

Table of Contents

Table of Contents	1
Pivotal CF Documentation	4
Getting Started with Pivotal CF	5
Using Operations Manager	15
Prerequisites to Deploying Operations Manager and Elastic Runtime	17
Preparing Your Firewall for Deploying Pivotal CF	19
Using Your Own Load Balancer	20
Using SSL with a Self-Signed Certificate	21
Using Ops Manager Resurrector on VMware vSphere	23
Connecting Elastic Runtime to LDAP	25
Configuring Network Segmentation in Ops Manager	27
Deploying Operations Manager to vSphere	31
Provisioning a Virtual Disk in vSphere	34
Deploying Operations Manager to vCloud Air and vCD	36
Understanding the Ops Manager Interface	43
Adding and Deleting Products	44
Configuring Ops Manager Director for VMware vSphere	46
Configuring Ops Manager Director for vCloud Air and vCloud	51
Creating New Elastic Runtime User Accounts	54
Logging into the Developer Console	55
Controlling Console User Activity with Environment Variables	56
Backing Up Pivotal CF	57
Upgrading Operations Manager	61
Upgrading Products in a Pivotal CF Deployment	64
Monitoring VMs in Pivotal CF	65
Deploying Pivotal Ops Metrics	67
Using Pivotal Ops Metrics	72
Pivotal CF Troubleshooting Guide	74
Troubleshooting Ops Manager for VMware vSphere	80
Advanced Troubleshooting with the BOSH CLI	82
Pivotal CF Security Overview and Policy	87
Cloud Foundry Concepts	89
Cloud Foundry Overview	90
Cloud Foundry Components	91
(Go)Router	93
Getting started	93
User Account and Authentication (UAA) Server	95
Cloud Controller	102
Droplet Execution Agent	104
Warden	107
Messaging (NATS)	112
How Applications Are Staged	114
Scaling Cloud Foundry	115
Orgs, Spaces, Roles, and Permissions	118
Cloud Foundry Security	120

Stacks	124
Cloud Foundry Glossary	125
Operator's Guide	126
Domains and Shared Domains	127
Isolating a Pivotal CF Deployment with a Private Network	129
Understanding the Elastic Runtime Network Architecture	132
Load Balancer	132
Router	132
Configuring Pivotal CF SSL Termination	133
Changing the Quota Plan of an Organization with cf CLI	135
Identifying Elastic Runtime Jobs Using vCenter	138
Administering Cloud Foundry	140
Adding Buildpacks to Cloud Foundry	141
Creating and Managing Users with the cf CLI	142
Creating and Managing Users with the UAA CLI (UAAC)	144
Application Security Groups	148
Using the Developer Console	150
Getting Started with the Developer Console	151
Understanding Developer Console Permissions	152
Managing Spaces Using the Developer Console	153
Managing User Accounts in Spaces Using the Developer Console	155
Developer Guide	157
Considerations for Designing and Running an Application in the Cloud	159
Deploy an Application	161
Domains, Subdomains, and Routes	164
Deploying with Application Manifests	168
Cloud Foundry Environment Variables	175
Using Blue-Green Deployment to Reduce Downtime and Risk	179
Application Logging in Cloud Foundry	182
Troubleshooting Application Deployment and Health	185
Identifying the API Endpoint for your Elastic Runtime Instance	189
cf Command Line Interface (CLI)	190
Installing the cf Command Line Interface	191
Getting Started with cf v6	192
Using cf with an HTTP Proxy Server	197
About Starting Applications	200
Scaling an Application Using cf scale	201
Services Overview	202
Adding a Service	204
Binding a Service Instance	206
Managing Service Instances with the CLI	207
Third-Party Managed Service Instances	209
User-Provided Service Instances	211
Configuring Play Framework Service Connections	214
Migrating a Database in Cloud Foundry	215
Using Third-Party Log Management Services	217
Configuring Selected Third-Party Log Management Services	219

Integrating Cloud Foundry with Splunk	224
Buildpacks	226
Buildpack Detection	227
Custom Buildpacks	228
Packaging Dependencies for Offline Buildpacks	231
Java Buildpack	233
Getting Started Deploying Spring Apps	234
Tips for Java Developers	240
Configure Service Connections for Grails	245
Configure Service Connections for Play Framework	247
Configure Service Connections for Spring	248
Cloud Foundry Eclipse Plugin	255
Cloud Foundry Java Client Library	270
Build Tool Integration	272
Node.js Buildpack	276
Tips for Node.js Applications	277
Environment Variables Defined by the Node Buildpack	280
Configure Service Connections for Node.js	281
Ruby Buildpack	283
Getting Started Deploying Ruby on Rails Apps	284
Tips for Ruby Developers	290
Configure Service Connections for Ruby	296
Services	298
Overview	299
Service Broker API	300
Managing Service Brokers	309
Access Control	312
Catalog Metadata	314
Binding Credentials	318
Dashboard Single Sign-On	320
Example Service Brokers	324
Packaging Pivotal One Products	325
Understanding Lifecycle Errands	342
Pivotal CF Release Notes and Known Issues	344
Pivotal Elastic Runtime v1.3.0.0 Release Notes	345
Ops Manager 1.3 Release Notes	346
Pivotal CF Developer Console v1.3.0.0 Release Notes	348
Pivotal Elastic Runtime v1.3 Known Issues	349
Ops Manager 1.3 Known Issues	350
Pivotal CF Developer Console v1.3.0.0 Known Issues	351

Pivotal CF Documentation

1. [Getting Started](#)
A quick guide to installing and getting started with Pivotal CF. If you are new to Pivotal CF, start here.
2. [Using Ops Manager](#)
A guide to using the Ops Manager interface to manage your Pivotal CF PaaS.
3. [Elastic Runtime Concepts](#)
An explanation of the components in Elastic Runtime and how they work.
4. [Operating Elastic Runtime](#)
A guide to running the Elastic Runtime component of Pivotal CF.
5. [Administering Elastic Runtime](#)
A guide to managing Elastic Runtime at the administrator level.
6. [Using the Developer Console](#)
A guide to using the web-based console 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 CF](#)
A guide to packaging Pivotal CF products.
11. [Release Notes](#)
Release notes and known issues for Pivotal Operations Manager, Pivotal Elastic Runtime and Pivotal MySQL Developer.

Getting Started with Pivotal CF

Welcome to Pivotal CF!

This guide is intended to walk you through deploying the installation virtual machine, setting up your PaaS, targeting Elastic Runtime, and pushing your first app. If at any time you experience a problem 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 CF and the Pivotal One product suite.

Step 1: Confirm System Requirements

Before you begin your Pivotal CF 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 Pivotal CF Installation Virtual Machine

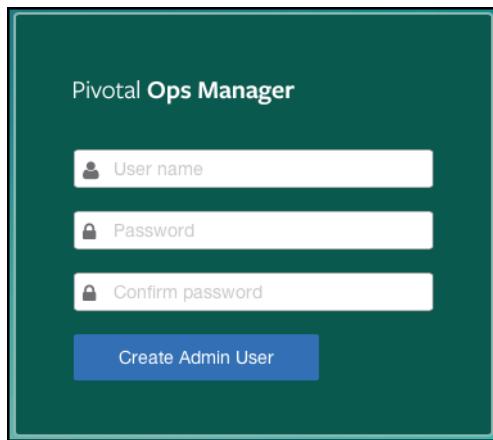
Download the Pivotal CF .ova file and deploy it. The procedure you follow depends on the IaaS you use:

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

Step 3: Set Up Your PaaS

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

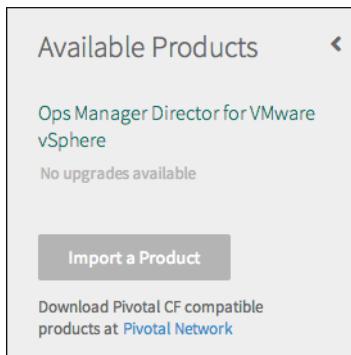
1. Browse to the interface IP address you specified in Step 2.



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

Import Products

1. Download Elastic Runtime and Pivotal MySQL Dev from [Pivotal Network](#).
2. From the Available Products view, click **Import a Product**.



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

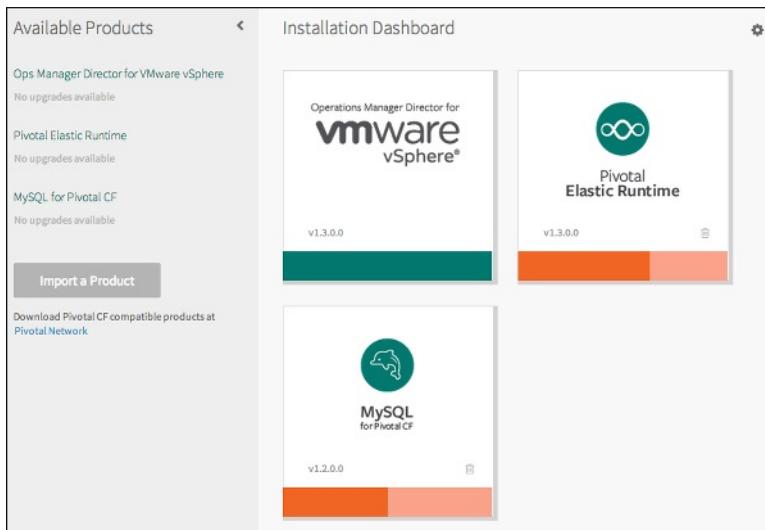
Configure Ops Manager Director

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

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

Configure Elastic Runtime

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



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

The screenshot shows the 'Pivotal Elastic Runtime' settings page. The 'Settings' tab is active. On the left, there is a sidebar with several configuration options, each preceded by a checked checkbox:

- Assign Networks
- Assign Availability Zones
- HAProxy
- Router IPs
- Cloud Controller
- External Endpoints
- SSO Config
- LDAP Config
- SMTP Config
- Lifecycle Errands
- Resource Config

The 'Assign Networks' option is currently selected. To its right is a panel titled 'Assign Networks' containing a dropdown menu set to 'default' and a blue 'Save' button.

3. (**vSphere Only**) Select **Assign Availability Zones**. These are the Availability Zones you [create](#) when configuring Ops Manager Director.

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

[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

Availability Zone Assignments

Place singleton jobs in

default

Balance other jobs in

default

[Save](#)

4. In the left column, select **HAProxy**.
5. 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 Pivotal CF SSL Termination](#) topic. For help understanding the Elastic Runtime architecture, refer to the [Architecture](#) topic.

HAProxy

Router IPs

Cloud Controller

External Endpoints

SSO Config

LDAP Config

SMTP Config

Lifecycle Errands

Resource Config

HAProxy IPs

SSL Certificate *

```
-----BEGIN CERTIFICATE-----
MIID0jCCAiKgAwIBAgIVAOcqflZLVjwVGU
DQ5kDxq+8vesfGMA0GCSqGSIb3DQE
BQUAMEIxCzAJBgNVBAYTAiVTMRAwDgY
-----END CERTIFICATE-----
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAsdSX/2+LZQPH4HKy
-----END RSA PRIVATE KEY-----
```

[Generate Self-Signed RSA Certificate](#)

Trust Self-Signed Certificates

[Save](#)

6. 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**.

- The **Certificate Key PEM** and **Private Key PEM** fields now contain certificate keys. Click **Save**.
 - Select **Router IPs**. 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 Pivotal CF that you registered with your load balancer. Refer to the [Using Your Own Load Balancer](#) topic for help using your own load balancer with Pivotal CF.
 - HAProxy load balancer:** Leave this field blank.
 - Click **Save**.
-
- Select **Cloud Controller** and enter the system and application domains.
 - The system domain defines your target when you push apps to Elastic Runtime
 - The application domain defines where Elastic Runtime should serve your apps

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks	Coordinates Pivotal Elastic Runtime application lifecycles
Assign Availability Zones	System Domain * <input type="text" value="poodle.wild.cf-app.com"/>
HAProxy	Apps Domain * <input type="text" value="poodle.wild.cf-app.com"/>
Router IPs	Cloud Controller DB Encryption Key <input type="text" value="Secret"/>
Cloud Controller	Maximum File Upload Size (MB) (min: 1024, max: 2048) * <input type="text" value="1024"/>
External Endpoints	<input type="checkbox"/> Disable Custom Buildpacks
SSO Config	App Memory (MB) (min: 10240, max: 102400) * <input type="text" value="10240"/>
LDAP Config	
SMTP Config	
Lifecycle Errands	
Resource Config	Service Instances (min: 100, max: 1000) * <input type="text" value="100"/>
<input type="button" value="Save"/>	

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

11. 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 CF](#) for more information.

12. Enter your intended maximum file upload size.

13. **(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 Pivotal CF documentation.

14. Click **Save**.

15. **(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**.

[Installation Dashboard](#)

Pivotal Elastic Runtime

Settings Status Credentials Logs

Assign Networks	Fowards syslog messages via TCP to a RELP syslog server. Leave these options blank unless you are providing a syslogd server.
Assign Availability Zones	
HAProxy	IP Address of Syslog Aggregator Host <input type="text"/>
Router IPs	Syslog Aggregator Port <input type="text"/>
Cloud Controller	
External Endpoints	<input type="button" value="Save"/>
Install CF	

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

- (Optional - Advanced)** If you are using the VMware SSO appliance for integration with Active Directory, complete this step. Select **SSO Config**, enter the URL of an SSO appliance to connect to Active Directory servers in your

organization, and click **Save**.

The screenshot shows the 'Pivotal Elastic Runtime' dashboard with the 'Settings' tab selected. On the left, a sidebar lists configuration steps: Assign Networks, Assign Availability Zones, HAProxy, Router IPs, Cloud Controller, External Endpoints, SSO Config (which is highlighted with a grey arrow), and LDAP Config. To the right, under 'SSO Config', there is a field labeled 'SSO URL' with an empty input box and a 'Save' button.

2. **(Optional - Advanced)** If you are using the LDAP endpoint for UAA, complete this step. Select **LDAP Config** and enter the following information:

- LDAP server URL
- LDAP credentials
- LDAP search base
- LDAP search filter
- LDAP server SSL certificate
- Alternate name for the LDAP server SSL certificate

Click **Save**.

The screenshot shows the 'Pivotal Elastic Runtime' dashboard with the 'Settings' tab selected. On the left, a sidebar lists configuration steps: Assign Networks, Assign Availability Zones, HAProxy, Router IPs, Cloud Controller, External Endpoints, SSO Config, LDAP Config (which is highlighted with a grey arrow), SMTP Config, Lifecycle Errands, and Resource Config. To the right, under 'LDAP Config', there are several fields: 'LDAP Server URL' (empty input box), 'LDAP Credentials' (two empty input boxes for 'Username' and 'Password'), 'LDAP Search Base' (empty input box), 'LDAP Search Filter' (empty input box), 'LDAP Server SSL Cert' (empty input box), 'LDAP Server SSL Cert AltName' (empty input box), and 'Email Attribute' (input box containing 'mail'). A 'Save' button is at the bottom.

3. **(Optional)** Select **SMTP Config**, enter your reply-to and SMTP email information, and click **Save**. The console 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.

Reply-To Email
admin@my.cf.com

From Email
admin@my.cf.com

Address of SMTP Server
172.16.64.5

Port of SMTP Server
25

HELO Domain
hi.com

SMTP Server Credentials

Username

Password

SMTP Authentication
plain

SMTP Enable Automatic STARTTLS

Save

4. 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 console and usage service deploys do not trigger this API call, and do not purge application usage events.

The screenshot shows the 'Lifecycle Errands' section of the Pivotal Elastic Runtime Settings. It includes a list of errands and their descriptions:

- Assign Networks**: Lifecycle errands are scripts that run at designated points during an installation.
- Assign Availability Zones**: Post-Install Errands
- HAProxy**: Push Console (Pushes the Pivotal Dev Console application to your Elastic Runtime installation)
- Router IPs**: Push App Usage Service (Monitors usage of an application pushed to your Elastic Runtime installation)
- Cloud Controller**: Run Smoke Tests (Runs Smoke Tests against your Elastic Runtime Installation)
- External Endpoints**
- SSO Config**
- LDAP Config**
- SMTP Config**
- Lifecycle Errands**

A blue 'Save' button is located at the bottom right.

- Select **Resource Sizes**, accept the defaults or make necessary changes, and click **Save**.
- Click the **Installation Dashboard** link to return to the Installation Dashboard.

Configure MySQL

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

Install Products

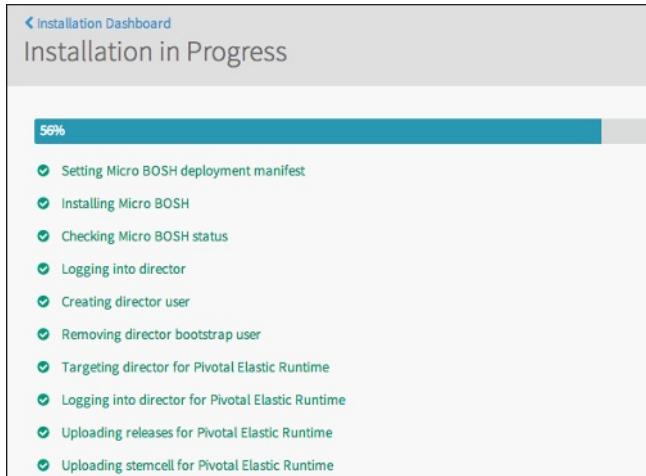
- Click **Install**.

The screenshot shows the 'Pending Changes' screen with three items selected for installation:

- INSTALL Ops Manager Director for VMware vSphere
- INSTALL Pivotal Elastic Runtime
- INSTALL Pivotal MySQL Dev

A large blue 'Install' button is at the bottom. Below it is a link to 'Recent Install Logs'.

- Your updated Pivotal CF installation begins deploying.



- 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 Pivotal CF 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 Pivotal CF, 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 tool to target your Elastic Runtime installation. Make sure you have [installed the cf tool](#). Refer to the Pivotal CF 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 the console, 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 CF 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 Pivotal CF](#)
- [Using Your Own Load Balancer](#)
- [Using SSL with a Self-Signed Certificate](#)
- [Using Ops Manager Resurrector on VMware vSphere](#)
- [Connecting Elastic Runtime to LDAP](#)
- [Configuring Network Segmentation in Ops Manager](#)

Deploying Operations Manager

- [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.

- [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 VMware vSphere](#)
- [Configuring Ops Manager Director for vCloud Air and vCloud](#)
- [Creating New Elastic Runtime User Accounts](#)
- [Logging into the Developer Console](#)
- [Controlling Console User Activity with Environment Variables](#)

Backing Up and Upgrading

- [Backing Up Pivotal CF](#)
- [Upgrading Operations Manager](#)
- [Upgrading Products in a Pivotal CF Deployment](#)

Monitoring, Logging, and Troubleshooting

- [Monitoring VMs in Pivotal CF](#)
- [Deploying Pivotal Ops Metrics](#)
- [Using Pivotal Ops Metrics](#)
- [Pivotal CF Troubleshooting Guide](#)

- [Troubleshooting Ops Manager for VMware vSphere](#)
- [Advanced Troubleshooting with the BOSH CLI](#)
- [Pivotal CF 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 Requirements

- vSphere 5.0, 5.1, or 5.5
- vSphere editions: standard and above
- 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 may turn this setting off. If you leave this setting on and the VM hosts do not support VT-x/EPT, then each VM will require 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.

vCD/vCloud Air Requirements

- vCD 5.1, 5.2, or 5.6 (vCloud Air)

Open Stack

Command line install only available through Pivotal Professional Services.

AWS

Command line install only available through Pivotal Professional Services.

General Requirements

- 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 could 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.
- 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	✓	
Console 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 Pivotal CF

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
```

Using Your Own Load Balancer

Pivotal CF uses HAProxy as the default load balancer for SSL termination. This guide describes how to use your own load balancer and forward traffic to the Elastic Runtime router IP.

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



7. 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.
8. In the left column, select **Router IPs**.
9. In the **Router IPs** field, enter the IP address(es) for Pivotal CF that you registered with your load balancer in Step 2. Save this change.



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

Using SSL with a Self-Signed Certificate

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.

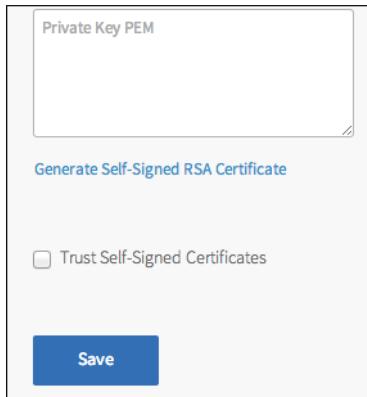
Follow these steps to use SSL with a self-signed certificate:

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.

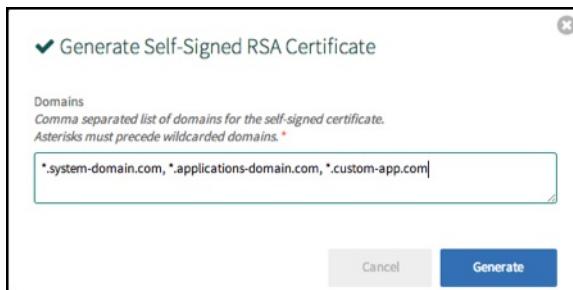
Generating a Self-Signed Certificate

To generate a self-signed certificate on your server:

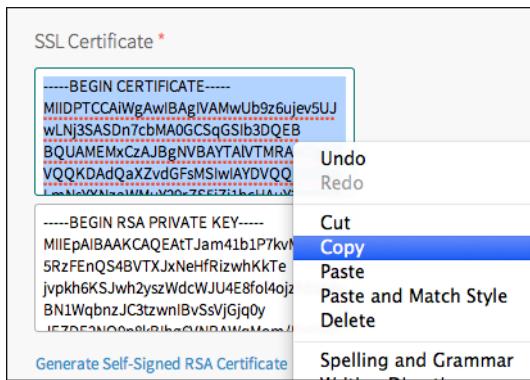
1. In Pivotal Ops Manager, click the **Pivotal Ops Metrics** tile.
2. Check **Enable SSL**.
3. Click **Generate Self-Signed RSA Certificate** and check the **Trust Self-Signed Certificates** box.



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 Ops Manager Resurrector on VMware vSphere

The Ops Manager Resurrector increases Elastic Runtime availability by:

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

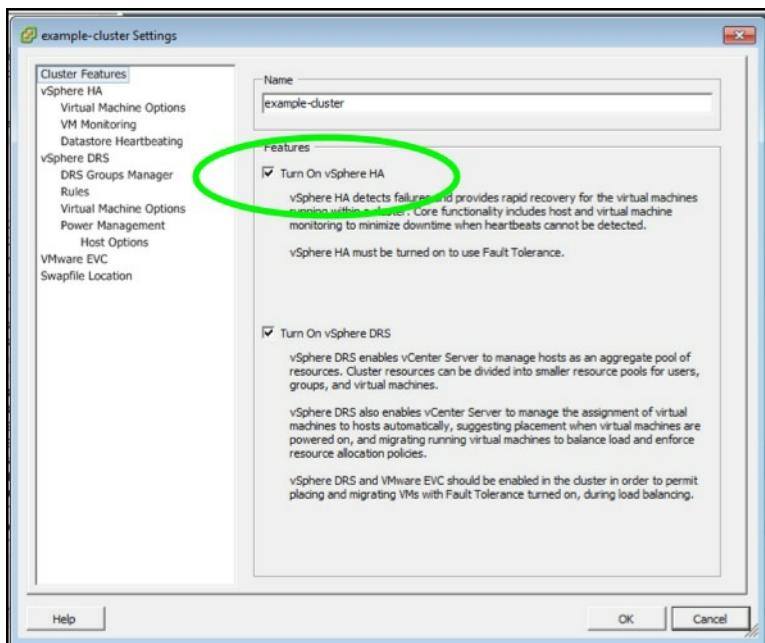
Ops Manager Resurrector continuously monitors the status of all virtual machines in an Elastic Runtime deployment. Additionally, the Resurrector monitors the BOSH Agent on each VM. If either the VM or the BOSH Agent fail, Resurrector restarts the virtual machine on another active host.

Note: The Ops Manager Resurrector cannot protect the virtual machine running Ops Manager. Pivotal recommends that you use vSphere High Availability to protect the Ops Manager VM.

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 CF deployment and select **Edit Settings**.
3. Check the **Turn on vSphere High Availability** checkbox.

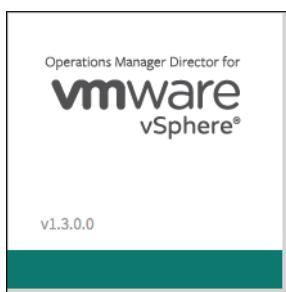


4. Click **OK** to enable vSphere High Availability on the cluster.

Enabling Ops Manager Resurrector

To enable the Ops Manager Resurrector:

1. Log into the Pivotal CF 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**.

The screenshot shows the "Director Config" settings page. On the left, a sidebar lists configuration tasks: vCenter Config, Director Config (which is selected and highlighted in grey), Create Availability Zones, Assign Availability Zones, Create Networks, Assign Networks, and Resource Config. The main panel is titled "Director Config" and contains two input fields: "NTP Servers (comma delimited)*" with the value "time.apple.com" and "Metrics IP Address" which is empty. A checked checkbox labeled "Enable VM Resurrector Plugin" is present. At the bottom is a blue "Save" button.

5. Ops Manager Resurrector is now enabled.

Connecting Elastic Runtime to LDAP

This topic describes connecting 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:

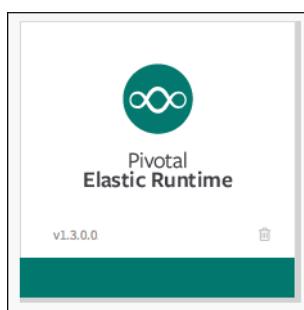
- Issues tokens for use by client applications when they act on behalf of Elastic Runtime users
- Authenticates users with their Elastic Runtime credentials
- Acts as an SSO service using Elastic Runtime or other credentials

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

For example, a domain named `cloud.example.com` typically uses the LDAP search base `ou=cloud,dc=example,dc=com`.

- **LDAP Search Filter:** A string that defines LDAP 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 surname equal to `Smith`.

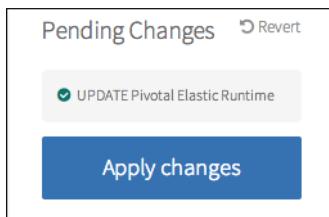
In the LDAP search filter string you use to configure Elastic Runtime, use `{0}` instead of the username. For example, use `cn={0}` to return all LDAP objects with the same surname as the username.

- **LDAP Server SSL Cert:** The server certificate to trust for self-signed certificates.
- **LDAP Server SSL Cert AltName:** The `AltName` of the self-signed SSL certificate.
- **Email Attribute:** The name of the attribute that contains the user email address. The default value is `mail`.

The form contains the following fields:

- LDAP Server URL
- LDAP Credentials
 - Username
 - Password
- LDAP Search Base
- LDAP Search Filter
- LDAP Server SSL Cert
- LDAP Server SSL Cert AltName
- Email Attribute *
 - mail

5. Click **Save**.
6. Click the **Installation Dashboard** link to return to the Installation Dashboard.
7. On the Installation Dashboard, click **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 \(UAAC\)](#).

Configuring Network Segmentation in Ops Manager

The network segmentation strategy you choose determines network traffic separation, and can have a significant impact on the security and performance of your deployment.

Network traffic within a Pivotal CF deployment takes one of three forms:

- **Management traffic:** Traffic connecting Pivotal CF to vCenter, as well as standalone SSH connections and vSphere client connections. This traffic may contain sensitive data.
- **Operations traffic:** Traffic to and from the Ops Manager Director associated with dynamic memory migration and storage operations. This traffic may contain the contents of the virtual machine dynamic memory or storage information from VMs.
- **Production traffic:** Traffic to and from product VMs.

You can choose one of the following network segmentation strategies:

- Full: Traffic forms separated by network
- Partial: Management and operations traffic separated from production traffic
- Single: All traffic on the same network

Full Network Segmentation

 **Note:** We recommend using full network segmentation in production environments because it provides the most security.

With full network segmentation, you install vCenter, the Ops Manager Director, and your products to different networks to separate management, operations, and production traffic.

Advantages

- Most secure because it separates the local network from the Internet and isolates each network traffic form to its own segment
- Best performance because it allocates hosts to separate networks, which results in the least local traffic

Disadvantages

- Most difficult network strategy to configure and manage

Configuring Full Network Segmentation

1. In vSphere, create three networks. For this example, call them network **A**, **B**, and **C**.
2. Install your vCenter on network A.
3. Deploy Ops Manager Director on network B. In the Ops Manager Director configuration interface, configure the Director as follows:
 - Click **Create Network**. Click **Add**. Add a network that maps to network A.
 - Click **Create Network**. Click **Add**. Add a network that maps to network B.
 - Click **Assign Network**. Click the **Infrastructure Network** drop-down menu and select the network that you created that maps to network A.
 - Click the **Deployment Network** drop-down menu and select the network that you created that maps to network B.

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

4. Install any other products on network C.
5. Ensure that network C can route to the IP address of the Ops Manager Director on the network you created that maps to network B.

[Return to the Getting Started Guide](#)

Partial Network Segmentation

With partial network segmentation, you install vCenter on one network and the Ops Manager Director and products on another. This separates management traffic from operations and production traffic.

Advantages

- Easier to configure and manage than the “Full Segmentation” strategy
- More secure than the “Single Network” strategy
- Better performance than the “Single Network” strategy

Disadvantages

- More difficult to configure and manage than the “Single Network” strategy
- Less secure than the “Full Segmentation” strategy
- Less performant than the “Full Segmentation” strategy

Configuring Partial Network Segmentation

1. In vSphere, create two networks. For this example, call them networks **A** and **B**.
2. Install your vCenter on network A.
3. Deploy Ops Manager Director on network B. In the Ops Manager Director configuration interface, configure the Director as follows:
 - Click **Create Network**. Click **Add**. Add a network that maps to network A.
 - Click **Create Network**. Click **Add**. Add a network that maps to network B.
 - Click **Assign Network**. Click the **Infrastructure Network** drop-down menu and select the network that you created that maps to network A.
 - Click the **Deployment Network** drop-down menu and select the network that you created that maps to network B.

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

4. Install any other products on network B.

[Return to the Getting Started Guide](#)

Single Network

With a single network, you deploy vCenter, the Ops Manager Director, and all products to the same network.

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

Advantages

- Easiest network strategy to configure and manage

Disadvantages

- Least secure because it does not separate the local network from the Internet and does not isolate the different network traffic forms
- Poor performance because all hosts on the same network results in a maximum amount of local traffic

Configuring for a Single Network

1. In vSphere, create one network. For this example, call it network **A**.
2. Install your vCenter on network A.
3. Deploy Ops Manager Director on network A. In the Ops Manager Director configuration interface, configure the Director as follows:
 - Click **Create Network**. Click **Add**. Add a network that maps to network A.
 - Click **Assign Network**. Click the **Infrastructure Network** drop-down menu and select the network that you created that maps to network A.
 - Click the **Deployment Network** drop-down menu and select the network that you created that maps to network A.

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

A

Save

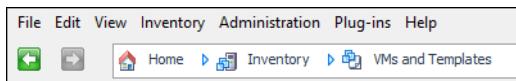
4. Install any other products on network A.

[Return to the Getting Started Guide](#)

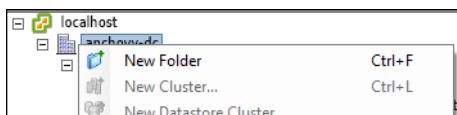
Deploying Operations Manager to vSphere

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.

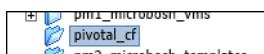
1. Refer to the [Known Issues](#) topic before getting started.
2. Download the Pivotal CF .ova file.
3. Log into vCenter.
4. Select the **VM and Templates** view.



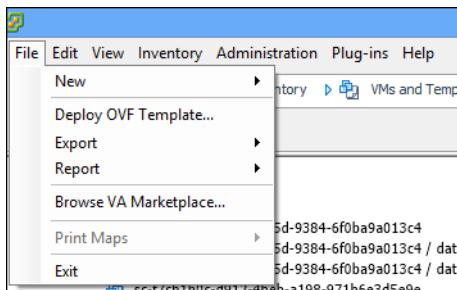
5. Right click on your datacenter and select **New Folder**.



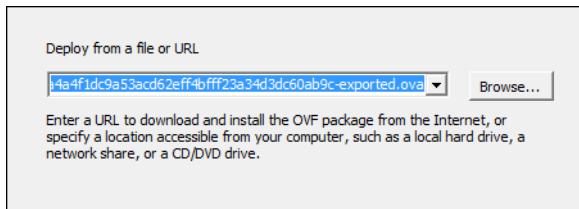
6. Name the folder "pivotal_cf" and select it.



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



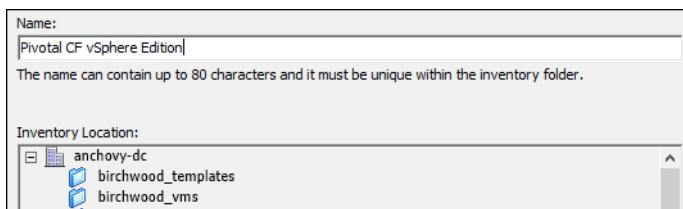
8. Select the .ova file and click **Next**.



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

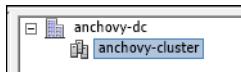
10. Accept the license agreement and click **Next**.

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



Note: The selected folder is the one you created.

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



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

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

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
172.16.64.2

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

Select a destination storage for the virtual machine files:

VM Storage Profile:	<input type="button" value="▼"/>					
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	315.55 GB	214.50 GB	VMFS5	Supports

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

Datastore:	anchovy-ds
Available space (GB):	4076.0
<input checked="" type="radio"/> Thick Provision Lazy Zeroed <input type="radio"/> Thick Provision Eager Zeroed <input type="radio"/> Thin Provision	

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

Source Networks	Destination Networks
Network 1	MattNetwork
	<input type="button" value="▼"/>
	MattNetwork
	VM Network
	VM Network Private

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 Pivotal CF 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'. Enter password <input type="password"/> Confirm password <input type="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 vCD

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

Configuring NAT Rules

Elastic Runtime accesses the Internet using a source NAT rule (SNAT).

Elastic Runtime's API endpoint, which is fronted by HAProxy, instance requires a destination NAT (DNAT) to forward traffic from a public IP. Ops Manager also requires a DNAT rule to connect to it from a public IP.

You need at least two public IP addresses for your virtual datacenter. One designated for Ops Manager and one for Elastic Runtime.

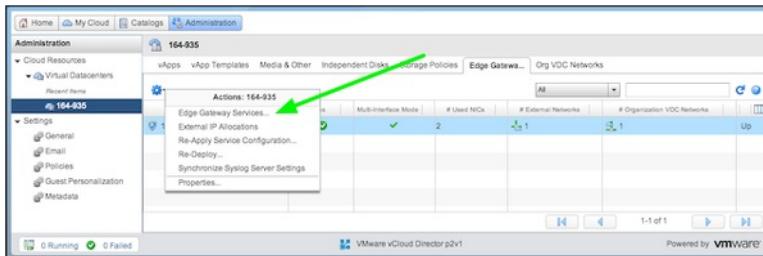
NAT rules are configured in the vShield Edge interface which will be described next.

Configuring vShield Edge

Follow these steps to configure vShield NAT rules. For more information about edge gateway services, see the [VMware vCloud Director](#) documentation.

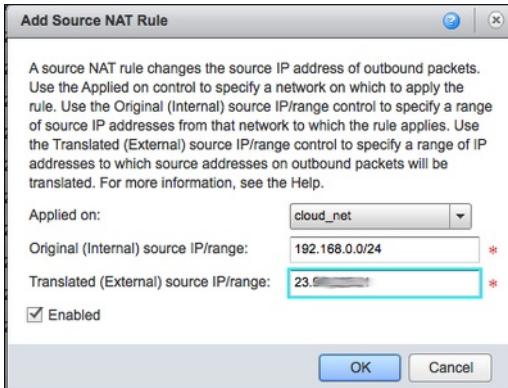
Open the vShield Edge Gateway Services Interface

1. Log into vCloud Air and click **Gateways**.
2. Click **Manage Advanced Gateway Settings** on the right side.
3. Select the gateway you want to configure, then click the gear icon and select **Edge Gateway Services**.

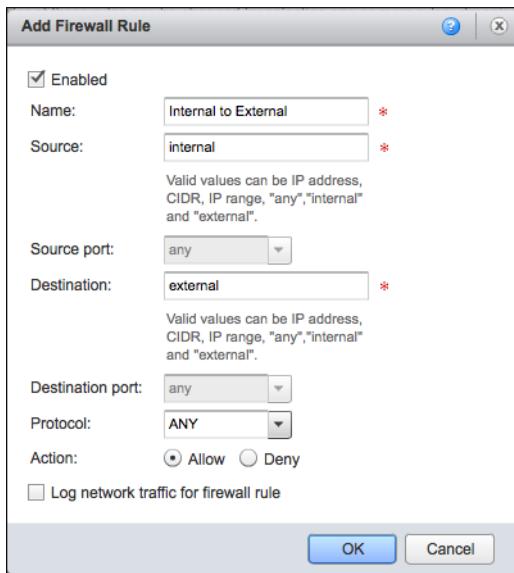


Creating an SNAT and Firewall Rule

1. Configure an SNAT rule allowing outbound connections through Ops Manager public IP address. Using the Elastic Runtime IP address can be problematic for DNS resolution.



2. Set up a firewall rule allowing outbound traffic from all internal IP addresses to all IP external addresses.



Allow Inbound SSH for Ops Manager

You may need to SSH to your Ops Manager VM which requires a DNAT rule.

1. Create a destination NAT (DNAT) rule with 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**
2. Create a 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

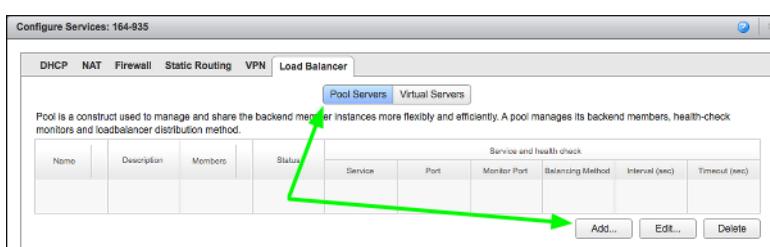
Repeat the steps above for ports 80, and 443 for the same public address.

Allow Inbound Web Traffic for Elastic Runtime

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

Set up Network Rules for Elastic Runtime DNS Resolution

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



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.

Add Load Balancer Member Pool

Configure Service

Service	Balancing Method	Port
HTTP	Round Robin	80
HTTPS	Round Robin	443
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**.

Add Load Balancer Member Pool

Configure Health-Check

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.

Edit Member

IP Address:	192.168.109.65	*
Ratio weight:	1	*
Specify how requests will be proportionately routed to an instance. Setting ratio weight to 0 will disable the member.		

7. Click **Finish**.

8. Click **Virtual Servers**.

DHCP NAT Firewall Static Routing VPN Load Balancer

Virtual Servers

Name	Description	Members	Status	Service and health check					
				Service	Port	Monitor Port	Balancing Method	Interval (sec)	Timeout (sec)
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, and HTTPS on port 443

11. Click **OK** to complete

Configure Services: 164-935

Virtual Servers

Name	IP Address	Description	Pool	Name	Port	Persistence	Logging	Enabled
Load Balancer	192.168.109.65		Load Balancer to Cloud Foundry	HTTP	80	None	-	✓
				HTTPS	443	Session		

Deploying Ops Manager to vCloud Air

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

Note: The vCloud Director Web Console (also on vCloud Air) only supports 32-bit browsers, including 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.

Locating Ops Manager on vCloud Air Public Catalog

1. Log into vCloud Air.
2. Choose the vDC env you want to deploy PCF
3. Click the link to open the vCD UI.
4. In the vDC UI, Navigate to **Catalogs > Public Catalog**.
5. Select **Pivotal Ops Manager vApp** from the Public Catalog.

Name	Version	Status
CentOS63-32bit	3	Ready
CentOS63-64bit	3	Ready
CentOS64-32bit	1	Ready
Pivotal Ops Manager VApp	1	Ready
Ubuntu Server 12.04 LTS (amd64 ...	4	Ready
Ubuntu Server 12.04 LTS (i386 201...	4	Ready

6. Click **Add to Cloud**.

Uploading Ops Manager to vCD

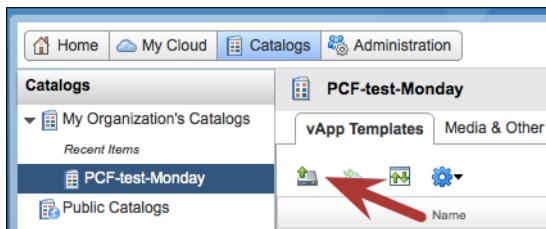
If you are not using vCloud Air, you must either upload the Ops Manager vApp into your catalog or use a vApp that was uploaded by a vCD administrator in your organization's catalog.

1. Download Pivotal CF Operations Manager for vCloud Air/vCloud from [Pivotal Network](#).
2. Log into vCloud.
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:

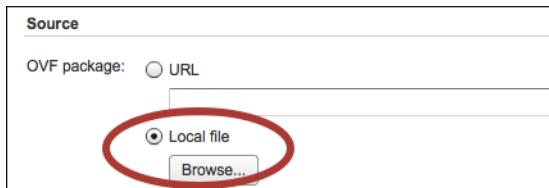
- Enter a name for the new catalog and click **Next**.
- Select a storage type and click **Next**.
- Specify sharing (if needed) and click **Next**.
- Review your settings and click **Finish**.

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

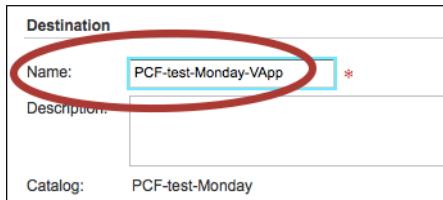


Note: The first time you upload software to vCloud Director, you are required to install the **Client Integration Plug-in** and restart all browsers.

- Select **Local file** and browse to your **ova** file.



- Enter a name for your Ops Manager vApp, enter a description (optional), and click **Upload**.

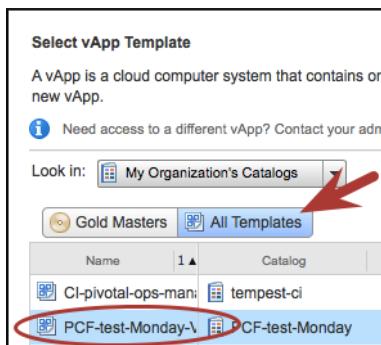


vCloud Director transfers the vApp Template (the OVF package) to a staging environment, then uploads it to your catalog.

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



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



Deploying Ops Manager

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

- Check the "I agree" checkbox to accept licenses and click **Next**.
- 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:	PCF-test-Monday-vApp	
Description:		

Virtual Datacenter

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

164-935	
---------	--

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:

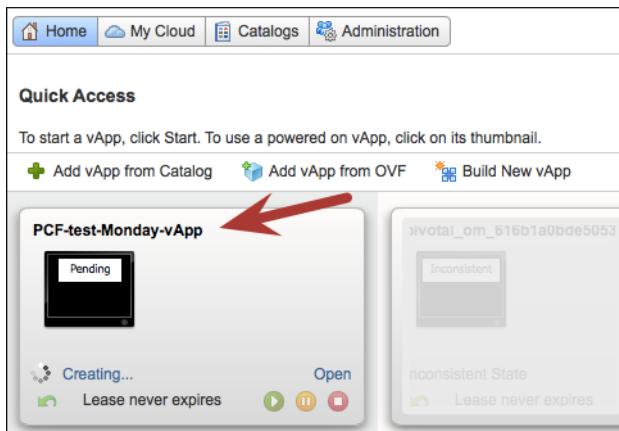
5. Enter the desired networking information, set an admin password for the Ops Manager vApp, and click **Next**.

Note: The order of the items on your screen may vary from the order shown here.

Uncategorized

DNS:	<input type="text"/>	The domain name servers for the Pivotal Ops Manager (comma separated). Leave blank if DHCP is desired.
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 'tempest'.
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

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.

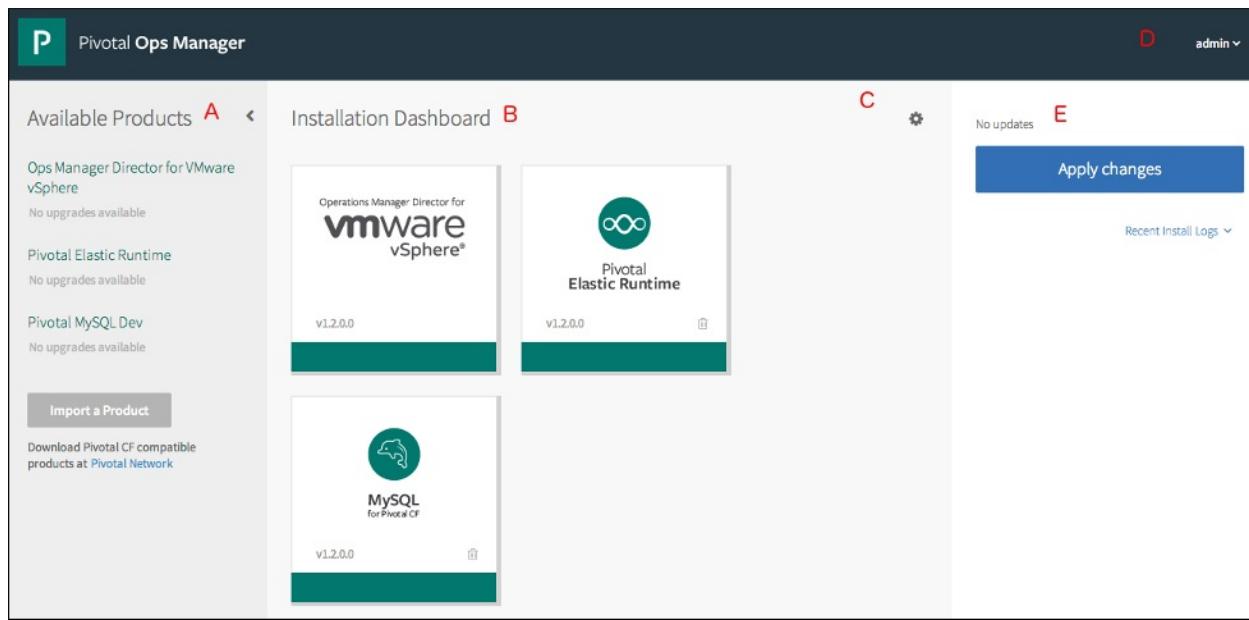


9. Once the VM boots, the interface is available at the IP address you specified.

[Return to the Getting Started Guide](#)

Understanding the Ops Manager Interface

This topic describes key features of the Pivotal CF 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 Pivotal CF 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.

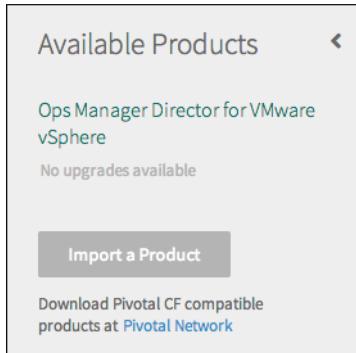
Note: When an update depends on prerequisites, the prerequisites automatically install first.

Adding and Deleting Products

Refer to this topic for help adding and deleting additional products from your Pivotal CF installation, such as Pivotal HD for Pivotal CF and Pivotal RabbitMQ.

Adding and Importing Products

1. Download Pivotal CF 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. To prevent a post-install 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 'Lifecycle Errands' section of the Pivotal MySQL Dev Installation Dashboard. At the top, there are tabs for 'Settings', 'Status', 'Credentials', and 'Logs'. The 'Lifecycle Errands' tab is selected. Below it, a sub-section titled 'Lifecycle Errands' is described as 'Lifecycle errands are scripts that run at designated points in time during an installation.' It lists two categories: 'Broker Registrar' and 'Post-Install Errands'. The 'Broker Registrar' entry includes a note: 'Registers broker with Cloud Controller and makes the 100mb plan public'.

For more information about lifecycle errands, see [Understanding Lifecycle Errands](#).

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**.

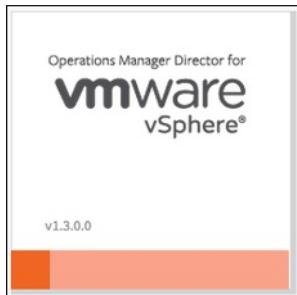
Note: You cannot delete the Ops Manager Director product.

2. In the Pending Changes view, click **Apply Changes**.
3. **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 VMware vSphere

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

1. Log in to Pivotal CF Operations Manager.
2. Click the **Ops Manager Director for VMware vSphere** tile.



3. Select **vCenter Config** and enter the following information:
 - vCenter IP address
 - vCenter credentials
 - Datacenter name
 - A comma-delimited list of datastore names

Click **Save**.

Installation Dashboard

Ops Manager Director for VMware vSphere

Settings Status Credentials Logs

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

vCenter Config

vCenter IP Address*
192.168.100.1

vCenter Credentials*
root

Datacenter Name*
drinks-dc

Datastore Names (comma delimited)*
coke-ds

Save

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

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**.
 - Click **Save**.

The screenshot shows the 'Director Config' section of the 'Settings' tab. It includes fields for 'NTP Servers (comma delimited)*' (set to 'time.nist.gov') and 'Metrics IP Address'. A checkbox for 'Enable VM Resurrector Plugin' is present. A 'Save' button is at the bottom.

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

The screenshot shows the 'Create Availability Zones' section of the 'Settings' tab. It includes fields for 'Name*' (set to 'az-1'), 'Cluster*' (set to 'drinks-cl'), and 'Resource Pool' (set to 'classic'). A 'Save' button is at the bottom.

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

The screenshot shows the 'Assign Availability Zones' step in the 'Settings' tab of the Ops Manager Director for VMware vSphere. The left sidebar lists steps: vCenter Config (checked), Director Config (checked), Create Availability Zones (unchecked), Assign Availability Zones (checked, highlighted with a grey arrow), Create Networks (unchecked), Assign Networks (unchecked), and Resource Config (checked). The main panel title is 'Assign Availability Zones'. It contains a note: '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.' Below this is a dropdown menu set to 'az-1' and a blue 'Save' button.

- 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 for VMware vSphere

- Settings
- Status
- Credentials
- Logs

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*

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 Sizes**, accept the defaults or make necessary changes, and click **Save**.

The screenshot shows the 'Settings' tab selected in the top navigation bar. The main content area displays configuration details for an 'Ops Manager Director' job. On the left, there are three checked items: 'vCenter credentials', 'vSphere configuration', and 'Network configuration'. To the right, a table provides resource allocation information:

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

A blue 'Save' button is located at the bottom right of the table.

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 CF Operations Manager.
2. Click the **VMware vCloud Air** tile.



3. Select **vCloud Air/vCloud Director credentials**, enter the URL of the vCloud Director and your credentials, and click **Save**.

Installation Dashboard

Ops Manager Director for VMware vCHS and vCloud Director

Settings Status Credentials Logs

vCHS/vCloud Director credentials

vCloud configuration

Network configuration

NTP servers

System Settings

Resource sizes

Credentials to connect to vCHS/vCloud Director

vCloud API URL *

The URL of the vCloud Director.

Organization name *

Login credentials *

Save

Notes:

- This user must have create and delete privileges for VMs and folders.
- The **Organization name** cannot include uppercase letters. Substitute lowercase letters for any uppercase letters present in the vCD organization name.

4. Select **vCloud configuration**, specify the datacenter name and storage profile name, and click **Save**.

[Installation Dashboard](#)

Ops Manager Director for VMware vCHS and vCloud Director

- [Settings](#)
- [Status](#)
- [Credentials](#)
- [Logs](#)

vCHS/vCloud Director credentials Configuration information for vCHS/vCloud

vCloud configuration Datacenter name * The name of the Director

Network configuration

NTP servers

System Settings

Resource sizes

Save

5. Select **Network configuration**, enter the network name, subnet, Pivotal CF IP ranges, DNS, and Gateway settings, and click **Save**. The Pivotal CF VMs will use the IP range you specify. **Reserved IP Ranges** are blacklisted from the installation.

[Installation Dashboard](#)

Ops Manager Director for VMware vCHS and vCloud Director

- [Settings](#)
- [Status](#)
- [Credentials](#)
- [Logs](#)

vCHS/vCloud Director credentials Network configuration for vCloud environment

vCloud configuration Network name * The name of the network

Network configuration

NTP servers

System Settings

Resource sizes

Reserved IP ranges

DNS *

Gateway *

Save

6. Select **NTP servers**, enter a comma-delimited list of time server addresses, and click **Save**.

[Installation Dashboard](#)

Ops Manager Director for VMware vCHS and vCloud Director

- [Settings](#)
- [Status](#)
- [Credentials](#)
- [Logs](#)

vCHS/vCloud Director credentials

vCloud configuration

Network configuration

NTP servers

System Settings

Resource sizes

NTP configuration

NTP servers (comma delimited) *

us.pool.ntp.org

One time

Save

7. Select **System Settings**, check or uncheck **Enable VM resurrector plugin**, and click **Save**.

[Installation Dashboard](#)

Ops Manager Director for VMware vCHS and vCloud Director

- [Settings](#)
- [Status](#)
- [Credentials](#)
- [Logs](#)

vCHS/vCloud Director credentials

vCloud configuration

Network configuration

NTP servers

System Settings

Resource sizes

Settings for Ops Manager Director

Enable VM resurrector plugin

Save

8. Select **Resource Sizes**, accept the defaults or make necessary changes, and click **Save**.

[Installation Dashboard](#)

Ops Manager Director for VMware vCHS and vCloud Director

- [Settings](#)
- [Status](#)
- [Credentials](#)
- [Logs](#)

vCHS/vCloud Director credentials

vCloud configuration

Network configuration

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

Save

9. 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 Developer Console, 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 Developer Console.

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 Developer Console 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 Developer Console. Refer to the Pivotal CF Developer Console documentation at docs.pivotal.io for more information about using the Developer Console.

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 Developer Console

Complete the following steps to log in to the Developer Console:

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 Developer Console using the UAA Administrator User credentials. In Pivotal Ops Manager, refer to **Pivotal Elastic Runtime > Credentials** for these credentials.

Controlling Console User Activity with Environment Variables

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

Understanding the Developer Console Environment Variables

You can control which users can create orgs and perform user management actions on the Developer Console 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 console 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 Developer Console with the new environment variable value.

Backing Up Pivotal CF

Refer to this topic for help backing up the contents of critical databases and backing up and restoring your Pivotal CF installation settings. Pivotal recommends that you back up your installation frequently.

 **Note:** Contact Pivotal Support for help restoring your Pivotal CF installation from the backups you created.

Back up and Restore Installation Settings

It is important to 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.

Exporting an installation

From the Product Dashboard, select **Actions > Export installation**. This option is only available after you have deployed at least one time.

Export installation exports the current Pivotal CF 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).

Importing an installation

From the Product Dashboard, select **Actions > Import installation**.

Import installation imports an existing Pivotal CF installation with all of its assets. When you import an installation, the prior installation will disappear and be replaced by the newly imported one.

Back Up the Cloud Controller DB Encryption Key

From the Product 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 / db0ef78a8b8e7af3
	Application staging upload credentials	staging_upload_user / e657e470a3ae0aab4b6f
	Bulk api credentials	bulk_api / 0d249db9ea47194fb97f
	Cloud Controller DB encryption credentials	db_encryption / 59011b5b6f6ec268ed50
	Cloud Controller DB Encryption Key	

Back Up Critical Databases

Refer to this section for help backing up and restoring databases associated with your Pivotal CF installation. You must restore the contents of these databases in order to have a functional restored PaaS. Once you have restored these databases, you must manually redeploy all apps.

Your Elastic Runtime deployment contains several critical data stores. You will back up each of them in turn. They are:

- Cloud Controller Database
- UAA Database
- Console Database
- The NFS Server

Download the BOSH Deployment Manifest

1. Install Ruby and the [BOSH CLI Ruby gem](#) on a machine outside your Pivotal CF system deployment.
2. In Ops Manager, browse to **Ops Manager Director > Status > Ops Manager Director** and note the IP address. This is the BOSH director IP address.

The screenshot shows the 'Status' tab of the 'Ops Manager Director' section in the Ops Manager interface. It displays a table with columns for 'JOB', 'INDEX', 'IPS', and 'CLOUD'. A single row is present for 'Ops Manager Director' with values 0, 10.0.0.3, and some redacted data. The IP address 10.0.0.3 is highlighted.

3. In Ops Manager, browse to **Ops Manager Director > Credentials > Ops Manager** and note the Director credentials.

The screenshot shows the 'Credentials' section for 'Ops Manager Director'. It lists four entries: 'VM credentials', 'BOSH agent credentials', 'Director credentials' (which is circled in red), and 'NATS credentials'. The 'Director credentials' row contains the 'director' user information.

4. Target the BOSH director using the address and credentials you noted:

```
$ bosh target 172.16.66.17
Target set to `microbosh-3ac4e7afaf8752dd349'
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. Locate the name of the current BOSH deployment:

```
$ bosh deployments
+-----+-----+-----+
| Name | Release(s) | Stemcell(s) |
+-----+-----+-----+
| cf-bee19d9c7d2f0c4fce8a | cf/170           | bosh-vsphere-esxi-ubuntu/2366 |
|                           | push-console-release/23   |
|                           | runtime-verification-errands/2 |
+-----+-----+-----+
Deployments total: 1
```

6. Use `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. The `DEPLOYMENT-NAME` is the name of the current BOSH deployment. Use `cf.yml` as the `LOCAL-SAVE-NAME`.

Example:

```
$ bosh download manifest cf-bee19d9c7d2f0c4fce8a cf.yml
Deployment manifest saved to `cf.yml'
```

Back Up the Cloud Controller Database

1. Run `bosh status` and record the UUID.

Example:

```
$ bosh status
Config
  /Users/pivotal/.bosh_config
Director
  Name      microbosh-vcloud-b340a7ecc329a7573fd7
  URL       https://192.240.155.231:25555
  Version   1.2524.0 (release:0501e461 bosh:0501e461)
  User      director
  UUID     5467db1e-1f2f-4558-b950-cd3c357d65c4
  CPI       vcloud
  dns      enabled (domain_name: microbosh)
  compiled_package_cache disabled
  snapshots disabled

Deployment
  not set
```

2. Open the downloaded `cf.yml` file in a text editor. Replace the value for `director_uuid:` with the UUID found by running `bosh status`.

For this example, `cf.yml` includes the line:

```
director_uuis: 5467db1e-1f2f-4558-b950-cd3c357d65c4
```

3. Use `bosh deployment cf.yml` to set the `cf.yml` file as your deployment manifest.

4. Stop the Cloud Controller process:

First, run the following command. Ensure that each section reports `No changes`.

```
$ bosh stop cloud_controller
You are about to stop cloud_controller/0
Detecting changes in deployment...

Releases
No changes

Compilation
No changes

Update
No changes

Resource pools
No changes

Networks
No changes

Jobs
No changes

Properties
No changes
```

Note: If the `bosh stop cloud_controller` command indicates that changes are pending, terminate the backup process and contact Pivotal Support. Proceeding with the deployment if there are pending changes will apply any enqueued changes, possibly putting your installation in a bad state. Backups are only supported with a clean installation.

If the output indicates that there are no changes, run the `--force` version of the same command:

```
bosh stop cloud_controller --force
```

5. In `cf.yml`, locate the ccdb component and note the address and password:

```
ccdb:
  address: 172.16.78.25
  port: 2544
  db_scheme: postgres
  roles:
    - tag: admin
      name: admin
      password: *****
```

6. Dump the data from the CCDB and save it:

```
pg_dump -h 172.16.78.25 -U admin -p 2544 ccdb > ccdb.sql
```

Note: Ensure that the `pg_dump` client version is the same as the version on the server.

Back Up the UAA Database

1. In `cf.yml`, locate the `uaadb` component and note the address and password:

```
uaadb:  
  address: 172.16.78.30  
  port: 2544  
  db_scheme: postgresql  
  roles:  
    - tag: admin  
      name: root  
      password: *****
```

2. Dump the data from the UAADB and save it:

```
pg_dump -h 172.16.78.30 -U root -p 2544 uaa > uaa.sql
```

Back Up the Console Database

1. In `cf.yml`, locate the console component and note the address and password:

```
consolodb:  
  address: 172.16.78.33  
  port: 2544  
  db_scheme: postgresql  
  roles:  
    - tag: admin  
      name: root  
      password: *****
```

2. Dump the data from the Console DB and save it:

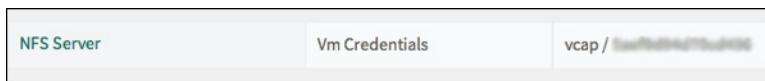
```
pg_dump -h 172.16.78.33 -U root -p 2544 console > console.sql
```

Back Up the NFS Server

1. In `cf.yml`, locate the NFS component and note the address:

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

2. In Ops Manager, browse to **Elastic Runtime > Credentials** and note the NFS server credentials.



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

```
ssh vcap@10.0.0.10 'cd /var/vcap/store && tar cz shared' > nfs.tar.gz
```

Note: The TAR file you create to back up the NFS server might be very large. To estimate the size of the TAR file before you create it, run the following command: `tar -cf - /dir/to/archive/ | wc -c`

Start Cloud Controller

Run `bosh start cloud_controller` to restart the Cloud Controller.

Upgrading Operations Manager

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

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

Starting with a Clean Slate

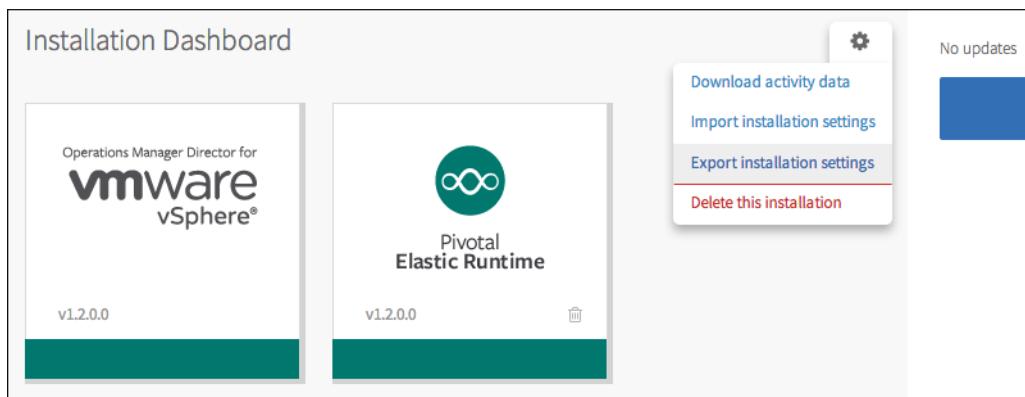
Follow these steps to upgrade Pivotal CF Operations Manager if you have no installed products, or you have no installed products you want to keep.

1. Browse to the Pivotal CF Ops Manager web interface and select **Delete 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](#)

Upgrading with Installed Products

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

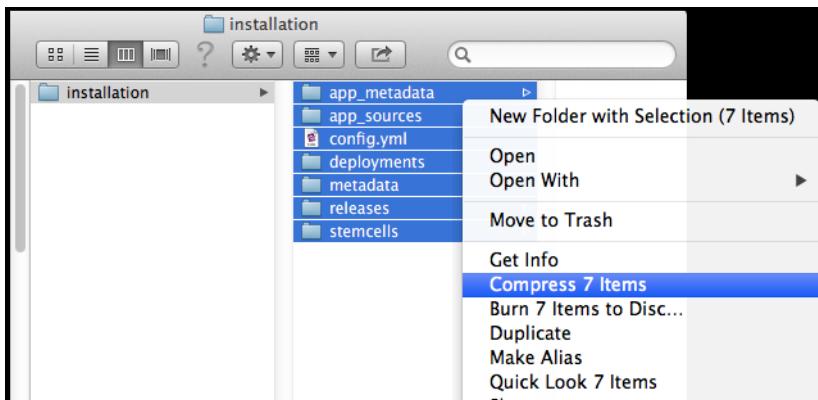
1. From the Product Dashboard, select **Actions > Export installation settings**.



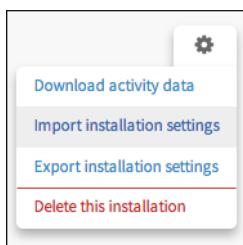
This exports the current Pivotal CF installation with all of its assets. When you export an installation, the export contains the base VM images and necessary packages, and references to the installation IP addresses. As a result, an exported file can be very large, as much as 5 GB or more.

- 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.

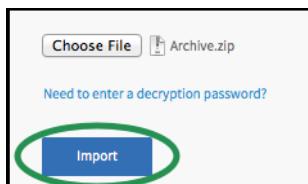
Note: When creating a zip file of an unzipped export, do not start compressing at the “installation” folder level. Instead, start compressing at the level containing the config.yml:



2. Download the latest Ops Manager VM Template from the [Pivotal Network](#) site.
3. Deploy the new Ops Manager VM. See [Deploying Operations Manager to vSphere](#) or [Deploying Operations Manager to vCloud Air and vCloud](#).
To avoid IP conflicts, use a different interface IP than the IP of the existing interface.
4. From the Product Dashboard, select **Actions > Import installation settings**.

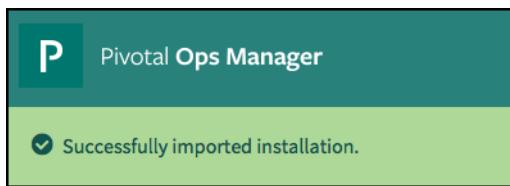


5. Click **Choose File**, browse to the installation zip file exported in Step 1, and click **Choose**.
6. Click **Import**.



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.

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



2. Verify the new installation functions correctly.
3. In vSphere, vCloud Air, or vCloud:
 - Record the IP address of the original Ops Manager VM.
 - Power off and delete the original Ops Manager VM.
 - Change the IP address of the newly installed Ops Manager VM to that of the original VM.

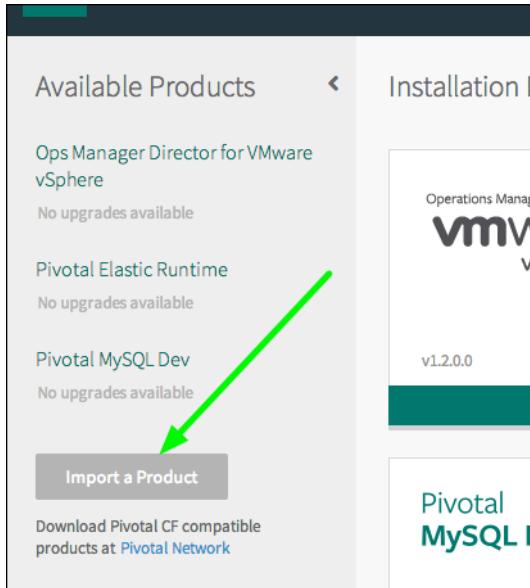
Note: Independently from upgrading Ops Manager, you can upgrade individual products such as Elastic Runtime, Pivotal MySQL, or RabbitMQ in your Pivotal CF deployment. See [Upgrading Products in a Pivotal CF Deployment](#).

Upgrading Products in a Pivotal CF Deployment

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

This topic describes how to upgrade individual products like Elastic Runtime, Pivotal MySQL, or RabbitMQ in your Pivotal CF deployment.

1. Browse to [Pivotal Networks](#) and sign in.
2. Download the latest Pivotal CF release for the product or products you want to upgrade.
3. Browse to the Pivotal CF Ops 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 Pivotal CF

This topic covers strategies for monitoring VM status and performance in Pivotal CF.

Monitoring VMs Using the Ops Manager Interface

Click any product tile and select the **Status** tab to view monitoring information.

Pivotal Elastic Runtime												
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 vm-6d43e59e-	0.12%	0.1%	9.7%	0.0%	41%	21%	N/A	Download	

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 Pivotal CF 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 CF 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.

Installation Dashboard

Metrics

Settings Status Credentials Logs

Assign Networks

Assign Availability Zones

JMX Provider

Resource Config

default

Save

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

Metrics

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 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
BQUAMDsxCzJBgNVBAYTANTMRawDgY
-----BEGIN RSA PRIVATE KEY-----
MIIEpjBAAKCAQEazINFM7S/rpn1rLyKM
aK1lyYFygii/NB9Z6E51SwXfCapS
HSof6zER1rSijwtgZ7nHobVjmp3UnWV5q
lxSseKpgSJaQkM+u5/zbwZj4gAg+FSo
-----END RSA PRIVATE KEY-----
```

[Generate Self-Signed RSA Certificate](#)

Save

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

Installation Dashboard

Pending Changes

INSTALL Metrics
 UPDATE Pivotal Elastic Runtime

Apply changes

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

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

- 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**.

The screenshot shows the 'Director Config' section of the 'Ops Manager Director for VMware vSphere' settings. On the left, there is a sidebar with several configuration items, each preceded by a green checkmark. The items listed are: vCenter Config, Director Config (which is currently selected and highlighted with a grey background), Create Availability Zones, Assign Availability Zones, Create Networks, Assign Networks, and Resource Config. To the right of the sidebar, the main panel has a title 'Director Config'. It contains two input fields: 'NTP Servers (comma delimited)*' with the value 'time.apple.com' and 'Metrics IP Address' with the value '10.85.52.127'. Below these fields is a checkbox labeled 'Enable VM Resurrector Plugin' which is unchecked. At the bottom right of the panel is a blue 'Save' button.

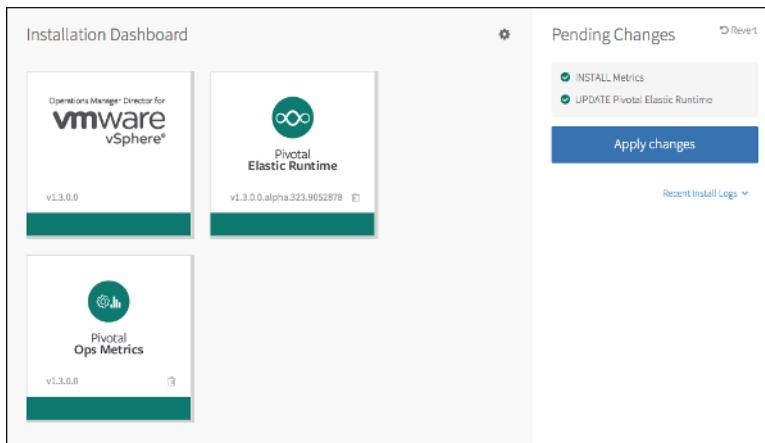
12. Return to the **Installation Dashboard**. Click the **Pivotal Elastic Runtime** tile and select **Resource Config**.

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

The screenshot shows the 'Resource Config' section of the 'Pivotal Elastic Runtime' settings. On the left, there is a sidebar with several configuration items, each preceded by a green checkmark. The items listed are: Assign Networks, Assign Availability Zones, HAProxy, Router IPs, Cloud Controller, External Endpoints, SSO Config, LDAP Config, SMTP Config, Lifecycle Errands, and Resource Config (which is currently selected and highlighted with a grey background). To the right of the sidebar, the main panel has a title 'Resource Config'. It displays a table with two columns: 'JOB' and 'INSTANCES'. The table lists various jobs and their corresponding instance counts. The 'Collector' job is explicitly shown with an instance count of 1. Other jobs listed include HAProxy, NATS, etcd, Health Manager, NFS Server, Cloud Controller Database, Cloud Controller, Clock Global, Cloud Controller Worker, Router, and DEA, all of which have an instance count of 1. The table has a total of 16 rows.

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**.



- 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 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)

Pivotal CF Troubleshooting Guide

This guide provides help with diagnosing and resolving issues encountered during a Pivotal CF install. For help troubleshooting issues that are specific to Pivotal CF 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 Pivotal CF 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.

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 Pivotal CF 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 Pivotal CF `.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 Pivotal CF 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.

Common Problems

Compare evidence that you have gathered to the problem descriptions below. If your problem is covered, try the recommended remediation procedures.

BOSH does not reinstall

You might want to reinstall BOSH for troubleshooting purposes. However, if Pivotal CF 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.

- 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-aea6a0386e18e", "id"=>"vm-360"}, "agent\_id"=>"56aea4ef-6aa9-4c39-8019-7024ccfdde4", "networks"=>{"default"=>{"ip"=>"192.168.86.19", "netmask"=>"255.255.255.0", "cloud\_properties"=>{"name"=>"VMNetwork"}, "default"=>["dns", "gateway"], "dns"=>["192.168.86.2", "192.168.86.17"], "gateway"=>"192.168.86.2", "dns\_record\_name"=>"0.nats.default.cf-d729343071061.microbosh", "mac"=>"00:50:56:9b:71:67"}, "disks"=>{"system"=>0, "ephemeral"=>1, "persistent"=>{}}, "ntp"=>[], "blobstore"=>{"provider"=>"dav", "options"=>{"endpoint"=>"http://192.168.86.17:25250", "user"=>"agent", "password"=>"agent"}, "mbus"=>"nats://nats:nats@192.168.86.17:4222", "env"=>{"bos"=>{"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

Scenario 1: Your Pivotal CF install exits with the following 403 error when you attempt to log in to the Developer Console:

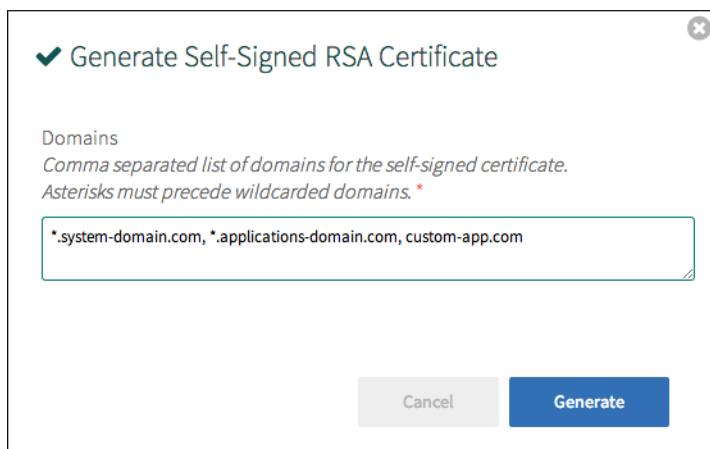
```
{"type": "step_finished", "id": "console.deploy"}  
/home/tempest-web/tempest/web/vendor/bundle/ruby/1.9.1/gems/mechanize-2.7.2/lib/mechanize/http/agent.rb:306:in  
'fetch': 403 => Net::HTTPForbidden for https://login.api.x.y/oauth/authorizeresponse_type=code&client_id=portal&redirect_uri=https%3...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

Scenario 2: Your Pivotal CF 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:

- Browse to the Operations Manager interface IP.
- Click the **Elastic Runtime** tile.
- Select **HAProxy** and click **Generate Self-Signed RSA Certificate**.
- 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 > 0

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

Pivotal CF displays an `Out of Disk Space` error if log files expand to fill all available disk space. If this happens, rebooting the Pivotal CF 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 Pivotal CF VM is incorrectly specified when deploying the Pivotal CF .ova file, installing Ops Manager Director fails at the “Installing Micro BOSH” step.

To resolve this issue, correct the DNS settings in the Pivotal CF Virtual Machine properties.

Installing Elastic Runtime fails

If the DNS information for the Pivotal CF 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 Pivotal CF Virtual Machine properties.

Viewing Developer Console Logs in a Terminal Window

The [Developer Console](#) provides a graphical user interface to help manage organizations, users, applications, and spaces.

When troubleshooting Developer Console performance, you might want to view the Developer Console application logs. To view the Developer Console application logs, follow these steps:

1. Run `cf login -a api.MY-SYSTEM-DOMAIN -u admin` from a command line to log in to Pivotal CF 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 Developer Console logs.

```
$ cf logs console  
Connected, tailing logs for app console in org system / space console as  
admin...
```

Changing Logging Levels for the Developer Console

The Developer Console recognizes the `LOG_LEVEL` environment variable. The `LOG_LEVEL` environment variable allows you to filter the messages reported in the Developer Console log files by severity level. The Developer Console defines severity levels using the Ruby standard library [Logger class ↗](#).

By default, the Developer Console `LOG_LEVEL` is set to `info`. The logs show more verbose messaging when you set the `LOG_LEVEL` to `debug`.

To change the Developer Console `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 Developer Console 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.

Troubleshooting Ops Manager for VMware vSphere

This guide provides help with diagnosing and resolving issues that are specific to Pivotal CF deployments on VMware vSphere.

For infrastructure-agnostic troubleshooting help, refer to the [Pivotal CF Troubleshooting Guide](#).

Common Issues

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

Pivotal CF Installation Fails

If you modify the vCenter Statistics Interval Duration setting from its default setting of 5 minutes, the Pivotal CF installation might fail at the MicroBOSH or BOSH deployment stage. Run an rb stack trace and search for an invoke method failure in `adapter_stub.rb` that has the following error message: `The specified parameter is not correct, interval`.

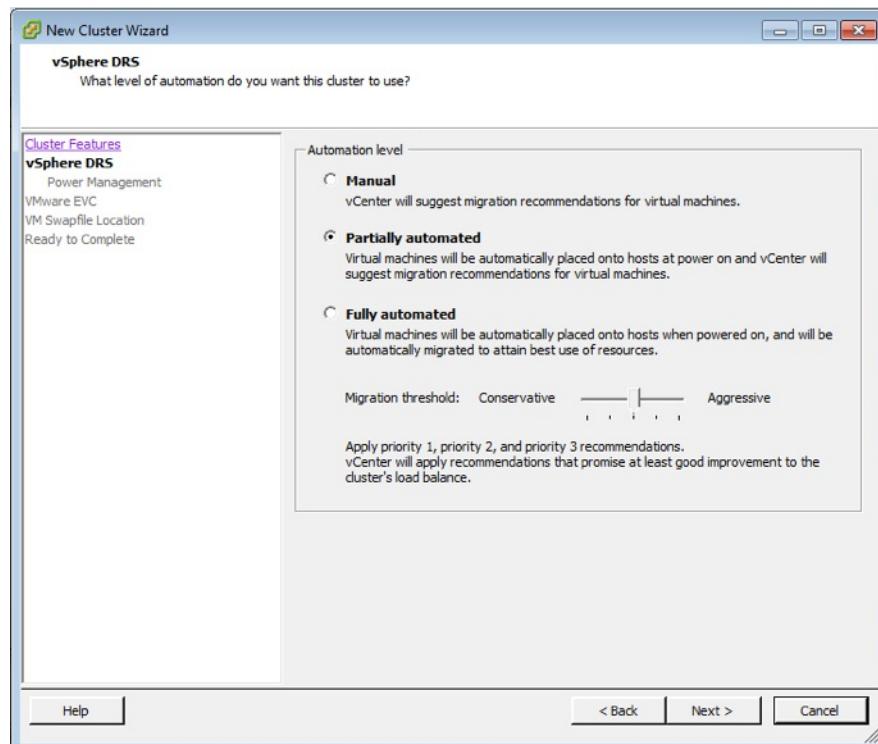
To resolve this issue, launch vCenter, navigate to **Administration > vCenter Server Settings > Statistics**, and reset vCenter Statistics Interval Duration setting to 5 minutes.

BOSH Automated Installation Fails

Before starting a 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 to workaround bug https://www.pivotaltracker.com/story/show/74705618
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
```

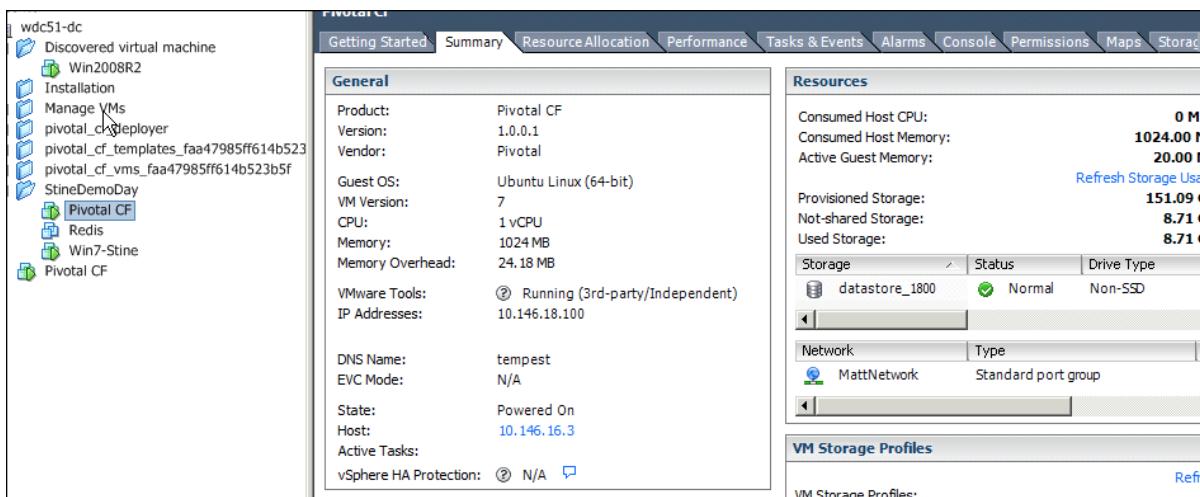
After editing the file run `sudo /etc/init.d/networking restart`. The interfaces files will be overwritten on reboot.

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 Pivotal CF 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 Pivotal CF .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.

Advanced Troubleshooting with the BOSH CLI

This page assumes you are running cf 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 CF installation.

 **Note:** For more troubleshooting information, refer to the [Troubleshooting Guide](#).

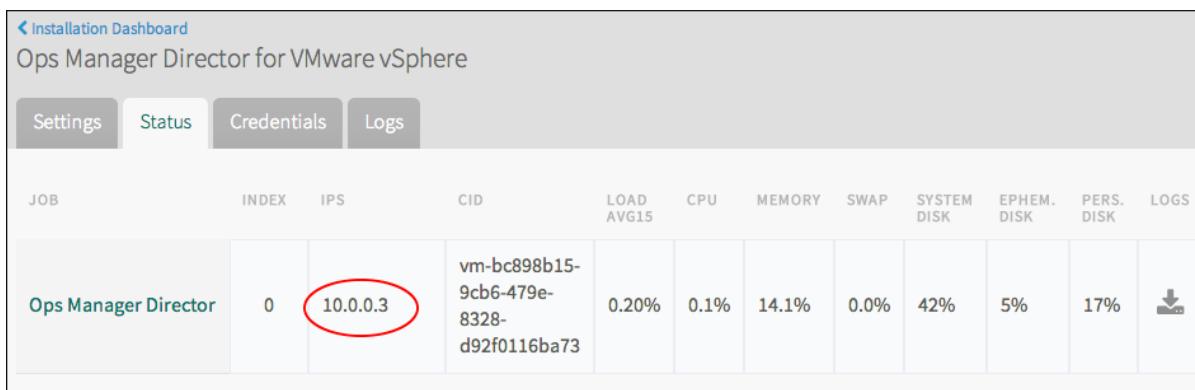
Prepare to Use the BOSH CLI

This section guides you through preparing to use the BOSH CLI.

Gather Information

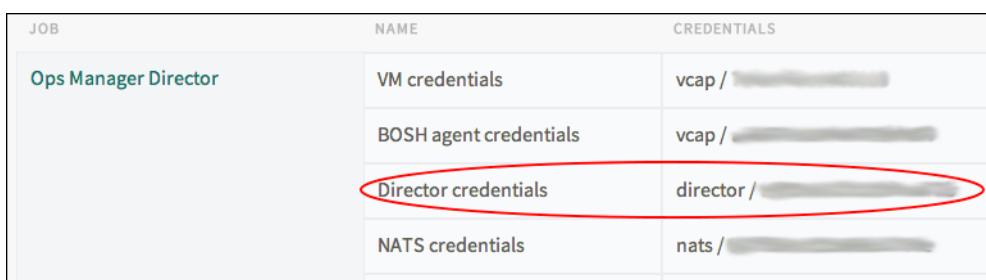
Before you begin troubleshooting with the BOSH CLI, collect the information you need from the Ops Manager interface. You must log out of the Ops Manager interface to use the BOSH CLI.

1. Open the Ops Manager interface. Ensure that there are no installations or updates in progress.
2. Click the **Ops Manager Director** tile and select the **Status** tab.
3. Record the IP address for the Ops Manager Director job. This is the IP address of the VM where the BOSH Director runs.



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

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



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

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. You will need the credentials used to import the Pivotal CF .ova or .ovf file into your virtualization system.

To SSH into the Ops Manager VM:

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: *****
```

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:

```
$ export BUNDLE_GEMFILE=/home/tempest-web/tempest/web/bosh.Gemfile
```

Note: This manual change to the `BUNDLE_GEMFILE` environment variable lasts 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 `bundle exec bash target OPS-MANAGER-DIRECTOR-IP-ADDRESS` to target your Ops Manager VM using the BOSH CLI.
4. Log in using the BOSH Director credentials:

```
$ bundle exec bash 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 `bundle exec bash deployment DEPLOYMENT-FILENAME.yml` to instruct the BOSH Director to apply CLI commands against the deployment described by the YAML file you identified:

```
$ bundle exec bash 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.

```
$ bundle exec bash 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

```
$ bundle exec 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

```
$ bundle exec 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

```
$ bundle exec 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

```
$ bundle exec 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 `bundle exec 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:

```
$ bundle exec 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
```

Pivotal CF Security Overview and Policy

This document outlines our security policy and is addressed to operators deploying Pivotal CF using Pivotal CF Operations Manager.

For a comprehensive overview of the security architecture of each Pivotal CF 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

Pivotal CF has many customer stakeholders who need to know about security updates. When there is a possible security vulnerability identified for a Pivotal CF component, we do the following:

1. Assess the impact to Pivotal CF.
2. If the vulnerability would affect a Pivotal CF 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 Pivotal CF 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](#)
- [Scaling Cloud Foundry](#)
- [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.

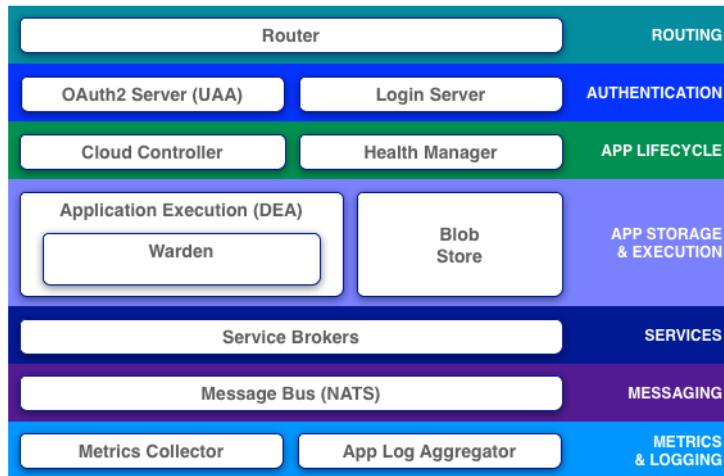
Who should use Cloud Foundry?

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, 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 c29tZXVzZXI6c29tZXBhc3M=  
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5  
> Host: 127.0.0.1:8080  
> Accept: */*  
>  
< HTTP/1.1 200 OK  
< Content-Type: application/json  
< Date: Mon, 25 Mar 2013 20:31:27 GMT  
< Transfer-Encoding: chunked  
<  
{"0295dd314aaf582f201e655cbd74ade5.cloudfoundry.me": ["127.0.0.1:34567"],  
 "03e316d6aa375d1dc1153700da5f1798.cloudfoundry.me": ["127.0.0.1:34568"]}
```

User Account and Authentication (UAA) Server

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. It can also 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

Fetch and install the UAA from Git:

```
$ git clone git://github.com/cloudfoundry/uaa.git  
$ cd uaa  
$ mvn install
```

Each module has a `mvn tomcat:run` target to run individually. Alternatively, you could import the modules as projects into STS (2.8.0 or better). The apps all work together and run on the same port (8080) with endpoints `/uaa`, `/app` and `/api`.

You will need Maven 3.0.4 or newer.

Deploy to Cloud Foundry

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

```
$ mvn package install  
$ cf push myuaa --path uaa/target
```

If you do that, choose a unique application id, not `myuaa`.

Local Command Line Usage

First run the UAA server as described above:

```
$ cd uaa  
$ mvn tomcat: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",  
  "commit_id": "111274e",  
  "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 client like `cf`. 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:

```
exp: 1355348409
user_name: marissa
scope: cloud_controller.read openid password.write scim.userids tokens.read tokens.write
email: marissa@test.org
aud: scim tokens openid cloud_controller password
jti: ea2fac72-3f51-4c8f-a7a6-5ffc117af542
user_id: ba14fea0-9d87-4f0c-b59e-32aaa8eb1434
client_id: cf
```

Remote Command Line Usage

The command line example in the previous section should work against a UAA running on the hosted Pivotal CF instance, 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.pivot.al.io/login
{
  "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
$ uaac target uaa.run.pivot.al.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 that a client like `cf` uses.

Integration tests

With all apps deployed into a running server on port 8080, the tests will include integration tests (a check is done before each test that the app is running). You can deploy them in your IDE or using the command line with `mvn tomcat:run`, then run the tests as normal.

For individual modules, or for the whole project, you can also run integration tests and the server from the command line:

```
$ mvn test -P integration
```

This might first require an initial `mvn install` from the parent directory to get the WARs in your local repo.

To make the tests work in various environments, you can modify the configuration of the server and the tests (e.g. the admin client) using a variety of mechanisms. The simplest is to provide additional Maven profiles on the command line:

```
$ (cd uaa; mvn test -P vcap)
```

This runs the integration tests against a UAA server running in a local vcap, so for example the service URL is set to `uaa.vcap.me` (by default). There are several Maven profiles to play with, and you can use them to run the server, the tests, or both:

- `local`: runs the server on the ROOT context `http://localhost:8080/`
- `vcap`: also runs the server on the ROOT context and points the tests at `uaa.vcap.me`.
- `devuaa`: points the tests at `http://devuaa.cloudfoundry.com` (an instance of UAA deployed on cloudfoundry) *NB this*

information is currently outdated

All these profiles set the `CLOUD_FOUNDRY_CONFIG_PATH` to pick up a `uaa.yml` and (if appropriate) set the context root for running the server.

Integration Tests

There is a simple cucumber feature spec (`--tag @uaa`) to verify that the UAA server is there. There is also a rake task to launch the integration tests from the `uaa` submodule in `vcap`. Typical usage for a local (`uaa.vcap.me`) instance:

```
$ cd vcap/tests  
$ rake bvt:run_uaa
```

You can change the most common important settings with environment variables (see below), or with a custom `uaa.yml`.

Note: You cannot use `MAVEN_OPTS` to set JVM system properties for the tests, but you can use it to set memory limits for the process, etc.

Custom YAML Configuration

To modify the runtime parameters, you can provide a `uaa.yml`:

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

then from `vcap-tests`:

```
$ CLOUD_FOUNDRY_CONFIG_PATH=/tmp rake bvt:run_uaa
```

or from `uaa/uaa`:

```
$ CLOUD_FOUNDRY_CONFIG_PATH=/tmp mvn test
```

The integration tests look for a YAML file in the following locations (later entries override earlier ones), and the webapp does the same when it starts up, so you can use the same config file for both:

```
classpath:uaa.yml  
file:${CLOUD_FOUNDRY_CONFIG_PATH}/uaa.yml  
file:${UAA_CONFIG_FILE}  
${UAA_CONFIG_URL}
```

Using Maven with Cloud Foundry or VCAP

To test against a vcap instance, use the Maven profile `vcap`. This profile switches off some of the tests that create random client and user accounts:

```
$ (cd uaa; mvn test -P vcap)
```

To change the target server, set `VCAP_BVT_TARGET` (the tests prefix it with `uaa.` to form the server url):

```
$ VCAP_BVT_TARGET=appcloud21.dev.mozycloud mvn test -P vcap
```

You can also override some of the other most important default settings using environment variables. The defaults come from `uaa.yml`, but tests will first search in an environment variable:

- `UAA_ADMIN_CLIENT_ID` the client id for bootstrapping client registrations needed for the rest of the tests.
- `UAA_ADMIN_CLIENT_SECRET` the client secret for bootstrapping client registrations

You can individually override all other settings from `uaa.yml` as system properties. To do this in an IDE, use the IDE features that allow you to modify the JVM in test runs. With Maven, use the `argLine` property to get settings passed onto the test JVM:

```
$ mvn -DargLine=-Duaa.test.username=foo test
```

This creates an account with `userName=foo` for testing (instead of using the default setting from `uaa.yml`).

If you prefer environment variables to system properties, you can use a custom `uaa.yml` with placeholders for your environment variables:

```
uaa:  
  test:  
    username: ${UAA_TEST_USERNAME:marissa}
```

This looks for an environment variable (or system property) `UAA_TEST_USERNAME` before defaulting to `marissa`. This is the trick used to expose `UAA_ADMIN_CLIENT_SECRET` etc. in the standard configuration.

Test with PostgreSQL or MySQL

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

To run the unit tests using PostgreSQL:

```
$ SPRING_PROFILES_ACTIVE=test,postgresql CLOUD_FOUNDRY_CONFIG_PATH=src/test/resources/test/profiles/postgresql mvn test
```

To run the unit tests using MySQL:

```
$ SPRING_PROFILES_ACTIVE=test,mysql CLOUD_FOUNDRY_CONFIG_PATH=src/test/resources/test/profiles/mysql mvn test
```

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).
- `gem` is a ruby gem (`cf-uaa-client`) for interacting with the UAA server.
- `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.
- `login` (sample) is an application that performs authentication for the UAA acting as a back end service. `login` is where Cloud Foundry administrators set up their authentication sources, e.g. LDAP/AD, SAML, OpenID (Google etc.) or social. The `run.pivotall.io` platform uses a different implementation of the [login server](#).

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 `mvn tomcat:run` to run it directly from `uaa` directory in the source tree (make sure the common jar is installed first using `mvn install` from the common subdirectory or from the top level directory). With Maven, it listens on port 8080.

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`. There is a Maven profile `local` to launch with context root `/`, and another called `vcap` to launch at `/` with a PostgreSQL backend.

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 users: `marissa` has password `koala`.

To use Postgresql for user data, activate one of the Spring profiles `hsqldb` or `postgresql`.

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

```
spring_profiles: postgresql
```

or by passing the `spring.profiles.active` parameter to the JVM. For example, to run with an embedded HSQL database:

```
$ mvn -Dspring.profiles.active=postgresql tomcat: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` 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

API Application

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

Run it using `mvn tomcat:run` from the `api` directory once all other Tomcat processes have been shut down. This deploys the app to a Tomcat manager on port 8080.

App 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 `mvn tomcat:run` from the `app` directory once all other Tomcat processes have been shut down.

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 `UAAPROFILE`. In the application source code (`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-<UAAPROFILE>.properties`. The naming convention adopted is that the `UAAPROFILE` 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, `deayml`, 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 `deayml`.

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 `deayml` 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

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 should not be 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 handled using [Control Groups](#). Each container is placed in its own control group, where it is configured to use an equal slice of CPU compared to other containers, and the maximum amount of memory it may use is set.

The following sections give a brief summary of the techniques used to implement the Linux backend for Warden. You can find a more detailed description in the `root/linux` directory of this repository.

Networking

Every container is assigned a network interface which 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 from and to the container can be forwarded using NATs. Additionally, you can filter and shape all traffic as needed, using readily available tools such as `iptables`.

Filesystem

Every container gets a private root filesystem. This filesystem is created by stacking a read-only filesystem and a read-write filesystem. This is implemented using `aufs` on Ubuntu versions from 10.04 to 11.10 and `overlayfs` on Ubuntu 12.04.

The read-only filesystem contains the minimal set of Ubuntu packages and Warden-specific modifications common to all containers. The read-write filesystem stores files overriding container-specific settings when necessary. Because all writes are applied to the read-write filesystem, containers can share the same read-only base filesystem.

The read-write filesystem is created by formatting a large sparse file. Because the size of this file is fixed, the filesystem that it contains cannot grow beyond this initial size.

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 }

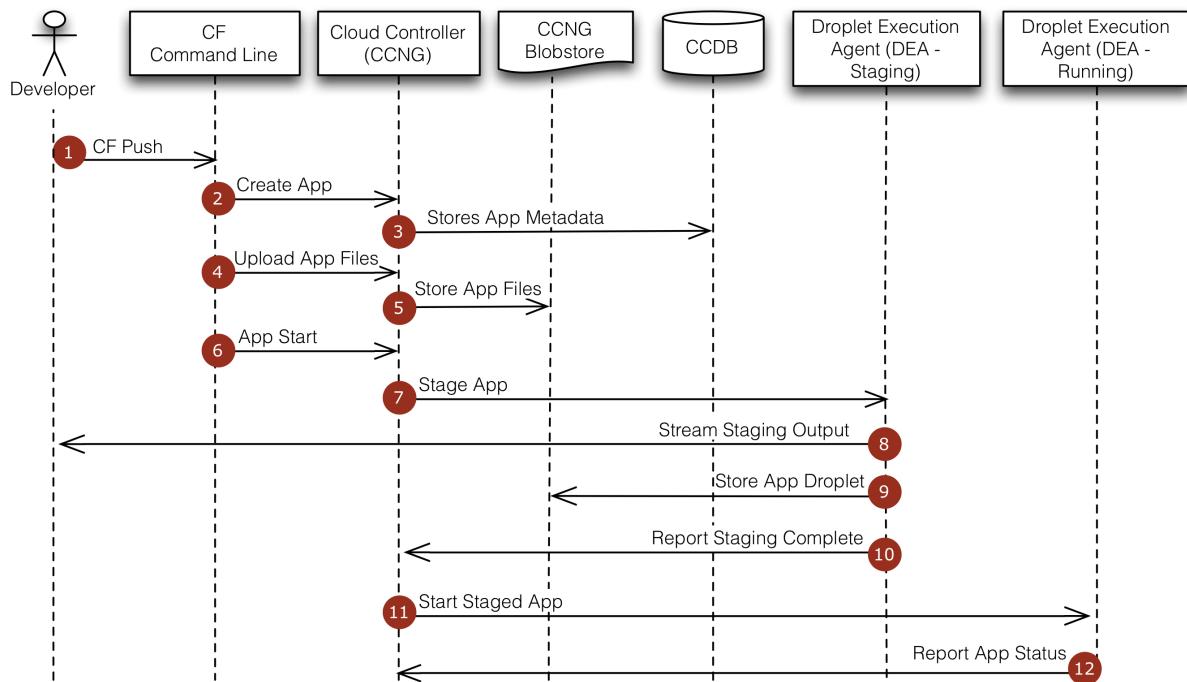
# Timeout unless a certain number of messages have been received
NATS.timeout(sid, TIMEOUT_IN_SECS, :expected => 2) { timeout_recv = true }

# 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



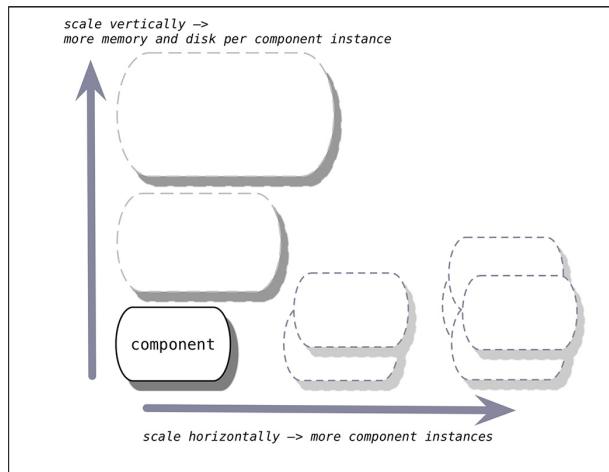
1. At the command line, the developer enters the directory containing her application and uses the cf command line tool to issue a push command.
2. The cf command line tool tells the Cloud Controller to create a record for the application.
3. The Cloud Controller stores the application metadata (e.g. 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.

Scaling Cloud Foundry

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

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.



Tradeoffs and benefits

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

DEAs:

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

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

Router:

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

Health Manager:

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

Cloud Controller:

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

Scaling Platform Availability

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

- For components that support multiple instances, increase the number of instances to achieve redundancy.
- For components that do not support multiple instances, choose a strategy for dealing with events that degrade

availability.

- For database services, plan for and configure backup and restore where possible.

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

Scalable processes

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

There are seven scalable processes:

- DEA: More DEAs add application capacity.
- Cloud Controller: More Cloud Controllers help with API request volume.
- Health Manager: Scaling to two Health Managers is sufficient.
- Router: Additional routers help bring more available bandwidth to ingress and egress.
- Loggregator Server: Deploying additional Loggregator servers splits traffic across them.
- Loggregator Router: Deploying additional Loggregator routers allows you to direct traffic to them in a round robin manner.
- UAA: Scaling to two UAAs is sufficient.

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:

In production deployments, it is a good idea to use your own load-balancing infrastructure with an associated high-availability strategy and configuration. Otherwise, use HAProxy as an alternative load-balancing solution.

NATS:

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.

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 and you need 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 consists of users grouped together for management purposes. All members of an org share a resource quota plan, 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

Cloud Foundry Security

Cloud Foundry protects customers from security threats by applying security controls and by isolating customer applications and data.

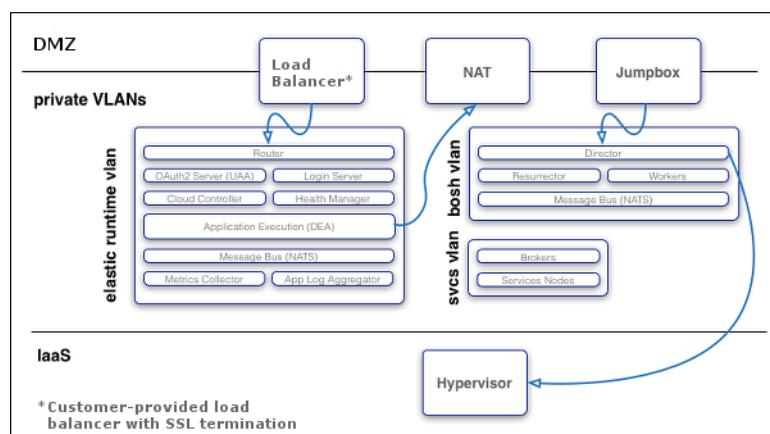
Cloud Foundry:

- 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 may want to become familiar with the [general system architecture](#).

System Boundaries and Access

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 proxy that maps to one or more routers, a NAT VM for outbound requests, and a jumpbox to access the BOSH Director.



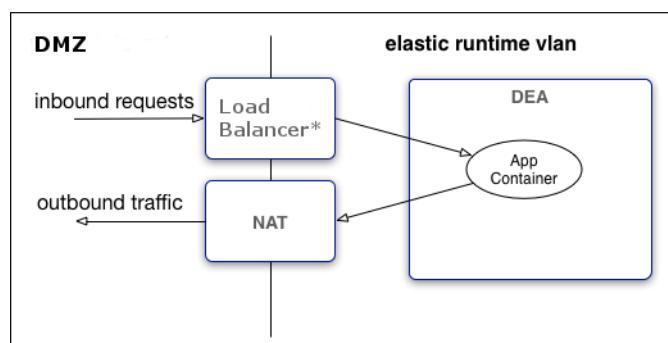
Because of the limited number of contact points with the public internet, the surface area for possible security vulnerabilities is minimized.

Protocols

All traffic from the public internet to the Cloud Controller and UAA happen over https. Inside the boundary of the system, components communicate over a pub-sub message bus, [gnatsd](#).

Application Traffic

The App Container IP address is allocated when the application instance is started. The application is also assigned an arbitrary port. (The application can determine which port to listen on from the VCAP_APP_PORT environment variable provided in the container environment.)



The Elastic Runtime router handles all inbound traffic to applications, routing traffic to one of the application instances.

If the deployment is configured as recommended with the cluster on a VLAN, then all outbound traffic goes through two levels of network address translation: container IP address to DEA IP address, then DEA to the NAT provisioned to connect the VLAN with the public internet.

Applications are prevented from connecting directly with other applications, or with system components, by the firewall rules specified in the operator-controlled BOSH deployment manifest.

Firewalls

Cloud Foundry uses firewalls to restrict access to systems from external networks and between internal components.

The firewall rules that prevent applications inside a Warden container from communicating with system components are configured in the BOSH manifest with the `allow_networks` and `deny_networks` properties. For convenience, Cloud Foundry provides a number of predefined firewall rules; these firewall rules should be reviewed and evaluated for necessity and desirability to an organization on a case-by-case base.

Cloud Foundry firewalls also restrict customer applications from establishing localhost connections over the loopback network interface to further isolate customer applications. These firewalls can be configured to further limit inbound and outbound connections as needed.

Spoofing

Managed firewalls prevent IP, MAC, and ARP spoofing between VMs within Cloud Foundry.

Cloud Foundry uses application isolation, operating system restrictions, and encrypted connections to further ensure risk is mitigated.

BOSH

Elastic runtime and associated services are deployed with BOSH. The BOSH Director is the core orchestrating component in BOSH. The BOSH Director controls VM creation and deployment, as well as other software and service lifecycle events.

Communication to the BOSH Director is secured over https. Authentication is through password security. The BOSH Director should be deployed on a subnet that is not publicly accessible. Operators access the BOSH Director from a jumpbox on the subnet or through a VPN.

BOSH launches VMs and communicates with those VMs over NATS, a publish-subscribe messaging system. NATS cannot be accessed from outside Cloud Foundry, thus ensuring published message can only originate from a component within your deployment.

BOSH provides a mechanism for operators to access the VMs.

BOSH provides an audit trail through the `bosh tasks` command. This command shows all actions that an operator has taken with BOSH.

BOSH allows you to set up individual log in accounts for each operator. BOSH operators have **root** access.

Data stored on BOSH VMs is not encrypted by BOSH but may be encrypted by your IaaS layer.

Authentication and Authorization

Access Mechanisms: All user access to Cloud Foundry, including API, CLI, and the Web-based console, is authenticated through User Account and Authentication Server [UAA](#).

The UAA is the identity management service for Cloud Foundry. As an OAuth2 provider, the UAA issues tokens for client applications to use when they act on behalf of Cloud Foundry users. The UAA can also authenticate users with their Cloud Foundry credentials, and can act as an SSO service using those credentials, or others.

All sensitive credentials are encrypted.

The VCAP_SERVICES environment variable contains credentials. It's not stored but rather is constructed from sensitive data that is encrypted in the Cloud Controller Database.

Authorization happens through the Cloud Controller, and access control is properly enforced through the Cloud Controller.

User passwords are stored in the UAA database using [bcrypt](#), an adaptive cryptographic key derivation function for passwords.

Segmenting User Access with Orgs and Spaces

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

Applications deployed to Cloud Foundry exist within a space. In turn, spaces live in orgs.

In order to even see the existence of an org or space, a user must be a member of it.

Cloud Foundry supports the following user roles:

Space Managers can add and manage users and enable features for a given space. Assign this role to managers or other users who need to administer the account.

Space Developers can create, delete, and manage applications and services, and have full access to all usage reports and logs. They can edit applications, including the number of instances and memory footprint.

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 space.

Org Managers can add and manage users, select and change their plan, and establish spending limits.

Org Auditors have read-only access to org information and reports.

Software Vulnerability Management

Cloud Foundry manages software vulnerability using stemcells and releases. New stemcells are created with patches for the latest security fixes to address any underlying operating system issues, while new Cloud Foundry releases are created with updates to address code issues.

Ensuring Application Code is Secured

Application developers push their code to Cloud Controller. It's secured through OAUTH2 (the UAA) and SSL (communication happens over https). Cloud Controller has role based permissions such that only authorized users are allowed to access that application.

Cloud Controller stores application configuration (environment variables) in an encrypted database table.

Applications code is then run inside a secure container utilizing [cgroups](#), user access permissions, and a firewalled network which prevents communication with system components. The firewall rules are configured in the BOSH manifest.

Application containers each have two IP addresses; outbound traffic goes through two NATs:

- From inside the container to the DEA
- From the DEA to the NAT router

Firewall rule inside the DEAs should be configured to ensure that all packets from inside are dropped or rejected.

The manifest templates in cf-release implement these networking rules.

Application Isolation with Containers

Each application deployed to Cloud Foundry runs within its own self-contained environment, a Warden container, and are blocked from directly interacting 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, while host-based firewalls restrict applications from establishing local network connections. 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 should not be aware of each others 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 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

Every container is assigned a network interface which 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 from and to the container can be forwarded using NAT. Additionally, all traffic can be filtered and shaped as needed, using readily available tools such as iptables.

Filesystem

Every container gets a private root filesystem. This filesystem is created by stacking a read-only filesystem and a read-write filesystem. This is implemented by using aufs on Ubuntu versions from 10.04 up to 11.10, and overlays on Ubuntu 12.04.

The read-only filesystem contains the minimal set of Ubuntu packages and Warden-specific modifications common to all containers. The read-write filesystem stores files overriding container-specific settings when necessary. Because all writes are applied to the read-write filesystem, containers can share the same read-only base filesystem.

The read-write filesystem is created by formatting a large sparse file. Because the size of this file is fixed, the filesystem that it contains cannot grow beyond this initial size.

Security Event Logging / Auditing

BOSH provides an audit trail through the `bosh tasks` command. This command shows all actions that an operator has taken with BOSH.

The cf-release provides a way to redirect cf component logs to a standard syslog server through a property in cf-release for syslog server: `syslog_aggregator`.

Tips for Running a Secure Deployment

- Ensure that your jumpbox is secure.
- In order to reduce the possibility of unauthorized access to the VMs within your BOSH managed cloud it is recommended that you deploy within a VLAN that limits network traffic to individual VMs.
- The firewall rules that prevent applications inside a Warden container from communicating with system components are configured in the BOSH manifest with the `allow_networks` and `deny_networks` properties.
- You can configure UAA clients and users in the BOSH manifest. Limit and manage these clients and users as you would any other kind of privileged account.
- Promiscuous network interfaces should not be allowed on the trusted network.
- Encrypt Data in Transit: Enable HTTPS for applications and SSL database connections to protect sensitive data transmitted to and from applications.
- Encrypt Sensitive Data at Rest: Customers with sensitive data can encrypt stored files and data within databases to meet their data security requirements. Data encryption can be deployed using industry standard encryption and the best practices for your language or framework.
- Authentication: To prevent unauthorized account access use a strong passphrase for both your Cloud Foundry user account and SSH keys, store SSH keys securely to prevent disclosure, replace keys if lost or disclosed, and use Cloud Foundry's RBAC model to restrict user access to only what is necessary to complete their tasks.
- Third-Party Services: In developing your application on Cloud Foundry, you may choose to use third party services for added functionality. Review and monitor data sharing and security practices with these providers.

Stacks

A stack is a prebuilt file system, including an operating system, that supports running applications with certain characteristics. Any DEA can support exactly one stack. To stage or run a Linux app using MySQL, a DEA running the `lucid64` stack must be available (and have free memory).

The scripts for building this stack (and later for building other stacks) reside in the [stacks](#) project.

The lucid64 stack

Currently, Cloud Foundry supports one stack: `lucid64`, a Ubuntu 10.04 64-bit system containing a number of common programs and libraries including:

- MySQL
- PostgreSQL
- sqlite
- imagemagick
- git
- ruby 1.9.3

Cloud Foundry Glossary

TERM	DEFINITION
API	Application Programming Interface
BOSH	A deployment orchestration solution.
CLI	Command Line Interface
DEA	Droplet Execution Agent. The DEA is the component in Cloud Foundry 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 Cloud Foundry 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 cf command line interface, the Cloud Controller API, and the Cloud Foundry Eclipse Plugin.
Org	An org is the top-most meta object within the Cloud Foundry infrastructure. Only an account with administrative privileges on a Cloud Foundry 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 also 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 Cloud Foundry 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 Cloud Foundry 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 CF operators.

Table of Contents

- [Domains and Shared Domains](#)
- [Isolating a Pivotal CF Deployment with a Private Network](#)
- [Understanding the Elastic Runtime Network Architecture](#)
- [Configuring Pivotal CF SSL Termination](#)
- [Changing the Quota Plan of an Organization with cf CLI](#)
- [Identifying Elastic Runtime Jobs Using vCenter](#)

Domains and Shared Domains

This page assumes you are using cf v6.

This page has information about domains and shared domains. For more information, see [Domains, Subdomains, 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
```

Isolating a Pivotal CF Deployment with a Private Network

You may want to place your Pivotal CF deployment on a private network, then route requests to Pivotal CF from a router or a load balancer on a public network. Isolating Pivotal CF in this way increases security and allows you to use as many IP addresses as you need by subnetting the private network.

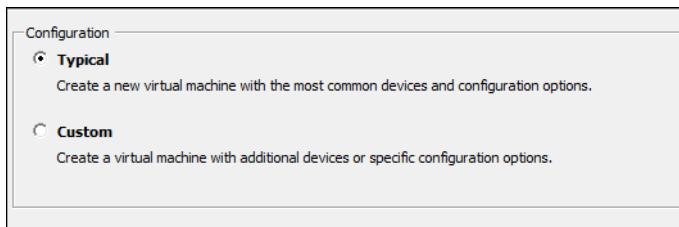
One way to accomplish this is to configure Network Address Translation (NAT) on a VM. The following example explains how to set up a virtual CentOS NAT box in vSphere, and use that to isolate a Pivotal CF deployment.

Step 1: Deploy a CentOS image to vSphere

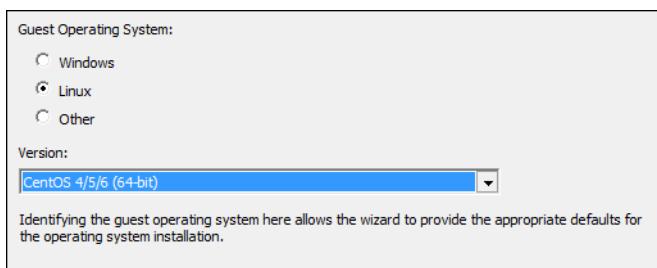
Connect to vCenter using the vSphere client. You will assign two network adapters to the VM. This results in the VM having two IP addresses:

- 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>`.

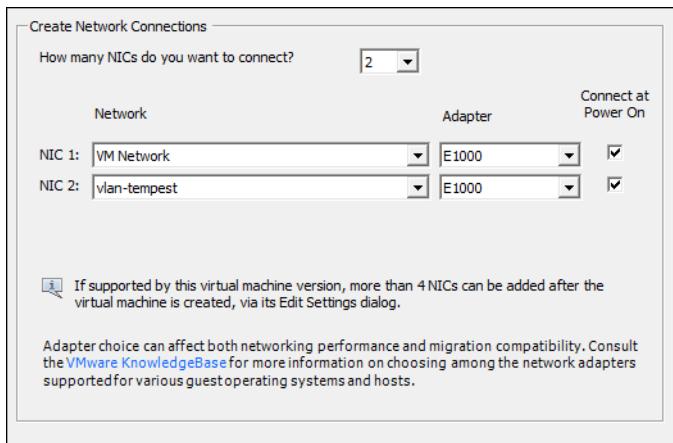
1. Download the latest CentOS image from <http://wiki.centos.org/Download>.
2. In vSphere, select **File > New > Virtual Machine**.
3. On the **Configuration** page, select the **Typical** configuration and click **Next**.



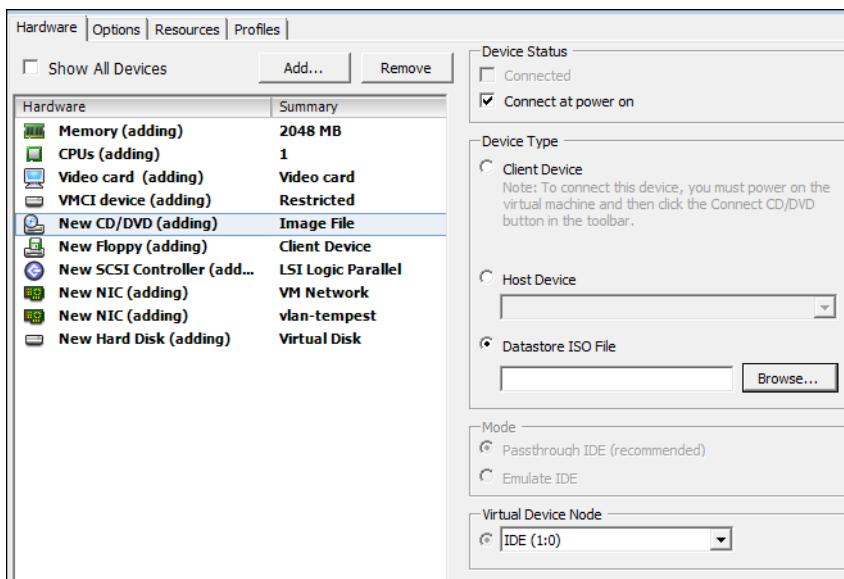
4. On the **Name and Location** page, name the virtual machine and click **Next**.
5. On the **Resource Pool** page, select a resource pool and click **Next**.
6. On the **Storage** page, select a destination storage for the VM and click **Next**.
7. On the **Guest Operating System** page, select **Linux** as the Guest Operating System and **CentOS 4/5/6 (64-bit)** as the version, then click **Next**.



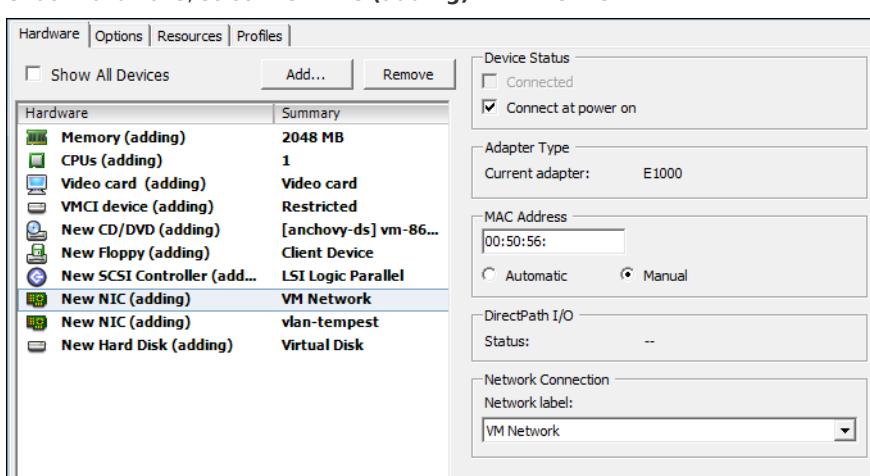
8. On the **Network** page, select **2** from the drop down menu to specify how many NICs you want to connect. Under **Network**, select **VM Network** for NIC 1 and the name of your private network for NIC 2. Then, click **Next**.



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

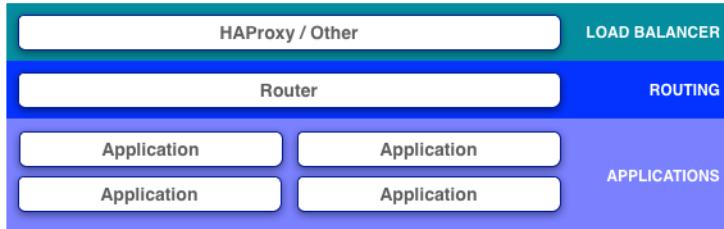
service iptables save
shutdown -r now
```

Step 3: Deploy the Ops Manager VM Template in the private network using the internal IP address

1. Download Pivotal CF 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 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 Pivotal CF 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 Pivotal CF SSL Termination

Pivotal CF deploys with SSL termination turned on by default, but does not deploy with a pre-configured load balancer. You must configure the Pivotal-deployed HAProxy load balancer or your own load balancer to use SSL termination.

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

Pivotal CF 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 Pivotal CF 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 Elastic Runtime.

1. Create an A record in your DNS that points to the HAProxy IP address.

For example, with `cf.example.com` as the main subdomain for your CF install and an HAProxy IP address `172.16.1.1`, you must create an A record in your DNS that serves `example.com` and points `*.cf` to `172.16.1.1`.

NAME	TYPE	DATA	DOMAIN
<code>*.cf</code>	A	<code>172.16.1.1</code>	<code>example.com</code>

2. Use the Linux `host` command to test your DNS entry. The `host` command should return your HAProxy IP address.

Example:

```
$ host cf.example.com
cf.example.com has address 172.16.1.1
$ host anything.example.com
anything.cf.example.com has address 172.16.1.1
```

3. 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 Pivotal CF Router IPs
- Adds appropriate `x-forwarded-for` and `x-forwarded-proto` HTTP headers

You must register static IP addresses for Pivotal CF with your load balancer and configure three fields in Pivotal Ops Manager.

1. Register one or more static IP address for Pivotal CF with your load balancer.
2. 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](#).
3. Configure one field on the Router IPs page in Elastic Runtime:
 - **Router IPs:** Enter the static IP address for Pivotal CF that you have registered with your load balancer.

 **Note:** When adding or removing Pivotal CF 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 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

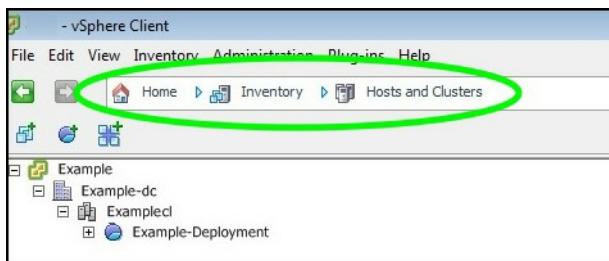
To effectively monitor, control, and manage the virtual machines making up your Elastic Runtime deployment, you may need to identify which VM corresponds to a particular job in Elastic Runtime. You can find the CID of a particular VM from Pivotal CF Ops Manager by navigating to **Elastic Runtime > Status**.

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.

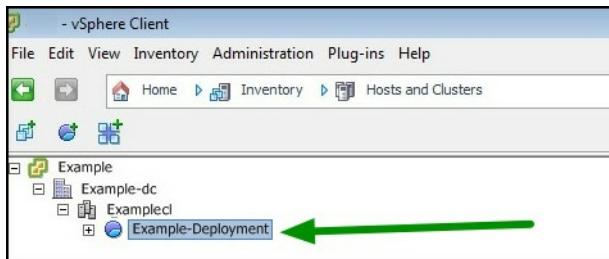
 **Note:** The CID shown in Ops Manager is the name of the machine in vCenter.

Identifying Elastic Runtime Jobs Using vCenter

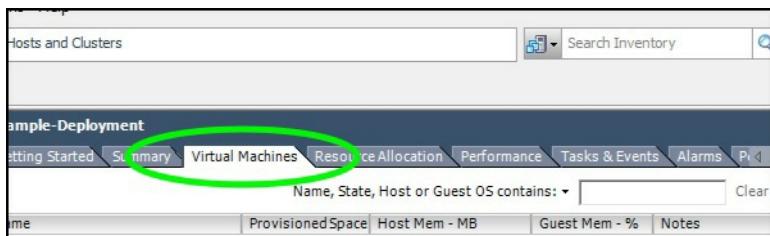
1. Launch the vSphere client and log in to the vCenter Server system.
2. Select the **Inventory > Hosts and Clusters** view.



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' summary page. On the left, there's a tree view of virtual machines and a status table. On the right, there's a detailed status table for each VM. A green arrow highlights the 'Status' dropdown menu, which lists various status types like Host, Provisioned Space, Used Space, etc.

Name	Provisioned Space	Host Mem - MB	Guest
vm-5d98bbf5-813f-4612-8	9.09 GB	1040	6
vm-a98b916f-c295-4a41-8	11.09 GB	597	3
Pivotal Ops Manager(zero)	151.09 GB	1036	45
vm-3e13d9e3-2323-457d-8	10.09 GB	598	4
vm-56012bf-e627-4651-8	19.09 GB	1023	88
sc-a49692cf-18e2-41ed-9	6.69 GB	0	0
vm-a0aaea51-6b7d-4639-8	9.09 GB	1045	12
vm-d26d43a2-11f9-4b52-8	9.09 GB	470	3
vm-e5bfff2f-e37-44ea-9	9.09 GB	468	4
vm-8e5030ae-f9d1-4548-8	20.09 GB	3130	0
sc-a1b24b3-dca3-4418-8	6.69 GB	0	0
vm-95662aee-5ec5-4e62-9	9.09 GB	551	4
sc-449692cf-18e2-41ed-9	6.69 GB	0	0
sc-b095e033-bba7-4e2d-8	6.69 GB	0	0

6. The job column displays the Elastic Runtime job associated with each virtual machine.

The screenshot shows the 'Resource Allocation' tab. On the left, there's a list of virtual machines. On the right, there's a dropdown menu for selecting a job. A green arrow highlights the 'job' dropdown menu, which lists various jobs like ccdb, cloud_controller, consolodb, dea, ha_proxy, health_manager, loggregator, loggregator_trafficcontroller, nats, nfs_server, riak_ts, router, saml_login, uaa, and uaadb.

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** tells cf what you want to call the buildpack.
- **path** tells cf where to find the buildpack. The path can point to a zip file, the URL of a zip file, or a local directory.
- **position** tells cf where to place the buildpack in the detection priority list. See [Buildpack Detection](#).
- **enable or disable** tells cf 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
```

Creating and Managing Users with the cf CLI

This page assumes you are using cf 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	<code>cf create-user USERNAME PASSWORD</code>	<code>cf create-user Alice pa55w0rd</code>
Delete a user	<code>cf delete-user USERNAME</code>	<code>cf delete-user Alice</code>

Creating Administrator Accounts

To create a new administrator account, use the [UAA CLI](#).

Note: The `cf` CLI cannot create new administrator accounts.

Org and App Space Roles

A user can have one or more roles. The combination of these roles defines the user's overall permissions in the org and within specific app spaces in that org.

Org Roles

Valid [org roles](#) are OrgManager, BillingManager, and OrgAuditor.

FUNCTION	COMMAND	EXAMPLE
View the organizations belonging to an account	<code>cf orgs</code>	<code>cf orgs</code>
View all users in an organization by role	<code>cf org-users ORGANIZATION_NAME</code>	<code>cf org-users my-example-org</code>
Assign an org role to a user	<code>cf set-org-role USERNAME ORGANIZATION_NAME ROLE</code>	<code>cf set-org-role Alice my-example-org OrgManager</code>
Remove an org role from a user	<code>cf unset-org-role USERNAME ORGANIZATION_NAME ROLE</code>	<code>cf unset-org-role Alice myexample-org OrgManager</code>

App Space Roles

Each app space role applies to a specific app space.

Note: By default, the org manager has app space manager permissions for all spaces within the organization.

Valid [app space roles](#) are SpaceManager, SpaceDeveloper, and SpaceAuditor.

FUNCTION	COMMAND	EXAMPLE
View the spaces in an org	cf spaces	cf spaces
View all users in a space by role	cf space-users ORGANIZATION_NAME SPACE_NAME	cf space-users my-example-org development
Assign a space role to a user	cf set-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE	cf set-space-role Alice my-example-org development SpaceAuditor
Remove a space role from a user	cf unset-space-role USERNAME ORGANIZATION_NAME SPACE_NAME ROLE	cf unset-space-role Alice my-example-org development SpaceAuditor

Creating and Managing Users with the UAA CLI (UAAC)

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 Admin Users

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
```

Creating Users

1. Obtain the credentials of an admin user created using UAAC as above, or refer to the 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 login.

```
$ 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 user created using UAAC as above, or refer to the the `uaa: scim` section of your deployment manifest for the user name and password of an admin user.
2. Use `cf login -u ADMIN-USERNAME -p ADMIN-PASSWORD` to login.
3. 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
```

4. 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 that you are using cf 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 an [org](#) or to a specific [space](#) in an org. A security group assigned to an org affects all spaces in the org.

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 org-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 and consist of 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 an org or a space for Cloud Foundry to apply the rules to outgoing traffic.

Binding to a Space

Run `cf bind-security-group SECURITY-GROUP ORG SPACE` to bind a security group to a 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.

Binding to an Org

To create an org-wide rule, you must bind a security group to the Default Staging or the Default Running security group set. Cloud Foundry applies the rules in every security group in each of these sets as follows:

- **Default Staging:** Rules are applied to every application staged in every space in the org. To bind a security group to the Default Staging set, run `cf bind-staging-security-group SECURITY-GROUP`.
- **Default Running:** Rules are applied to every application running in every space in the org. To bind a security group to the Default Running set, run `cf bind-running-security-group 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 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 Droplet Execution Agent (DEA) configuration section of your deployment manifest.
4. **Hard-coded REJECT rule:** Cloud Foundry returns a REJECT result for all outbound traffic from a container if not allowed by a higher-priority rule.

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 Developer Console

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 Developer Console](#)
- [Understanding Developer Console Permissions](#)
- [Managing Spaces Using the Developer Console](#)
- [Managing User Accounts in Spaces Using the Developer Console](#)
- [Managing User Permissions Using the Developer Console](#)

Getting Started with the Developer Console

The Developer Console is a tool to help manage organizations, users, applications, and spaces. Complete the following steps to invite new users to the Developer Console.

1. Log in to the Developer Console: [How to log in.](#)
2. The Developer Console displays information for your org and all of the spaces in your org.

The screenshot shows the Pivotal CF Developer Console interface. At the top, it says "My-Org". Below that, there's an "ORG" section with "My-Org" selected. Under "SPACES", there are three listed: "development", "staging", and "production". To the right, there's a summary: "3 Spaces", "1 Domain", and "1 Member". Below this, there are three cards for each space: "SPACE development" (3 Apps, 0 Services, 17% of Org Quota), "SPACE staging" (0 Apps, 0 Services, 0% of Org Quota), and "SPACE production" (0 Apps, 0 Services). A button "Add A Space" is at the bottom right.

3. In the Members tab, click **Invite New Members**.

The screenshot shows the "Members" tab in the Developer Console. It has tabs for "1 Space", "1 Domain", and "1 Member". Below that, there's a dropdown menu set to "org". On the right, there's a blue button "INVITE NEW MEMBERS". The main area shows a table with three columns: "MEMBER", "ORG MANAGER", and "ORG AUDITOR". There's one row for "admin", where the "ORG MANAGER" and "ORG AUDITOR" checkboxes are checked.

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 console 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 Developer Console Permissions

In most cases, the actions available to you on the Developer Console 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 Developer Console that usually only an Admin can perform with either the CLI or the Developer Console.

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 Developer Console to perform these actions.
- Org Managers, like Admins, can perform all actions using the Developer Console.
- Space Managers, like Admins and Org Managers, can assign and remove users from spaces using the Developer Console.
- Developer Console users of any role can create an org and view org and space users.

	CLI	DEVELOPER CONSOLE		
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 Developer Console

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 Developer Console: [How to log in](#).

The screenshot shows the Pivotal CF Developer Console interface. The top navigation bar says "Pivotal CF" and "My-Org". The left sidebar has sections for "ORG" (with "My-Org" selected), "SPACES" (listing "development", "staging", "production", and "Marketplace"), "Docs", "Support", and "Tools". The main content area is titled "My-Org" and shows "QUOTA" (17%, 1.75 GB of 10 GB Limit), "3 Spaces", "1 Domain", and "1 Member". Below this, there are three cards for the spaces: "SPACE development" (3 Apps, 0 Services, 17% of Org Quota), "SPACE staging" (0 Apps, 0 Services, 0% of Org Quotas), and "SPACE production" (0 Apps, 0 Services). A "Add A Space" button is also present.

Orgs

The current org is highlighted in the Developer Console navigation. The drop-down menu on the right displays other orgs. Use this menu to switch between orgs.

The screenshot shows the Pivotal CF Developer Console interface. The top navigation bar says "Pivotal CF". The left sidebar has sections for "ORG" (with "My-Org" selected) and "SPACES" (listing "development", "production", "staging", and "Marketplace"). The "My-Org" section has a dropdown arrow icon.

Spaces

The Developer Console navigation also shows the spaces in the current org. The current space is highlighted.

The screenshot shows the Pivotal CF Developer Console interface. The top navigation bar says "Pivotal CF". The left sidebar has sections for "ORG" (with "My-Org" selected) and "SPACES" (listing "development", "production", "staging", and "Marketplace"). The "development" space in the "SPACES" list is highlighted with a blue border.

- **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

Delete Space

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 Developer Console

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.

MEMBER	ORG MANAGER	ORG AUDITOR
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

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) Use commas to separate emails

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**

- 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.
- 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.
- Click **Send Invite**. The Developer Console 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.

- On the Org dashboard, click the **Members** tab.

1 Space | 1 Domain | 1 Member

org **INVITE NEW MEMBERS**

MEMBER	ORG MANAGER	ORG AUDITOR
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

- Edit the permissions assigned to each user by checking or unchecking the checkboxes under each user role. The Developer Console 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.

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](#)

[Configuring Domains, Subdomains, and Routes](#)

[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 cf v6](#)

[Using cf with an HTTP Proxy Server](#)

[Starting Applications using cf](#)

[Scaling an Application Using cf scale](#)

Services

[Services Overview](#)

[Adding a Service](#)

[Binding a Service Instance](#)

[Managing Service Instances](#)

[Third-Party Managed Services](#)

[User-Provided Services](#)

[Configuring Play Framework Service Connections](#)

[Migrating a Database in Cloud Foundry](#)

[Using Third Party Log Management Services](#)

[Configuring Selected Third Party Log Management Services](#)

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. There are a few reasons for this.

- **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 blob store 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 (for example to share state), 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.

HTTP and HTTPS Port Limitations

Applications running on Cloud Foundry receive requests using only the URLs configured for the application, and only on ports 80 (the standard HTTP port) and 443 (the standard HTTPS port).

Ignore Unnecessary Files When Pushing

By default, when you push an application, all files in the application's project directory tree, except version control files with file extensions `.svn`, `.git`, and `.darcs`, are uploaded to your Cloud Foundry instance. 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, or *evacuated*, on the DEA to be upgraded, then restarted on another DEA. To avoid the risk of an application being unavailable during Cloud Foundry upgrade processes, you should run more than one instance of an application.

Buildpacks and Language-Specific Considerations

Cloud Foundry stages application using buildpacks. Heroku developed the buildpack approach and made it available to the open source community. Cloud Foundry currently provides buildpacks for the several runtimes and frameworks.

Cloud Foundry Buildpacks

Cloud Foundry uses [buildpacks](#) to transform user-provided artifacts into runnable applications. The functionality of buildpacks varies, but many of them examine the user-provided artifact in order to properly download needed dependencies and configure applications to communicate with bound services. Heroku developed the buildpack approach and Cloud Foundry embraces it.

Deploy an Application

This page assumes that you are using cf v6.

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 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:

- [Prepare to Deploy](#)
- [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 URL of the Cloud Controller in your Elastic Runtime instance](#): This is the API endpoint, also known as the target URL, for your Cloud Foundry 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 subdomain and a domain.

- The subdomain is also known as the host portion of the URL. The name of an application is the default subdomain 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 domain when deploying, provided that the domain is registered and is mapped to the organization which contains the target application space.

For more information, see [About Domains, Subdomains 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 is likely to have. If your application is still in development, running a single instance makes it easier to troubleshoot. For any production application, you should run a minimum of two instances.
- **Memory Limit:** The maximum amount of memory that each instance of your application is allowed to consume. If an instance exceeds this limit, the instance is restarted. If the instance has to be restarted too often, it is terminated. To prevent this we recommend being generous with your memory limit.
- **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, `cf` 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.

`cf` 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 [Prepare to Deploy an Application](#).

For more information about the manifest file, see the [Deploying with Application Manifests](#).

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 `cf` 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 `cf`. See [Troubleshoot Application Deployment and Health](#).

Domains, Subdomains, and Routes

This page assumes you are using cf v6.

This page has information about how to specify the route (or URL) that Cloud Foundry uses to direct requests to an application. A route is made up of a *subdomain* (or host) and a *domain* that you can specify when you push an application.

Domains and Routes in Cloud Foundry

A **domain** is a domain name like `example.com`. Domains can also be multi-level and contain sub-domains like the “myapp” in `myapp.example.com`. In Cloud Foundry, domains are associated with orgs. Domain objects are not directly bound to apps.

Cloud Foundry also supports custom domains. You can map a registered domain of your own to a space in Cloud Foundry.

A **route**, based on a domain with an optional host as a prefix, may be associated with one or more applications. For example, `myapp` is the host and `example.com` is the domain in the route `myapp.example.com`. It is also possible to have a route that represents `example.com` without a host. This type of route is known as a zone apex domain or a root domain. Routes are children of domains and are directly bound to apps in Cloud Foundry.

A Cloud Foundry instance defines a default shared domain. Unless you specify a domain, `cf` uses the default shared domain in the route to your application.

Use Your Custom Domain in Cloud Foundry

There are three different models for using your own custom domain in Cloud Foundry:

- Using a shared root domain, such as `example.org`.
- Using a shared subdomain, such as `myapp.example.org`.
- Using a shared domain plus a host to create a url, such as `myapp.example.org`

Select the procedure below that corresponds with the type of domain you want to use.

Map a Root Domain

Some DNS providers support using an alternative custom record type, such as ANAME or ALIAS, for adding custom zone apex domains. These domains are also known as root domains.

Use a DNS ALIAS OR ANAME record to map a custom root domain to your app.

1. Create the domain in Cloud Foundry and associate it with an organization.
The command below defines the custom domain `example.org` in the “test-org” organization:

```
$ cf create-domain test-org example.org
```

2. Map the route to your app:

```
$ cf map-route myapp example.org
```

3. Configure your custom root domain with your DNS provider using an ALIAS or ANAME record.

ALIAS and ANAME records follow the pattern illustrated below:

RECORD	NAME	TARGET	NOTE
ALIAS or ANAME	empty or @	yourappname.example.com.	Refer to your DNS provider documentation to determine whether to use an empty or @ value for the Name entry.

Map a Root Domain Using Subdomain Redirection

If your DNS provider does not support ANAME or ALIAS records, use subdomain redirection to add a custom root domain. This technique is also known as domain forwarding. 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. This error does not display if you are not using SSL or are only using SSL with URLs in subdomain form, meaning `https://www.example.com`.

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	yourappname.cfapps.io	

Map a Single Subdomain

Use a DNS CNAME record to alias a multi-level custom subdomain.

1. Create the domain in Cloud Foundry and associate it with an organization.

The command below defines the custom domain `test.example.org` in the “test-org” organization:

```
$ cf create-domain test-org test.example.org
```

2. Map the route to your app:

```
$ cf map-route myapp test.example.org
```

3. Configure your custom domain with your DNS provider using a CNAME record.

CNAME records follow the pattern illustrated below:

RECORD	NAME	TARGET	NOTE
CNAME	test	yourappname.example.com.	Refer to your DNS provider documentation to determine whether the trailing `.` is required.

Map Multiple Subdomains

1. Create the domain in Cloud Foundry and associate it with an organization.

The command below defines the custom domain `myapp.example.org` in the “test-org” organization:

```
$ cf create-domain test-org example.org
```

2. Use the `-n HOSTNAME` parameter with the `maproute` command to specify a unique hostname for a route that uses a shared domain. Map the route to your app:

```
$ cf map-route myapp example.org -n myapp
```

3. Configure your custom domain with your DNS provider using a CNAME record.

CNAME records follow the pattern illustrated below:

RECORD	NAME	TARGET	NOTE
CNAME	myapp	yourappname.example.com.	Refer to your DNS provider documentation to determine whether the trailing `.` is required.

View Domains for an Org

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 user@example.org... OK  
  
name      status  
example.com  shared  
example.org   owned
```

Delete a Domain

You can delete a domain from Cloud Foundry with the `cf delete-domain` command:

```
$ cf delete-domain example.org
```

About Subdomains

In some cases, defining the subdomain portion of a route is optional. In general, though, this segment of a route is required to ensure the route is unique. Note in particular that you *must* define a subdomain in the following cases:

- If you assign the default domain defined for your Cloud Foundry instance.
- If you assign a custom domain to the application, and that domain is, or will be, assigned to other applications. In the case of a custom domain, a subdomain for it must be registered with your naming service along with the top-level domain.

You might choose not to assign a subdomain to an application that will not accept browser requests, or if you are using a custom domain for a single application only.

Assign Domain and Subdomain at push

When you run `cf push`, you can optionally specify a domain and subdomain for the application.

- Domain: Use the `-d` flag to specify one of the domains available to the targeted org.
- Subdomain: Use the `-n` flag to provide a string for the subdomain.

Example:

```
$ cf push myapp -d example.org -n myapp
```

The route cf creates for the application as a result of this command is `myapp.example.org`.

Assign Subdomain in Manifest

If you create or edit the manifest for an application, you can use the `host` (for subdomain) and `domain` attributes to define the components of the application route. For more information, see [Application Manifests](#).

List Routes

You can list routes for the current space with `cf routes` command. Note that the subdomain is shown as "host," separate from the domain segment. For example:

```
$ cf routes
Getting routes as user@example.org ...

host           domain     apps
myapp          example.org myapp1
                  example.org myapp2
ltest          example.com ltest
sinatra-hello-world example.com sinatra-hello-world
sinatra-to-do   example.com sinatra-to-do
```

Define or Change a Route from the Command Line

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
```

Assign a Route to an Application

Assign or change the route for a particular application with the `cf map-route` command. Specifying the subdomain 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
```

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 Using the Command Line

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 subdomain is optional.

Deploying with Application Manifests

This page assumes that you are using cf 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
```

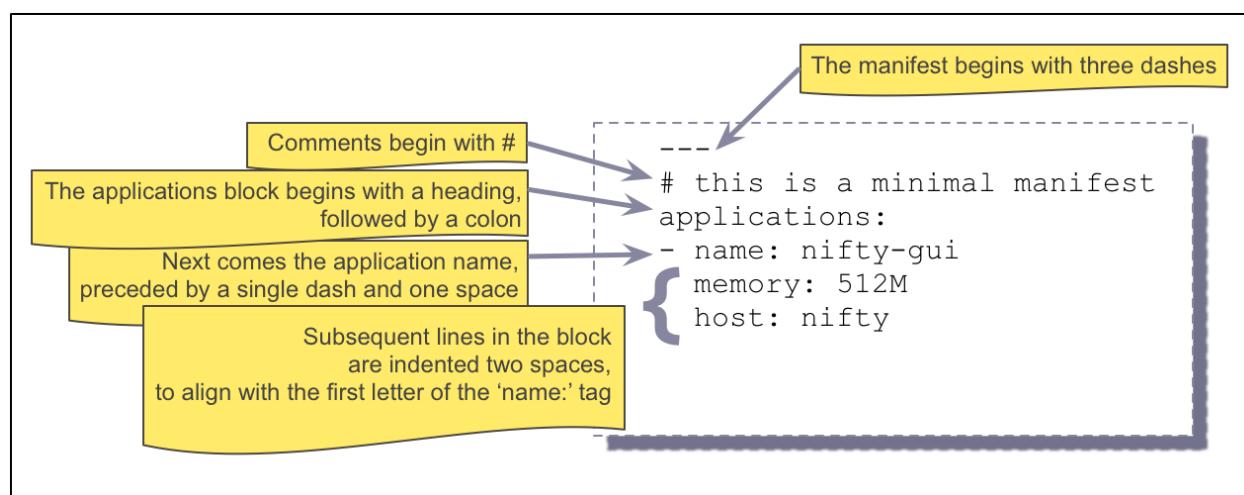
If the manifest is named something other than `manifest.yml`, use the `-f` option with both path *and* filename.

```
$ 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
```

A Minimal 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 minimal manifest illustrated below introduces some YAML indentation conventions that your manifests should follow.



The directory that contains the bits you want to push is a good place to put your manifest and to run `cf push`, because `cf` looks for a manifest in the current working directory.

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` 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` recursively pushes the contents of that directory instead of the current working directory.
- If the path is to a file, `cf` 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, `cf` 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

You can use the `command` attribute to set a custom start command for your application.

For example:

```
command: bundle exec rake VERBOSE=true
```

For some types of application, you must provide a custom start command, either from the command line or from the manifest. This is true of node.js applications, for example.

The command line option that overrides this attribute is `-c`.

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 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, `cf` 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 path attribute

You can use the `path` attribute to tell `cf` 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, `cf` 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, `cf` 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 `cf` 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` finds the bits you want to push?

The answer is to add a path line to each application description to lead `cf` to the correct bits. Assume that `cf` 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  
  subdomain: 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  
subdomain: 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 that you are using cf 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 Variable Values

The sections below describe methods of viewing the values of Cloud Foundry environment variables.

View Environment Variables using CLI

The cf command line interface provides two commands that can return environment variables.

To see the environment variables that you have set using the `cf set-env` command:

```
$ cf env my_app_name
```

To see the environment variables in the container environment:

```
$ cf files my_app_name logs/env.log
```

Variables Defined by the DEA

The subsections that follow describe the environment variables set by a DEA for an application at staging time.

You can access environment variables programmatically, including variables defined by the buildpack (if any). 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. This value is set as a result of the value you specify in an application manifest, or at the command line when pushing an application.

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
application_users, users	
instance_id	GUID that identifies the application.
instance_index	Index number of the instance.
application_version, version	GUID that identifies a version of the application that was pushed. Each time an application is pushed, this value is updated.
application_name, name	The name assigned to the application when it was pushed.
application_uris	The URI(s) assigned to the application.
started_at, start	The last time the application was started.
started_at_timestamp	Timestamp for the last time the application was started.
host	IP address of the application instance.
port	Port of the application instance.
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.

```
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.a1-app.cf-app.com"],"version":"c10  
63c1c-40b9-434e-a797-db240b587d32","name":"styx-james","uris":["styx-james.a1-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:PHxTPJSbkDakfK4cYwXHiIX9Q8p5Bxn@babar.elephantsql.com:5432/seilbmbd"
      }
    }
  ],
  "sendgrid": [
    {
      "name": "mysendgrid",
      "label": "sendgrid",
      "tags": [
        "smtp"
      ],
      "plan": "free",
      "credentials": {
        "hostname": "smtp.sendgrid.net",
        "username": "QvsXMbJ3rK",
        "password": "HCHMOYluTv"
      }
    }
  ]
}
```

Using Blue-Green Deployment to Reduce Downtime and Risk

This page assumes you are using cf 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.

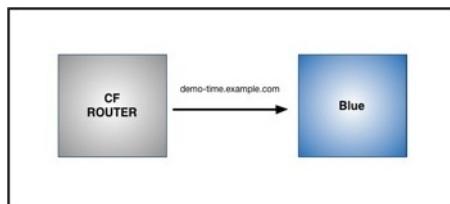
Step 1: Push an App

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

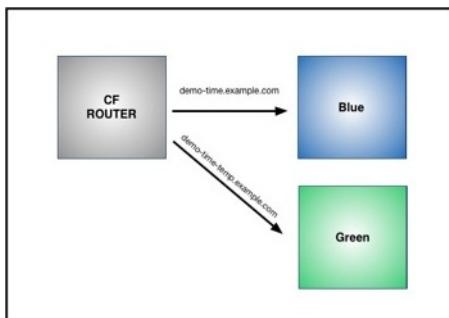
Now make a change to the application. First, replace the word "Blue" on the web page with "Green," then rebuild the source file for the application. Run `cf push` again, but use the name "Green" for the application and provide a different subdomain to create a temporary route:

```
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.



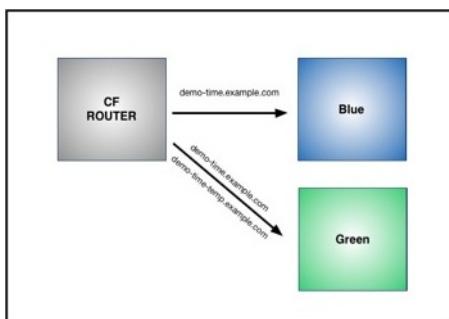
Step 3: Map Original Route to Green

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.



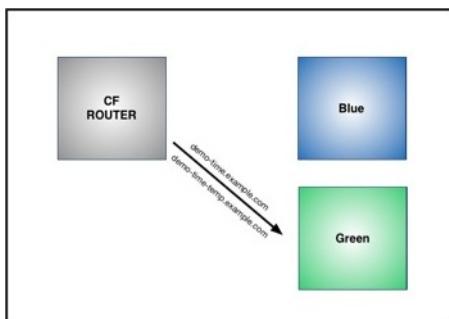
Step 4: Unmap Route to Blue

Once you verify Green is running as expected, stop routing requests to Blue using the `cf unmap` command:

```
cf unmap-route Blue example.com -n demo-time
Unbinding demo-time.example.com from blue... OK
```

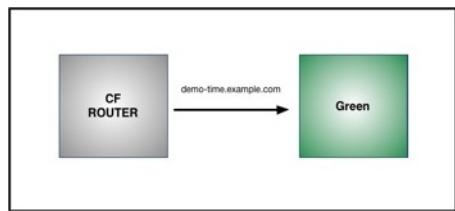
After cf unmap:

- 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` 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 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 that you are using cf v6.

Refer to this topic for help diagnosing and resolving common problems pushing and running apps on Cloud Foundry.

Common Problems

cf push times out

A cf push can time out during an upload or a staging. Symptoms can include a “504 Gateway Timeout” error or messages stating “Error uploading application” or “timed out waiting for async job to finish.” If this happens, try these techniques.

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 only pushing needed files. By default cf will push all the contents of the current working directory. Make sure you are pushing just your application’s directory. If your application is too large, or if it has many small files, Cloud Foundry may time out during the upload. You can also reduce the size of the upload by removing unneeded files or [specifying files to be ignored](#) in the `.cfignore` file.

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 see 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 see the memory usage of all the apps in a space, use `cf apps`.

Make sure your application is less than 1GB. By default cf will push all the contents of the current working directory. To reduce the number of bits you are pushing, make sure you are pushing just your application’s directory, and remove unneeded files or [specify files to be ignored](#) in the `.cfignore` file. These limits apply:

- The app bits to push cannot exceed 1GB.
- The droplet that results from compiling those bits cannot exceed 1.5GB; droplets are typically 1/3 larger than the pushed bits.
- The app bits, compiled droplet, and buildpack cache together cannot use more than 4GB 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 says “Unable to detect a supported application type.”

You can see 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 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:

```
----> 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 techniques.

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 that captures the symptoms of the problem you are troubleshooting.

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.

- View the environment variables that you have set using the `cf set-env` command:

```
cf env <appname>
```

- View the variables in the container environment:

```
cf files <appname> logs/env.log
```

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

  env.log                      932B
  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 `cf` 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` 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`.
- `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 Developer Console, or from the command line.

From the Developer Console

Log in to the Developer Console 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 cf is 6.x.

Contents in this section:

- [Installing cf v6](#)
- [Getting Started with cf v6](#)
- [Using cf with an HTTP Proxy Server](#)
- [Scaling an Application Using cf scale](#)

Installing the cf Command Line Interface

To install cf, download it from <https://github.com/cloudfoundry/cli/releases> and follow the instructions for your operating system.

Uninstall cf v5

If you previously used the cf v5 Ruby gem, you must uninstall this gem before installing cf v6.

To uninstall, run `gem uninstall cf`.

 **Note:** To ensure that your Ruby environment manager registers the change, close and reopen your terminal.

Install cf on Windows

To install cf on Windows:

1. Unpack the zip file.
2. Double click the `cf` executable.
3. When prompted, click **Install**, then **Close**.

Install cf on Mac OSX and Linux

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 cf version 6, see [Getting Started with cf v6](#).

Getting Started with cf v6

This guide introduces cf v5 users to what is new in cf v6.

cf v6 is simpler, faster and more powerful than v5. Consider these key differences:

- cf 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 v5 commands read manifests, only the push command in cf v6 reads manifests.
- While cf v5 used interactive prompts widely, cf 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 cf v6 which follows, we highlight new and improved features and how to use them.

Installation

cf 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 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.

Login

The `login` command in cf 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, cf prompts for:

- **API endpoint:** [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, cf 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 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](#).
- You no longer create services with push interactively or in a manifest. See [User-Provided Services](#) 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.

cf 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 cf 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.

User-Provided Service Instances

cf 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. cf 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 in interactive mode, use the `-p` option with a comma-separated list of parameter names. cf then prompts you for each parameter in turn.

```
cf uups SERVICE_INSTANCE -p "HOST, PORT, DATABASE, USERNAME, PASSWORD"
```

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 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`.

cf 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 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 cf v6 to help operators and developers work efficiently and comfortably.

cf 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 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

cf v6 command aliases and command line help feature improved consistency and convenience.

cf 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`.

cf 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 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 cf with an HTTP Proxy Server

If you have an HTTP proxy server on your network between a host running `cf` 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:

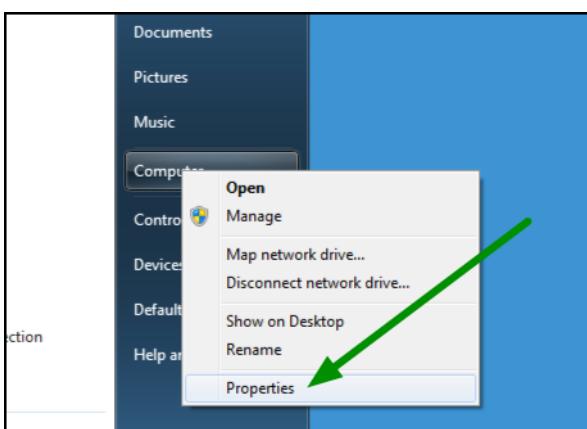
```
$ export HTTP_PROXY=http://my.proxyserver.com:8080 ↵
```

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:

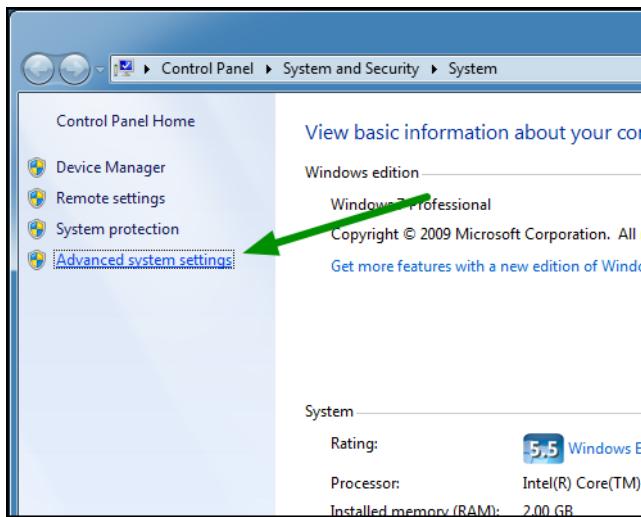
```
http_proxy=http://username:password@hostname:port;
export $HTTP_PROXY
```

Setting `HTTP_PROXY` in Windows

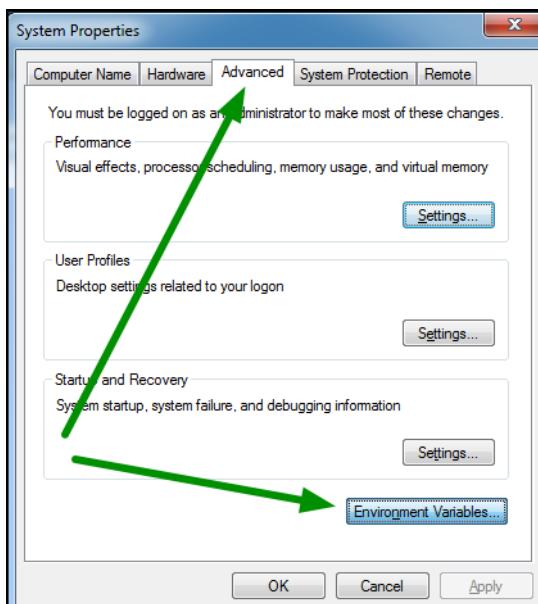
1. Open the Start menu. Right-click **Computer** and select **Properties**.



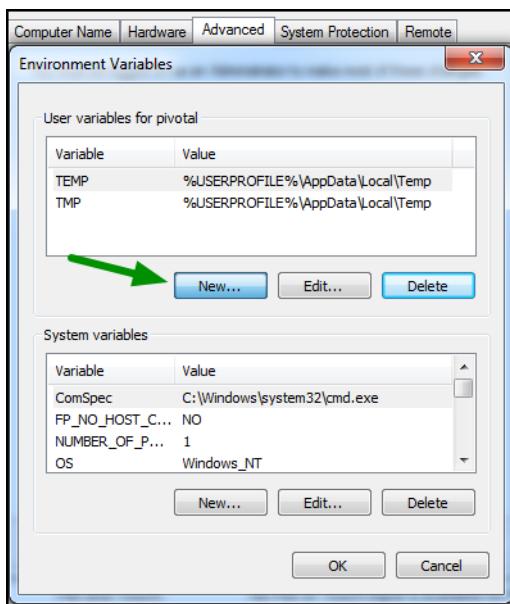
2. In the left pane of the System window, click **Advanced system settings**.



- In the System Properties window, select the **Advanced** tab, then click **Environment Variables**.



- 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**.

About Starting Applications

This page assumes that you are using cf 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 `cf` 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 `cf` 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 for Deploying while 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 on 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.

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 [Migrate a Database on Cloud Foundry](#).

Adding a Service

This page assumes that you are using cf 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 that you are using cf 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 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.

Note: This is an example. These services may not be available on your Cloud Foundry marketplace you target.

```
$ cf marketplace
Getting services from marketplace in org scoen / space test as scoen@gopivotal.com...
OK

service          plans           description
blazemeter      free-tier      The JMeter Load Testing Cloud
cleardb          spark          Highly available MySQL for your Apps.
cloudamqp        lemur          Managed HA RabbitMQ servers in the cloud
elephantsql      turtle         PostgreSQL as a Service
memcachedcloud   25mb          Enterprise-Class Memcached for Developers
mongolab         sandbox        Fully-managed MongoDB-as-a-Service
newrelic         standard       Manage and monitor your apps
rediscloud       25mb          Enterprise-Class Redis for Developers
searchly         starter        Search Made Simple.
sendgrid         free           SMTP service by SendGrid
```

Create a Service Instance

Use this command to create a service instance.

```
$ cf create-service cleardb spark cleardb-test
Creating service cleardb-test in org my-org / space test as me@example.com...
OK
```

Create a User-Provided Service Instance

User-provided service instances are service instances which have been 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 using the familiar `create-service` command, and provide whatever credentials your application requires.

- [User Provided Service Instances](#)

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 cleardb-test
Binding service cleardb-test 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 cleardb-test
Unbinding app my-app from service cleardb-test 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 cleardb-test

Are you sure you want to delete the service cleardb-test ? y
Deleting service cleardb-test in org my-org / space test as me@example.com...
OK
```

Third-Party Managed Service Instances

This page assumes that you are using cf 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 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/janclouduser/test-apps/test-msg-app/manifest.yml  
...  
Binding service test-mysql-01 to test-msg-app in org janclouduser-org / space development as janclouduser@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 `janclouduser` 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 janclouduser-org / space development as janclouduser@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 janclouduser-org / space development as janclouduser@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 janclouduser-org / space development as janclouduser@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 janclouduser-org / space development as janclouduser@example.com...
OK
...
```

5. To verify that the service instance is now recorded in `VCAP_SERVICES`, we search the environment log.

```
$ cf files db-app logs/env.log | grep VCAP_SERVICES

VCAP_SERVICES={"example-db-n/a":[{"name":"test-777","label":"example-db-n/","tags":["mysql","relational"],
"plan":"basic","credentials": {"jdbcUrl": "jdbc:mysql://bc3282dfc81aa8:24b81700@us-cdbr-east-05.example-db.net
:3306/ad_068816026ae218", "uri": "mysql://bc3282dfc81aa8:24b81700@us-cdbr-east-05.example-db.net:
1234/ad_0688168026ae218?reconnect=true", "name": "ad_0688168026ae218", "hostname": "us-cdbr-east-05.example-db.net",
"port": "1234", "username": "bc3282dfc81aa8", "password": "24b81700"}}]}
```

User-Provided Service Instances

This page assumes that you are using cf 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 cf; 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 Commands for Working with User-Provided Service Instances

cf 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. cf 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 in interactive mode, use the `-p` option with a comma-separated list of parameter names.

cf then prompts you for each parameter in turn.

```
cf uups SERVICE_INSTANCE -p "HOST, PORT, DATABASE, USERNAME, PASSWORD"
```

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 cf 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
```

- 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
```

- 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
```

- For the binding to take effect, we must push our app a second time.

```
$ cf push msgapp
Updating app msgapp in org my-org / space development as a.user@example.com...
OK
...
```

- To verify that the service instance is now recorded in [VCAP_SERVICES](#), search the environment log.

```
$ cf files msgapp logs/env.log | grep VCAP_SERVICES
VCAP_SERVICES={"user-provided":[{"name":"pubsub","label":"user-provided","tags":[],"credentials":{"binary":"pubsub.rb","host":"pubsub01.example.com","password":"p@29w0rd","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 that you are using cf 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. Obtain your database credentials by searching for the `VCAP_SERVICES` environment variable in the application environment log:
`cf files my-app logs/env.log | grep VCAP_SERVICES`
2. Connect to the database using your database credentials.
3. Migrate the database using SQL commands.
4. Update the application using `cf push`.

Migrate Occasionally

This method requires you to:

- Create a schema migration command or script.
- Run the migration command when deploying a single instance of the application.
- Re-deploy the application with the original start command and number of instances.

This method is efficient for occasional use because you can re-use the schema migration command or script.

1. Create a schema migration command or SQL script to migrate the database. For example:
`rake db:migrate`
2. Deploy a single instance of your application with the database migration command as the start command. For example:
`cf push APP -c 'rake db:migrate' -i 1`
3. Deploy your application again with the normal start command and desired number of instances. For example:
`cf push APP -c 'null' -i 4`

 **Note:** After this step the database has been migrated but the application itself has not started, because the normal start command is not used.

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 db:migrate && rails s
```

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 that you are using cf 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
```

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` followed by `cf push`.

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.

The screenshot shows the Papertrail dashboard with a light gray header containing the title 'Dashboard' and two buttons: 'Add Systems' and 'Create Group'. Below the header is a message box with a blue information icon and the text: 'Let's aggregate some logs. [Add your first system](#) in about 45 seconds, or [take a tour](#)'.

2. Click the **Alternatives** link.

The screenshot shows the 'Setup Systems' page. It displays the text 'Your systems will log to `logs.papertrailapp.com:11699`'. Below this, there is a link labeled '» Other log methods: Use an app hosting service or old syslog daemon? [Alternatives](#)'.

3. Select **I use Heroku**, enter a name, and click **Save**.

The screenshot shows a configuration form titled 'Choose your situation:' with three options:

- A **My syslogd only uses the default port**: Description: 'GNU syslogd and some embedded OSes will only log to port 514. A few Linux distros shipped with GNU syslogd (mostly older CentOS and Gentoo).'
- B **I use Heroku**: Description: 'Heroku uses TCP without TLS. Create one port per app.'
- C **My system's hostname changes**: Description: 'In rare cases, one system may change hostnames frequently. For example, a roaming laptop which sets its hostname based on DHCP (and roams across networks).'

On the right, there is a panel with the text: 'Let's create a destination for all logs from this app's dynos.' and 'We'll provide a syslog destination for a Heroku drain and [step-by-step setup](#)'. Below this is a field labeled 'What should we call it?' with the value 'CloudFoundry' and a note: 'Alphanumeric. Does not need to match app name.' At the bottom is a 'Save →' button.

4. Note the URL with port that is displayed after creating the system.

The screenshot shows a message box with the text 'CloudFoundry will log to `logs.papertrailapp.com:36129`'.

5. Create the log drain service in CF.

```
$ cf cups my-logs -l syslog://logs.papertrailapp.com:<port>
```

6. Bind the service to an app, then restart or re-stage as necessary.

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 AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
Mar 05 14:57:36 CloudFoundry AppId:960-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 AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} INFO: No beans of type
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean found
in application context. Skipping auto-reconfiguration.
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} Mar 05, 2014 10:57:36 PM
org.cloudfoundry.reconfiguration.AbstractServiceConfigurer configure
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} INFO: Class
org.springframework.amp.ConnectionFactory not
in classpath. Skipping auto-reconfiguration for it
Mar 05 14:57:36 CloudFoundry AppId:960-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 AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.681 INFO RequestMappingHandlerMapping:181 - Mapped
{"/albums/{id}",methods={PUT},params=[],headers[],consumes[],produces[],custom[]} onto public void
org.cloudfoundry.samples.music.domain.Album
org.cloudfoundry.samples.music.web.controllers.AlbumController.update(jav
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.681 INFO RequestMappingHandlerMapping:181 - Mapped
{"/albums/{id}",methods={GET},params[],headers[],consumes[],produces[],custom[]} onto public
{"/albums"},methods={GET},params[],headers[],consumes[],produces[],custom[]} onto public void
org.cloudfoundry.samples.music.web.controllers.AlbumController.albums(jav
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.682 INFO RequestMappingHandlerMapping:181 - Mapped
{"/albums/{id}",methods={POST},params[],headers[],consumes[],produces[],custom[]} onto public void
org.cloudfoundry.samples.music.domain.Album
org.cloudfoundry.samples.music.web.controllers.AlbumController.add(jav
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.682 INFO RequestMappingHandlerMapping:181 - Mapped
{"/info",methods={GET},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 AppId:960-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
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.684 INFO RequestMappingHandlerMapping:181 - Mapped
{"/info",methods={DELETE},params[],headers[],consumes[],produces[],custom[]} onto public
org.cloudfoundry.samples.music.domain.ApplicationInfo
Mar 05 14:57:36 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 22:57:36.684 INFO RequestMappingHandlerMapping:181 - Mapped
{"/info",methods={POST},params[],headers[],consumes[],produces[],custom[]} onto public
org.cloudfoundry.samples.music.domain.ApplicationInfo
Mar 05 14:57:36 CloudFoundry AppId:960-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
Mar 05 14:57:36 CloudFoundry AppId:960-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.showInfo()
Mar 05 14:57:36 CloudFoundry AppId:960-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 AppId:960-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 AppId:960-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 AppId:960-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 AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 2014 10:57:37 PM org.apache.coyote.AbstractProtocol start
Mar 05 14:57:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 2014 10:57:37 PM org.apache.coyote.http11.Http11Protocol start
Mar 05 14:57:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 2014 10:57:37 PM org.apache.catalina.startup.Catalina start
Mar 05 14:57:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} 2014 10:57:37 PM org.apache.catalina.core.StandardServer startup in 11278 ms
Mar 05 14:59:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} Tried to stop app that never received a start event
Mar 05 14:59:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{App/1} Stopped app instance (Index 1) with guid 40dc96a-c093-4bbb-8718-186f147f3e73
Mar 05 14:59:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{DEA} Stopped app instance (Index 1) with guid 40dc96a-c093-4bbb-8718-186f147f3e73
Mar 05 14:59:37 CloudFoundry AppId:960-c093-4bbb-8718-186f147f3e73/{DEA} Stopped app instance (Index 1) with guid 40dc96a-c093-4bbb-8718-186f147f3e73

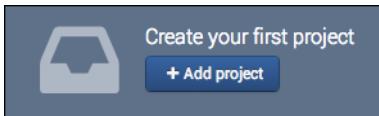
```

Example "access denied" (1.2.3.4 OR redis) -sshd
https://papertrailapp.com/groups/553803/events?centered_on_id=378003402845892611&q=program%3A4aa&c060-c093-4bbb-8718-186f147f3e73%27%5BApp%27%5D

Splunk Storm

From your Splunk Storm account:

- Click the **Add project** button.



- Enter the project details.

Add project

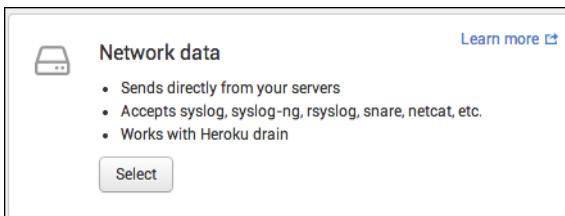
* Project name: Cloud Foundry

* 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**

- Create a new **input** for **Network data**.



- 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.

Network Data ▾

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

6. Create the log drain service in CF using the displayed TCP host and port.

```
$ cf cups my-logs -l syslog://<host:port>
```

7. Bind the service to an app, then restart or re-stage as necessary.

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 CF.

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 *	Cloud Foundry
Description	
Category	
Unless overwritten by Source metadata, the Collector will set the Source category of all messages to this value.	
<input type="button" value="Save"/> <input type="button" value="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 * <input type="text" value="CloudFoundry"/> Maximum name length is 128 characters	
Description <input type="text"/>	
Source Host <input type="text" value="LDAP_Server"/> Host name for the system from which the log files are being collected, e.g. LDAP_Server	
Source Category <input type="text"/> Log category metadata to use later for querying, e.g. OS_Security	
▶ Advanced ▶ Filters	
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

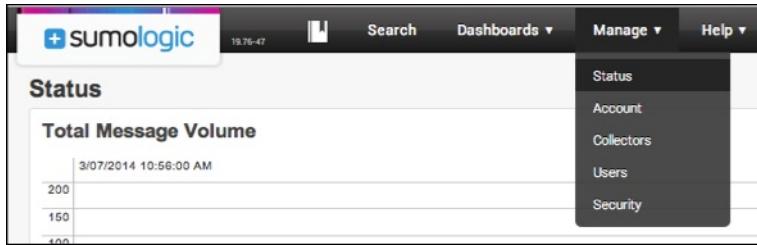
5. Once the source is created, a URL should be displayed. You can also view the URL by clicking the **Show URL** link beside the created source.

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

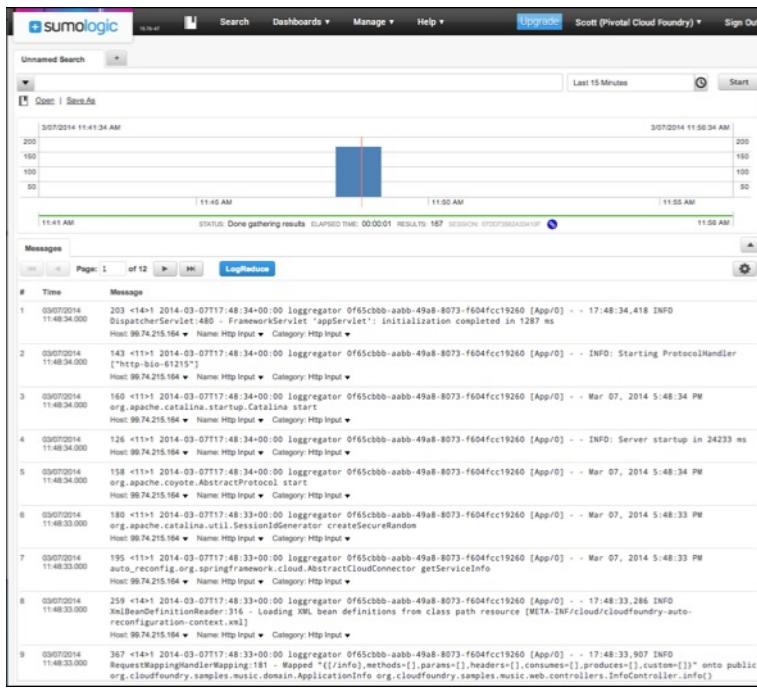
6. Create the log drain service in CF using the displayed URL.

```
$ cf cups my-logs -l <HTTP SOURCE-URL>
```

7. Bind the service to an app, then restart or re-stage as necessary.
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 that you are using cf 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 an [API extension for on-premises buildpacks](#).

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 would allow you to add support for it at the deployment stage. When an application written using Haskell is pushed, 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 CF back to the end user. The generally accepted pattern for this is to break out this functionality in to 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_type:
  web: bundle exec rackup config.ru -p $PORT
```

bin/package

The `package` script provides buildpack artifacts, which are provided to Cloud Foundry as system buildpacks. `package` is intended to provide a way for buildpack developers to package a buildpack with its dependencies. Implementing this is entirely optional. It is a Cloud Foundry extension to the [Heroku Buildpack API](#).

The result of running the package script must be a zip file that adheres to the api outlined above, using `detect`, `compile`, `release`.

The `package` script accepts two commands as arguments:

- `package online` produces a zip of the buildpack, preferably excluding unnecessary content such as tests.
- `package offline` produces a zip of the buildpack with all dependencies cached.

`package offline` produces a buildpack that does not need internet connectivity to run. For more information about the tradeoffs of the dependency storage options available for custom offline buildpacks, refer to the [Packaging Buildpack Dependencies](#) topic.

The `package offline` command is used to produce the [system buildpacks](#) bundled with Cloud Foundry.

For an example implementation, see the [Cloud Foundry Ruby buildpack](#).

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
```

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:

- [Tips for Java Developers](#)
- [Getting Started Deploying Spring Apps](#)
- [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 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.

Deploy a Spring Sample Application

This section describes how to deploy a Spring sample application to Elastic Runtime.

To learn how to deploy your own Spring app, see [Deploy Your Spring Application](#).

Prerequisites

- [Git](#) configured on your workstation and a [GitHub](#) account
- Basic Spring knowledge

Step 1: Get a Sample Spring App from GitHub

Run `git clone https://github.com/scottfrederick/spring-music.git` to clone the `spring_music` app from GitHub.

Step 2: 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 3: Deploy the Sample App

1. `cd` to the cloned app directory.
2. Run `./gradlew assemble`.
3. Run `cf push spring-music -n [HOST_NAME]`.

Example: `cf push spring-music -n jc0830-spring`

`cf push [APP_NAME]` pushes the specified app from the current directory to Elastic Runtime. Use the `-n` or `--random-route` flag to create a unique URL for the app to avoid URL collisions with other apps.

`cf push` automatically searches for an [application manifest YAML file](#) and also determines the appropriate [buildpack](#) for the app. A manifest specifies what `cf push` should do with the app and the buildpack provides framework and runtime support for your app.

The example shows the terminal output of deploying the `spring-music`. `cf push` uses the instructions in the manifest file to create the app, create and bind and the route, and upload the app to Elastic Runtime. It then follows the instructions in the manifest to start one instance of the app with 512M. After the app starts, the output displays the health and status of the app.

```
$ cf push spring-music -n jc0830-spring
Using manifest file /Users/pivotal/workspace/spring-music/manifest.yml

Updating app spring-music in org Cloud-Apps / space development as clouduser@example.com...
OK

Creating route jc0830-spring.example.com...
OK

Binding jc0830-spring.example.com to spring-music...
OK

Uploading spring-music...
Uploading app files from: /Users/pivotal/workspace/spring-music/build/libs/spring-music.war
Uploading 569.6K, 90 files
OK

Stopping app spring-music in org Cloud-Apps / space development as clouduser@example.com..
OK

Starting app spring-music in org Cloud-Apps / space development as clouduser@example.com...
OK
----> Downloaded app package (21M)
----> Downloaded app buildpack cache (40M)
...
----> Uploading droplet (60M)

0 of 1 instances running, 1 starting
1 of 1 instances running

App started

Showing health and status for app spring-music in org Cloud-Apps / space development as clouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: jc0830-spring.example.com

      state      since            cpu    memory      disk
#0   running   2014-08-28 06:31:37 PM  0.0%  277.8M of 512M  137.9M of 1G
```

Step 4: Test the Sample App

You've deployed your first app to Elastic Runtime! Now let's access your app online using its URL. In the terminal output, 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`. The image shows the resulting webpage for the `spring-music` app.

The screenshot shows a web application titled "Spring Music" with a green header bar. Below the header, there is a section titled "Albums" with a sub-section "View as: [grid] sort by: title artist year genre ▾ | +add an album". The main content area displays a grid of 12 album cards, each containing the album title, artist, year, genre, and a small preview image. The albums listed are:

- IV - Led Zeppelin (1971, Rock)
- Nevermind - Nirvana (1991, Rock)
- What's Going On - Marvin Gaye (1971, Rock)
- Are You Experienced? - Jimi Hendrix Experience (1967, Rock)
- The Joshua Tree - U2 (1987, Rock)
- Abbey Road - The Beatles (1969, Rock)
- Rumours - Fleetwood Mac (1977, Rock)
- Sun Sessions - Elvis Presley (1978, Rock)
- Thriller - Michael Jackson (1982, Pop)
- Exile on Main Street - The Rolling Stones (1972, Rock)
- Born to Run - Bruce Springsteen (1975, Rock)
- London Calling - The Clash (1980, Rock)

You can now use the `cf` CLI or the [Developer Console](#) to review information and administer your app and your Elastic Runtime account. For example, you could:

- Increase the number of app instances from 1 to 3, and redeploy the app with a new app name and host name.
- Create and bind a database service to the app and restart the app to detect the new service. For more information, refer to the [Creating and binding services](#) section of the `spring-music` GitHub README.

See the [Manage Your Application with the cf CLI](#) section for more information. See also [Using the Developer Console](#).

Deploy Your Spring Application

This section describes how to deploy your Spring application to Elastic Runtime.

Prerequisites

- A Spring app that runs locally on your workstation
- (**Optional**) [Git](#) configured on your workstation and a [GitHub](#) account
- Intermediate to advanced Spring knowledge
- 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. Refer to [Step 8: Configure the Deployment Manifest](#).

Step 1: Add the Cloud Foundry Java Client Library to Your Project

The Cloud Foundry Java Client Library enables you to query and manage app instances and account instance information of a Elastic Runtime account instance. For more information on including the library with your Maven or Gradle project and the specific objects that you can query, refer to the [Cloud Foundry Java Client Library](#) topic.

Note: Ensure that your Spring app runs locally before continuing with this procedure.

Step 2: (Optional) 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.

Step 3: (Optional) 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.

Step 4: 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

Step 5: 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, see [Step 8: Configure the Deployment Manifest](#).

When your app is running, you can use the `cf app [APP_NAME]` command to see memory utilization.

 **Note:** Elastic Runtime automatically allocates 512MB to an app when you do not specify a value.

Step 6: 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.

Step 7: Configure Service Connections for Your Spring App

Elastic Runtime provides extensive support for connecting a Spring application to services such as MySQL, PostgreSQL, MongoDB, Redis, and RabbitMQ. For more information on configuring a service connection for your app, refer to the [Configure Service Connections for Spring](#) topic.

Step 8: 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.

For example, you can set memory-related JVM options or a specific Java version in a custom buildpack and configure your manifest to use your custom buildpack:

1. Fork the [Cloud Foundry Java buildpack](#) and edit `config/open_jdk_jre.yml`.
2. Set the `buildpack` attribute in the manifest to the buildpack URL or name.

Refer to the [Deploying with Application Manifests](#) topic for more information.

Step 9: 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 10: Deploy Your Application

When you deploy your application, `cf push`:

- Creates a URL that is the route to your application. The URL is in the form HOST.DOMAIN, where HOST is the APP_NAME and DOMAIN is specified by your administrator. Your DOMAIN is `example.com`.
- Searches for an application manifest YAML file. A manifest specifies what `cf push` should do with the app.
- Determines the appropriate buildpack for the app. A buildpack provides framework and runtime support for your app.

 **Note:** You must use the CLI to deploy apps.

For example: `cf push my-app` creates the URL `my-app.example.com`.

1. `cd` to the app directory that you want to deploy.
2. Run `cf push [APP-NAME] -p [PATH-TO-WAR]`.

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 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.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. You can use the following options 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.
- Run `cf help push` to view other options for this command.

3. Launch a new terminal window and run `cf logs [APP-NAME]` to simultaneously view log activity while the app deploys.
4. 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.

```
App started

Showing health and status for app spring-music in org Cloud-Apps / space development as clouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: jc0830-spring.example.com

      state      since          cpu    memory       disk
#0   running   2014-08-28 06:31:37 PM  0.0%  277.8M of 512M  137.9M of 1G
```

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 v6](#) topic.

Note: You cannot perform certain tasks in the CLI or the [Developer Console](#) 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 Developer Console, depending on your user role, refer to the [Understanding Developer Console 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 [Developer Console](#), 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.

Learn More

For more information on how to develop, deploy, and manage applications, refer to the [Developer Guide](#).

Tips for Java Developers

This page assumes you are using cf 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>-<application-version>.war
```

Groovy

Groovy applications based on both [Ratpack](#) and a simple collection of files are supported.

Ratpack

Ratpack packages applications into two different styles; Cloud Foundry supports the `distZip` style. To build the ZIP and deploy it, run the following:

```
$ gradle distZip  
$ cf push <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 client: Upload of a large WAR large is faster and hence less likely to fail if you are using a recent version of cf. If you are using an older version of the cf client to upload a large, and having problems, try updating to the latest version of cf.
- 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.

Slow Starting Java or Grails Apps

It is not uncommon for a Java or Grails application to take a while to start. This, in conjunction with the current Java buildpack implementation, can pose a problem. The Java buildpack sets the Tomcat `bindOnInit` property to “false”, which causes Tomcat to not listen for HTTP requests until the application is deployed. If your application takes a long time to start, the DEA’s health check may fail by checking application health before the application can accept requests. A workaround is to fork the Java buildpack and change `bindOnInit` to “true” in `resources/tomcat/conf/server.xml`, although this has the downside that the application may appear to be running (as far as Cloud Foundry is concerned) before it is ready to serve requests.

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 (and contribute back) 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` like this:

```
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, the Cloud Foundry Grails plugin will re-configure the plugins when the app starts to provide the connection information to the plugins.

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 the Cloud Foundry auto-reconfiguration, you can choose to configure the Cloud Foundry service connections manually.

The best way to do the manual configuration is to use the `spring-cloud` library to get the details of the Cloud Foundry environment the application is running in. To use this library, add it to the `dependencies` section in 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-service-connector:1.0.0.RELEASE"
}
```

Then you can use the `spring-cloud` API in your `DataSource.groovy` file to set the connection parameters. If you were using all three types of database services as in the auto-configuration example, and the services were named “myapp-mysql”, “myapp-mongodb”, and “myapp-redis”, your configuration might look like that below. Auto-reconfiguration is disabled when a bean from the `spring-cloud` library is defined.

```

import org.springframework.cloud.CloudFactory

beans {
    cloud(CloudFactory) { bean ->
        bean.factoryMethod = "cloud"
    }
}

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 re-configures the relevant bean definitions to bind them to cloud services.

Cloud Foundry auto-reconfigures applications only if the following items are true for your application:

- Only one service instance of a given service type is bound to the application. In this context, MySQL and PostgreSQL are considered the same service type (relational database), so if both a MySQL and a PostgreSQL service are bound to the application, auto-reconfiguration will not occur.
- Only one bean of a matching type is in the Spring application context. For example, you can have only one bean of type `javax.sql.DataSource`.

With auto-reconfiguration, Cloud Foundry creates the database or connection factory bean itself, using its own values for properties such as host, port, username and so on. For example, if you have a single `javax.sql.DataSource` bean in your application context that Cloud Foundry auto-reconfigures and binds to its own database service, Cloud Foundry doesn't use the username, password and driver URL you originally specified. Rather, 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 doesn't 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

Sometimes you may not be able to take advantage of Cloud Foundry's auto-reconfiguration of your Spring application. This may be because you have multiple services of a given type or because you'd like to have more control over the configuration. In either case, you need to include the `spring-cloud` library in your list of application dependencies.

Update your application Maven `pom.xml` or Gradle `build.gradle` file to include a dependency on the `org.springframework.cloud:spring-service-connector` and `org.springframework.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>cloudfoundry-connector</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-service-connector</artifactId>
    <version>1.0.0</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>org.springframework.maven.milestone</id>
    <name>Spring Maven Milestone Repository</name>
    <url>http://repo.spring.io/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 [the service-scanning approach](#) until they need more explicit control. Java config also offers a way to expose application and service properties, should you choose to take a lower level access in creating service connectors yourself (or for debugging purposes, for example).

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 fails if there is no such unique service). 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")` (this follows how Spring's Java configuration works).

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 a 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 (that don't have a directly mapped method—typical for a newly introduced service or connecting to a private service in private PaaS) through the `service()` method. It 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 follows (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.

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 CloudFoundry 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() {  
            CloudEnvironment cloudEnvironment = new CloudEnvironment();  
            RdbmsServiceInfo serviceInfo = cloudEnvironment.getServiceInfo("my-postgres", RdbmsServiceInfo.class);  
            RdbmsServiceCreator serviceCreator = new RdbmsServiceCreator();  
            return serviceCreator.createService(serviceInfo);  
        }  
    }  
  
    @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}
```

where `{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 CloudFoundry.
<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 detects 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 `cloudfoundry-runtime` library, as shown here:

```
@Configuration
public class DataSourceConfig {
    @Bean
    public DataSource dataSource() {
        CloudEnvironment cloudEnvironment = new CloudEnvironment();
        RdbmsServiceInfo serviceInfo = cloudEnvironment.getServiceInfo("my-postgres", RdbmsServiceInfo.class);
        RdbmsServiceCreator serviceCreator = new RdbmsServiceCreator();
        return serviceCreator.createService(serviceInfo);
    }
}
```

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 a `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 a

`org.springframework.data.mongodb.MongoDbFactory` bean (from Spring Data MongoDB). The bean can be created by helper classes in the `cloudfoundry-runtime` library, as shown here:

```
@Configuration
public class MongoConfig {

    @Bean
    public MongoDbFactory mongoDbFactory() {
        CloudEnvironment cloudEnvironment = new CloudEnvironment();
        MongoServiceInfo serviceInfo = cloudEnvironment.getServiceInfo("my-mongodb", MongoServiceInfo.class);
        MongoServiceCreator serviceCreator = new MongoServiceCreator();
        return serviceCreator.createService(serviceInfo.get());
    }

    @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 a `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() {
        CloudEnvironment cloudEnvironment = new CloudEnvironment();
        RedisServiceInfo serviceInfo = cloudEnvironment.getServiceInfo("my-redis", RedisServiceInfo.class);
        RedisServiceCreator serviceCreator = new RedisServiceCreator();
        return serviceCreator.createService(serviceInfo);
    }

    @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 a `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 a `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 a `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() {
        CloudEnvironment cloudEnvironment = new CloudEnvironment();
        RabbitServiceInfo serviceInfo = cloudEnvironment.getServiceInfo("my-rabbit", RabbitServiceInfo.class);
        RabbitServiceCreator serviceCreator = new RabbitServiceCreator();
        return serviceCreator.createService(serviceInfo);
    }

    @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 v1.7.1 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.

v1.7.1 of the plugin provides several new features:

- Spring Boot support: In the Project Explorer, right-click on your Spring Boot project, select **Configure -> Enable as Cloud Foundry App**, then drag and drop the Spring Boot app to your Cloud Foundry server instance.
- Support for servers using self-signed certificates: When creating a new Cloud Foundry server through the Eclipse Servers wizard, if “self-signed certificate” is detected for a cloud URL, the user is prompted whether to proceed. If so, the decision is persisted as a preference for that URL for any future server instance creation and interactions using that URL, including cloning a server instance to another org/space.
- Application environment variables: Users can now specify environment variables both in the application deployment wizard, or edit them after an application is deployed through the Cloud Foundry server editor.
- Free-form application memory choices: Restriction on application memory choices have been removed. Users can specify any memory choice for applications in MB.
- Console enhancements: The Cloud Foundry console now supports loggregator, and loggregator content is streamed to the Cloud Foundry console when applications are running and the application’s console is open.

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:

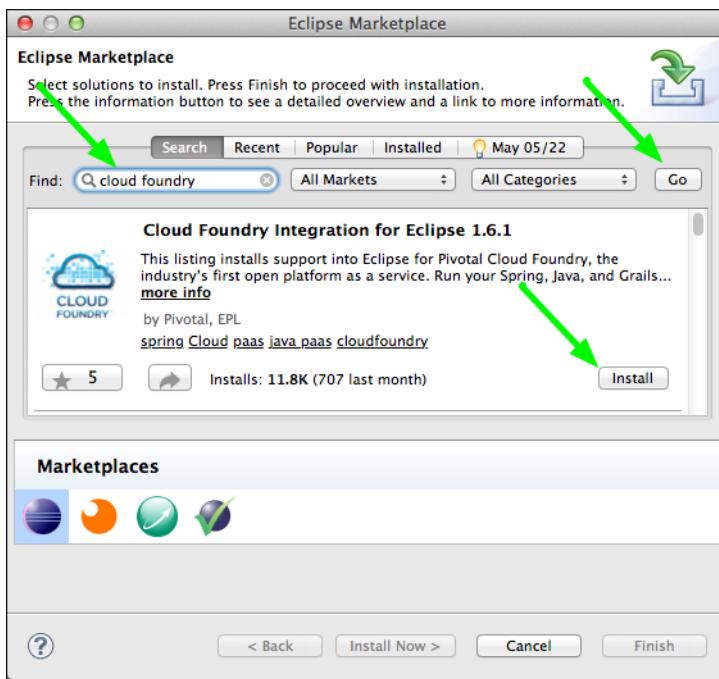
- [Install to Eclipse from Marketplace](#)
- [Install to STS from Extensions Tab](#)
- [Install from a Local Repository](#)

Install to Eclipse from Marketplace

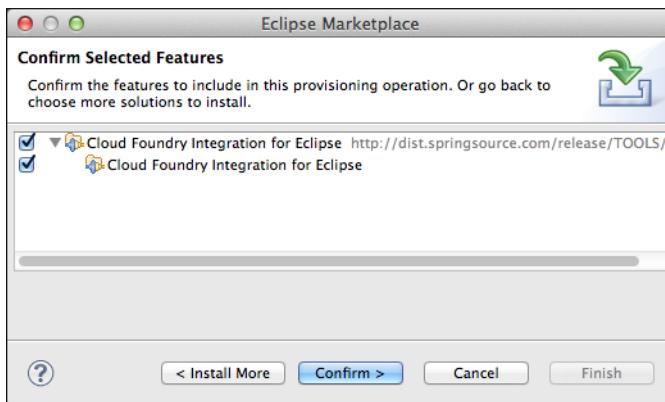
Follow these instructions to install the Cloud Foundry Eclipse Plugin to Eclipse from the Eclipse Marketplace.

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.

See [About the Plugin User Interface](#) below for a description of the plugin tabs and the information in each.

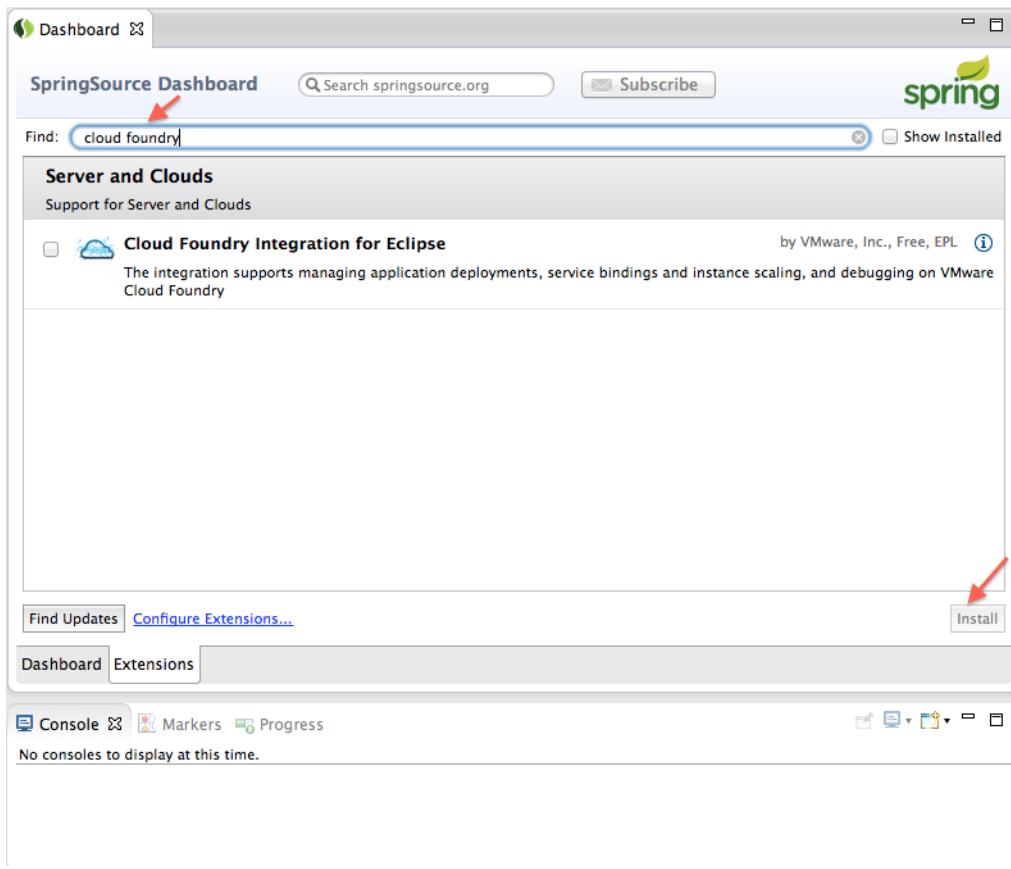


Install to STS from Extensions Tab

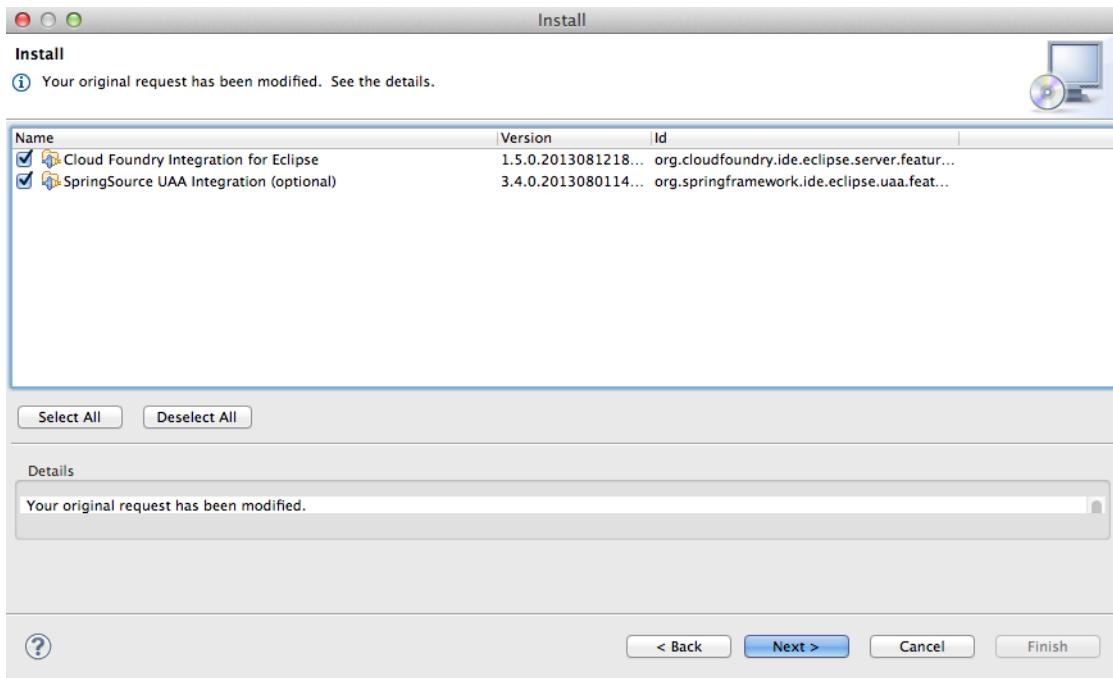
Follow these instructions to install the Cloud Foundry Eclipse Plugin to STS from the **Extensions** tab.

1. Start STS.
2. Select the **Extensions** tab from the STS Dashboard.
3. Enter “Cloud Foundry” in the **Find** field.

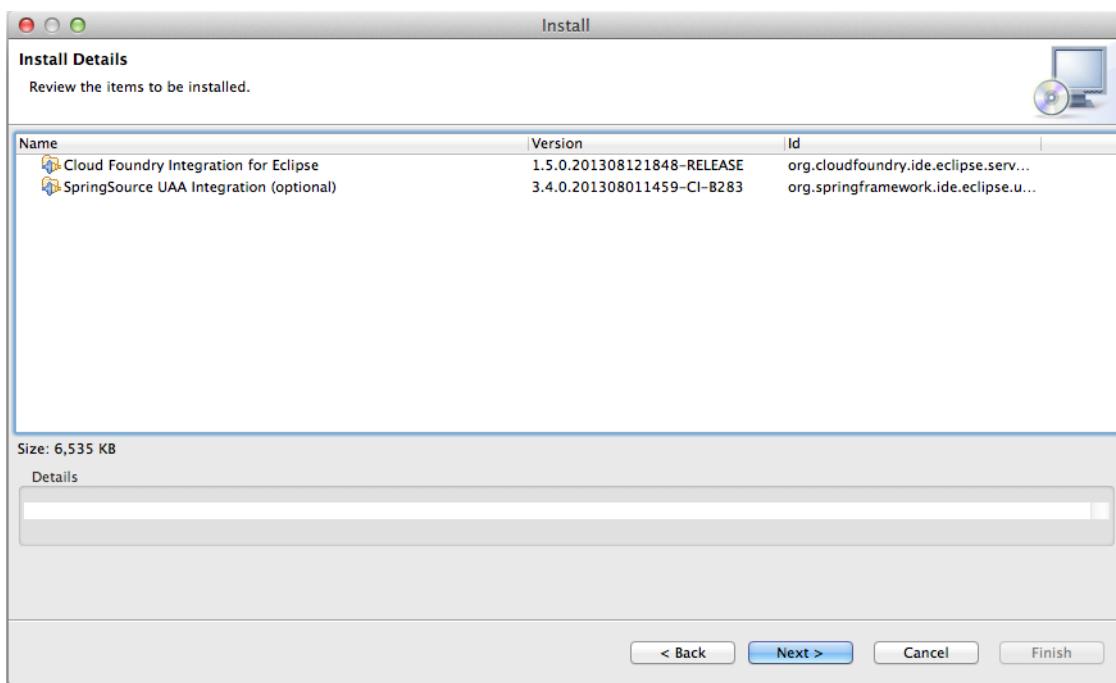
4. Select **Cloud Foundry Integration for Eclipse** and click **Install**.



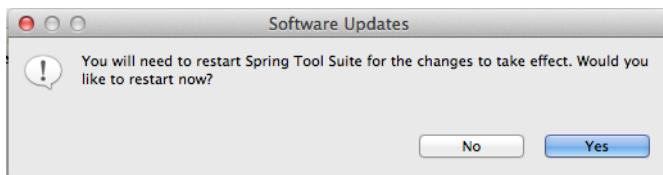
5. In the **Install** window, click **Next**.



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.

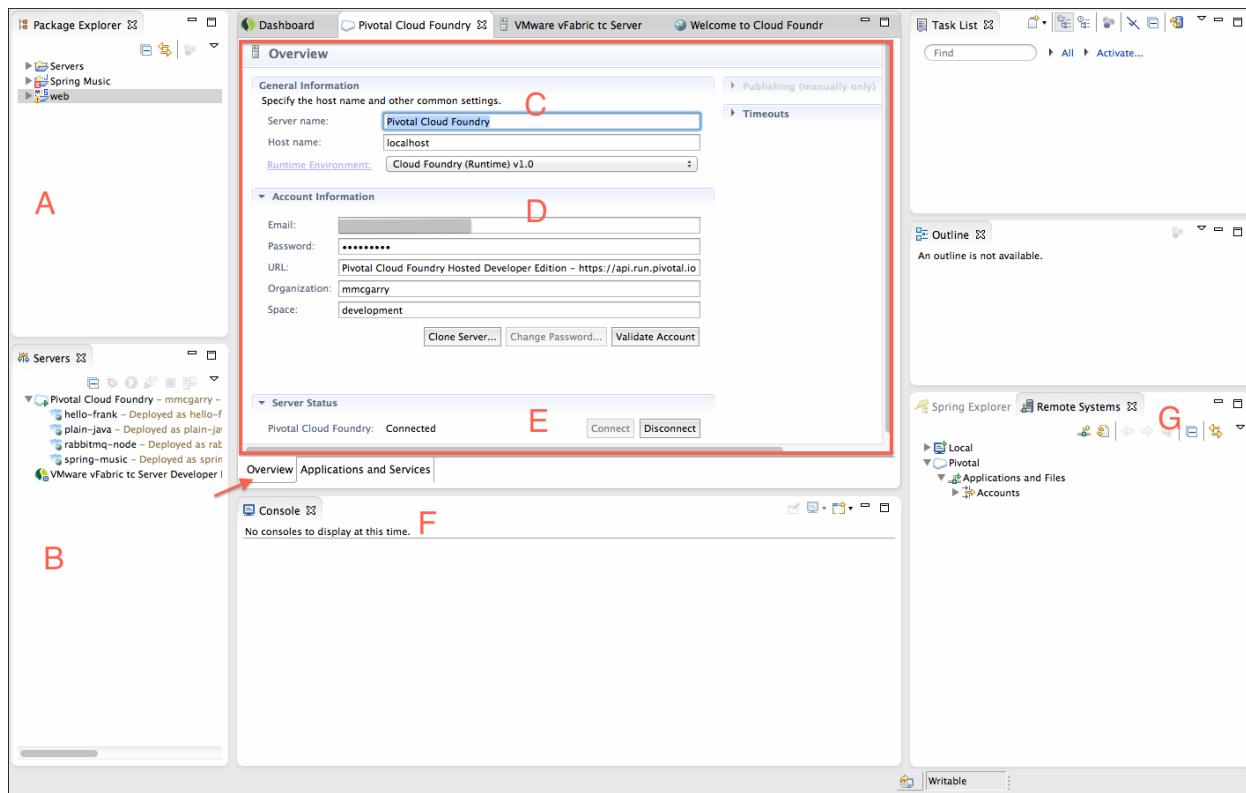
Note that the Cloud Foundry editor allows you to work with a single Cloud Foundry space; each space is represented by a distinct server instance in the **Servers** view (B). Multiple editors, each targeting a different space, can be open simultaneously. However, only one editor targeting a particular 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 follow panes are present when the **Applications and Services** tab is selected:

- H — The **Applications** pane lists the applications deployed to the target space.
- I — The **Services** pane lists the services provisioned in the targeted space.
- J — The **General** pane displays the following information for the application currently selected in the **Applications** pane:
 - **Name**
 - **Mapped URLs** – Lists URLs mapped to the application. You can click a URL to open a browser to the application within Eclipse (or STS); and click the pencil icon to add or remove mapped URLs. See [Manage Application URLs](#).
 - **Memory Limit** – The amount of memory allocated to the application. You can use the pull-down to change the memory limit.
 - **Instances** – The number of instances of the application that are deployed. You can use the pull-down to change number of instances.
 - **Start, Stop, Restart, Update and Restart** — The controls that appear depend on the current state of the application. The **Update and Restart** command will attempt an incremental push of only those application resources that have changed. It will not perform a full application push. See [Push Application Changes](#) below.

- K — The **Services** pane lists services that are bound to the application currently selected in the **Applications** pane. Note the icon in the upper right corner of the pane — it allows you to create a service, as described in [Create a Service](#).

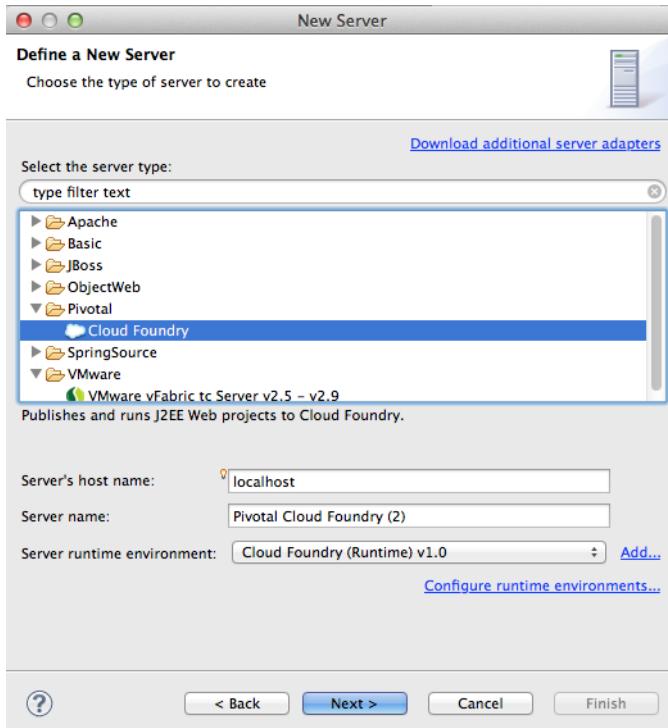
The screenshot shows the Pivotal Cloud Foundry interface with a red border around the main content area. The top navigation bar includes 'Dashboard', 'Pivotal Cloud Foundry', and 'VMware vFabric tc Server Developer Edition v2.9'. The left sidebar has 'Applications' and 'Services' sections. The 'Applications' section (labeled H) shows deployed applications: 'hello-frank', 'plain-java' (selected), and 'web'. The 'Services' section (labeled I) shows available services: '--mpmredis' (provider: redisclocloud, plan: 20mb) and 'elephantsql-e3c73' (provider: elephantsql, plan: turtle). The right side contains three panels: 'General' (labeled J) with application details (Name: plain-java [Started], Mapped URLs: plain-java, Memory limit: 512M, Instances: 1, Stop, Restart buttons); 'Application Services' (labeled K) showing bound services; and 'Instances' (labeled L) showing one instance (ID: 0, Host: 10.10.81.4, CPU: 5.1627..., Memory: 0M (512M), Disk: 87M (1024M), Uptime: 0h:0m:0s). A red arrow points from the 'Create a service' icon in the Applications pane to the 'Create a Service' link in the Services pane. A red box highlights the 'Applications and Services' tab at the bottom.

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.



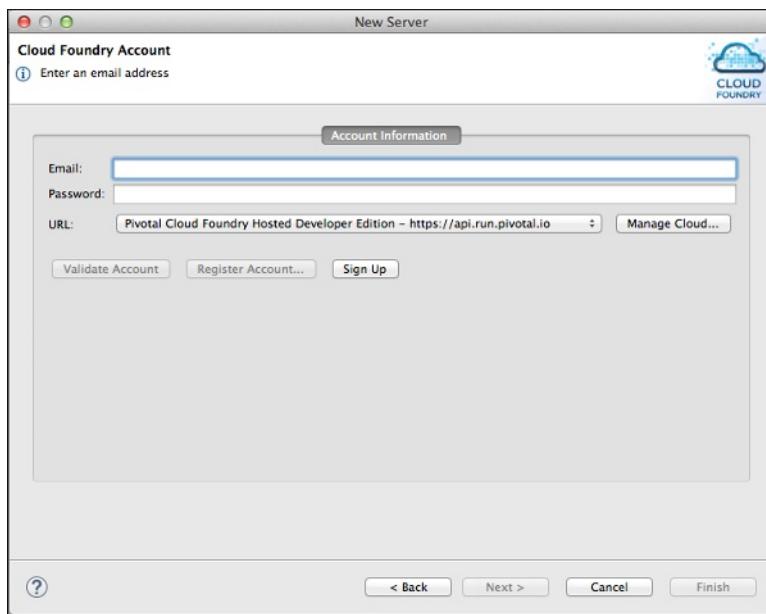
3. On the **Cloud Foundry Account** window, if you already have a Pivotal Cloud Foundry Hosted Developer Edition

account, enter your email account and password credentials and click **Validate Account**.

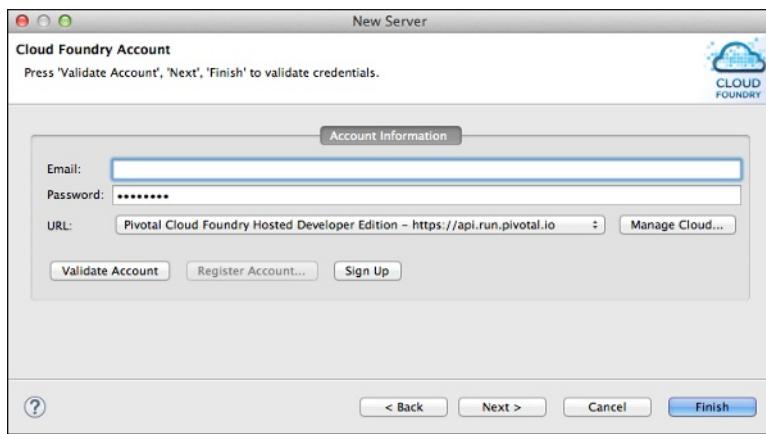
Note: By default, the **URL** field points to the Pivotal Cloud Foundry Hosted Developer Edition URL of <https://api.run.pivotal.io>. If you have a Pivotal Elastic Runtime account, refer to the [Logging into the Developer Console](#) topic to determine your Pivotal Elastic Runtime URL. Click **Manage 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.

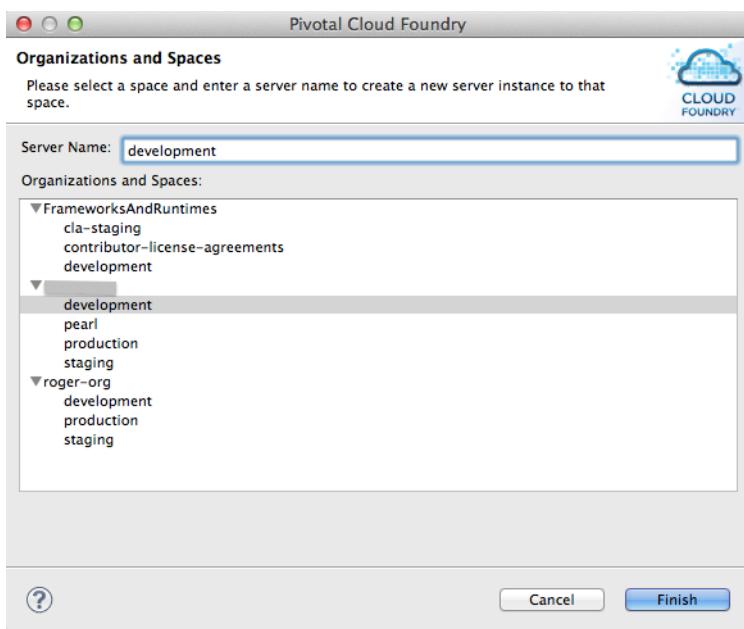


4. The **Cloud Foundry Account** window is refreshed and displays a message indicating whether or not your credentials were valid. Click **Next**.



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.



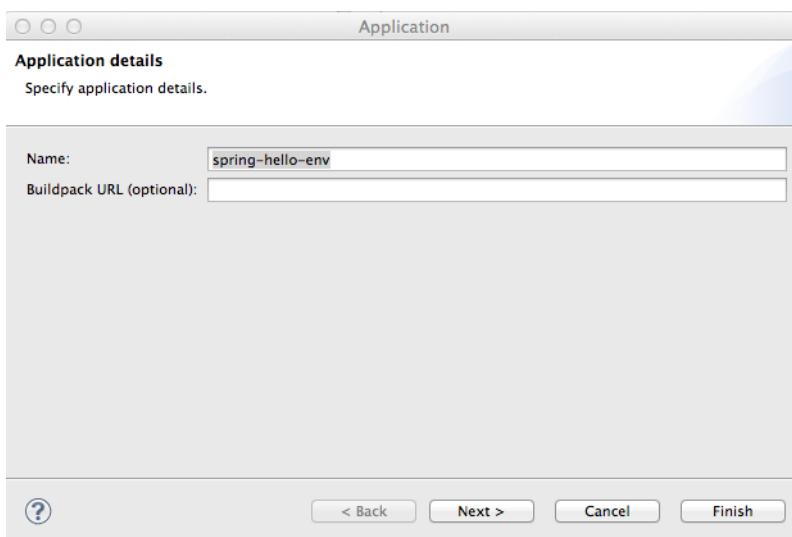
- Once you have successfully configured the Pivotal Cloud Foundry server, it will appear in the **Servers** view of the Eclipse or STS user interface. To familiarize yourself with the plugin user interface, see [About the Plugin User Interface](#). When you are ready, proceed to [Deploy an Application](#).

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.

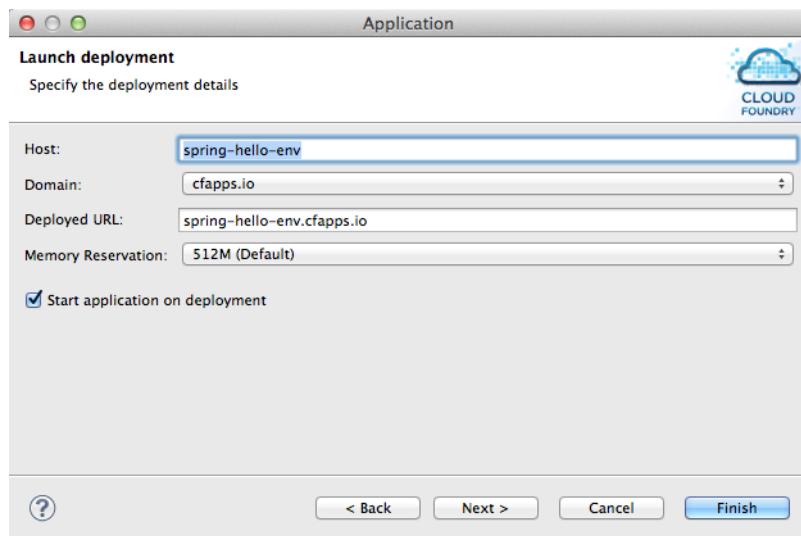
Domain — Contains the default domain. If you have mapped custom domains to the target space, they appear in the pull-down list.

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.

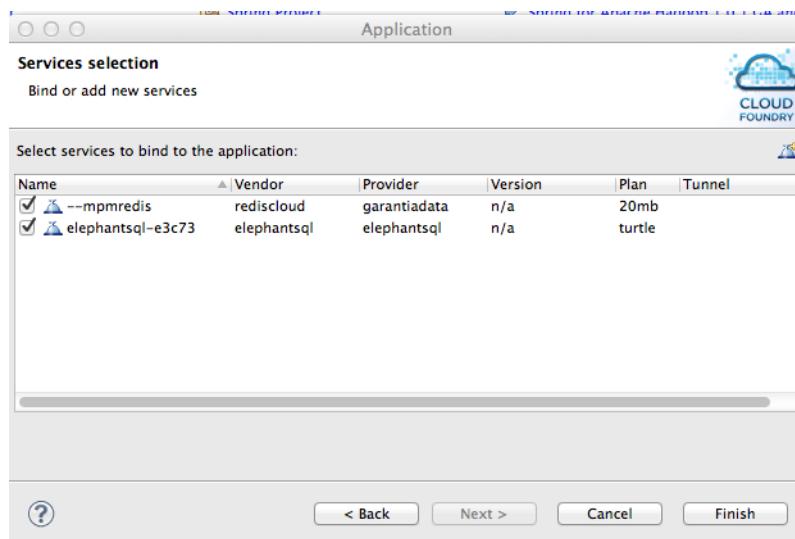
Deployed URL — By default, contains the value of the **Host** and **Domain** fields, separated by a period (.) character.

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](#).



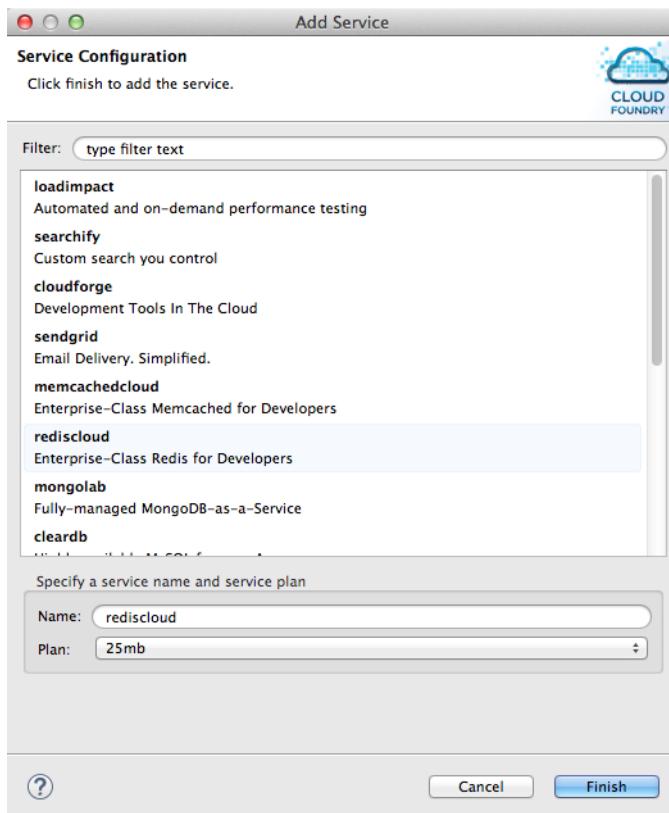
As the deployment proceeds, progress messages appear in the **Console** view. When deployment is complete, the application is listed in the **Applications** pane.

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

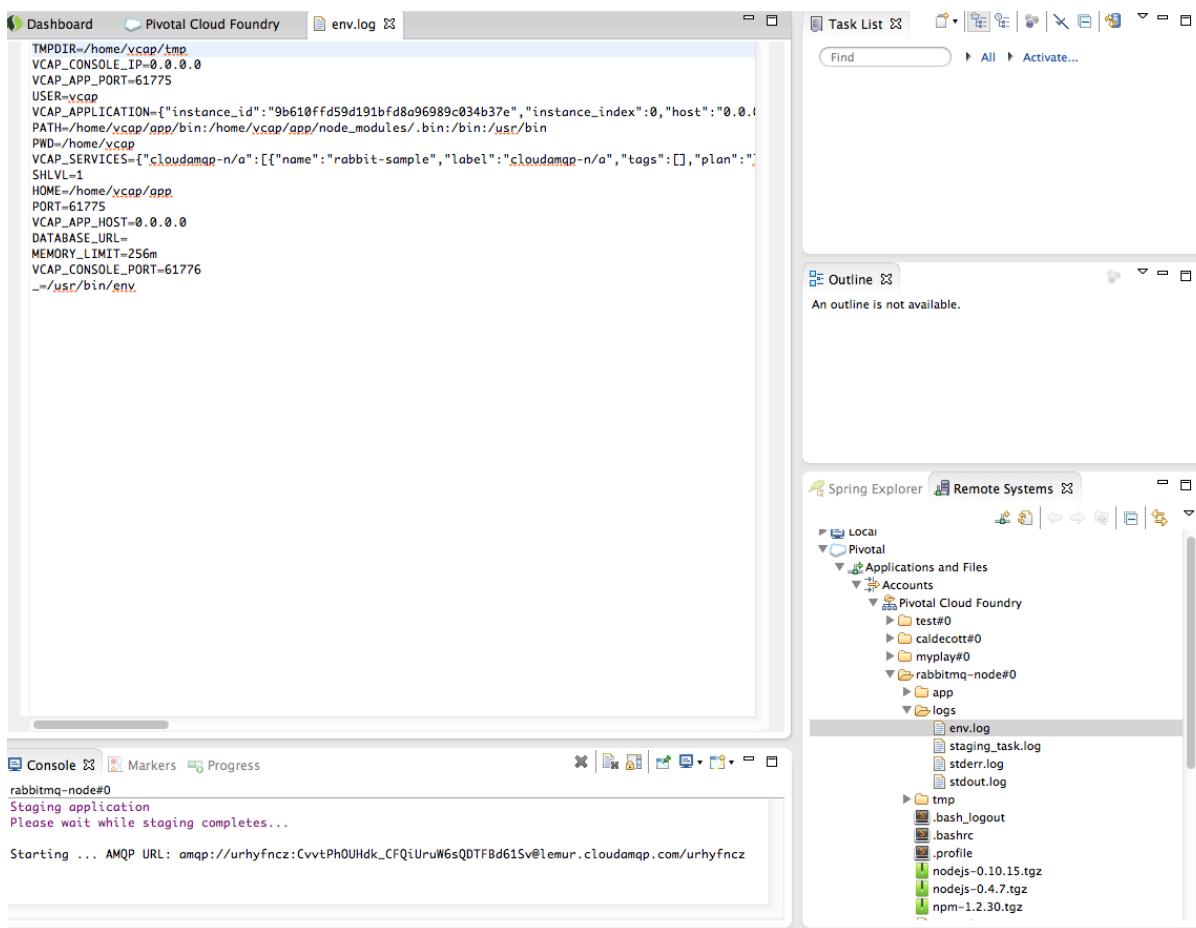
You can bind a service to an application when you deploy it. To bind a service to an application that is already deployed, drag the service from the **Services** pane to the **Application Services** pane. (See the area labelled "G" in the screenshot in the [Applications and Services](#) above.)

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 Tab](#).

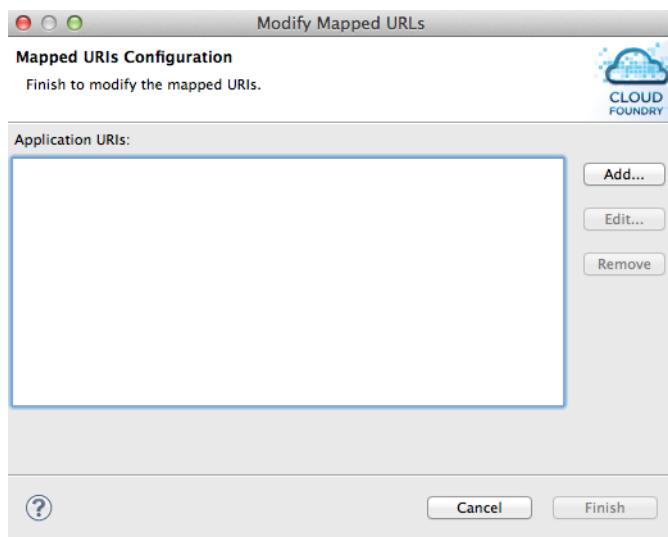
Push Application Changes

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:

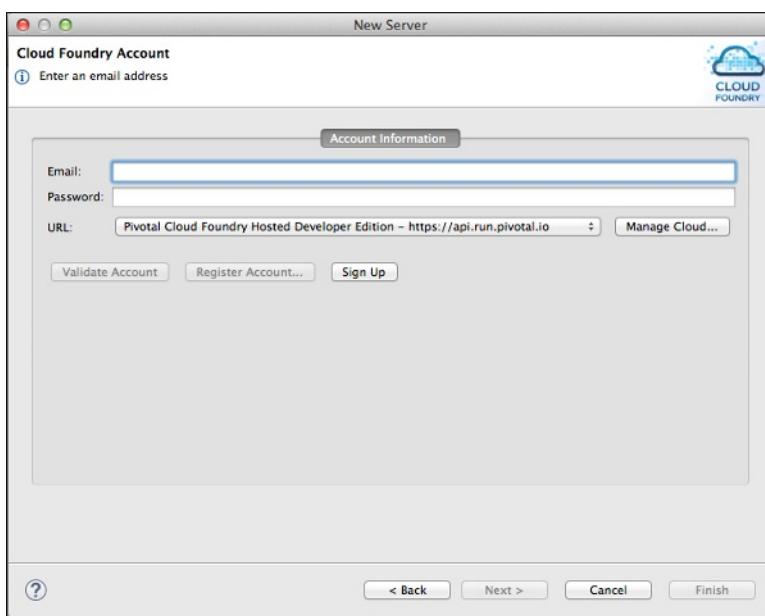
1. Either:
 - In the Cloud Foundry server instance editor “Overview” tab, click on the “Clone Server” button, or
 - Right-click a Cloud Foundry server instance in the **Servers** view, and select **Clone Server** from the context menu,
2. On the **Organizations and Spaces** window, select the space you 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 **Finish**.

Add a Cloud Foundry Instance URL

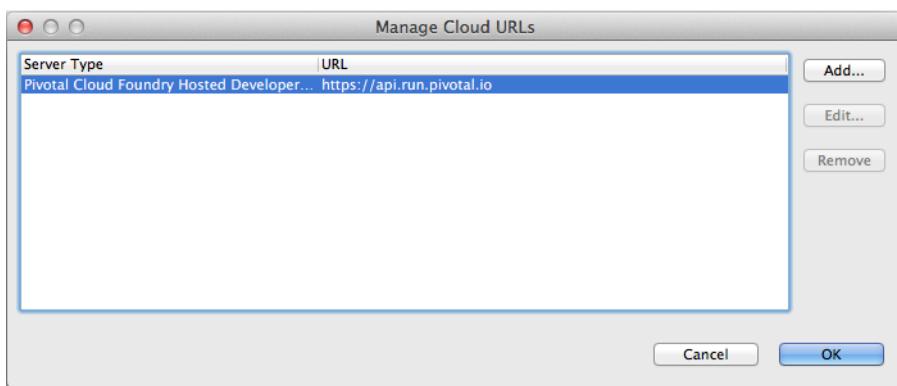
You can configure the plugin to work with any Cloud Foundry instances to which you have access. To do so:

1. Perform steps 1 and 2 of [Create a Cloud Foundry Server](#).

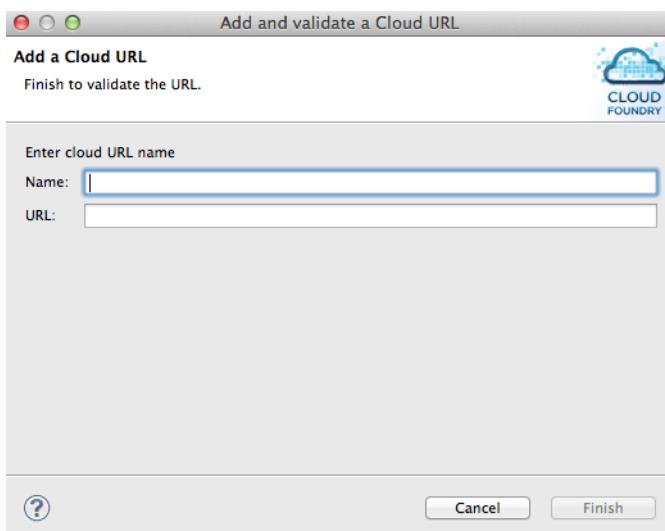
2. On the **Cloud Foundry Account** window, enter the email account and password you use to log on to the target instance, and then click **Manage Cloud URLs**



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 v6 and version 1.0.1 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>1.0.1</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>1.0.1</version>
    <configuration>
      <target>http://api.run.pivotals.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>1.0.1</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:1.0.1'
        ...
    }
}

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"
            provider = "rabbitmq"
            version = "n/a"
            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 cf-push
```

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 `cf.username` and `cf.password` with the Cloud Foundry security credentials parameters to the `gradle.properties` file as follows:

```
cf.username=user@example.com  
cf.password=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
cf-login	login -u USERNAME	\$ gradle cf-login
cf-logout	logout	\$ gradle cf-logout
cf-app	app APPNAME	\$ gradle cf-app [-Pcf.application=APPNAME]
cf-apps	apps	\$ gradle cf-apps
cf-target	api	\$ gradle cf-target
cf-push	push	\$ gradle cf-push [-Pcf.application=APPNAME] [-Pcf.uri=URL] [-Pcf.startApp=BOOLEAN]
cf-start	start APPNAME	\$ gradle cf-start [-Pcf.application=APPNAME]
cf-stop	stop APPNAME	\$ gradle cf-stop [-Pcf.application=APPNAME]
cf-restart	restart APPNAME	\$ gradle cf-stop [-Pcf.application=APPNAME]
cf-delete	delete APPNAME	\$ gradle cf-delete [-Pcf.application=APPNAME]
cf-scale	scale APPNAME -i INSTANCES	\$ gradle cf-scale [-Pcf.application=APPNAME] [-Pcf.instances=INTEGER]
cf-env	env APPNAME	\$ gradle cf-env [-Pcf.application=APPNAME]
cf-services	services	\$ gradle cf-services
cf-create-service	create-service SERVICE PLAN SERVICE_INSTANCE	\$ gradle cf-create-services
cf-delete-services	delete-service SERVICE_INSTANCE	\$ gradle cf-delete-service
cf-bind	bind-service APPNAME SERVICE_INSTANCE	\$ gradle cf-bind-services
cf-unbind	unbind-service APPNAME SERVICE_INSTANCE	\$ gradle cf-unbind

Node.js Buildpack

Use the Node.js buildpack with Node or JavaScript applications.

See the following topics:

- [Tips for Node.js Developers](#)
- [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 that you are using cf 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		To specify the versions of Node.js and npm an application requires, edit the application's <code>package.json</code> , as described in "node.js and npm versions" on https://github.com/cloudfoundry/heroku-buildpack-nodejs .
	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
	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		
npm	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		
	1.1.32		
	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 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 on Rails Apps](#)
- [Tips for Ruby Developers](#)
- [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 on Rails Apps

This guide is intended to walk you through deploying a Ruby on Rails (RoR) 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.

Deploy a RoR Sample Application

This section describes how to deploy a RoR sample application named `rails_sample_app`.

To learn how to deploy your own RoR app, see [Deploy Your RoR Application](#).

Prerequisites

- [Git](#) configured on your workstation and a [GitHub](#) account
- Basic RoR knowledge

Step 1: Get a Sample RoR App from GitHub

Run `git clone https://github.com/cloudfoundry-samples/rails_sample_app` to clone `rails_sample_app` from GitHub.

Step 2: 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 3: Deploy the Sample App

1. `cd` to the cloned app directory.
2. Run `cf create-service elephantsql turtle rails-postgres`.
3. Run `cf push rails_sample_app -n [HOST_NAME]`.

Example: `cf push rails_sample_app -n jc0825-rails`

`cf push [APP_NAME]` pushes the specified app from the current directory to Elastic Runtime. Use the `-n` flag to create a unique host name for the app to avoid URL collisions with other apps.

`cf push` automatically searches for an [application manifest YAML file](#) (`manifest.yml`) and also determines the appropriate [buildpack](#) for the app. A manifest specifies what `cf push` should do with the app and the buildpack provides framework and runtime support for your app.

The example 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 and the route, and upload the app to Elastic Runtime. It then binds the app to the `rails-postgres` service you defined in the previous step, 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

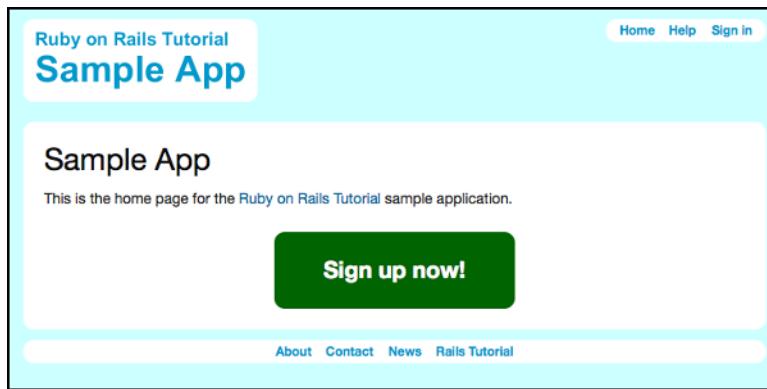
Showing health and status for app rails_sample_app in org Cloud-Apps / space development as clouduser@example.com...
OK

requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: jmt2014-rails.example.com

      state     since            cpu    memory      disk
#0   running   2014-08-25 03:32:10 PM  0.0%  68.4M of 256M  73.4M of 1G
```

Step 4: Test the Sample App

You've deployed your first app to Elastic Runtime! Now let's access your app online using its URL. In the terminal output, 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`. The image shows the resulting webpage for the `rails_sample_app`.



You can now use the `cf` CLI or the [Developer Console](#) to review information and administer your app and your Elastic Runtime account. For example, you could edit the `rails_sample_app/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 Developer Console](#).

Deploy Your RoR Application

This section describes how to deploy your RoR application to Elastic Runtime.

Prerequisites

- [Bundler](#) configured on your workstation

- (**Optional**) [Git](#) configured on your workstation and a [GitHub](#) account
- Intermediate to advanced RoR knowledge
- (**Optional**) Robust production web server for Ruby and RoR apps, such as Unicorn
- Your Rails 4.x app that runs locally on your workstation

Step 1: Add the rails_12factor Gem to Your Gemfile

1. Add `gem rails_12factor group: :production` to your Gemfile.

Note: This gem includes the `rails_serve_static_assets` gem that enables your Rails app to run in production with the static assets serving functionality. The `rails_serve_static_assets` gem does this by automatically modifying the `config.serve_static_assets` configuration parameter from `false` to `true`. Do not change the value for this option. For more information, refer to the [Rails_12factor GitHub README](#).

2. Run `bundle install` to update your Gemfile.

Step 2: (Optional) Configure Automatically Invoked Rake Tasks for Deployed Applications

Elastic Runtime does not provide a way to run a Rake task on a deployed application. For Elastic Runtime to automatically invoke a Rake task while an app is deployed, you must include the Rake task in your application and also configure the application start command with the `command` attribute in the application manifest, `manifest.yml`. 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
```

For information on how to configure a manifest file, refer to [Step 4: Configure the Manifest File](#).

Step 3: Configure a Service Instance for Your RoR Application

This section describes using the CLI to configure a PostgreSQL managed service instance for your app. You can use either the CLI or the [Developer Console](#) 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 bind your application to pre-provisioned external service instances.

For more information on creating and using service instances, refer to the [Services Overview](#) topic.

Run `cf marketplace` to view managed and user-provided services and plans available to you.

The example shows three of the available managed database-as-a-service providers and their offered plans: `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.

The example shows creating a service instance named `rails-postgres` that uses the `elephantsql` service and the `turtle` plan.

```
$ cf create-service elephantsql turtle rails-postgres
Creating service rails-postgres in org Cloud-Apps / space development as clouduser@example.com...
OK
```

Step 4: Configure the Manifest File

You can specify deployment options in a manifest that the `cf push` command uses.

For example, this section describes how to configure the manifest to bind the service that you created in the previous step. For more information about application manifests and supported attributes, refer to the [Deploying with Application Manifests](#) topic.

To bind a service to your app, add a `services` block to the `applications` block and specify the service instance you created in Step 3.

```
---
applications:
- name: rails-sample
  memory: 256M
  instances: 1
  path: .
  command: bundle exec rake db:migrate && bundle exec rails s -p $PORT
  services:
    - rails-postgres
```

You can also bind a service using the CLI or the [Developer Console](#).

Step 5: 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 6: (Optional) Configure a Production Server

Elastic Runtime uses the default 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 set a different production web server for your app:

1. Modify your app to use the production web server.

- Configure the `web` setting in a Procfile to point to the production web server.

When you deploy, `cf push` detects that your app requires the Cloud Foundry Ruby [buildpack](#), and that buildpack uses the modified Procfile to configure your app with the changed production web server settings. For an example of how to configure an application and a Procfile to use a different web server, refer to the [Heroku Deploying Rails Applications with Unicorn](#) topic.

Step 7: Deploy Your Application

When you deploy your application, `cf push`:

- Creates a URL that is the route to your application. The URL is in the form HOST.DOMAIN, where HOST is the APP_NAME and DOMAIN is specified by your administrator. Your DOMAIN is `example.com`.
- Searches for an application manifest YAML file. A manifest specifies what `cf push` should do with the app.
- Determines the appropriate buildpack for the app. A buildpack provides framework and runtime support for your app.

 **Note:** You must use the CLI to deploy apps.

For example: `cf push my-app` creates the URL `my-app.example.com`.

- `cd` to the app directory that you want to deploy.

- Run `cf push [APP-NAME]`.

The URL for your app must be unique from other apps that Elastic Runtime hosts or the push will fail. You can use the following options 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.
- Run `cf help push` to view other options for this command.

- Launch a new terminal window and run `cf logs [APP-NAME]` to simultaneously view log activity while the 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.

```
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: jmt0814-rails.example.com

state      since            cpu    memory      disk
#0  running   2014-08-13 02:45:51 PM  0.0%  69M of 256M  72.7M of 1G
```

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 v6](#) topic.

 **Note:** You cannot perform certain tasks in the CLI or the [Developer Console](#) 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 Developer Console, depending on your user role, refer to the [Understanding Developer Console 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 [Developer Console](#), 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: Configure a Service Instance for Your RoR Application](#) and [Step 4: Configure the Manifest File](#).
- 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 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.

Learn More

For more information on how to develop, deploy, and manage applications, refer to the [Developer Guide](#).

Tips for Ruby Developers

This page assumes that you are using cf 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, cf 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 on 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 && rails s
```

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 `cf` 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.
	1.3.2		

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
```

Configure Service Connections for Ruby

This page assumes that you are using cf 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/exampled"
      }
    }
  ]
}
```

Based on this `VCAP_SERVICES`, the Ruby buildpack creates the following `DATABASE_URL` environment variable:

```
DATABASE_URL = postgres://exampleuser:examplepass@babar.elephantsql.com:5432/exampled
```

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 [Migrate a Database on Cloud Foundry](#).

Troubleshooting

If you have trouble connecting to your service, run the `cf logs` command to view log messages for your application. The results of the `cf files my_app logs/env.log` command includes the value of the `VCAP_SERVICES` environment variable.

Example:

```
$ cf files my_app logs/env.log
Getting files for app my_app in org my_org / space my_space as a.user@example.com...
OK

TMPDIR=/home/vcap/tmp
VCAP_APP_PORT=64585
USER=vcap
VCAP_APPLICATION={"instance_id":"7dd14302a0804727a036b3b3f55300dc","instance_index":0,"host":"0.0.0.0",
"port":64585,"started_at":"2014-01-31 21:53:34 +0000","started_at_timestamp":1391205214,
"start":"2014-01-31 21:53:34 +0000","state_timestamp":1391205214,
"limits":{"mem":512,"disk":1024,"fds":16384}, "application_version": "c1901bd3-ad2a-40f5-a8fd-204a901d038e",
"application_name": "my_app", "application_uris": ["my_app.example.com"],
"version": "c1901bd3-ad2a-40f5-a8fd-204a901d038e", "name": "my_app", "uris": ["my_app.example.com"], "users": null}
PATH=/bin:/usr/bin
PWD=/home/vcap
VCAP_SERVICES={"p-mysql-n/a":[{"name": "p-mysql", "label": "p-mysql-n/a", "tags": ["postgres", "postgresql",
"relational"], "plan": "small_plan", "credentials": {"uri": "postgres://lrraxnih:eBawTKceuvKGZycBvYUv5Bd6B-X1m4a9t@sample.p-mysqlprovider.com:5432/lraaxnih"}}]}
SHLVL=1
HOME=/home/vcap/app
PORT=64585
VCAP_APP_HOST=0.0.0.0
DATABASE_URL=postgres://lrraxnih:eBawTKceuvKGZycBvYUv5Bd6B-X1m4a9t@sample.p-mysqlprovider.com:5432/lraaxnih
MEMORY_LIMIT=512m
_=~/usr/bin/env
```

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](#)

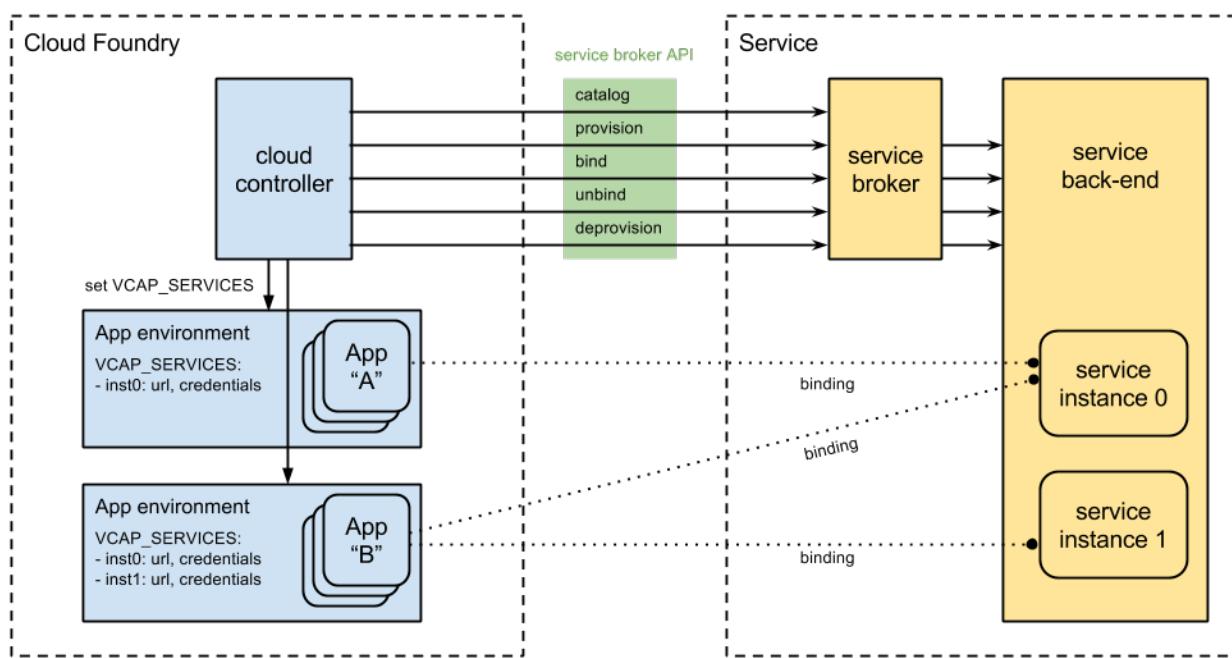
Overview

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
2.3	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 v2.2

The key change from v2.2 to v2.3 of the Services API is the addition of SSO functionality for users to access external service dashboard pages.

- A service can request a dashboard client in the service broker catalog.

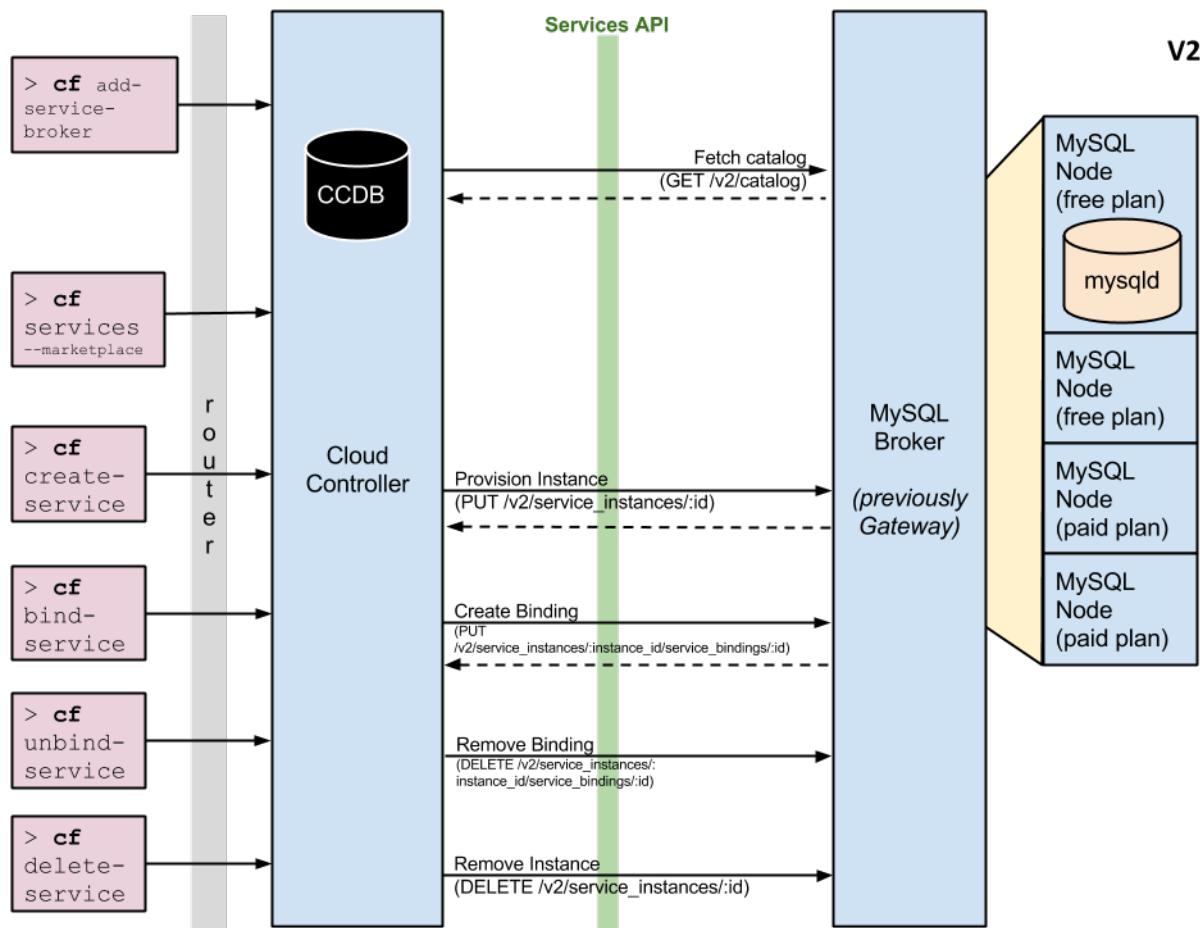
For details, see the [Dashboard SSO docs](#)

Dependencies

v2.3 of the services API has been supported since [final build 169](#) of [cf-release](#).

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: 2.3
```

Authentication

Cloud Controller (final release v145+) authenticates with the Broker using HTTP basic authentication (the `Authentication:` 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 http://username:password@broker-url/v2/catalog
```

Response

STATUS CODE	DESCRIPTION
200 OK	The expected response body is below

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. Eg. mysql, relational, redis, key-value, caching, messaging, amqp.
metadata	object	A list of metadata for a service offering. For more information, see Service Metadata .
requires	array-of-strings	A list of permissions that the user would have to give the service, if they provision it. The only permission currently supported is <code>syslog_drain</code> .
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/:id
```

Note: the `: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/:id -d '{
  "service_id": "service-guid-here",
  "plan_id": "plan-guid-here",
  "organization_guid": "org-guid-here",
  "space_guid": "space-guid-here"
}' -X PUT
```

In this case, `: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.
409 Conflict	Shall be returned if the requested service instance already exists. The expected response body is "{}"
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.

All other status codes will be interpreted as an error and cloud controller will inform the user that the provision failed.

We have chosen to require empty JSON in the response 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.

Body

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 ("9189kdfsk0vfnku" 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/9189kdfsk0vfnku"
}
```

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](#).

Request

Route

```
PUT /v2/service_instances/:instance_id/service_bindings/:id
```

Note: The `:id` of a service binding is provided by the Cloud Controller. `:instance_id` is the ID of a previously-provisioned service instance; `: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/:id -d '{
  "plan_id": "plan-guid-here",
  "service_id": "service-guid-here",
  "app_guid": "app-guid-here"
}' -X PUT
```

In this case, `:instance_id` refers to the id of an existing service instance in a previous provisioning, while `: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.
409 Conflict	Shall be returned if the requested binding already exists. The expected response body is "{}"
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.

All other status codes will be interpreted as an error and cloud controller will inform the user that the bind failed. Additionally, an unbind request will be sent to the broker to prevent an orphan being created on the broker.

We have chosen to require empty JSON in the response 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.

Body

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. The syslog_drain permission is required for logs to be automatically wired to applications.

```
{
  "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/:id
```

The `:id` in the URL is the identifier of a previously created binding (the same `: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
--------------------	------	-------------

QUERY-STRING FIELD	TYPE	DESCRIPTION
service_id*	string	ID of the service from the catalog. While not strictly necessary, some brokers might make use of this ID.
plan_id*	string	ID of the plan from the catalog. While not strictly necessary, some brokers might make use of this ID.

cURL

```
$ curl 'http://username:password@broker-url/v2/service_instances/:instance_id/
service_bindings/:id?service_id=service-id-here&plan_id=plan-id-here' -X DELETE
```

Response

STATUS CODE	DESCRIPTION
200 OK	Binding was deleted. The expected response body is "{}"
410 Gone	Shall be returned if the binding does not exist. The expected response body is "{}"

All other status codes will be interpreted as an error and cloud controller will inform the user that the unbind failed. The binding will remain in the cloud controller database.

We have chosen to require empty JSON in the response 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.

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/:id
```

The `:id` in the URL is the identifier of a previously provisioned instance (the same `: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/:id?service_id=
service-id-here&plan_id=plan-id-here' -X DELETE
```

Response

STATUS CODE	DESCRIPTION
200 OK	Service instance was deleted. The expected response body is “{}”
410 Gone	Shall be returned if the service instance does not exist. The expected response body is “{}”
All other status codes will be interpreted as an error and cloud controller will inform the user that the deprovision failed. The service instance will remain in the cloud controller database.	
We have chosen to require empty JSON in the response 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.	

Broker Errors

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) as well as a JSON-encoded error payload.

This payload allows the broker to expose a message to end-user or operator. If a payload is included, only the message from the description field in the payload will be shown. If there is no payload, a generic error message containing the HTTP response code will be shown.

Response

RESPONSE FIELD	TYPE	DESCRIPTION
description	string	An error message explaining why the request failed. This message will likely be forwarded to the person initiating the request.

Orphans

The Cloud Controller database and a Broker are expected to store identical copies of existing instances and bindings. These two lists may potentially become inconsistent. For example, if a broker times out during a delete request, the Cloud Controller will be unsure whether that resource still exists on the broker. Cloud Controller will implement the following orphan prevention techniques:

- If a broker fails to provision or bind an instance, the Cloud Controller will immediately issue a deprovision or unbind request.
- If a broker fails to unbind or deprovision an instance, the Cloud Controller will periodically retry that DELETE request until it succeeds (or generates a 410). Eventually it will give up, but this technique will help clean up resources that remain when a broker fails to delete.

Managing Service Brokers

This page assumes that you are using cf 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 do to when you see them.

List Service Brokers

```
$ cf service-brokers
Getting service brokers as admin...
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 command `update-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 update-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

This page assumes that you are using cf v6.

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.

Note: the CLI commands documented below can only be run by an administrator.

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.

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 DEVELOPER CONSOLE
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

BROKER API FIELD	TYPE	DESCRIPTION	CC API FIELD	PIVOTAL CLI	PIVOTAL DEVELOPER CONSOLE
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 DEVELOPER CONSOLE
name	CLI string	A short name for the service plan to be displayed in a catalog.	name	X	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	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: amount: { usd: float }, unit: string This indicates the cost in USD of the service plan, and how frequently the cost is occurred, such as "MONTHLY" or "per 1000 messages".	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: cloud_controller.write,openid,cloud_controller.read,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

UAU 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 [UAA OmniAuth Strategy](#). See this [OmniAuth wiki page](#) for instructions on how to create your own strategy.

The UAA OmniAuth strategy is used to first get an authorization code, as documented in [this section](#) of the UAA documentation. The user is redirected back to the service (as specified by the `callback_path` option or the default `auth/cloudfoundry/callback` path) with the authorization code. Before the application / action is dispatched, the OmniAuth strategy uses the authorization code to [get a token](#) and uses the token to request information from UAA to fill the `omniauth.auth` environment variable. When OmniAuth returns control to the application, the `omniauth.auth` environment variable hash will be filled with the token and user information obtained from UAA as seen in the [Auth Controller](#).

Restrictions

- UAA clients are scoped to services. There must be a `dashboard_client` entry for each service that uses SSO integration.
- Each `dashboard_client_id` must be unique across the CloudFoundry deployment.

Resources

- [OAuth2 ↗](#)
- [Example broker with SSO implementation ↗](#)
- [Cloud Controller API Docs ↗](#)
- [User Account and Authentication \(UAA\) Service APIs ↗](#)

Example Service Brokers

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.

Packaging Pivotal One Products

This document is intended for product teams to learn how to package, distribute, and upgrade Pivotal One 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 will 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 Ops 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. In its first, 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. The contents of that archive are the top-level directories listed below.

 **Note:** Archiving folders as subdirectories of other folders is not supported.

Product Package - Stemcells Directory

The stemcells directory contains stemcells for your release.

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 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: "1.1"
stemcell:
  name:   bosh-vsphere-esxi-ubuntu
  file:   bosh-stemcell-1657-ubuntu.tgz
  version: "1657"
compiled_package:
  name:   redis
  file:   redis-0.2-dev-ubuntu-1657.tgz
  version: 0.2-dev
releases:                                # [5]
-----
```

```

- 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 ...
image: iVBORw0KGgoAAAANSUhEUgAAAG # [6]
rank: 1
post_deploy_errands:
- name: test-redis # [7]

form_types:
- name: Redis
  label: Redis
  description: This is my little Redis
  property_inputs: # [9]
    - 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:
- name: snapshots
  type: collection
  configurable: true
  optional: false
  property_blueprints:
    - name: seconds
      type: integer
    - name: changes
      type: integer

job_types: # [11]
- name: redis
  resource_label: redis label for resource page
  template: redis
  release: redis

resource_definitions: # [12]
- 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 # [13]
  default: 1

static_ip: 0 # [14]
dynamic_ip: 1
max_in_flight: 1 # [15]
serial: false # [16]

instance_definitions:
- name: instances
  type: integer
  configurable: true
  default: 1

property_blueprints: # [17]

```

```

- name: vm_credentials
  type: salted_credentials
  default:
    identity: vcap
- name: password
  type: secret
  configurable: true
  optional: true
                                # [18]

manifest: |                         # [19]
  redis.password: (( password.secret ))
  redis.snapshots: (( .properties.snapshots.value ))

- name: test-redis
  resource_label: Test Redis
  template: test-redis
  release: redis-errands-release
  errand: true
                                # [20]
  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
      configurable: false
      default: 1

```

```
- 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_versions.
2. metadata_version: Every release of Ops manager expects a certain metadata schema. We are currently at 1.1.
3. stemcell: refers to the stemcell you placed in the stemcells directory (also applies to releases)
4. compiled_package: Optional. You can include compiled packages in the compiled_packages directory for faster install.
5. releases: An array of releases. BOSH Errand releases can be referred to here.
6. image: A base64 encoded version of the logo for your image.
7. post_deploy_errands: An array of BOSH errands that run after a successful deployment. The name of the errand must correspond with the job.
8. form_types: Each form that is displayed for user input is generated from a form type.
9. property_input: An HTML input for a form is a reference to a property_blueprint on a job_type (see below).
10. 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.
11. job_types: A list of the jobs that correspond with the job names in your BOSH manifest.
12. 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.
13. configurable: Must be true for user to edit, otherwise it’s disabled in the table or form.
14. static_ip / 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 try and use dynamic_ip when possible.
15. max_in_flight: Can be either a number or a ratio marked by a ‘%’ percentage symbol.
16. serial: true/false. If false this job will be deployed in parallel with other jobs.
17. property_blueprints: There are two kinds of blueprints. Property blueprints, their settings, types and defaults are further explored in the Types section.
18. optional: true/false. true means the value can be nil/blank. Defaults to false if this key is not specified.
19. manifest: Properties that you need to put in your BOSH manifest, underneath the appropriate job, are created here. This is how the form_type’s 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.
20. 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 - 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 will follow. It will be clear how to use these types when we look at manifest snippets.

string

Renders as a text field if configurable.

Accessor: value

integer

Renders as a text field if configurable. These property definitions allow you to additionally specify constraints as follows:

```
name: my-prop
type: integer
label: Foo
constraints:
  min: 200
  max: 300
  power_of_two: true
```

You don't 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

boolean

Renders as a checkbox if configurable.

Accessor: value

ca_certificate

Specially created for Rabbit because of the way SSL is done with Rabbit clusters. Renders as a textarea if configurable.

Accessor: value

domain

Used for things like foo.bar.com. Renders as a text field if configurable.

Accessor: value

wildcard_domain

Same as domain but can be used with a WildcardDomainVerifier.

Accessor: value

email

Renders as a text field if configurable. Validates that it's a sensible email address, e.g. foo@bar.com

Accessor: value

email_credentials

Renders as a pair of text fields as configurable, where the second field will be masked out with asterisks (*****).

Accessors: identity, password

host_mapping

Renders as textarea. Parses and validates as host mappings.

Accessors: value

http_url

Full URL with http or https protocol. Renders as text field.

Accessors: value

ip_address

e.g. 1.2.3.4. Renders as text field.

Accessor: value

ip_ranges

Comma-separated list of single IP addresses and IP ranges of the form 1.2.3.4-1.2.3.200. Renders as text input.

Accessor: value, parsed_ip_ranges (array)

multi_select_options

Renders a list of checkboxes (kind of like a multi-boolean). When giving a property definition this type, you must also specify a list of options as follows:

```
Same: my-prop type: multi_select_options label: Foo options: - name: checkbox1 label: Checkbox 1  
- name: checkbox2 label: Checkbox 2 - name: checkbox3 label: Checkbox 3
```

The value of such a property, that you'll have access to in the manifest parts of your metadata, will be an array of the names of the checked boxes.

Accessor: value (array of strings)

network_address_list

Comma separated list of network addresses

Accessor: value, parsed_network_addresses (array)

network_address

IP address, domain, or single host-name. Renders as a text field.

Accessor: value

network_address_port_pair

Renders as two text fields, the first one is validates as a network address, and the second as a port (i.e. an integer between 0 and 65535).

Accessor: address, port

port

Integer between 0 and 65535. Renders as text field.

Accessor: value

rsa_cert_credentials

A triple of private key, cert, and csr. If configurable, then in the UI it only renders a field for the private key and cert (both as textareas).

Accessors: public_key_pem, cert_and_private_key_pems

rsa_pkey_credentials

Just an RSA private key. Not supported in UI, so don't make it configurable.

Accessors: public_key_pem, private_key_pem

salted_credentials

A triple of identity, password, and salt. Not supported in UI, so don't make it configurable.

Accessors: sha512_hashed_password, identity, salt, password

simple_credentials

A pair of identity and password. Any strings will do. Renders as two text fields in the UI, with the password one being masked out with asterisks (******) out.

Accessors: identity, password

secret

A string that renders as a single password text input

Accessor: secret

smtp_authentication

Renders as a dropdown where you can select from 'plain', 'login', and 'cram_md5'. Specifically for Console, as this gets passed on to ActionMailer.

Accessor: value

More on Product Templates - Types with Random Values

You may wish to have some properties randomly generated rather than given by the user. Currently this behaviour is supported for the various _credentials types. To trigger auto-generation, the property_definition must be non-configurable. All of these types are "compound values" in the sense that they are made up of more than one basic thing, e.g. email_credentials is made up of an identity (email) and password. Auto-generation can be used to generate just part of a set of credentials when combined with partially specified default values. For example, a property blueprint like the following:

```
property_blueprints:
  - name: vm_credentials
    type: salted_credentials
    label: VM credentials
    configurable: false
    default:
      identity: vcap
```

Has its salt and password auto-generated, but not its identity.

email_credentials

Consists of identity and password. password can be auto-generated at deploy-time, but identity will not, so if you want the password auto-generated, specify a default for the identity.

rsa_cert_credentials

Generates a valid RSA private key, a CSR, and a self-signed cert. All in PEM format.

rsa_pkey_credentials

Generates a valid RSA private key (and there's a helper method to access the public key if needed)

salted_credentials

Generates an identity, salt, and password.

simple_credentials

Generates an identity and password.

secret

Generates a password.

More on Product Templates - Manifest Snippets

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 autogenerated.

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." Please 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. You can also use Ruby's ERB syntax, but requires you to know something about the evaluation context and Ops Manager's internal object graph. It is not recommended that anyone use ERB.

You can do the following things with spiff-like syntax:

```
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 access the identity access on the "bar"
      # property on this job (namely "foo")
      blah: (( bar.password ))
      hi: (( baz.value ))
  networkstuff:
    my_ip: (( first_ip ))           # this access 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:
  name: bosh-vsphere-esxi-ubuntu
  version: "1266"
  file: bosh-stemcell-1266-vsphere-esxi-ubuntu.tgz
  md5: e1ccb680bee9eda8ba7e81ff0b3421a8
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.
  template: 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 release cycle, and thus all previously released product installations should be migratable 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.

Product Package - compiled_packages directory

BOSH offers a new compiled feature as of December 2013. This feature will prevent BOSH from spending time booting compilation VMs to compile packages for a deployment.

Login to the Ops Manager Virtual Machine, sudo to root and run the following:

```
$ cd /home/tempest-web/tempest/web
$ BUNDLE_GEMFILE=bosh.Gemfile bundle exec bash target 172.16.66.17
$ BUNDLE_GEMFILE=bosh.Gemfile bundle exec bash deployments
+-----+-----+-----+
| Name | Release(s) | Stemcell(s) |
+-----+-----+-----+
| cf-2a8c2e075b8b46b5c | cf/154.3-dev | bosh-vsphere-esxi-ubuntu/1471 |
+-----+-----+-----+
$ BUNDLE_GEMFILE=bosh.Gemfile bundle exec bash export compiled_packages cf/154.3-dev bosh-vsphere-esxi-ubuntu/1471 /tmp
```

Understanding the Installation file

Pivotal Ops Manager has two types of YAML files it cares about: (1) product templates and (2) an “installation file”. Product templates are authored by you. It is used by Ops Manager to create forms and get user input. This file is never altered as a result of user input. The installation file is where all the user-entered data is stored, along with any auto-generated values such as non-configurable credentials, and IPs allocated to jobs. The job_types and property_blueprints of the product template files inform what the jobs and properties look like in the installation file. At install time, BOSH manifests are generated by first creating a template out of some boilerplate that Ops Manager provides and the “manifest” sections of the job_types in your product templates, and then rendering that template using values from the installation file.

Before you learn how to write content migrations, you must understand the structure of the installation file, as this is what you are migrating. You are not migrating the product template that you authored. This is a source of confusion for most.

To understand what the installation file looks like, you can login to your Ops Manager, decrypt, and view /var/tempest/workspaces/default/installation.yml

This file is encrypted using the login password for the Ops Manager’s admin user. You can decrypt it using this script as follows:

```
$ sudo wget http://bit.ly/1cW8tY7
$ mv 1cW8tY7 eos
$ sudo ruby eos decrypt <admin_password> installation.yml decrypted.yml
```

After decrypting the installation metadata, you will see something like the following:

```
---
installation_version: !binary |-
  MS4x
infrastructure:
  type: vsphere
  file_system:
    microbosh_vm_folder: pivotal_cf_vms_1da14f53e678361b2c32
    microbosh_template_folder: pivotal_cf_templates_1da14f53e678361b2c32
    microbosh_disk_path: pivotal_cf_disk_1da14f53e678361b2c32
components:
- type: microbosh
  guid: microbosh-15282863e97a58c4507a
  installation_name: microbosh-15282863e97a58c4507a
  product_version: 1.1.0.0
jobs:
- type: director
  guid: director-9fb5c60aed802815ef0d
  installation_name: director
  instances:
  - definition: instances
    value: 1
  resources:
  - definition: ram
    value: 3072
  - definition: ephemeral_disk
    value: 16384
  - definition: persistent_disk
    value: 20480
  - definition: cpu
    value: 4
  properties:
  - definition: vm_credentials
    value:
      identity: vcap
      salt: !binary |-
        MjMwOGU2MmU1YjJjNDE4ZQ==
      password: !binary |-
```

```

  properties:
    - definition: infrastructure
      value:
        identity: vcap
        password: !binary |-
          ODc3MzMmODlMjVlNDI2NGU5ZDU=
    - definition: director_credentials
      value:
        identity: director
        password: !binary |-
          MjNkMjMwZTdlnTBIzDcxZDAyZDc=
    - definition: nats_credentials
      value:
        identity: nats
        password: !binary |-
          ZGZkNTQ5ZGM1NjJkNjY2MmIxYzA=
    - definition: redis_credentials
      value:
        identity: redis
        password: !binary |-
          Zjg10TM1Yjc2ZDJmNzRjNjg3NmM=
    - definition: postgres_credentials
      value:
        identity: postgres
        password: !binary |-
          MTY0Zjh1NDkzYTM4YjBLMjJLNzk=
    - definition: blobstore_credentials
      value:
        identity: blobstore
        password: !binary |-
          Mzg5MDg5ZDU3ZDFKmjFjYzczoTY=
    - definition: health_monitor_credentials
      value:
        identity: health_monitor
        password: !binary |-
          NGRjYTQ2Njk1MDYxNDYzOWUxYjU=
    - definition: director_ssl
      value:
        private_key_pem: !binary |-
    - definition: resurrector_enabled
      value: true
  properties:
    - definition: infrastructure
      value:
        vsphere
    - definition: vcenter_ip
      value: 172.16.66.3
    - definition: login_credentials
      value:
        identity: root
        password: vmware
    - definition: network
      value: VM Network
    - definition: datacenter
      value: pineapple-dc
    - definition: cluster
      value: pineapple-cl
    - definition: datastore
      value: pineapple-ds
    - definition: resource_pool
      value: gold
    - definition: subnet
      value: 172.16.66.0/23
    - definition: reserved_ip_ranges
      value: 172.16.66.1-172.16.67.1, 172.16.67.100-172.16.67.255
    - definition: dns
      value: 10.80.130.1
    - definition: gateway
      value: 172.16.66.1
    - definition: ntp_servers
      value: 10.80.130.1
  ips:
    director-9fb5c60aed802815ef0d:
      - 172.16.67.2
      dynamic_for_static:
        - 172.16.67.3
    type: redis
    guid: redis-695b43f28ce9cf76610f
    installation_name: redis-695b43f28ce9cf76610f
    product_version: 1.0.0.6
  jobs:
    - type: redis
      guid: redis-81af61fabd99bc1a1486
      installation_name: redis
      instances:
        - definition: instances

```

```

value: 1
resources:
- definition: ram
  value: 1024
- definition: ephemeral_disk
  value: 2048
- definition: persistent_disk
  value: 3072
- definition: cpu
  value: 1
properties:
- definition: vm_credentials
  value:
    identity: vcap
    salt: !binary |-
      Yzk0ZGFmZDY4Nzc0MTc10Q==
    password: !binary |-
      NDY5MjZjODhjNGQ1ZTcyNw==
- definition: credentials
  value:
    identity: matt
    password: reider
- type: compilation
  guid: compilation-2794aeaf991cbac844d
  installation_name: compilation
  instances:
- definition: instances
  value: 1
resources:
- definition: ram
  value: 1024
- definition: ephemeral_disk
  value: 2048
- definition: persistent_disk
  value: 8192
- definition: cpu
  value: 1
ips:
  redis-81af61fabd99bc1a1486:
  - 172.16.67.44
  compilation-2794aeaf991cbac844d:
  - 172.16.67.45
  dynamic_for_static: []

```

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, please 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 / 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:
      - saml_login.sso_url
  - name: Verifiers::SmtpAuthenticationVerifier
    properties:
      - consoledb.smtp_address
      - consoledb.smtp_port
      - consoledb.smtp_hello_domain
      - consoledb.smtp_credentials
      - consoledb.smtp_authentication
      - consoledb.smtp_enable_starttls_auto
```

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-tempest.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>.

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 1.0.0.1 to 1.1.0.0

The following changes were made between version 1.0.0.1 released in November of 2013 and 1.1.0.0 released in March 2014:

install_time_verifiers

A product can now specify verifiers that can be run at install time (right after the user clicks the install button).

The name of the verifier specifies the ruby class name of the verifier to run (look in the app/models/verifiers folder for available verifiers). The properties key specifies which properties are passed to the verifier to instantiate it.

form_types

Forms are now specified in the metadata with a form_type key. Job types are now just containers for data and the UI is drawn from the form_types key. Each form type has a name (programmer facing), label (end user facing), a description string.

Forms can also optionally specify a verifier to run when the form is saved. They must also specify an array of property inputs that point to the properties in the jobs (ie. reference: job-name.property-name).

max_in_flight

Job types must specify max_in_flight (integer or %)

property_blueprint rename

property_definitions has been renamed to property_blueprints in job type

canaries

The number of canaries (integer) can now be specified for a job_type

CPU as power of two

CPU resource definition must specify a power_of_two constraint

MicroBOSH 1.1 features

To take advantage of the following features, a product must specify a dependency on MicroBOSH 1.1:

- compiled_packages can be specified for a product
- Product teams can specify jobs to deploy in parallel with serial: false

Typed values:

- Added network_address_list (comma separated list of network addresses)
- Added secret (password field to replace credentials when identity is unnecessary).

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 'Lifecycle Errands' section of the Pivotal MySQL Dev product tile. The 'Lifecycle Errands' tab is selected. It lists three errands:

- Lifecycle Errands**: Selected (indicated by a checked checkbox).
- Post-Install Errands**: Selected (indicated by a checked checkbox).
- Broker Registrar**: Not selected (indicated by an unchecked checkbox).

A tooltip for the 'Broker Registrar' errand states: "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.

Pivotal CF Release Notes and Known Issues

Release Notes

- [Pivotal Elastic Runtime](#)
- [Pivotal Operations Manager](#)
- [Pivotal Developer Console](#)
- [Pivotal HD](#)

Known Issues

- [Pivotal Elastic Runtime](#)
- [Pivotal Operations Manager](#)
- [Pivotal Developer Console](#)
- [Pivotal HD](#)

Pivotal Elastic Runtime v1.3.0.0 Release Notes

Changes since v1.2.2.0:

- Work on Feature Flags completed. See supporting documentation here: <http://apidocs.cloudfoundry.org/>
- Environment Variable Groups are now available and support administrators setting environment variable values across all applications.
- Application containers no longer create a listing of environment variables in log/env.log.
- Changed syslog config to send all syslog logs from the VM, the previous config only sent logs tagged with vcap. Nats logs are now sent to stderr so they end up in syslog.
- Included additional support for UDP and RELP (default is TCP) as syslog transport options.
- LDAP groups mapping to scopes is now supported. Administrators can now be derived from LDAP groups.
- Space Quotas are now supported.
- Added X-Forwarded-For to access log records.
- Support for I18N in the Pivotal CF CLI.
- Go, NodeJS, Java, PHP, Ruby, Python and PHP buildpacks are now all installed by default. The PHP buildpack has been reduced from 400+mb down to 64mb for droplet hello world and no longer includes cached dependencies in the droplet.
- Changed CC to use HM9000's bulk app state API. This should be more efficient for spaces with lots of apps.
- Added the /v2/apps/:guid/instances/:index endpoint, against which a DELETE kills the app instance at the specified index.
- This release is the first release where CF Security Groups are fully implemented. Security groups are used in this release. However, operators should review the Security Groups documentation to determine the correct application of these groups for their specific use cases. By default in this release, Security Groups are configured to allow outbound traffic from applications. CF Security Groups can be managed via the API or CF CLI
- Many missing CC API docs have now been added and are available at <http://apidocs.cloudfoundry.org/>

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 CF Developer Console v1.3.0.0 Release Notes

Changes since v1.2.0.0:

App dashboard

- Data on the app dashboard is refreshed every 20 seconds. When an update to the app (such as a start, stop, restart, or scale event) has been initiated on the app dashboard, data is refreshed every five seconds until the request is completed. The only exception is the recent logs feature, which requires a refresh to get the latest logs.

Usage service

- All identical app configurations (where the number of instances and amount of memory are the same) are rolled up into one duration of time.
- The default view is of the current calendar month, with the two previous months available as options.
- Although it cannot be displayed, data on the usage report will be saved beyond the three month window.
- If you delete an app, then push a new app with the same name, they will appear as separate apps.

Admin-only feature flags

- When the `ENABLE_NON_ADMIN_ORG_CREATION` env variable is set to `false` on the console app, only admin users will be able to create new orgs in Developer Console. This is the default behavior. When it is set to `true`, all users will be able to create new orgs in Developer Console.
- When the `ENABLE_NON_ADMIN_USER_MANAGEMENT` env variable is set to `false` on the console app, only admin users will be able to invite users and manage roles in Developer Console. This is the default behavior. When it is set to `true`, Org Managers and Space Managers will be able to invite users and manage roles in Developer Console.

Pivotal Elastic Runtime v1.3 Known Issues

- PCF Services (e.g.: MySQL for Pivotal CF, MongoDB for Pivotal CF or Pivotal HD for Pivotal CF) may lose connectivity during the upgrade of Pivotal CF Elastic Runtime due to an existing known issue with the Ruby BOSH Agent. Pivotal recommends that these services are upgraded to their 1.3 release versions along with the Elastic Runtime upgrade, as they include a new Go BOSH Agent that resolves the known issue. If you choose not to upgrade a service at this time and find that the installation has lost connectivity, run `bosh cck` against the affected VMs.

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.

Pivotal CF Developer Console v1.3.0.0 Known Issues

Issues

- On the Org Dashboard, user provided services are not shown in the services count.
- On the Org Dashboard, the quota graph is not dynamically updated.
- On the My Account page, leaving your last org will result in a 500 error.
- This space page is not dynamically updated, so current status is not reflected without a page refresh.