PRODUCT DOCUMENTATION

Pivotal GemFire® for Pivotal Cloud Foundry 1.7

Table of Contents

Pivotal GemFire for Pivotal Cloud Foundry

NOTE This product has been discontinued.

This documentation describes how to install, configure and use GemFire for Pivotal Cloud Foundry (PCF).

Product Snapshot

Current GemFire for Pivotal Cloud Foundry Details

Version: 1.7.0
Release Date: 19th September 2016
Software component version: GemFire 8.2.1, Spring Data GemFire 1.8.2, Java 8 (JDK/JRE 8u60)
Compatible Ops Manager Version(s): 1.7.x, 1.8.x
Compatible Elastic Runtime Version(s): 1.7.x, 1.8.x
Compatible Java Buildpack Version(s): 3.6, 3.7.1
vSphere support? Yes
AWS support? Yes
IPSec support? Yes

Upgrading to the Latest Version

This version of Pivotal GemFire for Pivotal Cloud Foundry supports upgrading from all the previous versions of the product. Running service instances are automatically upgraded one node at a time. If you configured your clusters with data redundancy, then the upgrade process requires no downtime and results in no data loss.

Note: Upgrades from Alpha or Beta versions of the software are not supported.

Documentation Index

The documentation contains the following topics:

For more information about Pivotal GemFire, see the Pivotal GemFire Documentation.

For more information about Pivotal Cloud Foundry, see the Pivotal Cloud Foundry Documentation.

Release Notes for GemFire for Pivotal Cloud Foundry

Overview

GemFire for Pivotal Cloud Foundry (PCF) supports the deployment of multiple Pivotal GemFire cluster configurations to PCF to support GemFire application development and deployment. GemFire for PCF enables administrators to customize three different cluster configurations to provide coarse control over the number of locators and servers available in each GemFire instance. In addition, developers can customize the GemFire instance that is bound to their app using GemFire cluster configuration commands or a new Cloud Foundry CLI interface.

Your feedback is welcome. Please provide any bugs, feature requests, or questions either to Pivotal Customer Support at https://support.pivotal.io or email pivotal-cf-feedback@pivotal.io.

v1.7.2.0

Release Date: 30th January 2017

New in this Release

- Security fixes:
  - Runs gfsh process as the `vcap` user instead of the `root` user

v1.6.6.0

Release Date: 30th January 2017

New in this Release

- Security fixes:
  - Runs gfsh process as the `vcap` user instead of the `root` user

v1.7.1.0

Release Date: 29th December 2016

New in this Release

- Stemcell AWS 3233.6: patches Ubuntu CVEs.
- Stemcell Azure 3233.7: patches Ubuntu CVEs.
- Security fixes:
  - Remove public route that exposed WAN replication credentials [CVE-2016-8220]
  - Add authentication to broker endpoints [CVE-2016-9880]

v1.6.5.0

Release Date: 29th December 2016

New in this Release

- Stemcell AWS 3233.6: patches Ubuntu CVEs.
- Stemcell Azure 3233.7: patches Ubuntu CVEs.
- Security fixes:
  - Remove public route that exposed WAN replication credentials [CVE-2016-8220]
  - Add authentication to broker endpoints [CVE-2016-9880]

v1.6.4.0

Release Date: 11th November 2016

New in this release

- Bug fixes:
  - Upgraded golang to v1.7

v1.6.3.0

Release Date: 28th October 2016

New in this release

- Stemcell AWS 3232.17: patches Ubuntu CVEs.
- Compatibility fixes for PCF v1.7 running on vSphere and Azure.

v1.7.0.0

Release Date: 19th September 2016

New in this Release

- Compatibility release for PCF v1.7 and PCF v1.8. The *GemFire for PCF* service now installs on both PCF v1.7 and PCF v1.8 without support for specific features, such as multiple availability zones.
- IPsec support: The *GemFire for PCF* tile now supports IPsec.
- Log format changed from free text to JSON
- Bug fixes:
  - DB synch issues
  - Upgraded golang to v1.7
  - Use crypto library for certificate chain discovery
  - Use of CF_DIAL_TIMEOUT
  - Service instance route are now unique across orgs and spaces
- Stemcell AWS 3232.17: patches Ubuntu CVEs
- Stemcell Azure 3232.17: patches Ubuntu CVEs

Known Issues and Limitations:

- There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x, v3.5.x. To resolve this, use the latest Java buildpack.
- Currently tile supports only a single Availability Zone (AZ).

v1.6.2

Release Date: 28th July 2016

New in this Release

- Compatibility release for PCF v1.7. The *GemFire for PCF* service now installs on PCF v1.7 without support for PCF v1.7–specific features, such as multiple availability zones.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks 3.4.x, 3.5.x. To resolve this, use the latest Java buildpack.

v1.6.1

Release Date: 22nd July 2016

New in this Release

- Stemcell AWS 3232.13: patches Ubuntu CVEs.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks 3.4.x and 3.5.x. To resolve this, update Java buildpack to 3.6. In Java buildpack 3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.6.0.0

Release Date: 31st May 2016

New in this Release

- Stemcell AWS 3232.4: patches Ubuntu CVEs.
- Stemcell Azure 3232.5: patches Ubuntu CVEs.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.5.0.0

Release Date: 9th May 2016

New in this Release

- The packaged GemFire is upgraded from v8.2.0.5 to v8.2.0.14.
- The `enable-network-partition-detection` property is now enabled to prevent against data corruption in split-brain scenarios.
- Cluster stability has been improved during restarts and during service instance deletions.
- Default timeout of `cf restart-gemfire` has been increased from 120 seconds to 900 seconds to accommodate slower-than-expected startup of servers and locators.

- When running on Azure, GemFire servers will have redundancy zones tied directly to Azure's fault domains so that redundancy for partitioned regions is satisfied across multiple virtual server racks.
- The `cf export-gemfire` command now gives feedback about missing arguments, making it easier to export your GemFire logs, cluster configs and properties.

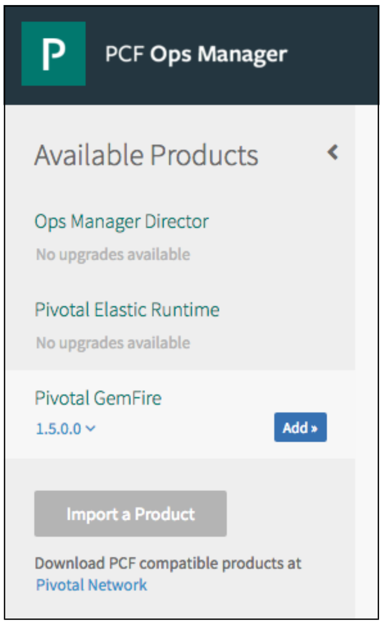
Upgrade Warning: Downtime Required

Upgrading to v1.5.0.0 will require downtime for your GemFire for PCF deployment.

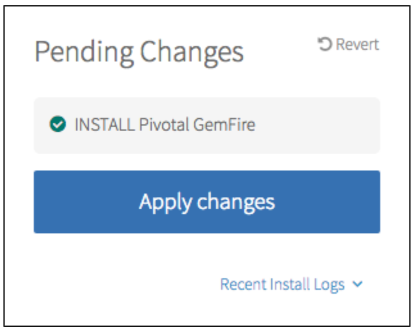
Since the `enable-network-partition-detection=true` property must be set on the entire cluster at the same time, we are unable to do a rolling upgrade. Previously deployed apps and service instances will continue to function normally after the upgrade, though GemFire for PCF will be unavailable during the upgrade.

For Ops Manager-deployed tiles:

1. Upload the v1.5.0.0 GemFire for PCF tile to Ops Manager.
2. Under *Available Products> Pivotal GemFire*, click **Add**.



3. `bosh target` your Ops Manager-deployed bosh director.
4. `bosh stop` your previously deployed GemFire for PCF tile. This will stop the GemFire for PCF service broker and all service instances.
5. In Ops Manager, click **Apply Changes**.



For bosh-deployed releases:

1. `bosh upload` the releases contained in this tile
2. `bosh stop` your existing GemFire deployment
3. `bosh deploy` the manifest contained in this tile
4. `bosh start` your new deployment

Known Issues and Limitations:

Upgrading to v1.5.0.0 will require downtime for your GemFire for PCF deployment as detailed in the previous section.

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.4.0.0

Release Date: 4th April 2016

New in this Release

- Adds support for using CLI plugin with a http proxy. To enable proxy usage, follow the same instructions for the CLI.
- Enables best practice OS tuning settings for server and locator VMs. Includes settings for:
  - `fs.file-max`
  - `vm.swappiness`
  - `net.core.somaxconn`
  - `net.core.netdev_max_backlog`
  - `net.core.rmem_max`
  - `net.core.wmem_max`
  - `net.ipv4.tcp_wmem`
  - `net.ipv4.tcp_rmem`
  - `net.ipv4.tcp_syncookies`
  - `net.ipv4.tcp_max_syn_backlog`.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.3.0.0

Release Date: 18th March 2016

New in this Release

- Add support for asynchronous service instance allocation and deallocation.
- Add basic auth for Gemfire agent endpoints.
- Improve error handling and reporting during service instance deallocation.
- Include stack traces in error logs for the service broker.
- Introduce log levels for the service broker. For now, they are only configurable at the release level and set to `info` by default. They are not exposed in the tile at this point.
- Increase the default limit for file descriptors to 32,000 for the GemFire VM.
- Ensure that the JVM is `kill`ed when it is Out of Memory.
- Upgrade stemcell to 3146.10, patching USN-2929-1.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.2.3.0

Release Date: 4th February 2016

New in this Release

- Stemcell 3146.6: patches Ubuntu CVEs.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.2.2.0

Release Date: 22nd January 2016

New in this Release

- Stemcell 3146.5: resolves CVE-2016-0728.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.2.1.0

Release Date: 12th January 2016

New in this Release

- Stemcell 3146.2: resolves CVE USN-2857-1.

Known Issues and Limitations:

There is a known issue where apps fail to stage when using the Java buildpacks v3.4.x and v3.5.x. To resolve this, update Java buildpack to v3.6. In Java buildpack v3.4, GemFire was upgraded from v8.0 to v8.2. GemFire v8.1 introduced a dependency on log4j, but log4j was not added to the Java buildpack v3.4. This dependency has been included in Java buildpack v3.6.

v1.2.0.0

Release Date: 1st December 2015

New in this Release

- Support for GemFire multi-site (WAN) replication.
- Support for the “Trusted Certificates” feature in Ops Manager. The certificates from “Trusted Certificates” are deployed to all GemFire nodes, and to the GemFire service broker.
  Note on self-signed and internal CA signed certificates: If you are using a self-signed certificate or a certificate signed by an internal or other not known certificate authority (CA), you must add the certificate (or certificate chain) to the “Trusted Certificates” in Ops Manager.
- GemFire upgraded to v8.2.0.5.

- Drain script improvements.
- Better error reporting for service instance allocation failures. Instead of returning “The maximum number of Service Instances already exist” message for all failures during a service instance allocation, each different type of failure returns a descriptive error message.
- Stemcell 3144.

Known Issues and Limitations

- Upgrading the GemFire tile on AWS can fail when the stemcell is changing as part of an upgrade. This problem usually manifests itself during the install process’ smoketest errand. The issue arises when, as part of an upgrade, a new VM has been created with the same IP but with a different MAC address. Because AWS suppresses unsolicited ARPs, other members may not know that their ARP cache has become invalid.
- The service supports only a single PCF availability zone. Although the service enables GemFire HA features such as redundancy and persistence, a failure of the targeted availability zone can result in the loss of GemFire region data.
  Note: If there is more than one available zone, you may see deployment failures or uneven sized clusters.
- Elastic scaling is not supported. You cannot change the number of cache servers and locators in a service plan without destroying and recreating the current service plan instances.
- If you deploy the service and then subsequently reduce the number of clusters, all allocated instances are destroyed.

v1.1.1.0

Release Date: 5th November 2015

New in this Release

- Support for Ops Manager v1.6 and Elastic Runtime v1.6. This release does not work with the previous releases of Ops Manager and Elastic Runtime.
- Stemcell 3100

Known Issues and Limitations

See Known Issues and Limitations for v1.1.0.0 below.

v1.1.0.0

Release Date: 22nd September 2015

New in this Release

- Upgraded Pivotal GemFire to v8.2
- Upgraded Java to v8, and JRE v1.8
- Spring-related enhancements:
  - Support for deploying a Spring XML configuration and application code to GemFire service instances.
    - If you use Spring “all the way” you can do so with GemFire service instances as well.
    - There is no need to translate the Spring configuration to GemFire XML.
    - Spring support is provided using configuration options of the GemFire for PCF CLI plug-in command, `cf restart-gemfire` .
- Support for using SSL encryption to access GemFire Service Instances from outside of PCF:
  - GemFire Pulse is now accessible via HTTPS.
  - GemFire REST clients can be configured to access a service instance’s REST API endpoint via HTTPS.
  - You can configure the GemFire shell (gfsh) to access a service instance over HTTPS.
- Additional fixes and improvements:
  - Drain script improvements
  - Improved handling of self-signed certificates
  - YAML syntax validation for cluster properties that are provided via the `--properties` option of the `cf restart-gemfire` CLI command
- Stemcell 3062

- Support for the experimental feature, HTTPS traffic to UAA

Known Issues and Limitations

- Upgrading the GemFire tile on AWS can fail when the stemcell is changing as part of the upgrade. The problem usually manifests itself during the install process' smoketest errand. The issue arises when, as part of upgrade, a new VM has been created with the same IP but a different MAC address. Because AWS suppresses unsolicited ARPs, other members may not know that their ARP cache has become invalid.
- The service supports only a single PCF availability zone. Although the service enables GemFire HA features such as redundancy and persistence, a failure of the targeted availability zone can result in the loss of GemFire region data.
  Note: If there is more than one available zone, you may see deployment failures or uneven sized clusters.
- The experimental HTTPS-only feature in Elastic Runtime v1.5 is not supported and may cause issues with this version of the product. Pivotal is working to provide full support for HTTPS-only traffic in a future release.
- Elastic scaling is not supported. You cannot change the number of cache servers and locators in a service plan without destroying and recreating the current service plan instances.
- If you deploy the service and then subsequently reduce the number of clusters, all allocated instances are destroyed.
- GemFire WAN replication is not supported.

v1.0.0.0

Release Date: 10 August 2015

Known Issues and Limitations

- The service supports only a single PCF availability zone. Although the service enables GemFire HA features such as redundancy and persistence, a failure of the targeted availability zone can result in the loss of GemFire region data.
  Note: If there is more than one available zone, you may see deployment failures or uneven sized clusters.
- The experimental HTTPS-only feature in Elastic Runtime v1.5 is not supported and may cause issues with this version of the product. Pivotal is working to provide full support for HTTPS-only traffic in a future release.
- Elastic scaling is not supported. You cannot change the number of cache servers and locators in a service plan without destroying and recreating the current service plan instances.
- If you deploy the service and then subsequently reduce the number of clusters, all allocated instances are destroyed.
- GemFire WAN replication is not supported.

Overview

GemFire for Pivotal Cloud Foundry (PCF) enables you to easily configure and provision complete Pivotal GemFire client/server clusters using Pivotal Cloud Foundry. A Cloud Foundry Administrator can easily configure three different types of GemFire clusters (service plans) using different numbers of locators and servers, as well as different resources per cluster node for each service plan. Each service plan instance uses GemFire best practices to automatically configure JVM resource utilization and garbage collection settings, taking into consideration the resources available on the cluster node. Service instances are dedicated, not multitenant, by default, but they can be used by different applications, if desired. Allocating a service instance provisions a GemFire cluster from the pool of available unallocated GemFire clusters that were instantiated during the tile installation. GemFire processes, locators and servers, run in dedicated VMs.

Pivotal provides the GemFire Pulse Web application gfsh command-line utility to help you monitor manage the service instances that you deploy. In addition, Pivotal provides a Cloud Foundry CLI plug-in that developers can use to perform the most common GemFire cluster management operations such as:

- Restarting the cluster
- Downloading cluster configuration files, log files, or statistics
- Uploading custom cluster configurations.

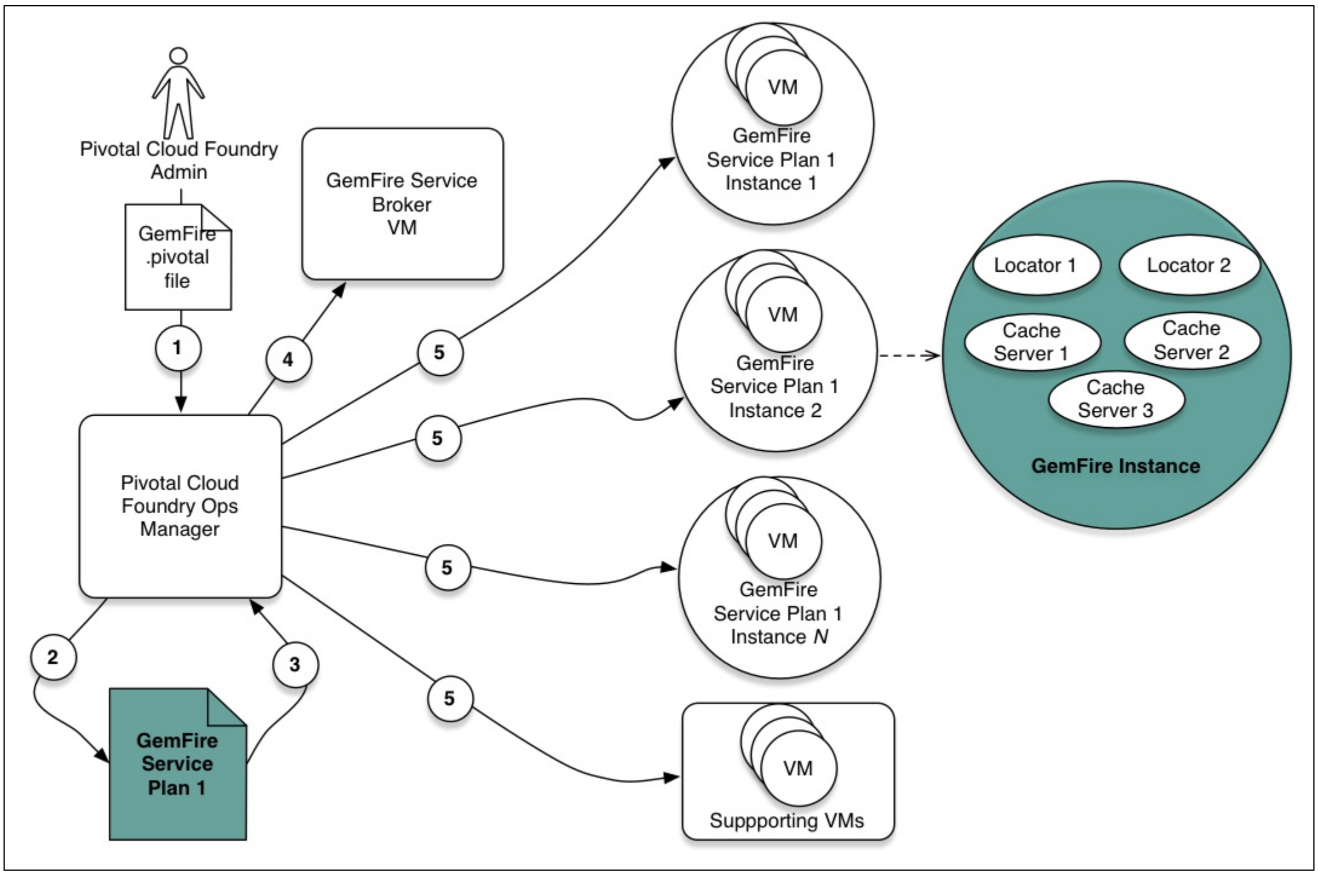
Finally, GemFire for Pivotal Cloud Foundry provides a custom rest URL to help you easily use the GemFire REST API with a bound service instance. Requests to the URL automatically round-robin to available cache servers, so there is no need to associate REST requests with a specific server.

How Does the Service Work?

PCF Administrator Workflow

The following diagram depicts the high-level workflow for deploying the GemFire service.

Figure 1. Installing and Configuring the Service


Pivotal Cloud Foundry Admin
GemFire .pivotal file
GemFire Service Broker VM
1
2
3
4
5
Pivotal Cloud Foundry Ops Manager
GemFire Service Plan 1
VM
GemFire Service Plan 1 Instance 1
GemFire Service Plan 1 Instance 2
GemFire Service Plan 1 Instance N
Suppporting VMs
Locator 1
Locator 2
Cache Server 1
Cache Server 2
Cache Server 3
GemFire Instance

1. Pivotal Cloud Foundry Administrator imports the GemFire service product file into Pivotal Cloud Foundry Operations Manager.

2. When a Pivotal Cloud Foundry Administrator configures the GemFire service for the first time, they define the number of discrete service instances for each GemFire Service Plan. A service instance is a GemFire cluster. GemFire for Pivotal Cloud Foundry enables you to configure instances for three different GemFire cluster configurations, where each instance can provide a different number of locators and cache servers, as well as memory and disk space resources.

3. After making their configuration choices, the Pivotal Cloud Foundry Administrator deploys the service, causing Pivotal Cloud Foundry Ops Manager to create and setup the appropriate set of virtual machines.

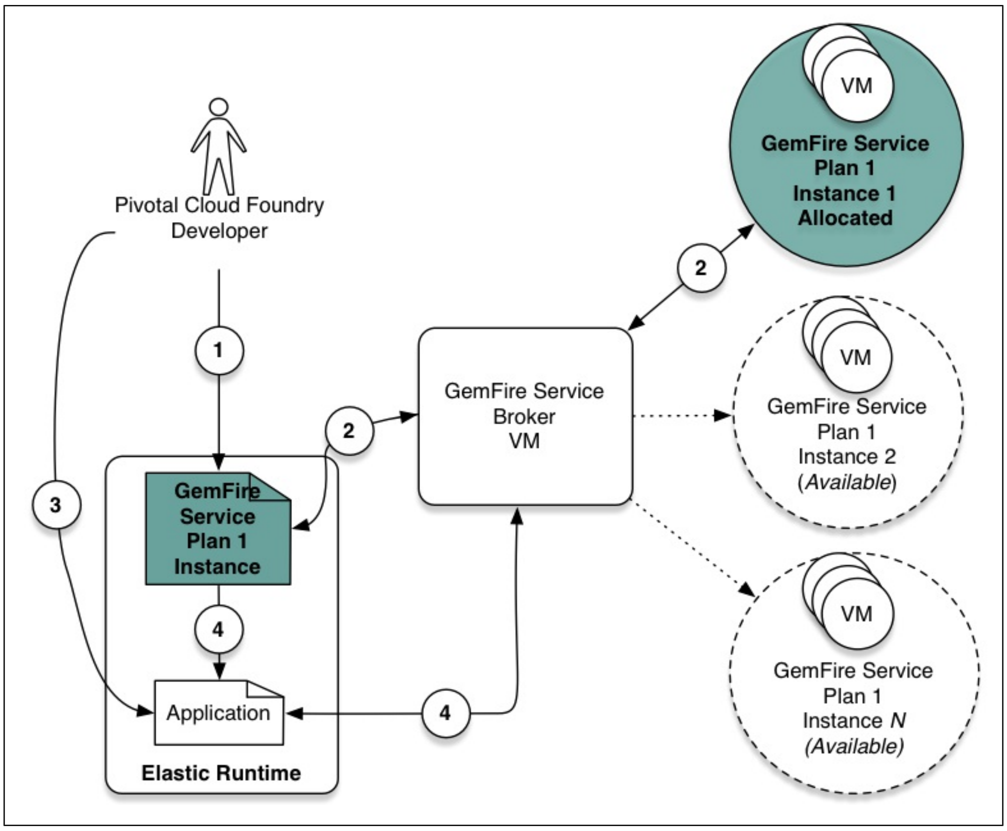
4. Operations Manager deploys a virtual machine to run the GemFire Service Broker, which is responsible for allocating GemFire Cluster service instances to Pivotal Cloud Foundry users and for passing relevant information into the applications that are bound to each service instance.

5. Operations Manager also deploys the specified number of GemFire service instances (clusters) as configured by the administrator. Lastly, there are a few temporary VMs used for install and uninstall errands.

After the service has been deployed in Operations Manager, it is ready for use by application developers in the Elastic Runtime environment. The following diagram depicts the high-level workflow that an application developer would use to consume the service.

PCF Developer (Service User) Workflow

Figure 2. Using the Service


Pivotal Cloud Foundry Developer
1
2
3
4
GemFire Service Plan 1 Instance
Application
Elastic Runtime
GemFire Service Broker VM
VM
GemFire Service Plan 1 Instance 1 Allocated
GemFire Service Plan 1 Instance 2 (*Available*)
GemFire Service Plan 1 Instance *N* (*Available*)

1. Pivotal Cloud Foundry Developer uses the CLI or Developer Console to create a GemFire service instance in the Elastic Runtime space where they are pushing the relevant applications.

2. Creating a service instance causes Elastic Runtime to contact the GemFire Service Broker, which allocates one un-allocated GemFire cluster of dedicated virtual machines to the GemFire service instance. In the response to Elastic Runtime, the Service Broker includes the URL to reach the GemFire Pulse monitoring tool for the service instance. The Elastic Runtime exposes this URL in the Developer Console's metadata for the service instance.

3. Pivotal Cloud Foundry Developer pushes their application to Elastic Runtime.

4. Pivotal Cloud Foundry Developer binds their application to the GemFire service instance. The GemFire Service Broker populates the application's VCAP_SERVICES environment variable with the metadata required to access the GemFire service instance's locators and cache servers.

5. Pivotal Cloud Foundry developer can configure, monitor, and manage the cluster using Pulse and/or the GemFire for Pivotal Cloud Foundry CLI plug-in.

Additional Resources

- Pivotal GemFire Product Documentation
- Pivotal GemFire Community Forum
- Pivotal GemFire KnowledgeBase
- Pivotal Cloud Foundry Documentation

Installing GemFire for Pivotal Cloud Foundry

Prerequisites

Before you begin your GemFire for Pivotal Cloud Foundry (PCF) deployment, your system needs to meet the following minimum requirements:

- **Pivotal Cloud Foundry Ops Manager for vSphere**, or
  Pivotal Cloud Foundry Ops Manager for vCloud Air or vCloud Director, or
  Pivotal Cloud Foundry Ops Manager for AWS
- **IPSec (optional):** If you wish to use IPSec in your Pivotal GemFire installation, please ensure you have the IPSec BOSH release deployed **before** installing the Elastic Runtime and GemFire tile. For instructions on deploying the IPSec BOSH release, see the instructions here.
- **Pivotal Cloud Foundry Elastic Runtime**
- Network access and credentials for the **Pivotal Cloud Foundry Ops Manager Web Console**
- **Capacity in the vSphere cluster for service instances you want to deploy.** By default, the GemFire service configures three different GemFire service plans. Each plan deploys the minimum of 2 locators, but deploy 3, 5, and 7 GemFire cache servers, respectively. Several supporting VMs are also required for service deployment. The resource allocations for the default GemFire service plan instances and components are as follows:

Table 1. Default Resource Requirements for a GemFire Service Instances

Service	Virtual Machines	CPU *	RAM (MB) *	Ephemeral Disk (MB) *	Persistent Disk (MB) *
GemFire locator (Plan 1)	2	1	1024	2048	1024
GemFire locator (Plan 2)	2	1	1024	2048	1024
GemFire locator (Plan 3)	2	1	1024	2048	1024
GemFire Server (Plan 1)	3	2	4096	6144	8192
GemFire Server (Plan 2)	3	2	4096	6144	8192
GemFire Server (Plan 3)	3	2	4096	6144	8192
GemFire Broker	1	2	4096	4096	4096
Broker Registrar	1	1	2048	2048	0
Broker Deregistrar	1	1	2048	2048	0
Cluster Smoke Test	1	1	512	2048	0
Cluster Agent Smoke Test	1	1	512	2048	0
Service Offering Smoke Test	1	1	512	2048	0
Compilation	2	1	1024	4096	0
Totals	**12**	**9**	**16384**	**24576**	**12800**

*Required for each VM

Service Configuration Defaults

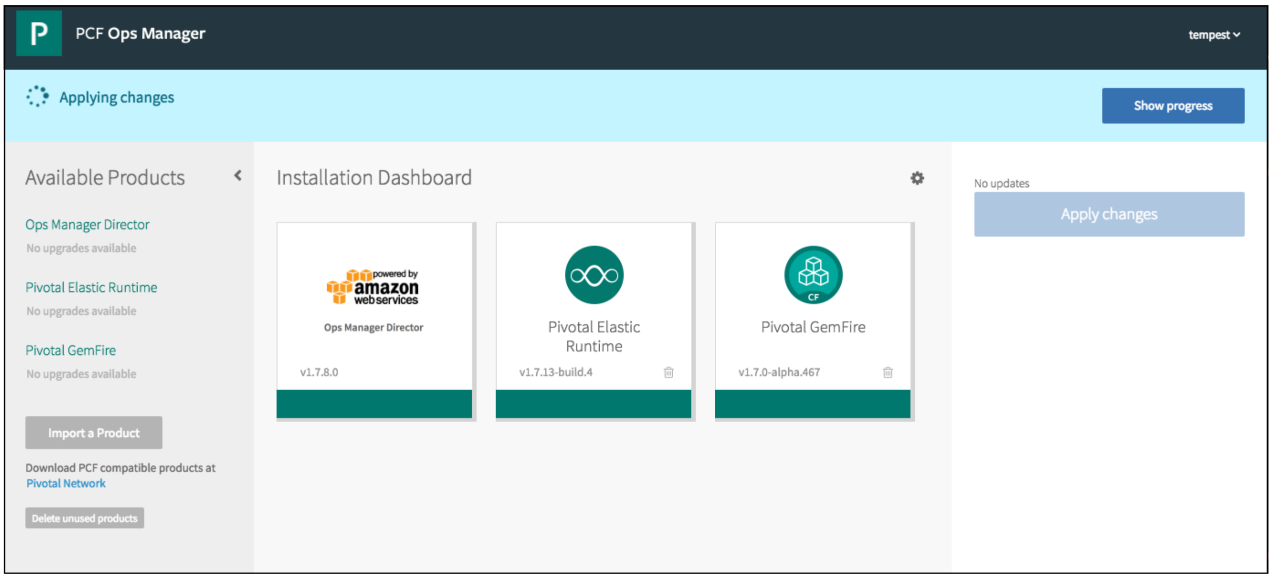
The GemFire for Pivotal Cloud Foundry service automatically implements the best practices an operator would normally employ when deploying GemFire. These configurations include:

- **Dynamic memory management**. Dynamically set the JVM maximum and minimum memory utilization for each instance of the GemFire software process based on the VM's total memory.
- **Garbage collection**. Enable and configure memory garbage collection for each instance of the GemFire software process.
- **Logging configuration**. Create and store logs at the config log level with appropriate limitations on sizing and automatic log rotation.
- **Statistics monitoring.** Create and store statistics files at a sample rate of 1,000, which corresponds to a sampling rate of once per second.
- **Data overflow configuration**. Overflow data to disk when crossing 70% memory utilization.
- **Memory overutilization protection**. Prevent further writes to memory when crossing 80% memory utilization.
- **Client authentication**. Require clients to authenticate when directly accessing cache servers.

The default configuration settings can be adjusted, if needed, on a per service instance basis.

Installation Steps

1. Download the *GemFire for Pivotal Cloud Foundry*tile from Pivotal Network.
2. Use a Web browser to log in to the **Pivotal Ops Manager** application. The **Pivotal Cloud Foundry Ops Manager Installation Dashboard** displays.
3. Click **Import a Product**.
   The **Add Products** screen displays.
4. Click **Choose File** and navigate to the file you downloaded.
   The file uploads to your Pivotal Cloud Foundry deployment.
5. Click **Add**.
   Pivotal Ops Manager adds a new tile for **GemFire for Pivotal Cloud Foundry** to the Installation Dashboard.



Self-Signed and Internal SSL Certificates

In production environments, we recommend that you use an SSL certificate signed by a reputable certificate authority (CA). Please ensure your SSL certificate has all the Subject Alternative Names (SANs):

- `*.system.example.com`
- `*.apps.example.com`
- `*.uaa.system.example.com`
- `*.login.system.example.com`

If your certificate is signed by a known authority and has the correct SANs, please disregard this section.

Internal Certificates

In the case of Internal Certificates (i.e. certificates signed by an internal Root CA), you need to configure the Ops Manager Director tile to trust the internal Root CA. To achieve this, please follow the steps below:

1. Add the internal Root CA is to the **Security Config** section under the Ops Manager Director Tile.

2. Copy the internal certificate and Root CA (certificate chain) to the Elastic Runtime tile by navigating to **Pivotal Elastic Runtime > Networking > Load Balancer with TLS enabled (or HAProxy)**

Self-Signed Certificates

When using self signed certificates generated by the Elastic Runtime Tile, you need to ensure that this certificate is provided to the Ops Manager Director Tile.

Generating Self-Sign Certificates using the Elastic Runtime Tile

The Elastic Runtime Tile allows you to generate self-signed certificates by navigating to **Pivotal Elastic Runtime > Networking > Generate RSA Certificate**. Please specify the following wildcard subdomains:

- `*.system.example.com`
- `*.apps.example.com`
- `*.uaa.system.example.com`
- `*.login.system.example.com`

Once you have generated your certificate, copy the certificate to the Ops Manager Director Tile and place it in the **Security Config** section.

Generating Self-SignedCertificates using OpenSSL

1. To use the OpenSSL CLI, run the following commands:

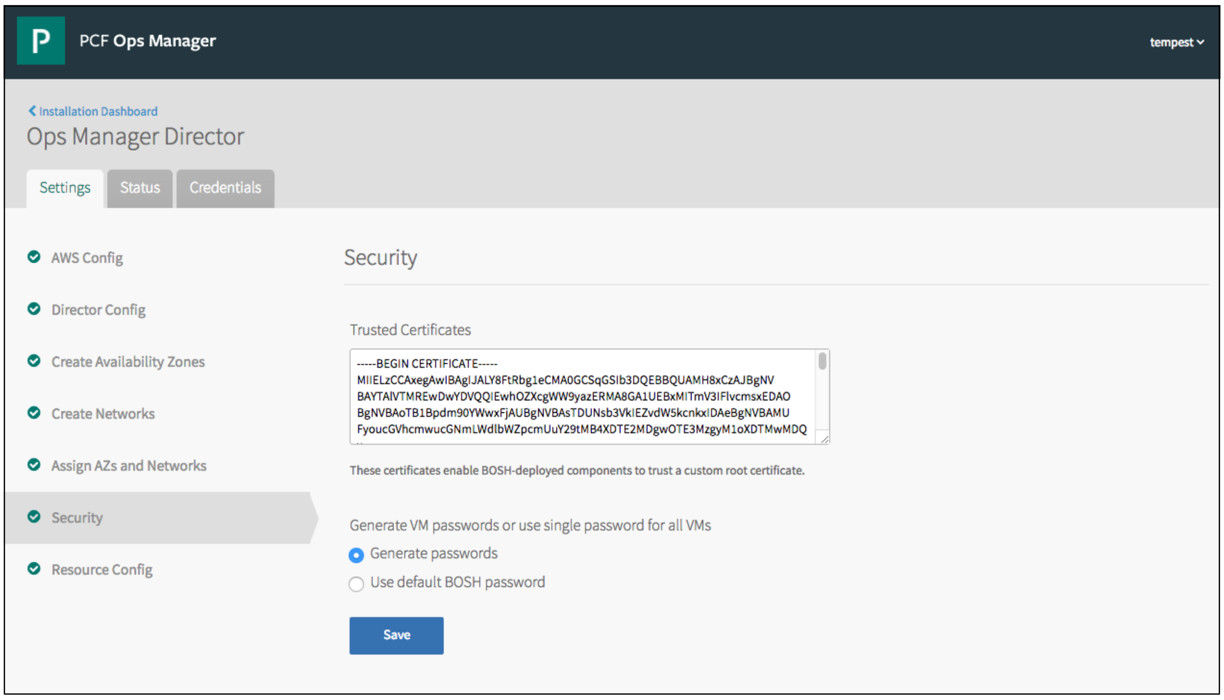
   a. Generate a private key: `openssl genrsa -des3 -out server.key 1024`
   b. Generate a Certificate Signing Request (CSR) with the `subjectAltName` entries, and edit the `subj` flag as needed:
   ```
   openssl req -new \ -key server.key \ -out server.csr \ -subj '/C=US/ST=State/L=Locality/O=Organization/OU=Organization
   Unit/CN=www.example.com/emailAddress=example@example.com/subjectAltName=DNS.1=*.system.example.com,DNS.2=*.apps.example.
   com,DNS.3=*.uaa.system.example.com,DNS.4=*.login.system.example.com'
   ```
   c. Remove Passphrase from Key: `cp server.key server.key.org openssl rsa -in server.key.org -out server.key`
   d. Generate a Self-Signed Certificate: `openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt`

2. Upload the SSL certificate to your load balancer. Instructions for AWS:

   a. Authenticate with the AWS CLI:
   ```
   $ aws configure AWS Access Key ID [None]: <ACCESS KEY GOES HERE> AWS Secret Access Key [None]: <SECRET ACCESS KEY GOES
   HERE> Default region name [None]: us-east-1 Default output format [None]: ENTER
   ```
   b. Upload your self-signed certificate in IAM:
   ```
   aws iam upload-server-certificate \ --server-certificate-name <NAME OF CERTIFICATE> \ --certificate-body
   file://server.crt \ --private-key file://server.key
   ```
   c. Upload your self-signed certificate to your Elastic Load Balancer (ELB):
   ```
   aws elb set-load-balancer-listener-ssl-certificate \ --load-balancer-name <ELB NAME> \ --load-balancer-port <PORT> \ --
   ssl-certificate-id arn:aws:iam::<IAM NUMBER>:server-certificate/<NAME OF CERTIFICATE>
   ```

3. Add the SSL certificate as a trusted certificate to your Ops Manager-deployed VMs.

   a. In Ops Manager, on the Director Tile, under the **Security** section, paste the certificate, and save.



Ops Manager Director

Security

Trusted Certificates

```
-----BEGIN CERTIFICATE-----
MIIELzCCAxegAwIBAgIJALY8FtRbg1eCMA0GCSqGSIb3DQEBBQUAMH8xCzAJBgNV
BAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazERMA8GA1UEBxMITmV3IFlvcmsxEDAO
BgNVBAoTB1Bpdm90YWwxFjAUBgNVBAsTDUNsb3VkIEZvdW5kcnkxIDAeBgNVBAMU
FyoucGVhcmwucGNmLWdlbWZpcmUuY29tMB4XDTE2MDgwOTE3MzgyM1oXDTMwMDQ
```

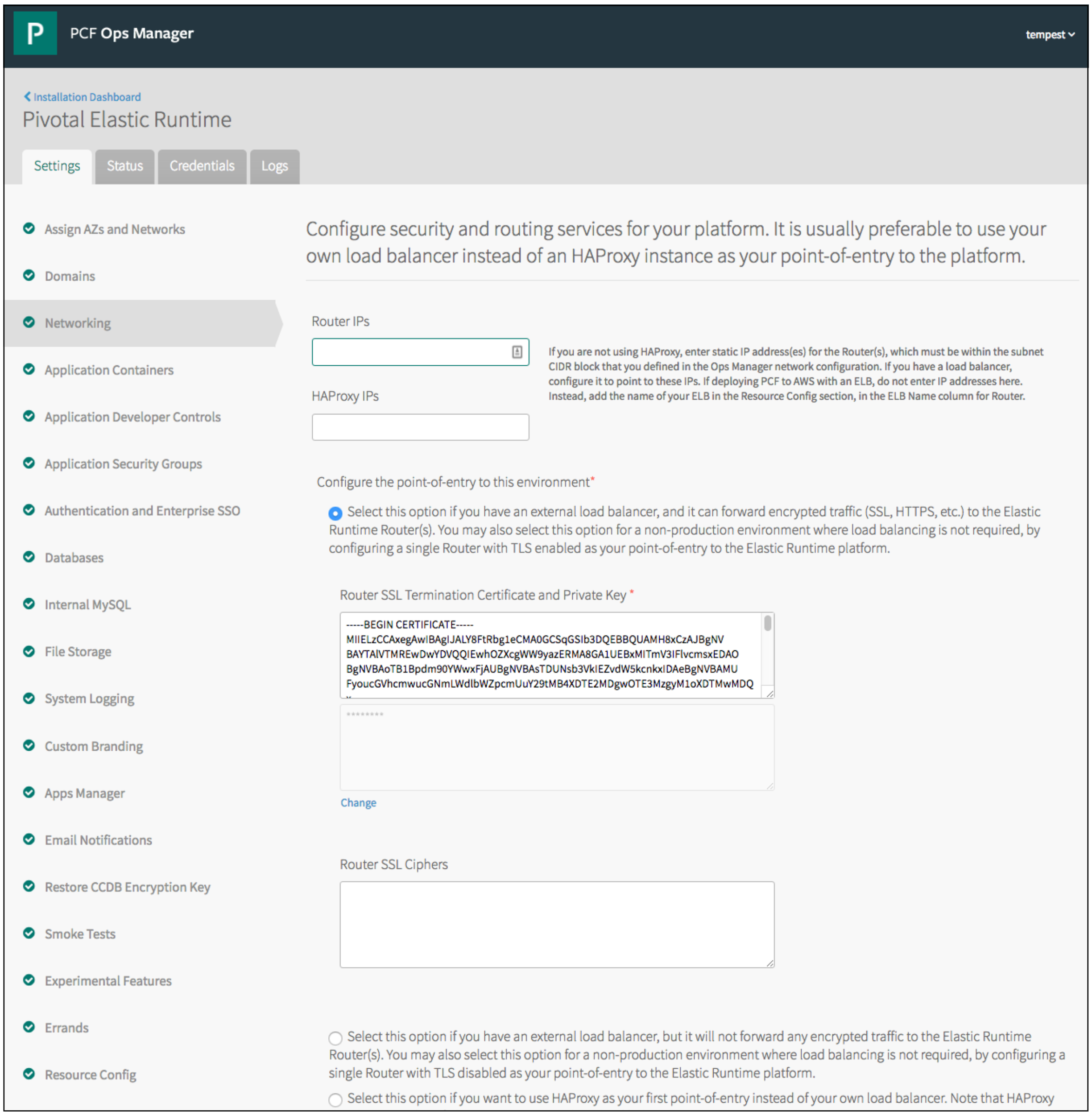
These certificates enable BOSH-deployed components to trust a custom root certificate.

Generate VM passwords or use single password for all VMs

[✓ Generate passwords]

Use default BOSH password

b. On the Elastic Runtime Tile, under the **Networking** section, paste both the certificate and the private key, be sure to check **Disable SSL certificate verification for this environment**, and save.



PCF Ops Manager

tempest

Installation Dashboard

Pivotal Elastic Runtime

Settings | Status | Credentials | Logs

- ✓ Assign AZs and Networks
- ✓ Domains
- ✓ Networking
- ✓ Application Containers
- ✓ Application Developer Controls
- ✓ Application Security Groups
- ✓ Authentication and Enterprise SSO
- ✓ Databases
- ✓ Internal MySQL
- ✓ File Storage
- ✓ System Logging
- ✓ Custom Branding
- ✓ Apps Manager
- ✓ Email Notifications
- ✓ Restore CCDB Encryption Key
- ✓ Smoke Tests
- ✓ Experimental Features
- ✓ Errands
- ✓ Resource Config

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

Router IPs

HAProxy IPs

If you are not using HAProxy, enter static IP address(es) for the Router(s), which must be within the subnet CIDR block that you defined in the Ops Manager network configuration. If you have a load balancer, configure it to point to these IPs. If deploying PCF to AWS with an ELB, do not enter IP addresses here. Instead, add the name of your ELB in the Resource Config section, in the ELB Name column for Router.

Configure the point-of-entry to this environment*

● Select this option if you have an external load balancer, and it can forward encrypted traffic (SSL, HTTPS, etc.) to the Elastic Runtime Router(s). You may also select this option for a non-production environment where load balancing is not required, by configuring a single Router with TLS enabled as your point-of-entry to the Elastic Runtime platform.

Router SSL Termination Certificate and Private Key *

```
-----BEGIN CERTIFICATE-----
MIIELzCCAxegAwIBAgIJALY8FtRbg1eCMA0GCSqGSIb3DQEBBQUAMH8xCzAJBgNV
BAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazERMA8GA1UEBxMITmV3IFlvcmsxEDAO
BgNVBAoTB1Bpdm90YWwxFjAUBgNVBAsTDUNsb3VkIEZvdW5kcnkxIDAeBgNVBAMU
FyoucGVhcmwucGNmLWdlbWZpcmUuY29tMB4XDTE2MDgwOTE3MzgyM1oXDTMwMDQ
```

****

Change

Router SSL Ciphers

○ Select this option if you have an external load balancer, but it will not forward any encrypted traffic to the Elastic Runtime Router(s). You may also select this option for a non-production environment where load balancing is not required, by configuring a single Router with TLS disabled as your point-of-entry to the Elastic Runtime platform.

○ Select this option if you want to use HAProxy as your first point-of-entry instead of your own load balancer. Note that HAProxy

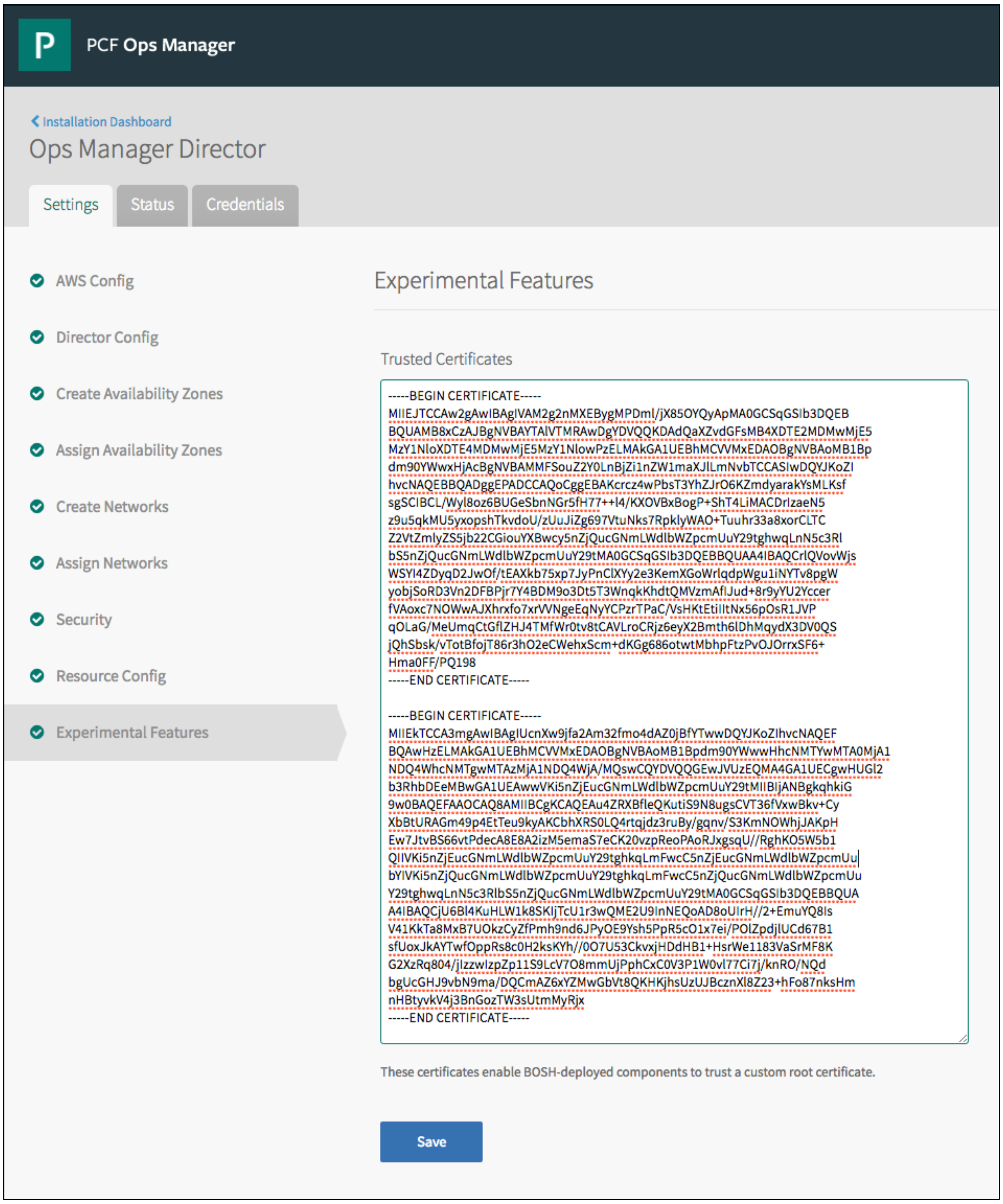
WAN Replication and Self-Signed Certificates

A WAN topology in GemFire is a client-server relationship. When using self-signed certificates, each client needs to trust the server's SSL certificate.

When using WAN replication with self-signed certificates, we strongly recommend using certificates that were generated with the OpenSSL CLI. This is because Ops Manager self-signed certificates are actually signed by Ops Manager's internal CA certificate. If you do use a certificate generated by Ops Manager, you'll need to trust the Ops Manager's internal CA certificate instead.

For example, when setting up WAN replication across two PCF installations who each use their own self signed certificates, named *West* and *East*:

1. In *West* Ops Manager, on the Director Tile, under **Experimental Features > Trusted Certificates**, add the certificate for *East* (appending if necessary), and save.



```
-----BEGIN CERTIFICATE-----
MIIEJTCCAw2gAwIBAgIVAM2g2nMXEBygMPDml/jX85OYQyApMA0GCSqGSIb3DQEB
BQUAMB8xCzAJBgNVBAYTAlVTMRAwDgYDVQQKDAdQaXZvdGFsMB4XDTE2MDMwMjE5
MzY1NloXDTE4MDMwMjE5MzY1NlowPzELMAkGA1UEBhMCVVMxEDAOBgNVBAoMB1Bp
dm90YWwxHjAcBgNVBAMMFSouZ2Y0LnBjZi1nZW1maXJlLmNvbTCCASIwDQYJKoZI
hvcNAQEBBQADggEPADCCAQoCggEBAKcrcz4wPbsT3YhZJrO6KZmdyarakYsMLKsf
sgSCIBCL/Wyl8oz6BUGeSbnNGr5fH77++l4/KXOVBxBogP+ShT4LiMACDrIzaeN5
z9u5qkMU5yxopshTkvdoU/zUuJiZg697VtuNks7RpklyWAO+Tuuhr33a8xorCLTC
Z2VtZmlyZS5jb22CGiouYXBwcy5nZjQucGNmLWdlbWZpcmUuY29tghwqLnN5c3Rl
bS5nZjQucGNmLWdlbWZpcmUuY29tMA0GCSqGSIb3DQEBBQUAA4IBAQCrlQVovWjs
WSYI4ZDyqD2JwOf/tEAXkb75xp7JyPnClXYy2e3KemXGoWrlqdpWgu1iNYTv8pgW
yobjSoRD3Vn2DFBPjr7Y4BDM9o3Dt5T3WnqkKhdtQMVzmAfIJud+8r9yYU2Yccer
fVAoxc7NOWwAJXhrxfo7xrVVNgeEqNyYCPzrTPaC/VsHKtEtiIltNx56pOsR1JVP
qOLaG/MeUmqCtGflZHJ4TMfWr0tv8tCAVLroCRjz6eyX2Bmth6lDhMqydX3DV0QS
jQhSbsk/vTotBfojT86r3hO2eCWehxScm+dKGg686otwtMbhpFtzPvOJOrrxSF6+
Hma0FF/PQ198
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIEkTCCA3mgAwIBAgIUcnXw9jfa2Am32fmo4dAZ0jBfYTwwDQYJKoZIhvcNAQEF
BQAwHzELMAkGA1UEBhMCVVMxEDAOBgNVBAoMB1Bpdm90YWwwHhcNMTYwMTA0MjA1
NDQ4WhcNMTgwMTAzMjA1NDQ4WjA/MQswCQYDVQQGEwJVUzEQMA4GA1UECgwHUGl2
b3RhbDEeMBwGA1UEAwwVKi5nZjEucGNmLWdlbWZpcmUuY29tMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu4ZRXBfleQKutiS9N8ugsCVT36fVxwBkv+Cy
XbBtURAGm49p4EtTeu9kyAKCbhXRS0LQ4rtqjdz3ruBy/gqnv/S3KmNOWhjJAKpH
Ew7JtvBS66vtPdecA8E8A2izM5emaS7eCK20vzpReoPAoRJxgsqU//RghKO5W5b1
QIIVKi5nZjEucGNmLWdlbWZpcmUuY29tghkqLmFwcC5nZjEucGNmLWdlbWZpcmUu
bYIVKi5nZjQucGNmLWdlbWZpcmUuY29tghkqLmFwcC5nZjQucGNmLWdlbWZpcmUu
Y29tghwqLnN5c3RlbS5nZjQucGNmLWdlbWZpcmUuY29tMA0GCSqGSIb3DQEBBQUA
A4IBAQCjU6Bl4KuHLW1k8SKIjTcU1r3wQME2U9InNEQoAD8oUIrH//2+EmuYQ8Is
V41KkTa8MxB7UOkzCyZfPmh9nd6JPyOE9Ysh5PpR5cO1x7ei/POlZpdjlUCd67B1
sfUoxJkAYTwfOppRs8c0H2ksKYh//0O7U53CkvxjHDdHB1+HsrWe1183VaSrMF8K
G2XzRq804/jIzzwIzpZp11S9LcV7O8mmUjPphCxC0V3P1W0vl77Ci7j/knRO/NQd
bgUcGHJ9vbN9ma/DQCmAZ6xYZMwGbVt8QKHKjhsUzUJBcznXl8Z23+hFo87nksHm
nHBtyvkV4j3BnGozTW3sUtmMyRjx
-----END CERTIFICATE-----
```

These certificates enable BOSH-deployed components to trust a custom root certificate.

2. In *East* Ops Manager, on the Director Tile, under **Experimental Features > Trusted Certificates**, add the certificate for *West* (appending if necessary), and save.

3. In both installations, you need to redeploy the GemFire for Pivotal Cloud Foundry tile for these changes to take effect. The simplest way to force a redeploy is to make a small change to any of the configuration properties, such as the Service Plan Configuration.

Then proceed to configure WAN replication across your multiple sites.

Creating GemFire Service Plans

Configure GemFire service instances to set the maximum capacity for each cluster provided in your Cloud Foundry environment.

Note: If you have already configured the service, reducing the number of resources and reconfiguring the service causes all allocated instances to be destroyed. See the Release Notes for more information about known problems and limitations.

Follow these steps to configure the GemFire service:

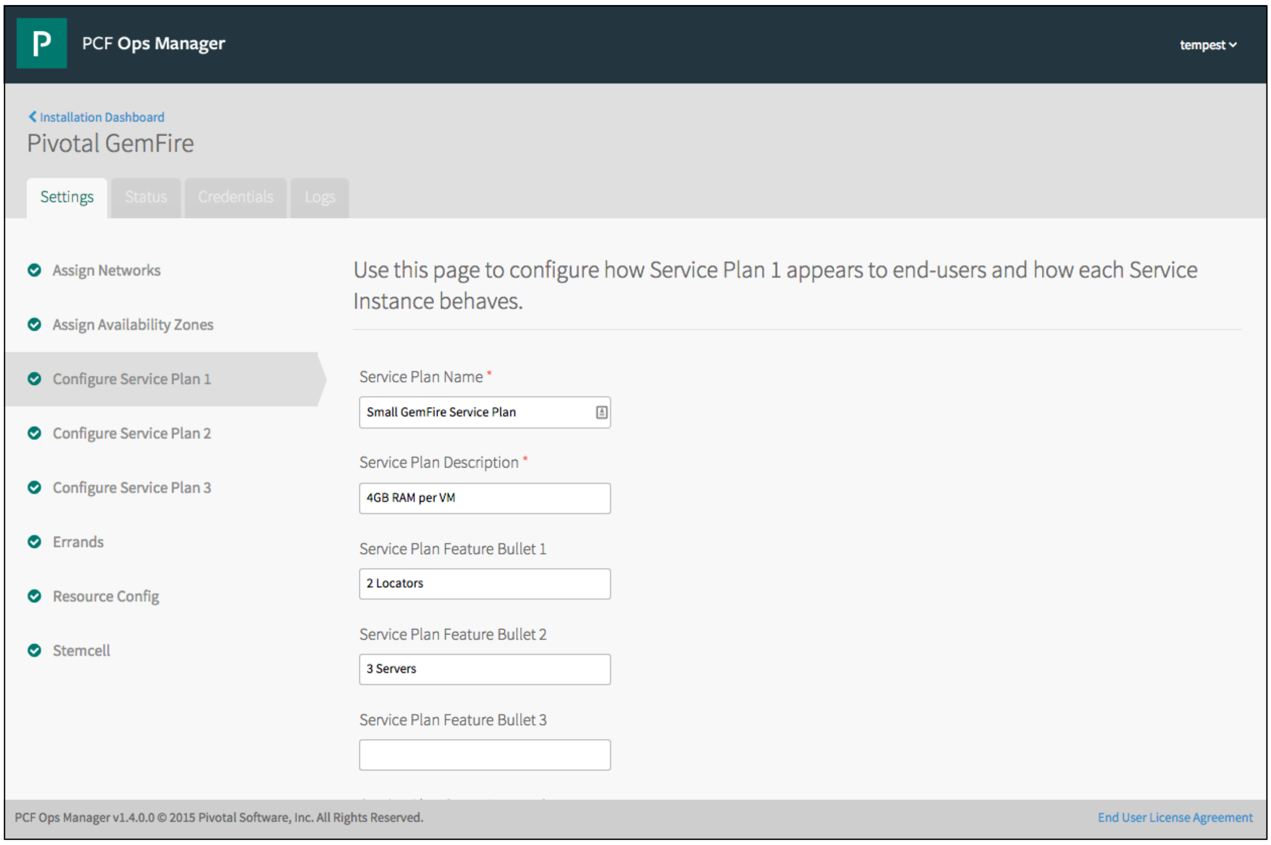
1. Select the **GemFire for Pivotal Cloud Foundry** tile to display the configuration page.

2. (Optional) Select the **Assign Networks** tab to specify the vSphere Network where Ops Manager deploys the GemFire Service Broker and related virtual machines. See Configuring Network Isolation Options in Ops Manager in the Pivotal Cloud Foundry documentation for more information.

3. Select **Assign Availability Zones** to specify the Availability Zone into which Ops Manager should deploy the GemFire Service Broker and virtual machines. Note that the current version of the GemFire service must be deployed to a single Availability Zone. See Configuring Ops Manager Director for VMware vSphere for more information.
Note: The current version of the service only supports deployment to one Availability Zone, even if multiple Availability Zones are available. Deploying the service to more than one AZ results in an invalid installation.

4. Select the **Configure Service Plan 1-3** tabs to configure the each of the three GemFire plans that will be available in the Elastic Runtime CLI and Web Console. GemFire for Pivotal Cloud Foundry provides three different service plans that you can configure to provide different GemFire cluster sizes. For each of the three service plan configurations:

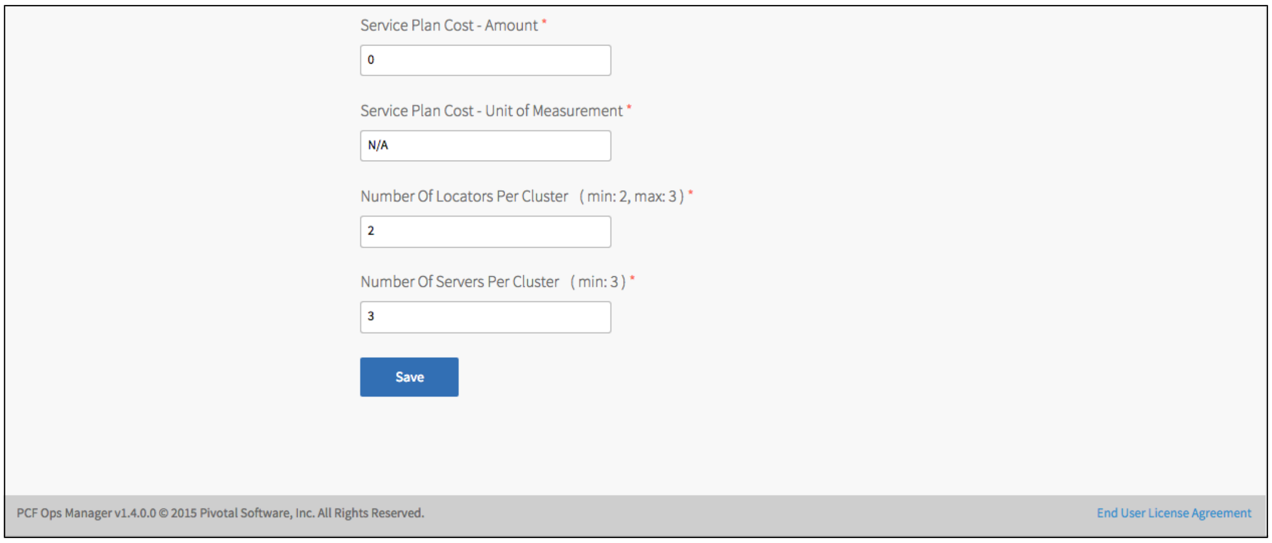
    a. (Optional) As a best practice, use the **Service Plan Name**, **Service Plan Description**, and **Service Plan Feature Bullet 1-3** fields to describe the size of each cluster. This information will appear in the Marketplace when developers are choosing from the available plans. For example:



Use this page to configure how Service Plan 1 appears to end-users and how each Service Instance behaves.

Service Plan Name *: Small GemFire Service Plan
Service Plan Description *: 4GB RAM per VM
Service Plan Feature Bullet 1: 2 Locators
Service Plan Feature Bullet 2: 3 Servers
Service Plan Feature Bullet 3:

2. Specify the number of locators *per cluster* to create in the GemFire cluster for this plan, as well as the number of cache servers *per cluster*. For example:



Service Plan Cost - Amount *

0

Service Plan Cost - Unit of Measurement *

N/A

Number Of Locators Per Cluster (min: 2, max: 3) *

2

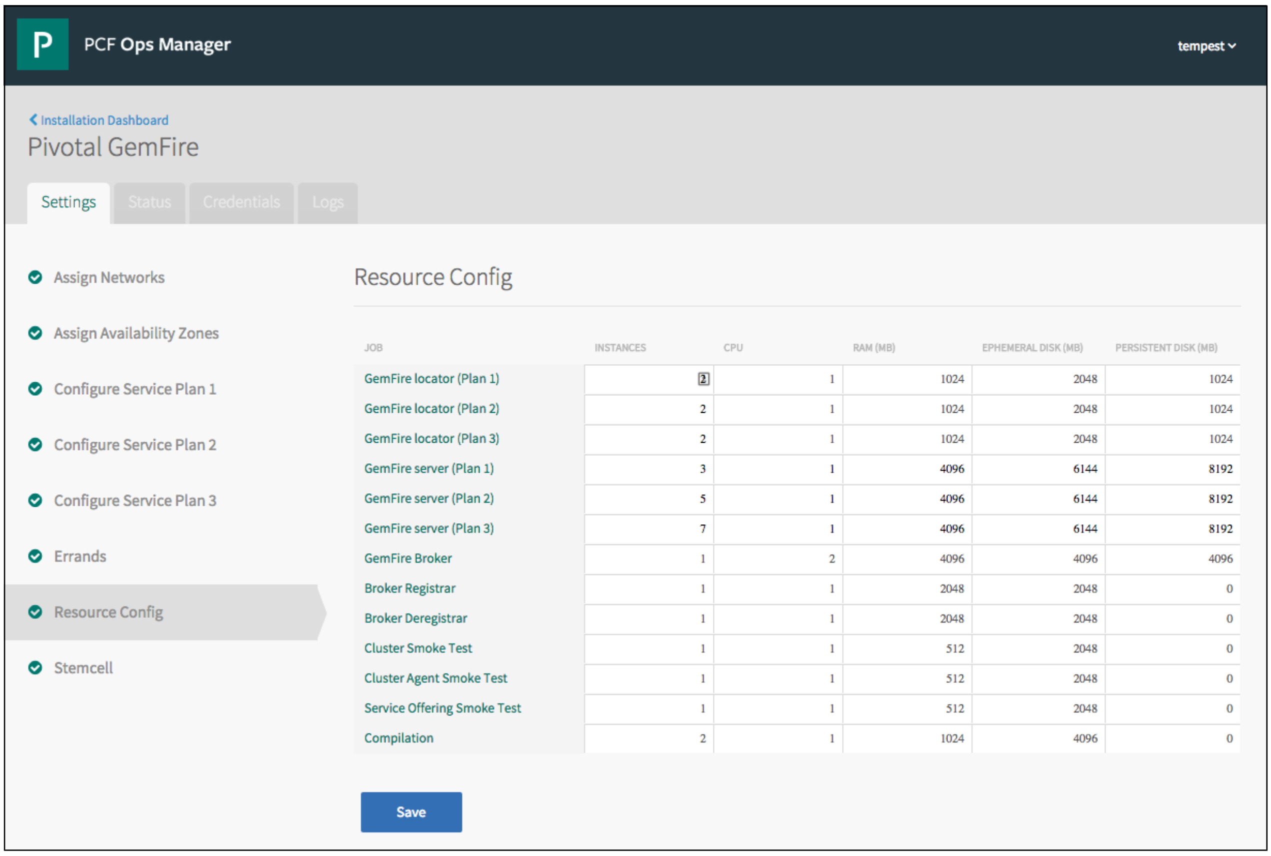
Number Of Servers Per Cluster (min: 3) *

3

Save
PCF Ops Manager v1.4.0.0 © 2015 Pivotal Software, Inc. All Rights Reserved.
End User License Agreement

Note: Each service plan must specify a *minimum* of two locators and three cache servers per GemFire cluster. Record the number of locators and servers used in each plan to help you configure the total plan resources in the next step. If you would like to defer a plan configuration or allocation until a later time, you can do that in the Resource Config tab (see below). 3. Click **Save** to save the current service plan configuration. 4. Repeat the above steps to configure each of the three GemFire service plans.

5. (Optional) Select the **Resource Config** tab to change the allocation of resources for of GemFire members in each service plan:



Resource Config

JOB	INSTANCES	CPU	RAM (MB)	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)
GemFire locator (Plan 1)	2	1	1024	2048	1024
GemFire locator (Plan 2)	2	1	1024	2048	1024
GemFire locator (Plan 3)	2	1	1024	2048	1024
GemFire server (Plan 1)	3	1	4096	6144	8192
GemFire server (Plan 2)	5	1	4096	6144	8192
GemFire server (Plan 3)	7	1	4096	6144	8192
GemFire Broker	1	2	4096	4096	4096
Broker Registrar	1	1	2048	2048	0
Broker Deregistrar	1	1	2048	2048	0
Cluster Smoke Test	1	1	512	2048	0
Cluster Agent Smoke Test	1	1	512	2048	0
Service Offering Smoke Test	1	1	512	2048	0
Compilation	2	1	1024	4096	0

a. Enter the total number of GemFire locator and server **Instances** that you want to create for each of the available service plans.
Note: The number of locator or server instances that you specify for a plan must be a multiple of the *per cluster* number locators and servers that you configured in the previous step. The only exception to this rule is if you want to defer allocating resources for the plan to a later time, in which case you can set the number of locators and servers to zero.
For example, if you configured GemFire Service Plan 1 to provide a small GemFire cluster using the minimum of 2 locators and 3 servers, but you want to wanted to make 3 instances of this plan available in the marketplace, then you would set **GemFire locator (Plan 1)** to 6 instances, and **GemFire server (Plan 1)** to 9 instances.
If you instead wanted to postpone configuring the plan and “save” your pending changes to the plan, then set the number of locators and servers to zero. You can then revisit the plan configuration and resource allocation at a later time. Be sure to set valid values for the cluster size and resources under **Resource Config** before you apply your final changes to the plan.
b. (Optional) For GemFire service plan cache servers only, enter the amount of **CPU**, **RAM (MB)**, **EPHEMERAL DISK (MB)**, and **PERSISTENT DISK**

(MB) resources to allocate for each server VM, based on your capacity requirements. For production deployments, Pivotal recommends increasing the number of CPUs to at least 2.
Approximately 1 GB of configured RAM is reserved for the VM operating system; all RAM above this amount is allocated to the JVM that runs the GemFire server.
Note: The resources allocations for GemFire locators and other service components are automatically configured to their recommended settings, and they cannot be changed.

6. Click **Save**.

7. Click the **Installation Dashboard** link.
   The **Installation Dashboard** screen displays.

8. Click **Apply Changes**.

   Pivotal Cloud Foundry Ops Manager deploys a single virtual machine to run the GemFire Service Broker, preallocates all additional VMs required for the three GemFire Service plan clusters. You can access information about the deployments from the Pivotal Cloud Foundry Ops Manager console.

   After the GemFire cluster service instances are deployed, the GemFire Service Broker automatically registers the service and its service plans in the Elastic Runtime Marketplace. Pivotal Cloud Foundry users can now create and bind to instances of the configured service plans. See Using the Pivotal GemFire Service on Pivotal Cloud Foundry.

Deferring Service Plan Configuration

As mentioned in the procedure above, it is possible to leave a service plan unallocated, and defer its configuration and resource allocation to a later time. To do this, set the number of locators and servers for the plan to zero in the **Resource Config** tab.

When you are ready to finalize the configuration, follow the same steps in Creating GemFire Service Plans to set any required properties, and allocate resources for the plan in the **Resource Config** tab. Finally, click **Apply changes** in the Ops Manager to create the service instances that you allocated.

Application Security Groups

To enable access to the GemFire for PCF tile service, you need to ensure your security group allows access to the Gemfire locator, and server VMs configured in your deployment. The IP addresses for these can be obtained from OpsManager by clicking on **GemFire Tile** and navigating to **Status** tab. All the IPS are mentioned under the column **IPS**. You should ensure the following TCP ports are enabled for the IP addresses: 7071, 40404, 55221. More details can be found at Application Security Groups.

```
{"protocol":"tcp","destination":"<YOUR-GEMFIRE-LOCATORS-AND-SERVERS-IP-RANGE>","ports":"7071, 40404, 55221"}
```

Warning

We recommend updating your PCF installation's default application security group as the default application security group in Pivotal Cloud Foundry allows **all** egress traffic.

Default application security group:

```
[
  {
      "destination": "0.0.0.0-169.254.169.253",
      "protocol": "all"
  },
  {
      "destination": "169.254.169.255-255.255.255.255",
      "protocol": "all"
  }
]
```

Additional application security group rules may be necessary to to ensure other PCF services continue to function correctly.

Using GemFire for Pivotal Cloud Foundry

As you would expect with any data service on Pivotal Cloud Foundry (PCF), the GemFire service simplifies the deployment and configuration of software that supports your applications. When you create a GemFire service instance, you are instantly provided with a dedicated cluster of GemFire members runing on dedicated VMs, that use JVM and GemFire settings automatically tuned for most use cases. You can then configure the cluster based on your application needs. For example, you might want to use an existing GemFire configuration from a GemFire development environment on your laptop. In that case, you can export the cluster configuration using GemFire's `gfsh export cluster-configuration`, and upload it to a GemFire service instance in your PCF environment using the GemFire service CLI for PCF. GemFire CLI is provided as a CF CLI plugin that includes commands for restarting, and configuring GemFire service instances, as well as downloading GemFire configuration and logs, on a per cluster basis.

Creating a GemFire Service Instance

The following procedure describes how to create a GemFire service instance in the Pivotal Cloud Foundry Elastic Runtime environment.

1. If you have not done so already, install the Pivotal Cloud Foundry Command Line Interface. See Installing the PCF CLI. Installation binaries are available here.

2. Log in to PCF using the PCF CLI.
   `$ cf login`

3. Run the following command to target the API endpoint, org and space where you want to create the service:
   `$ cf target -a <api-endpoint> -o <organization> -s <space name>`

4. Run the following command to view the available service plans.
   `$ cf marketplace`

   ```
   Getting services from marketplace in org staging / space staging as admin...
   OK
   service         plans                description
   p-gemfire       GemFireServicePlan1  Dedicated GemFire instance.
   TIP:  Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a
   given service.
   ```

5. Type the following command to create the service plan:
   `$ cf create-service GemFireServicePlan1 <service-plan-name> <service-instance-name>`
   where is the name of the Service Plan you see in the marketplace– in this example, "GemFireServicePlan1"– and is a descriptive name that you want to use for the service.
   For example:

   ```
   $ cf create-service p-gemfire GemFireServicePlan1 my-gemfire-test
   Creating service my-gemfire-test in org staging / space staging as admin...
   OK
   ```

Configuring a GemFire Service Instance

To configure a GemFire service instance (cluster) with your cache configuration and gemfire properties, you will need to use the GemFire service CF CLI plugin. If you have not installed the plugin yet, follow the instructions for installation described in Using the GemFire for Cloud Foundry CLI Plug-in.

Applying a cluster configuration involves uploading the configuration zip file to the cluster, and restarting the cluster. The configuration zip file is in the format used by the GemFire gfsh commands for exporting and importing cluster configurations (see Exporting and Importing Cluster Configurations in the Pivotal GemFire documentation). That allows for a cluster configuration to be exported from an existing deployment and used for a GemFire service instance on PCF. Alternatively, a cluster configuration zip file can be created manually from an existing GemFire cache xml configuration, GemFire properties, and any accompanying implementation and their dependency jars, as follows:

- Create a directory called `cluster`
- Copy your cache xml configuration file into the directory, and rename it to `cluster.xml`
- If there are any gemfire properties that should be set on the cluster, copy your gemfire properties file into the directory, and rename it to `cluster.properties`
- If there are any implementation jars and their dependency jars that should be deployed to the servers:
  - copy the implementation jars into `cluster` directory
  - copy any dependency jars into a directory called `lib` under `cluster` directory
- Create a zip file from `cluster` directory

The following example shows the contents of a configuration zip file:

```
$ unzip -t cluster_config.zip
Archive:  cluster_config.zip
    testing: cluster/                 OK
    testing: cluster/cluster.properties   OK
    testing: cluster/cluster.xml      OK
    testing: cluster/myCallbacks.jar     OK
    testing: cluster/lib/     OK
    testing: cluster/lib/myDependency1.jar      OK
    testing: cluster/lib/myDependency2.jar      OK
No errors detected in compressed data of cluster_config.zip.
```

In this example `myCallbacks.jar` contains the implementation of the callbacks referenced in the cache configuration, `cluster.xml`, and the lib directory contains the jars that myCalllbacks.jar depends on.

Upload and apply your configuration using `cf restart-gemfire` command with `--cluster-config` option. For example:

```
$ cf restart-gemfire my-gemfire-cluster --cluster-config ./cluster_config.zip
Broker HTTP Username>*****
Broker HTTP Password>*****
Cluster successfully restarted
```

In the above example, a cluster configuration contained in `cluster_config.zip` file in the current directory is uploaded to the service instance `my-gemfire-cluster`, and applied to it upon restarting the service instance (applying a cluster configuration requires that all the servers in the cluster be restarted).

See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password.

Configuring a Service Instance Using a Spring Application Context XML

If you use Spring you can configure your GemFire service instances using a Spring Application Context XML file instead of GemFire cache XML configuration. To use this approach you must include a Spring application context XML file in a jar that is provisioned in your configuration zip file, under a `lib` subdirectory of a `cluster` directory. In addition, you must place all implementation and dependency jars in the `cluster/lib` directory of the configuration zip file. The following are detailed steps for creating a Spring configuration zip file:

1. Create a directory called `cluster`, then a subdirectory under it called `lib`.
2. Place your Spring application context XML file in an implementation jar, and note the full classpath to the XML file as it appears in the JAR. You will need to reference this path later in the `cf restart-gemfire` command.

3. Copy all of the implementation and dependency JARs that should be deployed to the servers to the `cluster/lib` directory.

4. Create a zip file of the `cluster` directory.

For example, the following command shows the sample contents of a configuration zip file:

```
$ unzip -t cluster_spring_config.zip
Archive:  cluster_spring_config.zip
    testing: cluster/                 OK
    testing: cluster/lib/      OK
    testing: cluster/lib/myImplementation.jar     OK
    testing: cluster/lib/myDependency1.jar     OK
    testing: cluster/lib/myDependency2.jar     OK
No errors detected in compressed data of cluster_spring_config.zip.
```

In the above example, a Spring XML file could be placed in any of the JAR files under the `cluster/lib` directory. The configuration would then be uploaded and applied to a service instance using the `cf restart-gemfire` command with `--cluster-config` and `--spring-xml` options. For example:

```
cf restart-gemfire my-gemfire-cluster --cluster-config ./cluster_spring_config.zip --spring-xml /com/myCompany/myApp/myAppContext.xml
Broker HTTP Username>*****
Broker HTTP Password>*****
Cluster successfully restarted
```

In the above example, the cluster configuration file, `./cluster_spring_config.zip`, is uploaded to the `my-gemfire-cluster` service instance, and then applied after restarting the service instance. The `--spring-xml` option references the classpath to a Spring XML configuration file is included in one of the JAR files in `cluster_spring_config.zip`.

See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password.

Configuring JVM and GemFire Properties

The GemFire cluster configuration lacks the ability to set any JVM or GemFire immutable properties (the properties that have to be provided at the JVM startup). Those properties can be set using an optional `--properties` argument that takes a yaml file containing optional JVM and GemFire properties for locators and servers. For example:

```
$ cat ./properties.yml
properties:
  server:
      jvmargs:
          - "-XX:PermSize=96m"
          - "-XX:MaxPermSize=96m"
          - "-Dgemfire.OSProcess.ENABLE_OUTPUT_REDIRECTION=true"
      gemfire:
          statistic-sample-rate: 3000
  locator:
      jvmargs:
          - "-XX:PermSize=96m"
          - "-XX:MaxPermSize=96m"
      gemfire:
          statistic-sample-rate: 2000
```

The settings provided in this way will augment the existing (default) settings. This allows for the values of the existing properties to be modified, and new ones to be applied. The following command, for example:

```
$ cf restart-gemfire my-gemfire-cluster --cluster-config ./cluster_config.zip --properties ./properties.yml
Broker HTTP Username>*****
Broker HTTP Password>*****
Cluster successfully restarted
```

applies the properties from `./properties.yml` shown above to the servers and locators in the cluster `my-gemfire-cluster`, and the cluster configuration provided in `cluster_config.zip`. For more information about all the available CLI commands see Using the GemFire for Cloud Foundry CLI Plug-in.

See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password.

Cloud Deployment Considerations

Certain GemFire configuration settings are affected by the cloud nature of service instance deployments. Notable examples are any GemFire properties that take an IP address or hostname for value. Such properties should not be used. Also, disk directories for GemFire overflow and persistence disk stores are not supported. Disk stores must be configured without disk directories. As a result, disk stores are created in the working directory.

Network Partition Detection

As of v1.5.0.0, the GemFire for PCF service ships with the `enable-network-partition-detection` property enabled.

This built-in functionality of GemFire helps detect and resolve problems that result from adverse network events. Without this functionality, you run the risk of entering a split-brain situation where GemFire locators and servers cannot communicate with the rest of the cluster, leading to downtime and data loss.

Since the `enable-network-partition-detection` property can only be enabled or disabled across the entire cluster at once, the cluster must be brought down together. Hence, upgrades from an earlier version to v1.5.0.0 cannot happen in a rolling fashion. Further instructions can be found in our release notes.

Please refer to the Pivotal GemFire documentation for more information on Network Partitioning.

Working with a GemFire Service Instance

For the administration, configuration and monitoring of GemFire service instances the following tools are available:

- **GemFire Pulse** provides a graphical dashboard for monitoring vital, real-time health and performance of GemFire clusters, members, and regions. Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, WAN connections, and critical notifications
- **GemFire service plugin for CF CLI** provides CLI commands for configuring, and restarting GemFire clusters, as well as accessing the GemFire logs and statistics.
- **GemFire gfsh** provides remote access to GemFire clusters and many management, monitoring, and configuration features.

Accessing a Cluster via Pulse

Each GemFire service instance (cluster) has its own Pulse instance, which can be accessed via a *Manage* link located under the service instance in the Developer Console.

Restarting a Cluster

A GemFire cluster is restarted using the GemFire CLI command `cf restart-gemfire`. This command has multiple options (use it with `-h` to see all of them) that can be used in any combination to accomplish desired tasks during a restart.

This command also requires the Broker HTTP Username and Password. See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password.

Accessing GemFire Logs and Statistics, and Cluster Configuration

GemFire logs and statistics files for a cluster can be downloaded using the GemFire CLI command `cf export-gemfire`. For example:

```
$ cf export-gemfire cluster1 --logs ./cluster1_logs.zip
Broker HTTP Username>*****
Broker HTTP Password>*****
Successfully wrote logs and stats to `./cluster1_logs.zip`
```

`cluster1` is the name of the cluster to get the logs from.

`cf export-gemfire` is used to access the cluster configuration as well. For example:

```
$ cf export-gemfire demo1 --cluster-config ./demo1_config.zip --properties ./demo1_props.yaml
Broker HTTP Username>*****
Broker HTTP Password>*****
Successfully wrote cluster config to `./demo1_config.zip`
Successfully wrote cluster properties to `./demo1_props.yaml`
```

In this example, `cf export-gemfire` is used to download the cluster configuration to the file named `demo1_config.zip` and properties to the file named `demo1_props.yaml` for the cluster named `demo1`.

Note that the GemFire log files also include the full configuration.

See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password.

Accessing a Cluster via gfsh

GemFire clusters can be accessed via `gfsh` from any machine within the Cloud Foundry network.

First identify the locator IP address of your cluster with a service key:

```
$ cf create-service-key demo1 my-service-key
Creating service key my-service-key for service instance demo1 as admin...
OK
$ cf service-key demo1 my-service-key
Getting key my-service-key for service instance demo1 as admin...
{
 "locators": [
  "10.244.0.50[55221]",
  "10.244.0.51[55221]"
 ],
 "password": "<CLUSTER_USERNAME>",
 "username": "<CLUSTER_PASSWORD>"
}
```

Use the locator IP addresses to connect with gfsh running on a host within the Cloud Foundry network:

```
$ gfsh
gfsh>connect --locator=10.244.0.50[55221]
Connecting to Locator at [host=10.244.0.50, port=55221] ..
Connecting to Manager at [host=10.244.0.50, port=1099] ..
Successfully connected to: [host=10.244.0.50, port=1099]
```

Accessing GemFire Service Connection Information (Binding)

After you deploy a Java Buildpack application and bind it to a GemFire for Pivotal Cloud Foundry service instance, your application receives the GemFire cluster locator addresses and credentials in the VCAP_SERVICES environment variable. If you have enabled the GemFire REST API, then the REST URL is also provided as metadata in VCAP_SERVICES.

VCAP_SERVICES provides data as a JSON document, so you can use a variety of techniques to access the relevant connection information. SeeViewing Binding Meta Data for an example of the data provided in VCAP_SERVICES.

Note: GemFire for Pivotal Cloud Foundry only supports deploying GemFire client applications. You cannot deploy an application that participates in the bound GemFire cluster as a peer member.

Using a JSON Library to Acquire Connection Information

Because VCAP_SERVICES provides a JSON document, you can also use a Java JSON library to parse the data for connection information. This example code uses the `Jackson` library to parse the document:

```
package pivotal;

import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import pivotal.GemFireClient.Locator;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class EnvParser {

    private static EnvParser instance;

    private EnvParser() {
    }

    public static EnvParser getInstance() {
        if (instance != null) {
            return instance;
        }
        synchronized (EnvParser.class) {
            if (instance == null) {
                instance = new EnvParser();
            }
        }
        return instance;
    }

    public List<Locator> getLocators() throws JsonParseException, JsonMappingException, IOException {
        List<Locator> locatorList = new ArrayList<GemFireClient.Locator>();
        Map credentials = getCredentials();
        List<String> locators = (List<String>) credentials.get("locators");
        for (String locator : locators) {
            String locatorIP = locator.substring(0, locator.indexOf("["));
            String portString = locator.substring(locator.indexOf("[") + 1,
                    locator.indexOf("]"));
            locatorList.add(new Locator(locatorIP, Integer.parseInt(portString)));
        }
        return locatorList;
    }

    public String getUsername() throws JsonParseException, JsonMappingException, IOException {
        String username = null;
        Map credentials = getCredentials();
        username = (String) credentials.get("username");
        return username;
    }

    public String getPasssword() throws JsonParseException, JsonMappingException, IOException {
        String password = null;
        Map credentials = getCredentials();
        password = (String) credentials.get("password");
        return password;
    }

    private Map getCredentials() throws JsonParseException, JsonMappingException, IOException {
        Map credentials = null;
        String envContent = System.getenv().get("VCAP_SERVICES");
        List<Locator> locatorList = new ArrayList<GemFireClient.Locator>();
        ObjectMapper objectMapper = new ObjectMapper();
        Map services = objectMapper.readValue(envContent, Map.class);
        List gemfireService = getGemFireService(services);
        if (gemfireService != null) {
            Map serviceInstance = (Map) gemfireService.get(0);
            credentials = (Map) serviceInstance.get("credentials");
        }
        return credentials;

    }

    private List getGemFireService(Map services) {
        return  (List) services.get("p-gemfire");
    }
}
```

To authenticate your client application, you must implement the GemFire AuthInitialize interface, as shown in this example:

```
package pivotal;

import com.gemstone.gemfire.LogWriter;
import com.gemstone.gemfire.distributed.DistributedMember;
import com.gemstone.gemfire.security.AuthInitialize;
import com.gemstone.gemfire.security.AuthenticationFailedException;

import java.io.IOException;
import java.util.Properties;

public class ClientAuthInitialize implements AuthInitialize {

    private EnvParser env = EnvParser.getInstance();

    public static final String USER_NAME = "security-username";
    public static final String PASSWORD = "security-password";

    public static AuthInitialize create() {
        return new ClientAuthInitialize();
    }

    @Override
    public void close() {
    }

    @Override
    public Properties getCredentials(Properties arg0, DistributedMember arg1,
                                     boolean arg2) throws AuthenticationFailedException {
        Properties props = new Properties();
        try {
            String username = env.getUsername();
            String password = env.getPasssword();
            props.put(USER_NAME, username);
            props.put(PASSWORD, password);
        } catch (IOException e) {
            throw new AuthenticationFailedException("Exception reading username/password from env variables ", e);
        }
        return props;
    }

    @Override
    public void init(LogWriter arg0, LogWriter arg1)
            throws AuthenticationFailedException {
    }
}
```

The authenticated application can then create a GemFire ClientCache as follows:

```
Properties props = new Properties();
props.setProperty("security-client-auth-init", "pivotal.ClientAuthInitialize.create");
ClientCacheFactory ccf = new ClientCacheFactory(props);
try {
    List<URI> locatorList = EnvParser.getInstance().getLocators();
    for (URI locator : locatorList) {
        ccf.addPoolLocator(locator.getHost(), locator.getPort());
    }
    ClientCache client = ccf.create();
} catch (IOException e) {
          // handle
}
```

Using spring-cloud to Acquire Connection Information

As an alternative to use a Java JSON library, you can use `spring-cloud` with the Spring Cloud GemFire Connector to parse the JSON data. Note that this method does not require your application to be a Spring project or to have `spring-core` as a dependency. You only need to specify `spring-cloud` and `spring-cloud-gemfire-cloudfoundry-connector` as dependencies as shown in this exerpt:

```
<dependency>
        <groupId>com.gemstone.gemfire</groupId>
        <artifactId>gemfire</artifactId>
        <version>8.1.0</version>
    </dependency>
    <dependency>
        <groupId>io.pivotal.spring.cloud</groupId>
        <artifactId>spring-cloud-gemfire-cloudfoundry-connector</artifactId>
        <version>1.0.0.BUILD-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-core</artifactId>
        <version>1.2.0.RC1</version>
    </dependency>
...
        <repository>
            <id>spring-milestones</id>
            <url>http://repo.spring.io/libs-milestone/</url>
        </repository>
        <repository>
            <id>gemfire-repository</id>
            <name>Gemfire Repository</name>
            <url>http://dist.gemstone.com/maven/release</url>
        </repository>
```

For an application that binds to a GemFire service named `MyService,` you would then use the `CloudFactory` API to obtain a service connection from which you get metadata such as the available locators, username, and password. For example:

```
CloudFactory cloudFactory = new CloudFactory();
Cloud cloud = cloudFactory.getCloud();
GemfireServiceInfo myService = (GemfireServiceInfo) cloud.getServiceInfo("MyService");

URI[] locators = myService.getLocators();
String userName = myService.getUsername();
String password = myService.getPassword();
```

To authenticate your client application, you must implement the GemFire AuthInitialize interface, as shown in this example:

```
public class ClientAuthInitialize implements AuthInitialize {

  public static final String USER_NAME = "security-username";
  public static final String PASSWORD = "security-password";

  public GemfireServiceInfo serviceInfo;

  private ClientAuthInitialize() {
    CloudFactory cloudFactory = new CloudFactory();
    Cloud cloud = cloudFactory.getCloud();
    serviceInfo = (GemfireServiceInfo) cloud.getServiceInfo("MyService");
  }

  public static AuthInitialize create() {
    return new ClientAuthInitialize();
  }

  @Override
  public void close() {
  }

  @Override
  public Properties getCredentials(Properties arg0, DistributedMember arg1,
                                   boolean arg2) throws AuthenticationFailedException {
    Properties props = new Properties();

    String username = serviceInfo.getUsername();
    String password = serviceInfo.getPassword();
    props.put(USER_NAME, username);
    props.put(PASSWORD, password);

    return props;
  }

  @Override
  public void init(LogWriter arg0, LogWriter arg1)
      throws AuthenticationFailedException {
  }
}
```

The code to initialize the client, obtain authorization, and finally create a GemFire ClientCache would resemble:

```
CloudFactory cloudFactory = new CloudFactory();
Cloud cloud = cloudFactory.getCloud();
GemfireServiceInfo myService = (GemfireServiceInfo) cloud.getServiceInfo("MyService");
Properties props = new Properties();
props.setProperty("security-client-auth-init", "pivotal.ClientAuthInitialize.create");
ClientCacheFactory ccf = new ClientCacheFactory(props);
URI[] locators = myService.getLocators();
for (URI locator : locators) {
  ccf.addPoolLocator(locator.getHost(), locator.getPort());
}
ClientCache cache = ccf.create();
```

Acquiring a Connection from a Spring Application

If you are developing a Spring Application, then acquiring GemFire service connection information is a simple, two-step process:

1. Configure GemFire, `spring-cloud` and `spring-cloud-gemfire-cloudfoundry-connector` as dependencies in your project:

   ```
   dependencies {
   compile("com.gemstone.gemfire:gemfire:8.1.0")
   compile("io.pivotal.spring.cloud:spring-cloud-gemfire-spring-connector:1.0.0.BUILD-SNAPSHOT")
   compile("io.pivotal.spring.cloud:spring-cloud-gemfire-cloudfoundry-connector:1.0.0.BUILD-SNAPSHOT")
   }
   ```

2. In your application, auto-wire the GemFire ClientCache:

   ```
   @Configuration
   @ServiceScan
   @RestController
   public class MyController {

     @Autowired
     ClientCache cache;
   ```

Configuring a Java REST Client to Access a Service Instance via HTTPS

To support Java client applications accessing a service instance REST API endpoint via HTTPS, you must modify the Java Buildpack to provision security artifacts. You will need to obtain the correct SSL certificate from your Cloud Foundry HAProxy credentials and import it into the JRE truststore of the Java Buildpack. Importing the certificate then makes it available to your Java applications.

You configure the JRE with additional resources by adding the resource files to the appropriate location under the `resources` directory of the buildpack. For SSL certificates that should be added to the OpenJDK JRE truststore, the correct location is `resources/open_jdk_jre/lib/security` under the buildpack root. To add custom SSL certificates, add your cacerts file to `resources/open_jdk_jre/lib/security/cacerts`. This file is overlayed onto the OpenJDK distribution. The exact procedure to follow is:

1. Get the SSL certificate from your PCF Ops Manager by going to the Credentials tab in Elastic Runtime and copying the SSL RSA certificate from the HAProxy section. Save the certificate to a new file (for example, `myCert.cert`).

2. Import the saved certificate file to your trust store.

3. Clone the java build-pack.

4. Copy your truststore from Step 2 above into `resources/open_jdk_jre/lib/security/cacerts` under your buildpack root directory.

5. Rebuild your Java Buildpack. This example uses offline mode:

   ```
   $ cd ~/projects/java-buildpack
   $ bundle install
   $ bundle exec rake package OFFLINE=true
   ```

6. Upload the build pack. For example:

```
$ cf create-buildpack secure-java-buildpack build/java-buildpack-offline-<sha>.zip 1
```

7. Make sure client applications use HTTPS when forming the REST API endpoint URL.

8. Use the new buildpack when pushing your client applications. For example:

```
$ cf push -f app.yml -t 30 -b secure-java-buildpack
```

Deploying Applications for Use with the GemFire Service

This section provides tips on pushing your application to the Pivotal Cloud Foundry Elastic Runtime environment for use with the GemFire service. Some of the application deployment steps may differ depending on what kind of application you are deploying.

Re-deploying your application does not affect data stored in any existing service instances bound to the application.

See Deploy an Application in the Pivotal Cloud Foundry Documentation for detailed information on pushing CF applications.

Using the Java Buildpack

Important: Pivotal recommends that you use the latest Java Buildpack when pushing your applications. Specify the location of the buildpack using the `-b` parameter:

```
$ cf push <your-app-name> -p <location-of-your-app-file> \
-b https://github.com/cloudfoundry/java-buildpack.git
```

To avoid having to restage your application after binding the service, you can push the application initially with the `--no-start` command:

```
$ cf push <your-app-name> -p <location-of-your-app-file> \
 -b https://github.com/cloudfoundry/java-buildpack.git --no-start
```

Binding an Application to the GemFire Service

The following procedures describe how to bind a GemFire service instance to your Pivotal Cloud Foundry application.

PCF Developer Console Instructions

1. Log in to the **Pivotal Cloud Foundry Developer Console**.

2. Select your **Org** from the drop-down list on the left. This should be the same Org where you created the service instance.

3. Select the space where your GemFire service instance and bound application have been deployed.

4. Select the application that you wish to bind to the service. A page displays showing the already bound services and instances for this application.

5. Click **Bind**. A list of available services displays.

6. Click the **Bind** button for the GemFire service you want to bind to this application.

7. Using the Pivotal Cloud Foundry CLI, start or restage your application. See Pushing or Restaging Applications After Changes.

CLI Instructions

Alternately via the CLI:

1. Log in to your Pivotal Cloud Foundry environment using the Pivotal Cloud Foundry CLI.
   `$ cf login`

2. Run the following command to target the specific org and space where you want to create the service plan.
   `$ cf target -o <organization> -s <space_name>`

3. Run the following command to view running service instances:
   `$ cf services`

   ```
   Getting services in org staging / space staging as admin...
   OK
   name              service          plan                 bound apps
   my-gemfire-test   gemfire           GemFireSmallPlan
   ```

4. Run the following command to bind the application to the service instance:
   `$ cf bind-service <application> <service-instance-name>`
   For example:

   ```
   $ cf bind-service gfe-app my-gemfire-test
   Binding service my-gemfire-test to app gfe-app
   in org staging / space  staging as admin...
   OK
   TIP: Use 'cf restage' to ensure your env variable changes take effect
   ```

5. Restage your application.
   `$ cf restage <application>`

Pushing or Restaging Applications After Service Changes

To ensure that your application picks up the correct environment variables, you must restage or re-push your applications after binding them to the GemFire service. This can currently be done using the CLI. In addition, if you make any other changes to the GemFire Service while it is bound to your application (for example, add, modify or delete the service), you will need to re-push or restage your application afterwards.

Pushing or re-staging your application does not affect data stored in the existing service instance.

To restage or re-push your application using the Pivotal Cloud Foundry CLI:

1. Log in to your Pivotal Cloud Foundry environment using the Pivotal Cloud Foundry CLI.
   `$ cf login`

2. Run the following command to target the API endpoint, org and space where you want to push or restage the application. For example:
   `$ cf target -a <api-endpoint> -o <organization> -s <space_name>`

3. Push or restage your existing application:
   `$ cf push <application> -p <location-of-your-app-file> -b <buildpack-location>`
   or
   `$ cf restage <application>`
   Alternately, if you pushed your application without starting it, you can start your application now to pick up the newly bound service.
   `$ cf start <application>`

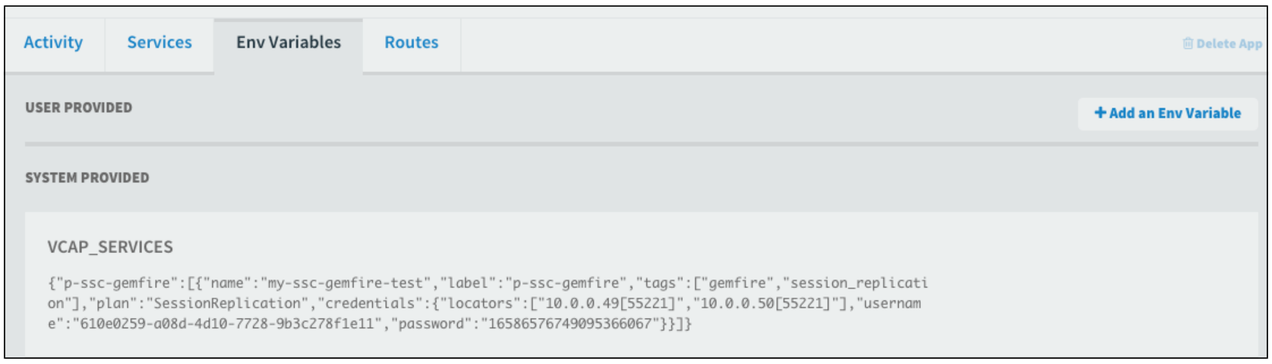
For more details on deploying applications, see Deploy an Application in the Pivotal Cloud Foundry documentation.

Viewing Binding Meta Data

To view the binding variables, use the following procedures.

PCF Developer Console Instructions

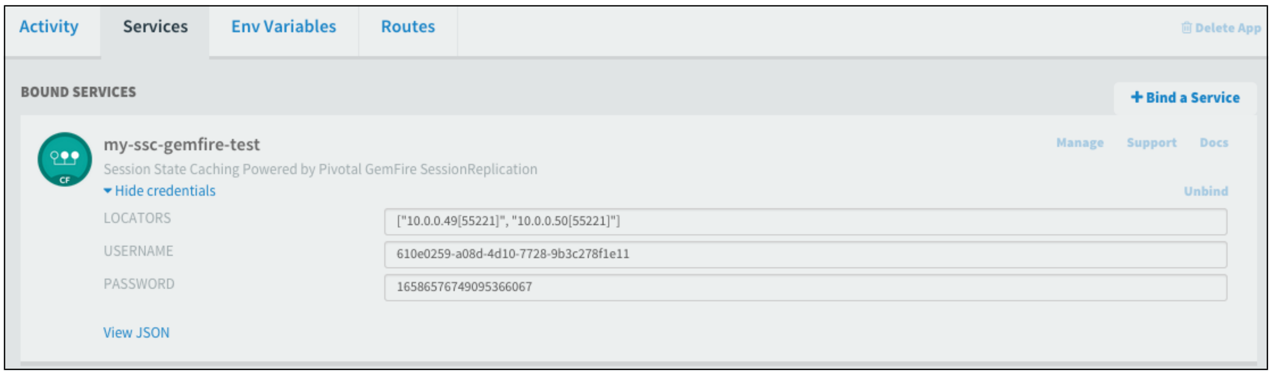
1. Log in to the **Pivotal Cloud Foundry Developer Console**.
2. Select your **Org** from the drop-down list on the left.
3. Select the space where your GemFire service instance and bound application have been deployed.
4. Select the application that you have bound to the GemFire service.
5. Click on the **Env Variables** tab. The environment variables for the service binding display:



VCAP_SERVICES

```
{"p-ssc-gemfire":[{"name":"my-ssc-gemfire-test","label":"p-ssc-gemfire","tags":["gemfire","session_replicati
on"],"plan":"SessionReplication","credentials":{"locators":["10.0.0.49[55221]","10.0.0.50[55221]"],"usernam
e":"610e0259-a08d-4d10-7728-9b3c278f1e11","password":"16586576749095366067"}}]}
```

Alternately, you can also view credentials used by the service when binding to the application by clicking on the **Services** tab and clicking **Show credentials**.
Service credentials and locator address information displays:



my-ssc-gemfire-test
Session State Caching Powered by Pivotal GemFire SessionReplication

LOCATORS	["10.0.0.49[55221]", "10.0.0.50[55221]"]
USERNAME	610e0259-a08d-4d10-7728-9b3c278f1e11
PASSWORD	16586576749095366067

CLI Instructions

To view the binding variables in the CLI, type the following command after you have bound your application to the GemFire service:

```
$ cf env <application-
name>
```

If successful, you should see similar to the following in the returned output:

```
Getting env variables for app   in org  / space  as ...
OK
System-Provided:
{
 "VCAP_SERVICES": {
  "p-gemfire": [
   {
    "credentials": {
     "locators": [
      "10.0.0.55[55221]",
      "10.0.0.56[55221]"
     ],
     "password": "15587128842615488747",
     "rest_url": "",
     "username": "dacaf950-1633-4741-7dc5-eb11ce5f33e2"
    },
    "label": "p-gemfire",
    "name": "MyService",
    "plan": "GemFireServicePlan1",
    "tags": [
     "gemfire"
    ]
   }
  ]
 }
}
No user-defined env variables have been set
No running env variables have been set
No staging env variables have been set
```

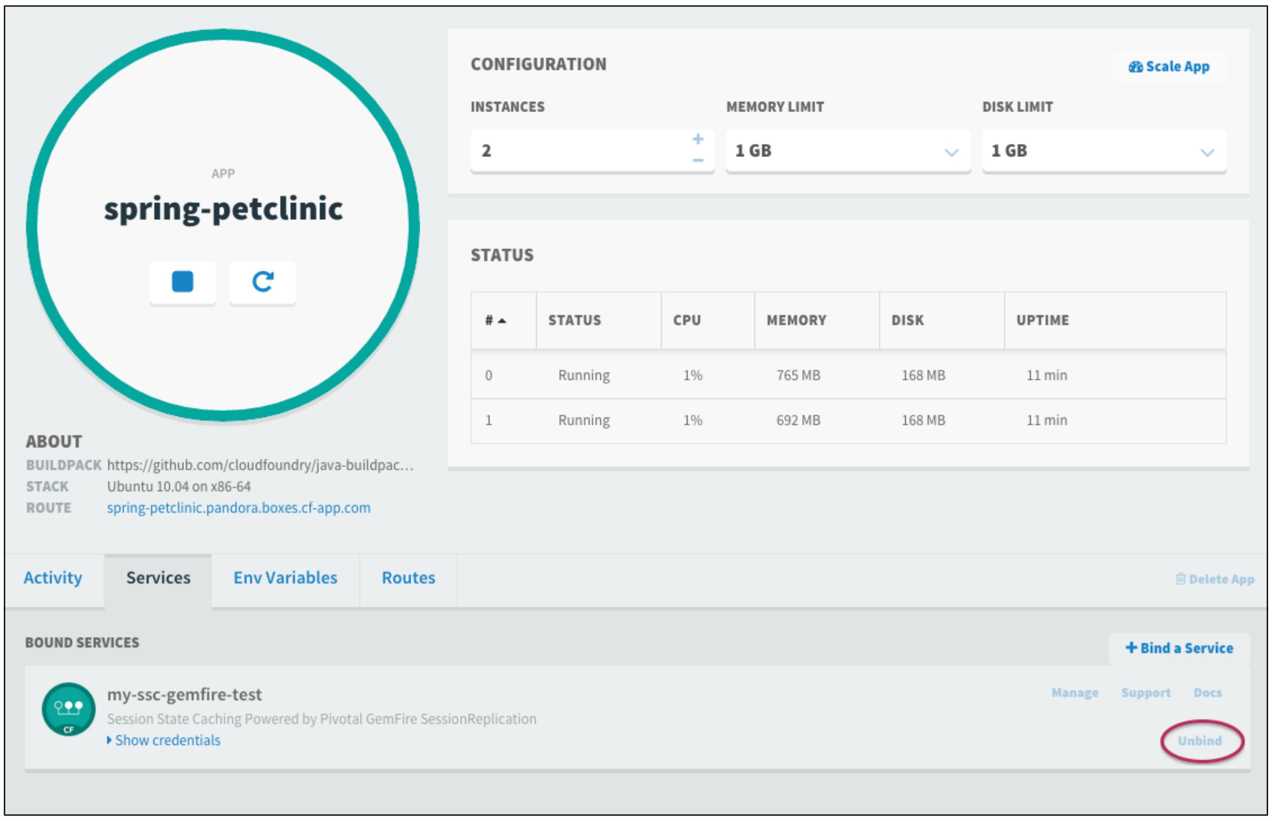
Unbinding an Application from the GemFire Service

When you create a GemFire service instance, the GemFire Service Broker allocates a specific cluster of GemFire locators and servers. You can bind, unbind and the bind again to that particular service instance as often as you want. The service instance always uses the same cluster and the Service Broker does not do anything to the data in that service instance.

PCF Developer Console Instructions

To unbind the application from the GemFire service:

1. Log in to the **Pivotal Cloud Foundry Developer Console**.
2. Select your **Org** from the drop-down list on the left.
3. Select the space where your GemFire service instance and bound application have been deployed.
4. Select the application that you have bound to the GemFire service. A page displays that show the bound services and instances for this application.
5. Locate the bound service instance you want to unbind and click **Unbind**.



APP
spring-petclinic

CONFIGURATION

INSTANCES	MEMORY LIMIT	DISK LIMIT
2	1 GB	1 GB

STATUS

#	STATUS	CPU	MEMORY	DISK	UPTIME
0	Running	1%	765 MB	168 MB	11 min
1	Running	1%	692 MB	168 MB	11 min

ABOUT
BUILDPACK https://github.com/cloudfoundry/java-buildpac...
STACK Ubuntu 10.04 on x86-64
ROUTE spring-petclinic.pandora.boxes.cf-app.com

BOUND SERVICES
my-ssc-gemfire-test
Session State Caching Powered by Pivotal GemFire SessionReplication

6. A confirmation dialog box displays. Click **Unbind** again.
7. If successful, the following message will appear at the top of the screen:

Service successfully unbound from the application. TIP: Use 'cf push' to ensure your env variable changes take effect.

8. Use the Pivotal Cloud Foundry CLI to push or restage your application for the changes to take effect. See Pushing or Restaging Applications After Changes.

CLI Instructions

1. Log in to your Pivotal Cloud Foundry environment using the Pivotal Cloud Foundry CLI.
   `$ cf login`

2. Run the following command to target the org and space where you want to push or restage the application. For example:
   `$ cf target -o <organization> -s <space_name>`

3. Run the following command:
   `$ cf unbind-service <application> <service-instance-name>`
   where is the name of the GemFire instance you are unbinding from the specified .
   For example:

```
$ cf unbind-service gfe-app my-gemfire-test
Unbinding app gfe-app from service my-gemfire-test in org staging / space
staging as admin...
OK
```

4. Restage or re-push your application for the application changes to take effect. See Pushing or Restaging Applications After Changes.

Deleting a GemFire Service Instance

When you delete a GemFire service instance, all applications that are bound to that service are automatically unbound and any data in the service instance is cleared. In addition, the allocated service instance (GemFire cluster) is returned to the pool of available clusters and those locators and cache servers are now available to future applications.

PCF Developer Console Instructions

To delete a service instance using the Pivotal Cloud Foundry Developer Console:

1. Log in to the **Pivotal Cloud Foundry Developer Console**.

2. Select your **Org** from the drop-down list on the left.

3. Locate the row under Services that contains the service instance you want to delete and click **Delete**.

4. If you had applications that were bound to this service, you may need to restage or re-push your application for the application changes to take effect. See Pushing or Restaging Applications After Changes.

CLI Instructions

1. Log in to your Pivotal Cloud Foundry environment using the Pivotal Cloud Foundry CLI.
   `$ cf login`

2. Run the following command to target the org and space where you want to push or restage the application. For example:
   `$ cf target -o <organization> -s <space_name>`

3. Run the following command:
   `$ cf delete-service <service-instance-name>`
   where is the name of the GemFire service you are deleting. Enter 'y' when prompted.
   For example:

```
$ cf delete-service my-gemfire-test
Really delete the service my-gemfire-test?> y
Deleting service my-gemfire-test in org staging / space staging as admin...
```

```
OK
```

4. If you had applications that were bound to this service, you may need to restage or re-push your application for the application changes to take effect. See Pushing or Restaging Applications After Changes.

Configuring Multi-site (WAN) Connections Between Two or More GemFire Service Instances

You can configure GemFire service instances for multi-site (WAN) communication. For information about GemFire multi-site replication capabilities, see Multi-site (WAN) Configuration in the *GemFire User's Guide*.

A key prerequisite for multi-site replication is network connectivity between the GemFire instances to be configured for multi-site connections. GemFire nodes on the opposite sides of a multi-site (WAN) link must be able to establish direct connections to each other.

The GemFire service for PCF provides functionality that makes the WAN configuration process easier for GemFire clusters in PCF. The service provides support for configuring service instances so that they discover each other and establish WAN connections. You must still create WAN senders and receivers, and configure data regions for multi-site replication.

The steps to enable WAN connections between two GemFire service instances are as follows:

1. For each service instance, obtain the WAN configuration URL using the command:

   ```
   cf show-wan-config-url <service_instance>
   ```

2. When you create the cluster configuration for each service instance, configure the WAN senders and receivers, and data regions that should be replicated over the WAN.

3. Set the GemFire property `distributed-system-id` to a unique integer value in the properties yml file for each service instance (see the previous section for more information about configuring GemFire properties for a service instance).

4. Restart each service instance with the WAN configuration URLs for the other instances passed as argument to the `--enable-wan` option of the `cf restart-gemfire` command. For example:

   ```
   cf restart-gemfire <service_instance_A> \
     --cluster-config <cluster_A_config_zip> \
       --properties <properties_cluster_A_yml> \
       --enable-wan <cluster_B_WAN_config_URL>
   ```

   The above command restarts the locators in addition to the servers. This command also requires the Broker HTTP Username and Password. See the section on Broker Credentials on how to obtain the Broker HTTP Username and Password. After restarting, the clusters are enabled for WAN communication to each other.

Using the GemFire for Cloud Foundry CLI Plug-in

Installation

Follow this procedure to install the GemFire CLI plug-in:

1. Download the plug-in binary from Pivotal Network.
2. Enable execute permissions on the downloaded file. For example:

   ```
   $ chmod a+x ./cf-gemfire-cli-darwin-amd64
   ```

3. Install the plug-in binary:

   ```
   $ cf install-plugin ./cf-gemfire-cli-darwin-amd64-1.2.0

   Installing plugin ./cf-gemfire-cli-darwin-amd64-1.2.0...
   OK
   Plugin GemFire v1.2.0 successfully installed.
   ```

4. (Optional) Use the `cf plugins` command to verify that you have successfully installed the GemFire plugin:

```
$ cf plugins
Listing Installed Plugins...
OK

Plugin Name   Version   Command Name        Command Help
GemFire       1.2.0     gemfire             GemFire plugin command's help text
GemFire       1.2.0     restart-gemfire     Restart GemFire cache servers (Also used for applying configuration changes)
GemFire       1.2.0     export-gemfire      Retrieve GemFire artifacts, such as logs and stats
GemFire       1.2.0     show-wan-config-url Display the WAN configuration URL for the service instance
```

See Using cf CLI Plugins in the Pivotal Cloud Foundry (PCF) documentation for general information about installing, uninstalling, or listing available Cloud Foundry Command Line Interface plug-ins.

Overview

Use the `cf gemfire` command to display basic usage information for the plugin:

```
$ cf gemfire
NAME:
  cf - Cloud Foundry plugin to interact with GemFire service instances

USAGE:
  cf [global options] command [command options] [arguments...]

VERSION:
  1.2.0

AUTHOR(S):
  Pivotal Inc.

COMMANDS:
  restart-gemfire     SERVICE_INSTANCE --cluster-config CONFIG.ZIP | -c CONFIG.ZIP
  export-gemfire      SERVICE_INSTANCE
  show-wan-config-url SERVICE_INSTANCE
  help, h             Shows a list of commands or help for one command

GLOBAL OPTIONS:
  --help, -h          show help
  --version, -v       print the version
```

Execute any of the commands with the `-h` argument to display usage information. For example:

```
$ cf export-gemfire -h
NAME:
   export-gemfire - Retrieve GemFire artifacts, such as logs and stats

USAGE:
   cf export-gemfire SERVICE_INSTANCE [command options]

DESCRIPTION:
   Retrieve artifacts from a given service instance (cluster).

OPTIONS:
   --logs, -l           path for downloaded log file archive
   --cluster-config, -c  path for downloaded cluster configuration
   --properties, -p      path for downloaded cluster properties
```

Broker HTTP Username and Password

The following commands require additional authentication:

- `export-gemfire`
- `restart-gemfire`

The Broker username and password can be found on your PCF Ops Manager on the Gemfire for PCF Tile. See the Credentials tab. Under the name**Gemfire Broker Credentials** they will be in the form of `username / password`. These credentials will be required when running the commands `export-gemfire` & `restart-gemfire` by a command line prompt.

Optionally these commands can be run non-interactively by using the flags `--broker-username` or `-U` and `--broker-password` or `-P`.

export-gemfire

`cf export-gemfire` enables you to export artifacts from a given service instance, such as the GemFire logs and statistics, and the cluster configuration and properties. The cluster configuration and properties exported in this way can be applied to another service instance using the `restart-gemfire` command.

The `cf export-gemfire --cluster-config` command downloads the configuration in the format provided by the GemFire Cluster Configuration service. See Overview of the Cluster Configuration Service in the Pivotal GemFire documentation for more information.

You'll be prompted for credentials `Broker HTTP Username` & `Broker HTTP Password` when this command is invoked. Please see the note on how to obtain the credentials. To run this command non-interactively, the broker username and password can be provided with `--broker-username` or `-U` and `--broker-password` or `-P` flags.

Usage:

```
cf export-gemfire SERVICE_INSTANCE [command options]
```

Optional arguments include:

- `--logs PATH` — Downloads the GemFire logs and statistics files for the cluster to the path and zip file given by PATH
- `--cluster-config PATH` — Downloads the GemFire cluster configuration to the path and zip file given by PATH
- `--properties PATH` — Downloads the cluster properties yaml file to the path and yaml file given by PATH

Example:

```
cf export-gemfire clusterA --logs ./clusterA_logs.zip --properties ./clusterA_props.yml --cluster-config ./clusterA_config.zip
```

restart-gemfire

`cf restart-gemfire` enables you to restart a GemFire cluster and perform a number of cluster tasks that either require a cluster restart (such as applying a cluster cache configuration) or that are convenient to perform during a restart (such as clearing out log files). The command allows for "daisy-chaining" options so you can perform multiple tasks with a single restart. For example, you can use a single application of this command to apply a new cluster configuration, clear the log files, and enable the REST API.

Note: Using `--reset-defaults` with `--cluster-config` first causes the cluster configuration to be restored to the original configuration. The new configuration (provided with `--cluster-config`) is then applied. Both operations take take place during the same restart.

You'll be prompted for credentials `Broker HTTP Username` & `Broker HTTP Password` when this command is invoked. Please see the note on how to obtain the credentials. To run this command non-interactively, the broker username and password can be provided with `--broker-username` or `-U` and `--broker-password` or `-P` flags.

The only required argument to this command is the name of a GemFire for Pivotal Cloud Foundry service instance that you have deployed. All other arguments are optional.

Usage:

```
cf restart-gemfire SERVICE_INSTANCE [--clear-logs] [--clear-disks]
[--cluster-config PATH] [--default-server enable | disable]
[--enable-wan COMMA_SEPARATED_LIST_OF_WAN_CONFIG_URLS]
[--include-locators] [--spring-xml CLASSPATH_TO_XML] [--properties PATH] [--reset-defaults]
[--rest-api enable | disable] [--timeout "SECONDS"]
```

Optional arguments include:

- `--clear-logs` — Clears all of the GemFire logs and statistics files for the cluster.
- `--clear-disks` — Clears all of the disk stores, removing all persisted data. **CAUTION: This option removes all persisted data from the cluster!**
- `--cluster-config PATH` — Uploads the cluster configuration file given by PATH.
- `--default-server=enable|disable` — Determines whether to start up the default GemFire cache server.
- `--enable-wan COMMA_SEPARATED_LIST_OF_WAN_CONFIG_URLS` — Enables a GemFire WAN replication channel to each of the service instances identified by the WAN configuration URLs you provide. You can obtain the WAN configuration URLs by invoking `cf show-wan-config-url` on each of the service instances to which you want to enable a WAN connection. Then provide the configuration URLs as comma separated argument list. **Setting this option restarts the GemFire locators.**
- `--include-locators` — Includes the locators in all requested actions. If you omit this option, the command only applies to cache servers.
- `--spring-xml` — Specifies the classpath to a Spring XML file in a cluster configuration JAR. This option applies the Spring XML file in the JAR to the cluster. This option must be used along with the `--cluster-config` option to provision the containing JAR file.
- `--properties PATH` — Applies the properties in the YAML file at the specified PATH to the GemFire cluster. See *Configuring JVM and GemFire Properties* in Configuring a GemFire Service Instance for more information.
- `--reset-defaults` — Restores the original, default, configuration for the cluster; clears archives and disk stores. **CAUTION: This option removes all persisted data from the cluster!**
- `--rest-api=enable|disable` — Enables or disables the GemFire REST developer API. Enabling the REST API sets up a single front end URL for REST connections. Connections to the URL are load balanced between all available servers.
- `--timeout SECONDS` — Provides the timeout value, in seconds, for the restart command.

show-wan-config-url

`cf show-wan-config-url` shows the WAN configuration URL for the given service instance in HTTP Authentication format with placeholders for the broker username & password. Replace the placeholders in the URL before passing it as input to the `--enable-wan` option of `cf restart-gemfire` to configure other service instances for WAN communication to this instance.

Please see the note on how to obtain the credentials to replace the placeholders `BROKER_USERNAME` and `BROKER_PASSWORD` in the URL.

Usage:

```
cf show-wan-config-url SERVICE_INSTANCE
```

Example:

```
$ cf show-wan-config-url instance_one
=> https://BROKER_USERNAME:BROKER_PASSWORD@gemfire-broker.gf2.pcf-gemfire.com/admin/wan/credentials/93f7add8-d077-4489-b16b-4905c51d3d46

Change the placeholders BROKER_USERNAME and BROKER_PASSWORD in the cli output before passing it as input to  `cf restart-gemfire --enable-wan`
```

```
$ cf restart-gemfire --enable-wan https://my-username:my-password@gemfire-broker.gf2.pcf-gemfire.com/admin/wan/credentials/93f7add8-d077-4489-
```

Troubleshooting

Service Installation Troubleshooting

Problem: Could not deploy a service instance in the Elastic Runtime

If you try to deploy (and allocate) more Session State Caching service instances than are available as defined by your Pivotal Cloud Foundry (PCF) Administrator, you may receive the following error message:

> 500 ERROR. YOU HAVE REACHED THIS PAGE BECAUSE AN ERROR OCCURRED. PLEASE CONTACT YOUR OPERATOR

You need to delete existing service instances or ask the admin to deploy more instances.

Problem: Service plan tile does not show up in the Developer Console Marketplace

Verify that the register errand under lifecycle errands is selected in the Ops Manager tile configuration screen.

Application Troubleshooting

Problem: Application does not pick up changes to the service instance

Make sure you restage (or re-push) your application if you have made configuration changes to the underlying service instance.

Service Broker Troubleshooting

The Service Broker is responsible for managing the lifecycle of service instances. If you encounter issues with service instance management (creating, destroying, binding, unbinding etc.), the broker logs may contain some clues.

The logs can be found on broker VMs at `/var/vcap/sys/log/broker/broker.stdout.log` and are logged in JSON format. You can use a utility such as jq to parse these logs and extract information. `log_level` of each message is number 0-3 corresponding to the following log levels: `DEBUG` , `INFO` , `ERROR` and `FATAL` . Here are some examples (using jq 1.5):

Display all ERROR logs:

```
jq '. | select(.log_level == 2)' broker.stdout
```

Search log for messages containing a substring:

```
jq '. | select(.message | contains("some-substring"))' broker.stdout
```

Check out the jq manual for more usage details.

Broker endpoints for advanced troubleshooting

- GET `/admin/credentials/{locator ip}` : Given an IP address of a locator, will retrieve the credentials for the servers in the Gemfire cluster.

```
curl 10.0.16.153:8080/admin/credentials/10.0.16.155
```

```
{
 "credentials": [
   {
    "password": "apassword",
    "username": "ausername",
    "locators": [
      "10.0.16.154",
      "10.0.16.155"
    ]
   }
 ]
}
```

- GET `/admin/wan/credentials/{service instance GUID}` : Given the GUID of a Gemfire service instance, will retrieve the WAN credentials for that server

```
curl 10.0.16.153:8080/admin/wan/credentials/7dd9446b-20c6-4fbc-a31f-8416fd96abbd
```

```
{
 "password": "apassword",
 "username": "ausername",
 "locators": [
   "10.0.16.154[55221]",
   "10.0.16.155[55221]"
 ]
}
```

- GET `/admin/instance_counts` : Displays the number of available service instances associated with the broker

```
curl 10.0.16.153:8080/admin/instance_counts
```

```
[
  {
    "count": 3,
    "status": "AVAILABLE"
  }
]
```

- GET `/admin/cluster_config/{service instance GUID}` : Retrieves the cluster config as a zip file which contains cluster.xml and cluster.properties

```
curl 10.0.16.153:8080/admin/cluster_config/47574053-7050-4911-9277-5725b27e1e38 > cluster_config.zip
```

- POST `/admin/cluster_config/{service instance GUID}` : Updates the cluster config of the specified service instance

Options:

```
{
  'reset-defaults': bool,
  'clear-logs': bool,
  'clear-disk-stores': bool,
  'cluster_config': zip file (cluster/cluster.xml, cluster/cluster.properties and any custom code to be deployed on GemFire server),
  'cluster_properties': file, unsure
}
```

```
curl -v -F reset-defaults=true 10.0.16.153:8080/admin/cluster_config/47574053-7050-4911-9277-5725b27e1e38
```

- GET `/admin/archives/{service instance GUID}` : Retrieves the log files of the cluster as a zip file

```
curl 10.0.16.153:8080/admin/archives/47574053-7050-4911-9277-5725b27e1e38 > archives.zip`
```

- DELETE `/v2/service_instances/{service instance GUID}` : Deletes a service instance.

```
curl -v -X DELETE http://10.0.18.151:8080/v2/service_instances/608363f1-4811-480d-bd95-b2ad9833cac5
```

Uninstalling GemFire for Pivotal Cloud Foundry

- Prerequisites
- Uninstalling

Prerequisites

Before you delete your **GemFire for Pivotal Cloud Foundry** tile, please ensure all GemFire service instances have been deleted. If you fail to delete all GemFire service instances before deleting the tile, it may cause issues if you ever wish to re-install the GemFire tile.

To check if there are any GemFire service instances present in your Pivotal Cloud Foundry deployment, you can run the simple shell script shown below:

```
#!/usr/bin/env bash

for org in $(cf orgs | awk 'NR>3') ; do
  cf target -o ${org} > /dev/null
  for space in $(cf spaces | awk 'NR>3') ; do
    cf target -s ${space} > /dev/null
    cf services | grep p-gemfire
  done
done
```

Uninstalling

See the following instructions for uninstalling Pivotal Cloud Foundry tiles.