pivotal Cloud Foundry Products

This document is intended for product teams to learn how to package, distribute, and upgrade [Pivotal Cloud Foundry](#) (PCF) products across releases.

Important: Packaging custom products is an alpha feature supported through a Pivotal Professional Services engagement or as part of an Enterprise License Agreement.

Assumptions

This document might be difficult to follow unless you have a fully functional BOSH release of your product including a working manifest, release, and stemcell. You should also have a working knowledge of Pivotal Cloud Foundry Operations Manager and know, for example, what terms such as “resources page” or “product tile” refer to.

What is Ops Manager?

Pivotal Ops Manager is visual interface for installing and upgrading distributed software in the cloud. It is a vSphere Virtual Appliance that launches a Ruby on Rails application and orchestrates Pivotal’s BOSH platform to manage virtual machines on vSphere. As of this writing, it can install Cloud Foundry, MySQL, Rabbit MQ, Pivotal HD, and any other product that has been packaged as a BOSH release.

How should a product be packaged?

The Ops Manager team refers to a packaged product as a “product zip” because it is archived using .zip format. The extension, however, should be .pivotal, to prevent users from extracting the .zip. Be sure to create and distribute an archive that uses the final release files of your product, and not development files.

The contents of the archive are the top-level directories listed below.

Note: Creating a product zip from folders as subdirectories of other folders is not supported.

Product Package - Releases Directory

The releases directory contains your product’s release file. This is created using the BOSH command `bosh create release --with-tarball`. Multiple releases can be specified.

Product Package - Product Templates

The metadata directory contains a YAML (.yml) file, named whatever you wish, with the product template that Ops Manager needs to prompt the user and do an install. This product template references your release and stemcell, creates the user input forms, and generates a BOSH manifest when a user clicks **Apply Changes** or **Install** in Ops Manager.

To understand how to author product templates / metadata, let’s look at an example and explain the elements of interest.

```
---
name: p-redis
product_version: 1.0.0.6
metadata_version: "1.4"
stemcell_criteria:
  os: ubuntu-trusty
  requires_cpi: false
  version: "2858"
releases:
  - name: redis
    file: redis-0.2-dev.tgz
    version: 0.2-dev
  - name: redis-errands-release
    file: redis-errands-release-0.16-dev.tgz
    version: 0.16-dev
```

```

label: redis
description: Redis is a key-value store that ...
icon_image: iVBORw0KGgoAAAANSUhEUgAAAG # [5]
rank: 1
post_deploy_errands:
- name: test-redis # [6]

form_types:
- name: Redis
  label: Redis
  description: This is my little Redis
  property_inputs: # [8]
  - reference: redis.password

- name: redis_snapshotting_collection
  label: Snapshotting
  description: Create a snapshotting policy
  property_inputs:
  - reference: .snapshots
    label: Snapshots
    description: Enter the seconds and number of changes for each snapshot
    property_inputs:
    - reference: seconds
      label: Seconds
      description: Number of seconds before a save occurs
    - reference: changes
      label: Changes
      description: Number of changes to save in the elapsed seconds

property_blueprints: # [9a]
- name: snapshots
  type: collection
  configurable: true
  optional: false
  property_blueprints:
  - name: seconds
    type: integer
  - name: changes
    type: integer

job_types: # [10]
- name: redis
  resource_label: redis label for resource page
job_templates:
- redis
release: redis

resource_definitions: # [11]
- name: ram
  type: integer
  configurable: true
  default: 1024

- name: ephemeral_disk
  type: integer
  configurable: true
  default: 2048

- name: persistent_disk
  type: integer
  configurable: true
  default: 3072

- name: cpu
  type: integer
  configurable: true # [12]
  default: 1

static_ip: 0 # [13]
dynamic_ip: 1 # [13]
max_in_flight: 1 # [14]
serial: false # [15]

instance_definitions:
- name: instances
  type: integer
  configurable: true
  default: 1

property_blueprints: # [16]
- name: vm_credentials
  type: salted_credentials
  default:
    identity: vcap
  - name: password

```

```

type: secret
configurable: true
optional: true
# [17]

manifest: |
  redis.password: (( password.value ))
  redis.snapshots: (( .properties.snapshots.value ))

- name: test-redis
  resource_label: Test Redis
  job_templates:
    - test-redis
  release: redis-errands-release
  errand: true
# [18]
  resource_definitions:
    - name: ram
      type: integer
      configurable: true
      default: 1024
      constraints:
        min: 1024
    - name: ephemeral_disk
      type: integer
      configurable: true
      default: 1024
      constraints:
        min: 1024
    - name: persistent_disk
      type: integer
      configurable: false
      default: 0
    - name: cpu
      type: integer
      configurable: true
      default: 1
      constraints:
        min: 1
  static_ip: 0
  dynamic_ip: 1
  max_in_flight: 1
  instance_definitions:
    - name: instances
      type: integer
      configurable: false
      default: 1
  property_blueprints:
    - name: vm_credentials
      type: salted_credentials
      default:
        identity: vcap

- name: compilation
  resource_label: compilation

  resource_definitions:
    - name: ram
      type: integer
      configurable: true
      default: 1_024
    - name: ephemeral_disk
      type: integer
      configurable: true
      default: 2_048
    - name: persistent_disk
      type: integer
      configurable: true
      default: 8_192
    - name: cpu
      type: integer
      configurable: true
      default: 1

  static_ip: 0
  dynamic_ip: 1
  max_in_flight: 1

  instance_definitions:
    - name: instances
      type: integer
      default: 1

```

1. `name` and `product_version`: These uniquely identify a product. Ops Manager allows “upgrades” between products with the same `name` in the metadata, and different `product_version`s.
2. `metadata_version`: Every release of Ops manager expects a certain metadata schema. We are currently at 1.4.
3. `stemcell_criteria`: refers to the desired stemcell.
4. `releases`: An array of releases. BOSH Errand releases can be referred to here.
5. `icon_image`: A base64 encoded version of the logo for your image.
6. `post_deploy_errands`: An array of BOSH errands that run after a successful deployment. The name of the errand must correspond with the job.
7. `form_types`: Each form that is displayed for user input is generated from a form type.
8. `property_inputs`: An HTML input for a form is a reference to a `property_blueprint` on a `job_type` (see below).
9. Collections. A collection is a data structure containing multiple property inputs, a bit like a database table. When rendered as HTML the collection will display add, edit, and delete indicators. A user can create many records in a collection.

Note that the `reference: .snapshots` key-value pair in the `property_inputs` section of the `redis_snapshotting_collection` references [9a], a global `property_blueprints` section. This `property_blueprints` section describes the global `.snapshots` collection property, and not the `property_blueprints` on a `job_type`.

10. `job_types`: A list of the jobs that correspond with the job names in your BOSH manifest.
11. `resource_definitions`: These are the sections that create the sizing table in Ops Manager’s resource page. Note: CPU must be a power of 2.
12. `configurable`: Must be true for user to edit, otherwise it is disabled in the table or form.
13. `static_ip` and `dynamic_ip`: Corresponds to the BOSH setting for a job’s IP allocation. If `static_ip` is 0, `dynamic_ip` must be 1 and vice versa. You should use `dynamic_ip` when possible.
14. `max_in_flight`: Can be either a number or a ratio marked by a '%' percentage symbol.
15. `serial`: `true / false`. If `false`, this job will be deployed in parallel with other jobs.
16. `property_blueprints`: There are two kinds of blueprints. Property blueprints, their settings, types and defaults are further explored in the Types section.
17. `optional`: `true / false`. `true` means the value can be nil or blank. Defaults to `false` if this key is not specified.
18. `manifest`: Properties that you need to put in your BOSH manifest, underneath the appropriate job, are created here. This is how the `form_type` value is passed to the manifest (see example to the left). For more information on passing variables as property values, refer to the [Manifest section](#).
19. `errand`: Optional. `true / false`. Set to `true` to run this job as an errand. This must correspond to the `post_deploy_errand` listed in #7 above. For more information about errands, see [Understanding Lifecycle Errands](#).

More on Product Templates - Property Blueprints

Specifying the type of a property blueprint determines how the field shows up in the UI. It also determines what validation gets done when the user submits a form. Certain types have special features, e.g. you can specify min/max constraints for integer types, and you can have the various credential types be auto-generated.

For a type to show up in the UI, it must be configurable.

For each type, one or many accessors are defined below. It will be clear how to use these types when we look at manifest snippets.

string

- Accessor: value

- Rendered Content: text input

integer

Allows you to additionally specify constraints as follows:

```
name: my-prop
type: integer
label: Foo
constraints:
  min: 200
  max: 300
  power_of_two: true
```

You do not need to specify constraints, and if you do, you can specify either min, or max, or both. The `power_of_two` constraint should be used for the cpu resource definition for products deployed on VMware vSphere or vCloud Air / vCloud.

- Accessor: value
- Rendered Content: text input

boolean

- True/False value
- Accessor: value
- Rendered Content: checkbox

dropdown

```
- name: favorite_color
  type: dropdown_select
  label: Favorite Color
  configurable: true
  default: red
  options:
    - name: red
      label: Red
    - name: navy_blue
      label: Navy Blue
```

- Accessor: value
- Rendered Content: dropdown list of the options

domain

- Accessor: value
- Rendered Content: text input

wildcard_domain

For use with a WildcardDomainVerifier.

- Accessor: value
- Rendered Content: text input

uuid

- Accessor: value
- Rendered Content: Not available
- Generates: UUID

string_list

Takes a comma-delimited list of strings.

- Accessor: value
- Rendered Content: text input

 **Note:** The value is an array of non-empty strings from the input.

text

- Accessor: value
- Rendered Content: textarea

selector

Allows for different sets of property blueprints and manifest templates based on user selection.

```
property_blueprints:
  - name: selector_example
    type: selector
    configurable: true
    default: Not Configured
    option_templates:
      - name: not_configured
        select_value: Not Configured
        named_manifests:
          - name: my_snippet
            manifest: |
              is_this_configured: no
      - name: custom_configuration
        select_value: External DB
        named_manifests:
          - name: my_snippet
            manifest: |
              is_this_configured: yes
              configured_property: (( .properties.selector_example.custom_configuration.your_name.value ))
    property_blueprints:
      - name: your_name
        type: string
        configurable: true
```

- Accessors:
 - value: The `select_value` for the selected option
 - selected_option: Allows you to retrieve a parsed manifest snippet for the selected option
 - NAMED_OPTION: Provides access to the property blueprints for a named option
- Rendered Content: Radio buttons that show the nested property blueprints for the selected option.

Accessors example:

```
job_types:
  - name: example_selector_usage
    manifest: |
      custom_configured_name: (( .properties.selector_example.custom_configuration.your_name.value ))
      selected_option_manifest: (( .properties.selector_example.selected_option.parsed_manifest(my_snippet) ))
```

ldap_url

Valid LDAP URI.

- Accessor: value
- Rendered Content: text input

collection

A collection of entries. Each entry has all of the nested properties.

```
property_blueprints:
  - name: collection_example
    type: collection
    configurable: true
    optional: false
    property_blueprints:
      - name: first_nested_property
        type: string
      - name: second_nested_property
        type: string
```

- Accessor: value
- Rendered Content: An add button that provides a form to enter nested property values. A list component that allows editing and removing existing entries.

email

Validates that it is an email address.

- Accessor: value
- Rendered Content: text input

http_url

Full URL with HTTP or HTTPS protocol.

- Accessors: value
- Rendered Content: text input

ip_address

Validates a single IPv4 address.

- Accessor: value
- Rendered Content: text input

ip_ranges

Comma-separated list of single IPv4 addresses and/or IPv4 ranges of the form '1.2.3.4-1.2.3.200.'

- Accessor: value, `parsed_ip_ranges` (array)
- Rendered Content: text input

multi_select_options

When giving a property definition this type, you must also specify a list of options as follows:

```
name: my-prop
type: multi_select_options
label: Foo
options:
  - name: checkbox1
    label: Checkbox 1
  - name: checkbox2
    label: Checkbox 2
  - name: checkbox3
    label: Checkbox 3
```

- Accessor: value (array of selected option names)
- Rendered Content: checkbox(es)

network_address_list

Comma-separated list of network addresses.

- Accessor: value, parsed_network_addresses (array)
- Rendered Content: text input

network_address

IP address, domain, or single host-name. Renders as a text field.

- Accessor: value
- Rendered Content: text input

port

Integer between 0 and 65535.

- Accessor: value
- Rendered Content: text input

rsa_cert_credentials

A triple of private key, cert, and csr.

Accessors: public_key_pem, cert_and_private_key_pems
Rendered Content: textarea fields for the private key and cert
Generates: valid RSA private key, CSR, self-signed cert. All in PEM format.

rsa_pkey_credentials

RSA private key.

- Accessors: `public_key_pem`, `private_key_pem`
- Rendered Content: Not available
- Generates: valid RSA private key

salted_credentials

A triple of identity, password, and salt.

- Accessors: `sha512_hashed_password`, identity, salt, password
- Rendered Content: Not available
- Generates: identity, salt, password

simple_credentials

A pair of identity and password.

- Accessors: identity, password
- Rendered Content: text input for identity and password input for password
- Generates: identity, password

secret

- Accessor: secret
- Rendered Content: password input
- Generates: secret

smtp_authentication

Authentication type for SMTP.

- Accessor: value

- Rendered Content: dropdown where you can select from 'plain', 'login', and 'cram_md5'

Ops Manager-provided accessors

These accessors are available within the manifest for a job when you use double parentheses.

Example usage:

```
job_types:
  - name: first_job
    manifest: |
      my_first_ip: (( first_ip ))
      other_job_ip: (( .second_job.first_ip ))
  - name: second_job
```

Available accessors

`availability_zone` : The availability zone of the job instance

`ephemeral_disk` : The size of the ephemeral disk attached to the job's VM, given in megabytes

`first_ip` : The first ip assigned to the job

`instances` : The number of job instances

`ips` : An array of ips that the job has been assigned to

`ips_by_availability_zone` : Provides a hash. The keys are the identifiers for the different availability zones. The value is an array of ips assigned to the job for that availability zone.

`name` : Name of the job

`persistent_disk` : The size of the persistent disk attached to the job's VM, given in megabytes

`ram` : The RAM assigned to the job in megabytes

VM Credentials

Note: The `vm_credentials` property is required to be present on a job property `blueprints` array. These are the credentials given to the created job VM.

```
job_types:
  - name: my_job
    property_blueprints:
      - name: vm_credentials
        type: salted_credentials
        label: VM credentials
        configurable: false
        default:
          identity: vcap
```

More on Product Templates - Manifest

When a user clicks the **Install** or **Apply Changes** buttons in Ops Manager, a BOSH deployment manifest is generated and a BOSH deploy begins.

A BOSH manifest includes a number of properties for each job. These come from the `job_types` in your product template, but the values the user entered are evaluated from the accessors listed in the table above. You can also create properties using the `manifest` section of your product template, which is most common for things that are neither entered by the user nor auto-generated.

To refer to things that were either entered by a user, or auto generated, you can use a special reference syntax, which the Ops Manager team refers to as "spiff syntax." Note that this is not the same as the spiff command line tool many of

the other teams are using to generate manifests, it simply has a syntax that looks vaguely like spiff syntax.

Note: You can use Ruby's ERB syntax, but requires you to know something about the evaluation context and Ops Manager's internal object graph. Use of ERB is not recommended.

Double parentheses allow you to reference properties and job attributes.

```
job_types:
  - name: other_job
    property_blueprints:
      - name: creds
        type: simple_credentials
  - name: foo
    property_blueprints:
      - name: bar
        type: simple_credentials
        configurable: true
      - name: baz
        type: integer
        configurable: true
  manifest: |
    something:
      something_else: (( bar.identity )) # this accesses the identity access on the "bar"
      # property on this job (namely "foo")
      blah: (( bar.password ))
      hi: (( baz.value ))
  networkstuff:
    my_ip: (( first_ip ))           # this accesses the first IP allocated to this job
    bobs_ip: (( .other_job.first_ip )) # the leading . means to go up a level and find
      # a different job in this metadata, namely
      # "other_job", and then ask it for things such as
      # first_ip or an accessor on some property
    bobs_password: (( .other_job.creds.password ))
  stuff_i_need_from_other_deployment:
    runtime_uaa_ip: (( ..cf.uaa.first_ip )) # two leading .. means to go up two levels and
      # find a different product, namely "cf"
```

More about Product Templates - Dependency Management

If your product relies on other products, or you have specific jobs that rely on other products, you need to specify this in your product template. By creating a dependency, you will be warning users that they must meet prerequisites before installing. If the dependency is at a job level, and the user sets the number of instances to '0', there will be no dependency enforced.

Service brokers are a great example of why dependencies could be set at the job level. Service brokers have no purpose other than being used with Elastic Runtime, and they are tied to specific releases of the Cloud Controller. By setting dependency in the broker, and setting the number of instances to zero, a user could use the service independently without having to worry about it functioning with Elastic Runtime.

You should specify `provides_product_version` in your product template, even though it appears somewhat redundant as `name` and `product_version` capture the same information. Additionally, you may use `requires_product_versions` to depend on something else. This can be specified at the top-level of a product template, implying the entire product depends on another one; or it can be specified at an individual `job_type` level.

Here's how this looks in Elastic Runtime's product template:

```
---
name: cf
product_version: 1.0.0.1
metadata_version: "1.1"
stemcell_criteria:
  os: ubuntu-trusty
  version: "2858"
  requires_cpi: false
releases:
  - name: cf
    version: "147.20-dev"
    file: cf-147.20-dev.tgz
    md5: 6b36ae7d613e5857447d789717f598ce
provides_product_versions:
  - name: cf
    version: 1.0.0.1
requires_product_versions:
  - name: microbosh
    version: "~> 1.0"
```

Here's how to create dependencies by job:

```
job_types:
  - name: database-server
    label: My DB Server
    resource_label: My DB Server
    description: Multi-tenant DB server
    job_templates:
      - db-server
    requires_product_versions:
      - name: cf
        version: "~> 1.0"
```

Product Package - content_migrations directory

The content migrations directory contains YAML files that migrate installation data for products they previously installed. As of this writing we are within the 1.x.x.x release cycle, and thus all previously released product installations should be able to be migrated to the next. If you change the product template structure, like add, edit, or delete a `job_type`, or `property_blueprint`, you will need to create a `content_migration` so that the customer's installation can migrate to these new settings.

How are Upgrades Packaged?

When your team has a new release, you must create a content migration. These migrations are YAML files with certain rules for updating the installation hash discussed in the previous section. This transformation uses a Ruby gem called Transmogrifier. The migration file must be placed in the `content_migrations` directory in the product zip.

Migrating the Product Version

The first and simplest value that must be migrated is the product version. Say, for example, that your last release is 1.0.0.3 and you are going to release version 2.0.0.0 next week. Your last version's product template looks like this:

```
---
name: p-redis
product_version: 1.0.0.3
metadata_version: "1.1"
```

A content_migration YAML to migrate this version to the new 2.0.0.0 release would look like this:

```
product: redis
installation_version: "1.1"
to_version: "2.0.0.0"

migrations:
  - from_version: 1.0.0.3
    rules:
      - type: update
        selector: "product_version"
        to: "2.0.0.0"
```

Note: `installation_version` is the schema version of the `installation.yml` file, which contains an installation's state. This is not the same as the metadata version, which refers to the schema of the product template that you are authoring.

If your customers have a couple of Redis versions released you can migrate more than one version with the same migration as follows:

```
product: redis
installation_version: "1.1"
to_version: "2.0.0.0"

migrations:
  - from_version: 1.0.0.3
    rules:
      - type: update
        selector: "product_version"
        to: "2.0.0.0"
  - from_version: 1.0.0.4
    rules:
      - type: update
        selector: "product_version"
        to: "2.0.0.0"
```

Migrating Property Names and Types

The example above, in which you modified a version number, uses the update type of migration. You can also use create and delete.

Example 1. Adding a New Property

Imagine that your 2.0.0.0 Redis release has a new property called `max_memory`. In this case, all you need to do is add this property to your product template so that Ops Manager will display it and set it in any new installation BOSH manifests. You do not need a `content_migration` since the new installation file will be created when you click **Apply Changes**.

```
property_blueprints:
...
  - name: max_memory
    type: string
    label: max memory
    description: Enter the Max Memory for the Redis Cache
    configurable: true
    default: "2mb"

  manifest: |
    redis.password: (( credentials.password ))
    redis.max_memory: (( max_memory.value ))
```

Example 2. Updating a Property Names

Redis uses admin passwords, without user names. Our original product template has a property called `credentials` that uses a `simple_credentials` type. Unfortunately there was no `simple_password` type which just prompts for password (there will be soon). To refresh your memory:

```
property_blueprints:
...
  - name: credentials
    type: simple_credentials
    label: credentials
    description: it is a secret
    configurable: true

  manifest: |
    redis.password: (( credentials.password ))
```

Imagine the new type called `simple_password` was already available in Ops Manager. What would the steps be to migrate our password to this type? First, we would need to modify our product template to the different type, give it a new name, and reference it by that name. Our new product template would look like this:

```

property_blueprints:
...
- name: redis_password # NEW NAME (was called 'credentials')
  type: simple_password # NEW TYPE!
  label: redis password
  description: it is a secret
  configurable: true

manifest:
  redis.password: (( redis_password.password )) # NEW REFERENCE!

```

Next, we would look at our installation file and see if there are things to migrate:

```

- definition: credentials # Hmmm.. this uses the old name!
  value:
    identity: matt # We don't need this, Redis never used it anyhow...
    password: reider

```

Our Migration needs to change the name of property definition from credentials to `redis_password` it would also be cleaner to delete the identity property as well.

Our complete migration file is as follows:

```

...
product: redis
installation_version: "1.1"
to_version: "2.0.0.0"

migrations:
- product_version: 1.0.0.3
  rules:
    - type: update
      selector: "product_version"
      to: "2.0.0.0"
    - type: update
      selector: jobs.[type=redis].properties.[definition=credentials]
      to: redis_password
    - type: delete
      selector: jobs.[type=redis].properties.[definition=credentials].value.identity

```

Verifiers

Product templates include two types of verifiers: form verifiers and install time verifiers. Most verifiers are built to check availability of resources and IP addresses or existence of network endpoints. If your product requires some kind of verifier, let the team know so we can build it. Verifiers are not shown in the product template sections earlier in this document.

Form Verifiers

Form verifiers run when a user saves a form. You name the verifier in the `form_type` section, under the name of the form. The following verifier checks that Elastic Runtime's router IP is free or available:

```

- name: router
  label: Router IPs
  description: "Enter the IP address(es) for the Cloud Foundry Router."
  verifier:
    name: Verifiers::StaticIpsVerifier
  property_inputs:
    - reference: router.static_ips

```

Install Time Verifier Example

Install time verifiers run when a user clicks the **Apply Changes** or **Install** buttons. Usually you will run the same verifiers at install or form save, but they are broken out in case there are exceptions.

Here is an excerpt from the Elastic Runtime product template that verifies that the SSO Appliance, and SMTP servers, are available at the addresses the user entered:

```
install_time_verifiers:
  - name: Verifiers::SsoUrlVerifier
    properties:
      url: saml_login.sso_url
  - name: Verifiers::SmtpAuthenticationVerifier
    properties:
      credentials: consoledb.smtp_credentials
      address: consoledb.smtp_address
      port: consoledb.smtp_port
      helo_domain: consoledb.smtp_helo_domain
      authentication: consoledb.smtp_authentication
      enable_starttls_auto: consoledb.smtp_enable_starttls_auto
```

Verifier Definitions

Parameters are key-pair values where the value is property reference whose type matches that of the parameter.

```
Name: Verifiers::LDAPBindVerifier
Parameters:
- url: ldap_url
- credentials: simple_credentials

Name: Verifiers::SmtpAuthenticationVerifier
Parameters:
- credentials: simple_credentials
- address: network_address
- port: integer
- helo_domain: domain
- authentication: smtp_authentication
- enable_starttls_auto: boolean

Name: Verifiers::SsoUrlVerifier
Parameters:
- url: http_url

Name: Verifiers::WildcardDomainVerifier
Parameters:
- domain: wildcard_domain

Name: Verifiers::StaticIpsVerifier
Parameters: none

Name: Verifiers::MysqlDatabaseVerifier
Parameters:
- host: network_address or string
- port: integer
- username: string
- password: string or secret
- database: string

Name: Verifiers::BlobstoreVerifier
Parameters:
- access_key_id: string
- secret_access_key: string or secret
- bucket_name: string
```

Tips and Tricks for Metadata Authors

Authoring product templates currently lacks tooling and is not for the faint of heart. If you find that your product template leads to 500 errors in the Ops Manager, you should start by validating that it can be parsed.

Checking your YAML

```
$ gem install psych
$ irb
> require 'psych'
> contents = File.read('p-hd-tempест.yml')
> Psych.load(contents)
```

If you get an error, such as “Psych::SyntaxError: mapping values are not allowed in this context at line 24” it means there is likely a problem with spacing, or formatting, at line 23 or 24 of your file.

SSHing into Ops Manager / Shortening the Feedback Cycle

Product Teams have reported that the feedback cycle for product template changes is long. Changing your product template, uploading an entire product to Ops Manager, and watching it fail, can be frustrating. We are working to make this easier.

One way to shorten the feedback cycle is possible by SSHing into the Ops Manager VM (username = tempest, password is whatever you set when you deployed the .ova). Once in the Ops Manager VM, you can edit product templates directly at `/var/tempest/meta-data`.

After editing metadata, you must restart the Ops Manager web application.

```
$ sudo service tempest-web stop  
$ sudo service tempest-web start
```

Restoring Previous Installation File

There is an undocumented feature in Ops Manager to restore the previous version of the installation file's state, including all Stemcells and Releases that were part of that installation file. This should not be confused with restoring the installation (VMs) itself. To restore an installation's state browse to <https://{your ops manager}/restore>.

Starting Fresh

If your installation file seems corrupted for some reason, and you want to start over again without installing a new .ova, you can go into vCenter, kill any deployed VMs, and then clean all of the files from the directory `/var/tempest/workspace`.

Using the Ops Manager API

Using CURL, it is possible to automate most of the actions possible via the Ops Manager web interface. To view the Ops Manager API documentation, visit <https://{ops Manager}/docs>.

Warning: The Ops Manager API is an experimental feature that is not fully implemented and could change without notice. Pivotal is developing an officially supported Ops Manager API that will replace many of these endpoints in a subsequent release.

Automating Ops Manager deployment

You can automate the deployment of an Ops Manager .ova to vSphere or vCloud Air / vCloud without using vCenter.

```
$ git clone git@github.com:pivotal-cf/installation.git  
$ cd installation/gems/vsphere_clients  
$ gem install vsphere_clients-*gem  
$ cd ../ova_manager  
$ gem install ova_manager-*gem
```

Once you have completed these steps you can write a script to both deploy an .ova as follows:

```
require 'ova_manager'
require 'vsphere_clients'

OvaManager::Deployer.new({
  host: "172.16.74.3",
  user: "root",
  password: "vmware"
},{{
  datacenter: "mozzarella-dc",
  cluster: "mozzarella-cl",
  datastore: "mozzarella-ds",
  network: "VM Network",
  folder: "Matt",
  resource_pool: ""
}}).deploy("/Users/mreider/Downloads/z.ova", {
  ip: "172.16.74.150",
  netmask: "255.255.254.0",
  gateway: "172.16.74.1",
  dns: "10.80.130.1",
  ntp_servers: "10.80.130.1",
  vm_password: "admin"
})
```

The script to destroy an .ova looks like this:

```
require 'ova_manager'
require 'vsphere_clients'

OvaManager::Destroyer.new("mozzarella-dc", {
  host: "172.16.74.3",
  user: "root",
  password: "vmware"
}).clean_folder("Matt")
```

Changes from 1.3 to 1.4

You can import your existing 1.3 product into a Ops Manager 1.4. To take advantage of 1.4 features, you must update the following pieces of your metadata.

The following changes were made between version 1.3 released in November 2014 and 1.4 released in February 2015:

metadata_version

The value should be 1.4

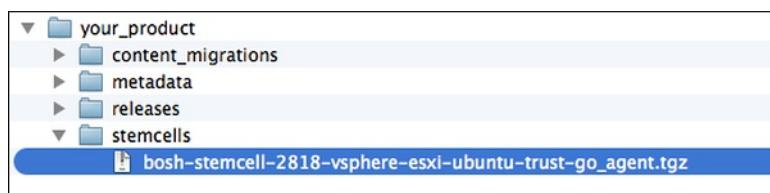
icon_image

- Old key: `image`
- New key: `icon_image`

(128x128) format instead of the old image (163x110). The new icon should be centered with the product “label” text displayed centered underneath.

stemcell

You should remove your stemcell in order for your product to be IaaS-agnostic. Remove your stemcell from the “stemcells” subdirectory within your product directory as shown in the image below:



Your product template should refer to stemcells by only version and OS, as the following example shows:

```
stemcell_criteria:
  os: ubuntu-trusty
  version: '2858'
  requires_cpi: false
```

If you import or update a 1.3 product to Amazon Web Services (AWS), note the following:

- After importing, click **Import Stemcell** in Ops Manager to upload the referenced stemcell version, "xen-hvm," for AWS.
- If you update, your upload will contain unnecessary stemcell files that take up space.

job_templates

The 'template' key under individual jobs has been renamed to 'job_templates'. Furthermore, it only supports an array as a value.

compiled_packages

Since compiled packages are a product of an infrastructure-specific stemcell and a release, compiled packages are no longer supported.

New Ops Manager accessors

```
ips_by_availability_zone
```

Provides a hash. The keys are the identifiers for the different availability zones. The value is an array of ips assigned to the job for that availability zone.

```
job_types:
  - name: first_job
  manifest: |
    my_ips_by_az: (( ips_by_availability_zone ))
```

Will be expanded to:

```
job_types:
  - name: first_job
  manifest: |
    my_ips_by_az:
      zone1-guid: [192.168.1.100, 192.168.1.101]
      zone2-guid: [192.168.2.100]
```

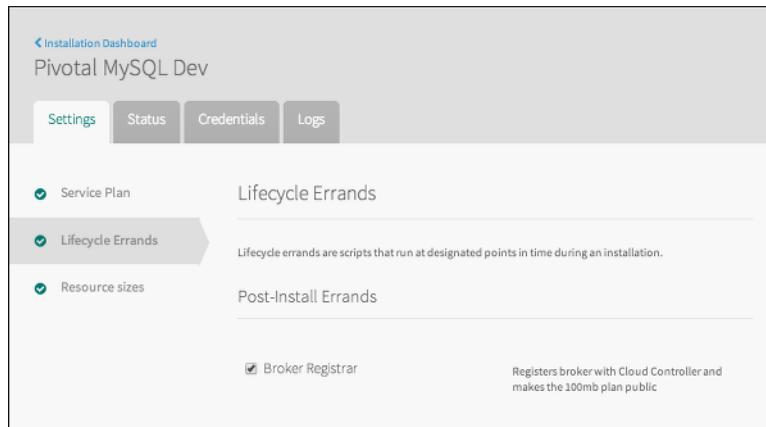
Understanding Lifecycle Errands

Lifecycle errands are scripts that run at designated points in time during a product installation. Product teams create errands as part of a product package, and a product can only run errands it includes.

Products can have two kinds of lifecycle errands:

Post-Install: Post-install errands run after a product installs, before Ops Manager makes the product available for use.

Most post-install errands run by default. An operator can prevent a post-install errand from running by deselecting the checkbox for the errand on the **Settings** tab of the product tile in Ops Manager before installing the product.



The screenshot shows the 'Installation Dashboard' for 'Pivotal MySQL Dev'. The 'Settings' tab is active. Under the 'Lifecycle Errands' section, there is a list with a checkbox. The checkbox is checked, and a tooltip explains its function: 'Registers broker with Cloud Controller and makes the 100mb plan public'.

Typical post-install errands include smoke or acceptance tests, databases initialization or database migration, and service broker registration.

Pre-Delete: Pre-delete errands run after an operator chooses to delete a product, before Ops Manager deletes the product. Ops Manager does not display pre-delete errands, and an operator cannot prevent a pre-delete errand from running.

Typical pre-delete errands include clean up of application artifacts and service broker de-registration.

Post-Install Errand Example

Pivotal MySQL has a **Broker Registrar** post-install errand. This errand registers the service broker with the Cloud Controller and makes service plans public. After successfully installing Elastic Runtime, but before making Elastic Runtime available for use, Ops Manager runs the **Broker Registrar** post-install errand.

If an operator deselects the checkbox for the **Broker Registrar** errand before installing Elastic Runtime, the service broker is not registered with the Cloud Controller, and the service plans are not made public.

Pre-Delete Errand Example

Pivotal MySQL has a **Broker Deregistrar** pre-delete errand. This errand:

- Purges the service offering
- Purges all service instances
- Purges all application bindings
- Deletes the service broker from the Cloud Controller

When an operator chooses to delete the Pivotal MySQL product, Ops Manager first runs the **Broker Deregistrar** pre-delete errand, then deletes the product.

Ops Manager does not display the **Broker Deregistrar** pre-delete errand, and an operator cannot prevent the errand from running when Pivotal MySQL is deleted.

Decrypting and Encrypting Installation Files

This topic is intended for product teams who package, distribute, and upgrade [Pivotal Cloud Foundry](#) (PCF) products across releases.

This topic describes the installation and product template YAML files in your Pivotal Cloud Foundry Operations Manager VM. It also provides instructions on viewing and modifying the installation YAML files so that product teams can migrate content. For more information on performing content migrations, refer to the [How are Upgrades Packaged?](#) section of the *Packaging Pivotal One Products* topic.

Understanding Ops Manager Installation and Product Template Files

During the installation process, Ops Manager combines information from the installation and product template YAML files to generate the manifests that define your deployment.

Installation files: PCF stores user-entered data and automatically generated values for Ops Manager in installation files. PCF encrypts and stores these files in the following IaaS-specific directory:

- **AWS:** `/var/ubuntu/workspaces/default`
- **VMWare IaaS Product:** `/var/tempest/workspaces/default`

You must decrypt them to view the contents, edit the files as necessary for your content migration, then re-encrypt them.

Product templates: Ops Manager uses product templates to create forms and obtain user input. The `job_types` and `property_blueprint` key-value pairs in a product template determine how the `jobs` and `properties` sections display in the installation file. Ops Manager stores product templates in the following IaaS-specific directory:

- **AWS:** `/var/ubuntu/meta-data`
- **VMWare IaaS Product:** `/var/tempest/meta-data`

You can edit these files. User input does not alter these files.

Decrypting and Encrypting Installation Files

1. Clone the GitHub `pivotal-cf encrypt-decrypt scripts` repo:

```
git clone https://github.com/pivotal-cf/encrypt-decrypt-scripts
```

This repo contains the following files:

- `decrypt-ops.sh`
- `encrypt-ops.sh`
- `eos.rb`

2. Copy the files from your local machine to your Ops Manager VM.

Replace `LOCAL_USER@LOCAL_SERVER` with the username and the IP address of your machine.

Use the correct `REMOTE_USER` name for your IaaS, as follows:

- **AWS:** `ubuntu`
- **VMWare IaaS Product:** `tempest`

```
$ ssh LOCAL_USER@LOCAL_SERVER
$ scp ~/PATH_TO_FILES/*.sh REMOTE_USER@REMOTE_SERVER:/home/REMOTE_USER
$ scp ~/PATH_TO_FILES/eos.rb REMOTE_USER@REMOTE_SERVER:/home/REMOTE_USER
```

3. Log in to the Ops Manager VM using SSH and the correct username for your IaaS.

AWS: Replace `REMOTE_SERVER` with the IP address for your Ops Manager VM. When prompted, provide your `ops_manager` SSH key information that you created when configuring AWS components for PCF. For more information, refer to the [Create Two Key Pairs](#) section of the *Configuring AWS for PCF* topic.

```
$ ssh ubuntu@REMOTE_SERVER
```

VMWare IaaS Product: Replace `REMOTE_SERVER` with the IP address for your Ops Manager VM. Also provide the admin password that you set when you installed Ops Manager to your vSphere, vCloud Air, or vCloud IaaS. For more information, refer to the following:

- vCloud Air and vCloud: Refer to the step on defining password information for the Ops Manager vApp in the [Complete the vApp Wizard and Deploy Ops Manager](#) section of the *Deploying Operations Manager to vCloud Air and vCloud* topic.
- vSphere: Refer to the step on defining password information for the VM admin user in the [Deploying Operations Manager to vSphere](#) topic.

```
$ ssh tempest@REMOTE_SERVER
tempest@REMOTE_SERVER's password:
```

4. Use `sudo cp` to copy the files to the protected `/var/REMOTE_USER/workspaces/default` directory.

```
$ sudo cp *.sh /var/REMOTE_USER/workspaces/default
$ sudo cp eos.rb /var/REMOTE_USER/workspaces/default
```

5. In the `/var/REMOTE_USER/workspaces/default` directory, change the permissions of the `decrypt-ops.sh` and `encrypt-ops.sh` scripts.

```
$ sudo chmod 755 decrypt.sh
$ sudo chmod 755 encrypt.sh
```

6. In the `/var/REMOTE_USER/workspaces/default` directory, run the `decrypt` command to create the `decrypted-installation.yml` and `decrypted-installed.yml` files.

Replace `PASSWORD` with your Ops Manager `REMOTE_USER` UserID password.

```
$ sudo ./decrypt.sh PASSWORD
```

This script does the following:

- Searches the `/var/REMOTE_USER/workspaces/default` for the installation YAML files and creates decrypted copies.
- Creates a directory `decrypted` and stores the decrypted files in this directory.

7. Review and edit the files as necessary for your content migration.

8. Run the `encrypt` command on the modified decrypted files.

Replace `PASSWORD` with your Ops Manager `REMOTE_USER` UserID password.

```
$ sudo ./encrypt.sh PASSWORD
```

This script encrypts the decrypted files and stores the resulting output in the `/var/REMOTE_USER/workspaces/default` directory, which overwrites the existing `installation.yml` file and adds a new file, `installed-installation.yml`. `encrypt-ops.sh` does not overwrite the `actual-installation.yml` file.

 **Note:** Ensure that you back up the edited and encrypted `installation.yml` and `installed-installation.yml` files for future reference.

Pivotal Cloud Foundry Release Notes and Known Issues

Release Notes

- [Pivotal Elastic Runtime](#)
- [Pivotal Operations Manager](#)
- [Pivotal Apps Manager](#)

Known Issues

- [Pivotal Elastic Runtime](#)
- [Pivotal Operations Manager](#)
- [Pivotal Apps Manager](#)

Pivotal Elastic Runtime v1.4.0.0 Release Notes

Changes since v1.3.5.0:

Elastic Runtime

New Features

New Stack: Trusty 14.04

There is a new stack based on Ubuntu Trusty 14.04 LTS.

The lucid64 stack that has been part of Pivotal Cloud Foundry Elastic Runtime for several years as the root file system for containers will reach end of support for security fixes on April 29th, 2015 by Canonical. Developers or Operators will need to take action to ensure existing applications migrate to using the new stack.

In PCF Elastic Runtime 1.4 support has been added for the new stack called cflinuxfs2 derived from Ubuntu Trusty 14.04.

What do you need to do?

If you have an application or app as a service running on Pivotal Cloud Foundry, ensure that your application works when run with the new stack and the corresponding new buildpacks. We recommend using a blue/green deployment strategy to ensure there is no down time for the app. Simply running the following command will result in a small amount of down time as old instances are stopped and the app is restaged with the new stack.

```
cf push app-name -s cflinuxfs2
```

If you have custom buildpacks, you need to ensure that your buildpacks work when run with the new stack.

List of buildpacks that now support cflinuxfs2:

- Java buildpack
- NodeJS buildpack
- PHP buildpack
- Go buildpack
- Python buildpack
- Ruby buildpack

Operator process for rolling out the new stack

To stay current with security fixes, operators should configure the default stack for new apps to cflinuxfs2 in the Elastic Runtime configuration tab. This means all new applications will get the new default stack:

← Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks This configures the default stack used when pushing a new app.

Assign Availability Zones

Root Filesystem Default Root Filesystem*

cflinuxfs2 - derived from Ubuntu 14.04 (Trusty Tahr)

lucid64 - derived from Ubuntu 10.04 (Lucid Lynx)

System Database Config

File Storage Config

IPs and Ports

MySQL Proxy Config

Cloud Controller

Save

Applications that were created prior to cflinuxfs2 being the default stack will have the lucid64 stack set and therefore developers or operators need to take action.

Operators should decide when to notify users that they will be vulnerable to not getting new security patches once lucid64 reaches end of support. Operators should send notice that lucid64 is close to end of life, and developers should migrate to the new cflinuxfs2 stack. Larger deployments should manage carefully forcing restaging of apps onto the new stack.

lucid64 reaches end of support for security fixes on April 29th, 2015

The lucid64 stack will be removed from the Elastic Runtime tile in the next minor release.

When an Elastic Runtime release without lucid64 is deployed, apps that have not yet switched to cflinuxfs2 will not start because their stack will no longer be available.

Operators will need to force the change to the cflinuxfs2 stack and restart the applications. There is a cf CLI plugin [stack-changer](#) available that can be used by administrators to help determine what applications have not migrated. It can then also be used to move apps to the new stack in batches.

Once the migration is complete, to remove lucid64 from the list of stacks that cloud controller returns, an admin will need to delete the stack via the CC API.

```
cf curl /v2/stacks/:lucid64_stack_guid -X DELETE
```

Known issue: For apps that use native compiled binaries, when the stack is changed, the app may fail to start. If this occurs, the current work around is to delete the app and then push the app again.

External RDS and S3 Options:

There are new features in the tile which allow for utilization of an external relational-database service for the Elastic Runtime databases and an external Amazon S3 bucket for Elastic Runtime's file storage. This feature is especially useful for those who want to install their product on AWS and take advantage of the RDS and S3 services, and also reduce the number of VMs that Elastic Runtime would otherwise require to serve as the databases.

Highly Available MySQL Service

MySQL, used by Notifications, Autoscaling, and Apps Manager, can now be deployed in a highly-available configuration. This introduces a proxy tier which routes MySQL connections from internal components to healthy cluster nodes.

API/cf CLI

- Org and space users can now list other users in the org/space [details](#)

- `cf org-users ORG`
- `cf space-users ORG SPACE`

- Allow admins to create wildcard routes [details ↗](#)
 - This feature allows cf admins to specify a wildcard route `cf create-route SPACE DOMAIN -n ***`
 - The host name must be specified exactly as `***`
 - If you map this wildcard route to an app, requests to any matching routes will be routed to the app. Exact matches of routes will take precedence over the wildcard route.
- Org Managers can now share private domains across specific organizations [details ↗](#)
- Added `VCAP_APPLICATION` to the `/v2/apps/:guid/env` endpoint [details ↗](#)
 - `cf env APP_NAME`
- Added `/v2/apps/:guid/copy_bits` endpoint [details ↗](#)
 - `cf env APP_NAME`
- Added new endpoint to return memory usage of an org `/v2/organizations/:guid/memory_usage` [details ↗](#)
- More flexible nested domain creation rules [details ↗](#)
- Deprecated AppEvent in Cloud Controller [details ↗](#)
- Inlined relations api performance enhancements [details ↗](#)
- Respect true, t, false, and f when querying boolean values [details ↗](#)
- `/v2/apps` can now be filtered by organization_guid [details ↗](#)
- Users can now specify longer start commands via the API [details ↗](#)
- `/v2/routes` can now be queried by organization guid [details ↗](#)

Improved stability

- Added drain for restarts due to memory usage [details ↗](#)
- Changed HM9000 to communicate with Cloud Controller over HTTP, and not NATS to address an issue where HM9000's API stops responding to requests [details ↗](#)
- Updated Cloud Controller's Ruby version to 2.1.4 [details ↗](#)
- Improve GoRouter stability (memory & GC times) [details ↗](#)
- Bumped apcera's NATS client in yagnats and handle callbacks [details ↗](#)
- Bump apcera/nats to v1.0.6 [details ↗](#)
- Retry polling on the state of the upload if error is an HTTP timeout [details ↗](#)
- Change v2 directory server port to 32766 [details ↗](#)
- Updated golang version in cf-release to 1.3.1
- Upgraded gorouter, hm9000 to golang 1.4.2 [details ↗](#)
- Upgraded DEA and Warden to Ruby 2.1.4 [details ↗](#)
- Update collector, login, and uaa to Ruby 2.1.4 and remove Ruby 1.9.3 [details ↗](#)

Services API

- Update Service Plan completed [details ↗](#)
- Users can change the plan for a service instance, if support for the feature is declared by the service broker [details ↗](#)
- Removed unnecessary warning when plans without instances are deleted during broker catalog sync [details ↗](#)
- Pass through message from broker to client for 409 responses [details ↗](#)
- Work completed on Service Audit Events [details ↗](#)
- Cloud Controller now correctly returns a 502 when broker returns a malformed response [details ↗](#)
- Service Broker client in Cloud Controller now uses a different library to prevent retry on timeout, as retries are now handled by orphan mitigation mechanisms [details ↗](#)
- Service Instance Orphan Mitigation improvements [details ↗](#)
- `/v2/managed_service_instances` is now deprecated [details ↗](#)
- Renaming broker does not cause CC to fetch service broker catalog [details ↗](#)
- Work continues on support for Asynchronous Service Instance Operations [details ↗](#)
 - Improved error handling and descriptive error messages

- Operations are block while in progress
- Brokers can declare a polling interval, admins can configure a default interval and max attempts
- Audit events created for async operations
- API docs being published, changes for this feature are marked Experimental and subject to backward incompatible change

Misc

- Update default `request_timeout_in_seconds` to 900s (15min) [details ↗](#)
- Upgrade HAProxy to 1.5.10 [details ↗](#)
- make `ha_proxy` respect disabling timeout [details ↗](#)
- Expose IP address and port on vars with prefix `CF_INSTANCE` in the container [details ↗](#)
- `VCAP_SERVICES`, `DATABASE_URL`, and `VCAP_APPLICATION` no longer does BASH variable substitution [details ↗](#)
- Route endpoints can now request their own staleness threshold [details ↗](#)
- Cloud Controller worker names now include job index [details ↗](#)
- Allow core files to be generated in warden [details ↗](#)
- Sets the status code for upgrade to WebSocket and TCP scenarios [details ↗](#)
- Update nginx for cloud controller to nginx-1.6.2 and pcre to pcre-8.36 [details ↗](#)
- Re-enabled SIGCHLD on children of wshd [details ↗](#)
- Write hm9000 logs to a file instead of STDOUT [details ↗](#)
- cloud controller worker and clock jobs now logs to files instead of STDOUT [details ↗](#)
- Output timestamps in all ctl logs [details ↗](#)
- nginx_status endpoint is always enabled [details ↗](#)
- Org Auditors can now see space related /v2/events [details ↗](#)
- Escape `VCAP_SERVICES` and `VCAP_APPLICATION` env variables [details ↗](#)
- cloudfoundry/gorouter #75: Allow keepalives to the front-end proxy [details ↗](#)
- Add content-type to blobstore file creation [details ↗](#)
- Security Groups applied to the space are visible to non-admins with correct permissions [details ↗](#)
- Improved service broker error handling and user-facing error messages
- Improved service orphan mitigation; failed provision requests trigger delete requests with retry and backoff

Bug Fixes

- Fixed several NATS split brain issues
- Fixed several container transitions on the DEA
- Fixed an issue where logs were not being captured when an app is shutting down
- Fixed an issue where the DEA needed twice the space requested to successfully place an app [details ↗](#)
- Fix key type issue in graphite historian [details ↗](#)
- Fixed Mysql 5.6 issue on timestamp creation [details ↗](#)
- Fixed issue where deadlocks were happening on delayed job workers [details ↗](#)
- Fixed SpaceDeveloper able to move a service instance from one space to another [details ↗](#)
- Fix incompatibility with MySQL 5.5 when setting timezone [details ↗](#)
- Fixed time consistency in cloud controller [details ↗](#)
- Renaming a service instance no longer causes instance to become stuck in 'update in progress' [details ↗](#)
- Fixed an issue where deleting a route bound to a running app with multiple routes would actually result in a small amount of downtime for the app [details ↗](#)
- Fixed a bug when an app has a lot of fingerprints matched during resource matching, it should still be able to stage [details ↗](#)
- Recursive deletion of a space no longer aborts after first failure to delete a resource within (deleting all resources that can be deleted) and deletion of service instances and bindings are not rolled back, preventing orphans. Deletion of applications during recursive deletion of a space remains in a transaction; failure to delete one app will roll back deletion of all apps in the space.

UAA and Login Server

New Features

- Improved SAML SSO Setup via Ops Manager Console:
Earlier Single Sign-On for the platform could be set up only by providing the Meta Data URL for the SAML Identity Provider. We now support two ways to set up the Identity provider for SSO: Meta Data URL and Meta Data XML. This feature extends support for Identity Providers that do not support a Meta Data URL. For more information, refer to [Configuring Single Sign-On](#).
- Mapping LDAP Groups to Administrator Role via Ops Manager:
The LDAP Configuration page now supports configuring the LDAP Group information. This allows for LDAP groups to be mapped to the Administrator Role in Cloud Foundry. For more information, refer to [Connecting Elastic Runtime to LDAP](#).

Logging, Analytics and Metrics

New Features

- Loggregator Firehose.
- Changing syslog drain location no longer requires application restaging.
- Updated LoggregatorEmitter to 4.0.0.
- System now Diego-enabled.
- Application forcibly closing output stream closes both standard out and standard error to logging system - fixes logging agent stability issues.
- New tuning parameters for cc polling.
 - properties.syslog_drain_binder.update_interval_seconds
 - properties.syslog_drain_binder.polling_batch_size
- Standardized `--config` flag across all components.

Component Features And Bug Fixes

- Numerous general system fixes.
 - golang 1.3.X now default for most components.
 - Bug fixes for recovering when NATS connection is lost.
 - Honor system's usess/flag.
 - Add timestamps to every HttpStartStop envelope.
- Metron specific features and fixes.
 - Syslog configuration consolidated into Metron package.
 - No longer report metrics for downed component.
 - Fixed where application shutdown logs were not being collected.
 - Stability fixes between dropsonde and Metron.
 - Tracks cumulative values now in counters.
- Doppler features and fixes.
 - Increased buffer size for messages to better handle drops.
 - Improved channel operations for better durability.
- TrafficController / NOAA features and fixes.
 - Streaming endpoint can use a cookie to get the oauth token (allows js clients to stream logs).
 - Numerous NOAA additions for firehose.
 - Attempts to reconnect after unexpected disconnect.
 - Can fulfill Diego container metrics requests.
- Syslog features and fixes.
 - Syslog configuration consolidated into metron package.
 - Syslog aggregator package removed (use metron package for syslog forwarding).

Bug Fixes

- Bug fixed where application shutdown logs were not being collected by loggregator.
- Bug fixes for recovering when NATS connection is lost.

Notifications

- A new centrally-managed application service to notify users about platform and application events such as:
 - New service invitations
 - Planned downtime alerts
 - Application performance warnings
- Manage existing internal notifications and custom messages via an API so that administrators can:
 - Create and edit custom message templates in HTML and plain text
 - Send notifications to individuals, spaces, organizations or all users
- You can configure your own SMTP server in the runtime tile.

Autoscaling

- Auto-scaling is now officially GA after a successful beta period
- Set min and max instances by % CPU utilization or calendar schedule
- Configured from apps manager.

Buildpacks

New Features

- Buildpacks are now downloaded to the DEA when the DEA starts instead of waiting for first app push
- Upgrading php buildpack to v3.1.0
 - Added support for `cflinuxfs2` stack.
 - On the `lucid64` stack, the following **changes** were made to package support:
 - Removed support for PHP 5.4.35, added support for PHP 5.4.38, in addition to still-supported 5.4.36 and 5.4.37.
 - Removed support for PHP 5.5.19, added support for PHP 5.5.22, in addition to still-supported 5.5.20 and 5.5.21.
 - Removed support for PHP 5.6.3, added support for PHP 5.6.6, in addition to still-supported 5.6.4 and 5.6.5.
 - Added support for Apache httpd 2.4.12, in addition to still-supported 2.4.10.
 - Added support for mod_lua to httpd 2.4.10 (also supported in httpd 2.4.12).
 - Removed support for nginx 1.7.7, however, nginx 1.5.{11,12,13}, 1.6.{0,1,2}, and 1.7.{8,9,10} are still supported.
 - Replaced support for composer 1.0.0-alpha8 with 1.0.0-alpha9.
 - Replaced support for newrelic 4.15.0.74 with 4.18.0.89.
 - On the `cflinuxfs2` stack, **only** the following packages are supported:
 - PHP 5.4.{36,37,38}, 5.5.{20.21.22}, and 5.6.{4,5,6}.
 - HHVM 3.5.0 and 3.6.0
 - Apache httpd 2.4.{10,12}.
 - nginx 1.5.13, 1.6.{0,1,2}, and 1.7.{8,9,10}.
 - composer 1.0.0-alpha9
 - newrelic 4.18.0.89
 - The buildpack has increased significantly in size with this release, from 458M to 1.1G.
- Upgrading ruby buildpack to v1.3.0
 - Added support for `cflinuxfs2` stack.
 - Binary files now permanently hosted on a CF-managed S3 bucket.
 - On `lucid64`, the following **changes** were made to package support:

- Removed support for ruby 1.9.2. The binary for this version, included in previous [ruby-buildpack](#) releases, is not functional. There is a [track of work](#) scheduled to recreate binaries for all our stacks; but for this release, generating replacement binaries was deprioritized. That said, [ruby 1.9.2 has reached end-of-life](#), and at this moment we do not plan to support 1.9.2 in future [cflinuxfs2](#) buildpack releases.
- On [cflinuxfs2](#), only the following interpreter versions are supported:
 - ruby 2.2.0, 2.1.{2,3,4,5}, 2.0.0, and 1.9.3.
 - jruby 1.7.{1-11}.
 - See [manifest.yml](#) for full details.
- The buildpack has increased in size with this release, from 825M to 922M.
- Upgrading python buildpack to v1.2.0
 - Added support for [cflinuxfs2](#) stack.
 - Binary files now permanently hosted on a CF-managed S3 bucket.
 - On the [lucid64](#) stack, no changes in python version support were made.
 - The [cflinuxfs](#) stack supports the same python versions as [lucid64](#) with the following exceptions:
 - Removed support for python 2.7.1, however 2.7.{0,2,3,6,7,8,9} are still supported.
 - Removed support for python 3.2.0, however 3.2.{1,2,3} are still supported.
 - See [manifest.yml](#) for full details.
 - The buildpack has increased in size with this release, from 315M to 654M.
- Upgrading nodejs buildpack to v1.2.0
 - Added support for [cflinuxfs2](#) stack.
 - Binary files now permanently hosted on a CF-managed S3 bucket.
 - On the [lucid64](#) stack, the following **changes** were made to package support:
 - Removed binaries for node v0.8.6, v0.10.14, and 0.11.4 for all stacks. The binaries for these versions, included in previous [nodejs-buildpack](#) releases, are not functional. There is a [track of work](#) scheduled to recreate these binaries for all our stacks; but for this release, generating replacement binaries was deprioritized.
 - On the [cflinuxfs2](#) stack, the same packages are supported as on the [lucid64](#) stack, which are too numerous to list here. See [manifest.yml](#) for full details.
 - The buildpack has decreased in size with this release, from 418M to 403M.
- Upgrading go buildpack to v1.2.0
 - Added support for [cflinuxfs2](#) stack.
 - The buildpack, at 675M, has not materially increased in size with this release.
 - Support version 1.4.1
 - Default to version 1.4.1 if no version specified
 - (<https://www.pivotaltracker.com/story/show/86334724>)
 - Update buildpack-packager to v.2.0.0
 - (<https://www.pivotaltracker.com/story/show/84805200>)
- Update to Java Buildpack 2.7.1 [details](#) [release](#)
 - Improved Access Logging (via [Guillaume Berche](#) and [Kevin Kessler](#))
 - Improved AppDynamics Integration (via [Troy Astle](#), [Max Brunsfeld](#), Mike Youngstrom, and [Shaozhen Ding](#))
 - New Relic on Java 8
 - Secure Dependency Retrieval via HTTPS
 - Java 8 as Default
 - Tomcat 8 as Default (via [Dave Head-Rapson](#))
 - Print [OutOfMemory](#) diagnostics to Loggregator (via [Troy Astle](#))
 - Support for custom CA certificates when download from a repository (via [Dave Head-Rapson](#))
 - Tomcat Redis Session Replication support for alternate Redis services (via [Neil Aitken](#))
 - Improved SSL Diagnostics
 - Support for GemFire session replication.
 - Better [JAVA_OPTS](#) escaping (via [Sridhar Vennela](#))
 - Documentation updates (via [Makoto Motegi](#), [Mohammad Asif Siddiqui](#), and [Ann Paungam](#))
 - Windows build improvements (via [Josh Ghiloni](#))
 - JRebel support

Security - CVE fixes have been implemented since v1.3.0.0 and released via security patches.

- BASH Shellshock CVE-2014-6271, CVE-2014-7169, CVE-2014-7186, and CVE-2014-7187
- Updated HAProxy to disable SSLv3. This addresses POODLE CVE-2014-3566
- GHOST CVE-2015-0235 [details ↗](#)
- Updated lucid64 and cflinuxfs2 to address CVE-2013-7423, CVE-2014-9402, CVE-2015-1472, CVE-2015-1473 [detail ↗](#)
- Updated ca-certificates for lucid64 to address USN-2509-1 [detail ↗](#)
- CVE-2014-9680 [details ↗](#)
- lucid64 and cflinuxfs2. This addresses USN-2537-1, CVE-2015-0209, CVE-2015-0286, CVE-2015-0287, CVE-2015-0288, CVE-2015-0289, CVE-2015-0292, CVE-2015-0293 [details ↗](#)
- [CVE-2014-9636 ↗](#) [details ↗](#)
- Updated rootfs to address openssl vulnerabilities [details ↗](#) [ubuntu-notice ↗](#)
- Updated default cipher string for HAProxy [details ↗](#)
- Application Security Groups now support logging of the first packet of outbound tcp traffic [details ↗](#) [apidoc ↗](#)

Ops Manager 1.3 Release Notes

Operator Features

Operators can...

- modify Static IPs for products with jobs that expose them
- change instance counts and Static IPs will be recalculated and reassigned
- specify more than one network for an installation
- designate a specific network to deploy individual products
- see IP addresses for multiple networks on product status pages
- be protected from entering IP addresses that clash across products and networks
- specify which network is designated as Infrastructure vs. Deployment for Director
- be protected from changing deployment networks (Agents would be orphaned)
- change networks for any product before or after installation occurs
- deploy to networks and availability zones without configuring them when only one exists
- see helpful instructions if there are no networks or availability zones
- specify more than one availability zone (cluster + resource pool) for an installation
- balance jobs across availability zones on deployment
- re-balance jobs across availability zones if an availability zone is added or removed
- see which availability zones an instance resides on product status pages
- automatically upgrade to latest Director without needing to click “Add”
- install the most recent version by default if there is more than one Director
- be protected from reverting to old versions of Director
- use Ops Manager with some particular optimized pages (load times)
- specify a vCenter folder for my network
- make changes to networks that impact other settings without reference problems
- make changes to availability zones that impact other settings without reference problems
- specify which availability zone in which to place singleton jobs
- specify which availability zones in which to balance multi-instance jobs
- have older releases cleaned out of blobstores after installations complete

Product Author Features

Product Authors can...

- add products to installations via API
- display slug fields in collection forms
- use drop-down property types
- use Ops Manager templating format rather than ERB
- specify the Ops Manager version to target
- use more than one verifier per form
- see the Ops Manager SHA in the debug page
- use Ops Manager templating to reference Director deployment_ip and NTP settings

Bug fixes

- Logs can be downloaded.
- NFS server manifest corrections.

- Collection subfields can be optional.
- Resource pools are optional.
- Integers and booleans in collections work properly.
- Multi-select in collections work properly.
- Configuring products before Director works properly.
- Saving forms dims entire page rather than top ¾.
- Secrets will not be confused with scientific notation (Java YAML problem).
- Clusters and resource pools are verified before installation.
- Networks are verified before installation.
- Configured status for products works properly.
- Dynamic IP allocation works properly.

Pivotal Cloud Foundry Apps Manager v1.4.0.0 Release Notes

Changes since v1.3.0.0:

New name

- Apps Manager is the new name of Developer Console.
- The push-console errand has been renamed push-apps-manager and now deploys to the `apps-manager` space in the `system` org.

Less configuration and faster deploys

- Apps Manager now uses the Notifications service to send registration, invitation, and password reset emails.
- Previous versions of Apps Manager required three apps to function: the main app, the scheduler app, and the workers app. In v1.4, Apps Manager has been reduced to just the main app. Various refactoring efforts have also removed unused code from the main app.
- As a result, the time to deploy Apps Manager is significantly faster in v1.4.

New database options

- Previous versions of Apps Manager required a Postgres database. In v1.4, cloud operators have an additional option of using a MySQL database for new installations. Migrations from v1.3 or earlier Postgres databases to v1.4 MySQL databases are not supported.

App dashboard

- New streaming logs feature enables developers to monitor app and platform activity in real-time from the browser.
- New usability improvements include better workflows for creating and binding service instances, seeing when an app was last pushed, seeing which buildpack was used to deploy, viewing recent logs, viewing and managing env variables, and viewing all system-set env variables.
- Users with the space auditor role can now see events, logs, and a read-only list of bound service instances.

Org dashboard

- Adding new roles to an existing user is now possible via the invitation flow.
- Org quota usage is now refreshed every 30 seconds and calculated from a new Cloud Controller endpoint.

Usage service

- The usage report has a new streamlined design.
- New feature exposes service usage events as a billing metric for internal chargeback or showback.
- New security constraints limit access to the Usage Service and Usage Report to users with the Org Manager role (per org) or `cloud_controller.admin` scope (all orgs).

Admin-only feature flags

- When the `ENABLE_NON_ADMIN_ORG_CREATION` env variable is set to `false` on the `apps-manager` app, only admin users will be able to create new orgs in Apps Manager. This is the default behavior. When it is set to `true`, all users will be able to create new orgs in Apps Manager.
- When the `ENABLE_NON_ADMIN_USER_MANAGEMENT` env variable is set to `false` on the `apps-manager` app, only admin users can invite users and manage roles in Apps Manager. This is the default behavior. Also, users will not be able to access the registration or password reset pages. When it is set to `true`, Org Managers and Space Managers can invite users and manage roles in Apps Manager.

Pivotal Elastic Runtime v1.4 Known Issues

- 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.
- With PCF 1.4, we have added the ability to map LDAP groups to the Administrator role. This allows LDAP users belonging to that group to perform the administrator activities rather than using an internal admin account which is created by the installer.

We have introduced two new fields under the LDAP Configuration in Elastic Runtime Tile: LDAP Group Search Base and Group Search Filter. If LDAP is enabled for Pivotal Cloud Foundry, please follow the steps below after upgrade to set the LDAP Group Search Base and LDAP Group Search Filter. If these values are not set, LDAP authentication will not function.

The LDAP group search base is the location in the LDAP directory tree from which the LDAP Group search begins. For example, a domain named "cloud.example.com" typically uses the following LDAP Group Search Base:

`ou=Groups,dc=example,dc=com`. The LDAP Group Search filter is a string that defines LDAP Group search criteria. The standard value is `member={0}`.

Ops Manager 1.3 Known Issues

- Changes to products cannot be combined with deletions. Make a change, click Apply, and wait for the change to be installed. Do a deletion separately.
- Resource Pools cannot be deleted from Availability Zones. If a resource pool is inadvertently deleted, run [BOSH recreate](#).
- Exporting an installation can result in a 500 error if the Ops Manager VM is less than 2GB in RAM.
- The progress bar can stall during an installation, which requires the user to refresh the page.
- The subnet form on the Network page can take invalid formats. The correct format is CIDR notation.
- Ops Manager fails to create VMs using the Cisco NSX 1000v switch. The vSphere Distributed Switch is recommended.

Pivotal Cloud Foundry Apps Manager v1.4.0.0 Known Issues

New issues

- Users with the `cloud_controller.admin` scope do not see events on the App Dashboard unless they also have the Space Developer role.

Existing issues

- When running PCF with self-signed certs, streaming logs requires an extra step to accept the cert from Loggregator.
- On the Org Dashboard, user-provided services do not display in the services count.
- On the My Account page, leaving your last org results in a 500 error.
- The space page is not dynamically updated, so you must refresh to see the current status.