

VMware GemFire for TAS Documentation 1.9

VMware GemFire for TAS 1.9

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2023 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

About Pivotal Cloud Cache	12
Product Snapshot	12
Pivotal Cloud Cache and Other Pivotal Cloud Foundry Services	14
Pivotal Cloud Cache Architecture	15
Pivotal GemFire® Basics	15
The PCC Cluster	15
Member Communication	17
Recommended Usage and Limitations	18
Limitations	18
Security	19
TLS Encryption for the PCC Service Instance	19
Networking	19
Security within the GemFire Cluster	20
Pivotal Cloud Cache Operator Guide	21
Preparing for Installation	21
Networking for On-Demand Services	22
Service Network Requirement	22
Default Network and Service Network	22
PCC Instances Across WAN	25
Preparing for Transport Layer Security (TLS)	25
Overview	25
Provide or Generate a CA Certificate	26
Add the CA Certificate	26
Create a User Account and Authentication (UAA) Client	28
Installing and Configuring Pivotal Cloud Cache	28
Configure Tile Properties	28

Assign Availability Zones and Networks	30
Settings	30
Settings: Smoke Test Plan	30
Settings: Allow Outbound Internet Access Settings	31
Settings: Default Distributed System ID	31
Configure Service Plans	32
Configure a Small Footprint Plan	34
Configure a Dev Plan	37
Syslog	39
Service Instance Upgrades	39
Security	40
Errands	41
VM Memory Allocation	41
Memory Allocation for Colocated Plans	41
Memory Allocation for Non-colocated Plans	42
Configuring User Account and Authentication (UAA) Roles	42
The Roles	42
Configuring the Roles	43
Setting Service Instance Quotas	44
Create Global-level Quotas	44
Create Plan-level Quotas	45
Create and Set Org-level Quotas	45
Create and Set Space-level Quotas	45
View Current Org and Space-level Quotas	46
Monitor Quota Use and Service Instance Count	46
Calculate Resource Costs for On-Demand Plans	47
Calculate Maximum Resource Cost Per On-Demand Plan	48
Calculate Maximum Resource Cost for All On-Demand Plans	49
Calculate Actual Resource Cost of all On-Demand Plans	49
Upgrading Pivotal Cloud Cache	49
Enable Individual Service Instance Upgrades	50
Updating Pivotal Cloud Cache Plans	50
Uninstalling Pivotal Cloud Cache	51

Backing Up and Restoring Service Instances	51
Before You Begin	51
Backing Up a PCC Service Instance	52
Restoring a PCC Service Instance	53
Enable Service-Instance Sharing	54
Rotating Certificates	54
WAN-Connected Service Instances	55
Generate New TLS CA Certificates	55
Collect All the Certificates	56
Add All the Certificates to the Tiles	57
Apply Changes for the First Time	58
Set New Services TLS CA Certificate	58
Apply Changes for the Second Time	59
Remove the Old Services TLS CA Certificates	60
Apply Changes for the Third Time	60
Restoring a WAN Connection	61
Working with Service Instances	69
View Available Plans	70
Create or Delete a Service Instance	71
Create a Service Instance	71
Provide Optional Parameters	71
Enable Session State Caching with the Java Buildpack	72
Enable Session State Caching Using Spring Session	72
Dev Plans	73
Delete a Service Instance	73
Upgrade a Single Service Instance	73
Updating a Pivotal Cloud Cache Service Instance	74
Rebalancing a Cluster	74
Restarting a Cluster	75
Changes to the Service Plan	75

Monitoring Pivotal Cloud Cache Service Instances	76
Service Instance Metrics	76
Member Count	76
Total Available Heap Size	77
Total Used Heap Size	77
Total Available Heap Size as a Percentage	77
Per Member Metrics	78
Memory Used as a Percentage	78
Count of Java Garbage Collections	78
CPU Utilization Percentage	78
Average Latency of Get Operations	79
Average Latency of Put Operations	79
JVM pauses	80
File Descriptor Limit	80
Open File Descriptors	80
Quantity of Remaining File Descriptors	81
Threads Waiting for a Reply	81
Gateway Sender and Gateway Receiver Metrics	82
Queue Size for the Gateway Sender	82
Events Received at the Gateway Sender	82
Events Queued by the Gateway Sender	82
Events Received by the Gateway Receiver	82
Disk Metrics	82
Average Latency of Disk Writes	83
Quantity of Bytes on Disk	83
Quantity of Available Bytes on Disk	83
Experimental Metrics	83
Total Memory Consumption	84
Service-Instance Sharing	85
Share a Service Instance	86
Bind an App to a Shared Service Instance	86
App Authentication	87
Set Up Service Instances Across a Wide Area Network (WAN)	87
Establishing Mutually Trusted TLS Credentials	87
Assumptions	88

Establish Mutual Trust	88
Set Up a Bidirectional System	89
Assumptions	89
Create Clusters	89
Create Service Keys	90
Establish a Bidirectional Connection	92
Create Gateway Senders and Regions	96
Verify Bidirectional WAN Setup	97
Set Up a Unidirectional System with TLS	98
Assumptions	99
Create Clusters	99
Create Service Keys	99
Establish a Unidirectional TLS Connection	102
Create a Gateway Sender and Regions	106
Verify Unidirectional WAN Setup	107
Set Up an Additional Bidirectional Interaction	108
Assumptions	108
Get Cluster A's Remote Credentials	108
Create the New Cluster	110
Create Gateway Senders and Regions	115
Set Up an Additional Unidirectional Interaction	116
Assumptions	116
Get Cluster A's Remote Credentials	116
Create the New Cluster	118
Create Gateway Senders and Regions	122
Setting Up Servers for an Inline Cache	123
Implement a Cache Loader for Read Misses	123
Implement an Asynchronous Event Queue and Cache Listener for Write Behind	124
Implement a Cache Writer for Write Through	125
Configure Using gfsh for Write Behind	126
Configure Using gfsh for Write Through	127
Accessing a Service Instance	127
Create a Service Key	128

Connect with gfsh over HTTPS	131
Create a Truststore	131
Establish the Connection with HTTPS	132
Establish the Connection with HTTPS in a Development Environment	133
Determine Your TLS Termination	133
Using gfsh	134
gfsh Command Restrictions	134
Do Not Export from a GemFire Cluster to a Cloud Cache Cluster	135
Create Regions	135
Working with Disk Stores	136
Create a Disk Store	136
Destroy a Disk Store	136
Use the Pulse Dashboard	136
Access Service Instance Metrics	137
Service Instance (Cluster-wide) Metrics	137
Member (per-VM) Metrics	137
Access Service Broker Metrics	137
Export gfsh Logs	138
Deploy an App JAR File to the Servers	138
Use the GemFire-Greenplum Connector	139
Scripting Common Operations	139
Running a gfsh Script	139
Example Scripts	139
Connecting a Spring Boot App to Pivotal Cloud Cache with Session State Caching	140
Use the Tomcat App	140
Use a Spring Session Data GemFire App	141
Upgrade PCC and Spring Session Data GemFire	141
Creating Continuous Queries Using Spring Data GemFire	142
Application Development	144
Design Patterns	145
The Inline Cache	145
The Cache-Aside Cache	145
Bidirectional Replication Across a WAN	147

Blue-Green Disaster Recovery	147
CQRS Pattern Across a WAN	148
Hub-and-Spoke Topology with WAN Replication	149
Follow-the-Sun Pattern	150
Region Design	152
Keys	152
Partitioned Regions	152
Partitioned Region Types for Creating Regions on the Server	154
Replicated Regions	154
Replicated Region Types for Creating Regions on the Server	155
Persistence	155
Overflow	156
Regions as Used by the App	156
An Example to Demonstrate Region Design	157
Handling Events	157
Continuous Queries	157
Example Applications	158
A Simple Java App	158
A Spring Boot App	159
Top Down Explanation	159
The App Controller	160
TLS-Enabled Cluster Demonstration	160
Continuous Queries in the App	161
Steps to Run an App	161
Java Build Pack Requirements	161
Bind an App to a Service Instance	161
Developing an App Under TLS	162
HTTP Session State Caching	162
Deactivate Near Caching Within the App	162
Deactivate Caching Using an External Repository for Configuration	162
Deactivate Caching Using a Custom Java Buildpack	165
Troubleshooting	167

Acquire Artifacts for Troubleshooting	167
Gather GemFire Logs	167
View Statistics Files and Logs	167
Acquire Thread Dumps	167
Acquire the Deployment Name	168
Troubleshooting for Operators	169
Smoke Test Failures	169
General Connectivity	169
Troubleshooting for Developers	170
 Pivotal Cloud Cache Release Notes	 171
v1.9.2	171
v1.9.1	171
v1.9.0	171

About Pivotal Cloud Cache

Pivotal Cloud Cache (PCC) is a high-performance, high-availability caching layer for Pivotal Cloud Foundry (PCF). PCC offers an in-memory key-value store. It delivers low-latency responses to a large number of concurrent data-access requests.

PCC provides a service broker to create in-memory data clusters on demand. These clusters are dedicated to the PCF space and tuned for specific use cases defined by your service plan. Service operators can create multiple plans to support different use cases.

PCC uses Pivotal GemFire. The [Pivotal GemFire API Documentation](#) details the API for client access to data objects within Pivotal GemFire. PCC supports writing JavaScript apps with the [Cloud Cache Node.js Client](#).

This documentation performs the following functions:

- Describes the features and architecture of PCC
- Provides the PCF operator with instructions for installing, configuring, and maintaining PCC
- Provides app developers instructions for choosing a service plan and creating and deleting PCC service instances
- Provides app developers instructions for binding apps

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Product Snapshot

The following table provides version and version-support information about Pivotal Cloud Cache:

Element	Details
Version	1.9.2
Release date	January 31, 2020
Software component version	GemFire v9.8.6
Minimum Pivotal GemFire Native Client version	v10.0.3
Compatible Ops Manager version(s)	v2.6, v2.7, v2.10
Compatible Pivotal Application Service (PAS) version(s)	v2.6.x, v2.7.x

IaaS support	AWS, Azure, GCP, OpenStack, vSphere
IPsec support	Yes
Required BOSH stemcell version	Xenial v621.41 or a more recent version
Minimum Java buildpack version required for apps	v4.18 A v4.36 Java buildpack requires customization to work with an app that will be bound to a session replication service. For more information about customization, see Tomcat Session State Caching .

The following table identifies the Apache Geode or Tanzu GemFire version incorporated into the Tanzu GemFire for VMs version.

Tanzu GemFire for VMs version	Tanzu GemFire version	Apache Geode version
1.14.5	v1.15.1	
1.14.4		v1.14.4
1.14.3		v1.14.3
1.14.2		v1.14.2
1.14.1*		v1.14.1
1.14.0*		v1.14.0
1.13.7		v1.13.8
1.13.6		v1.13.7
1.13.5		v1.13.6
1.13.4*		v1.13.5
1.13.3*		v1.13.4
1.13.2*		v1.13.3
1.13.1*		v1.13.2
1.13.0*		v1.13.1
1.13.0 Beta*		Build 312 for v1.13
1.12.4	9.10.13	
1.12.3	9.10.8	
1.12.2	9.10.6	
1.12.1	9.10.5	
1.12.0	9.10.2	
1.11.3	9.9.5	
1.11.2	9.9.3	
1.11.1	9.9.2	

1.11.0	9.9.1
1.10.9	9.9.7
1.10.8	9.9.6
1.10.7	9.9.6
1.10.6*	9.9.6
1.10.5*	9.9.5
1.10.4*	9.9.4
1.10.3*	9.9.3
1.10.2*	9.9.2
1.10.1*	9.9.1
1.10.0*	9.9.0
1.9.2	9.8.6
1.9.1	9.8.4
1.9.0*	9.8.3

* This version is not available on [VMware Tanzu Network](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Pivotal Cloud Cache and Other Pivotal Cloud Foundry Services

As well as Pivotal Cloud Cache, other services offer *on-demand* service plans. These plans let developers provision service instances when they want.

These contrast with the older *pre-provisioned* service plans, which require operators to provision the service instances during installation and configuration through the service tile UI.

The following table lists which service tiles offer on-demand and pre-provisioned service plans:

Service tile	Standalone product related to the service	Supports on-demand	Supports pre-provisioned
RabbitMQ for PCF	Pivotal RabbitMQ	Yes	Yes. Only recommended for test environments.
Redis for PCF	Redis	Yes	Yes (shared-VM plan). Only recommended for test environments.
MySQL for PCF	MySQL	Yes	No
Pivotal Cloud Cache	Pivotal GemFire	Yes	No

For services that offer both on-demand and pre-provisioned plans, you can choose the plan you want to use when configuring the tile.

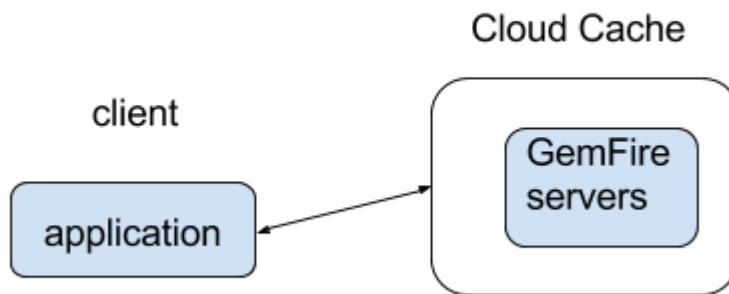
[Create a pull request or raise an issue on the source for this page in GitHub](#)

Pivotal Cloud Cache Architecture

Pivotal GemFire® Basics

Pivotal GemFire is the data store within Pivotal Cloud Cache (PCC). A PCC service instance requires a small amount of administrative GemFire setup, and any app will use a limited portion of the GemFire API.

The PCC architectural model is a client-server model. The clients are apps or microservices, and the servers are a set of GemFire servers maintained by a PCC service instance. The GemFire servers provide a low-latency, consistent, fault-tolerant data store within PCC.



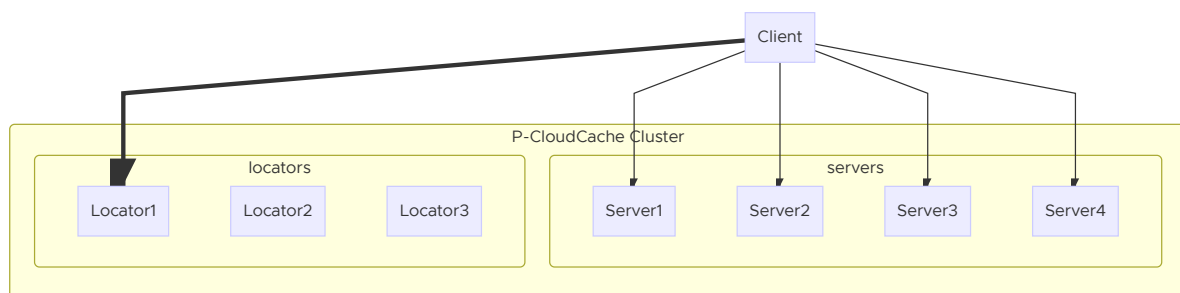
GemFire holds data in key/value pairs. Each pair is called an **entry**. Entries are logically grouped into sets called **regions**. A region is a map (or dictionary) data structure.

The app (client) uses PCC as a cache. A cache lookup (read) is a get operation on a GemFire region. The cache operation of a cache write is a put operation on a GemFire region. The GemFire command-line interface, called gfsh, facilitates region administration. Use gfsh to create and destroy regions within the PCC service instance.

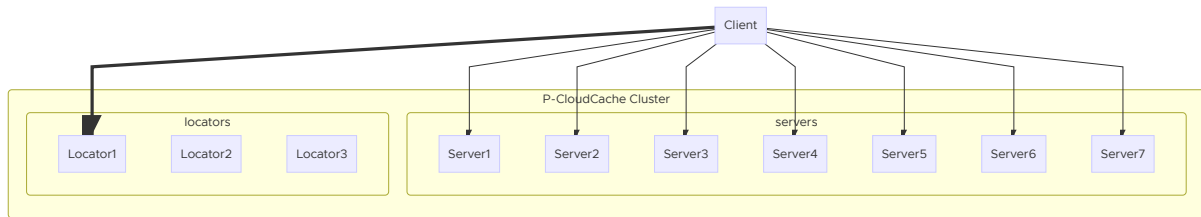
The PCC Cluster

PCC deploys cache clusters that use Pivotal GemFire to provide high availability, replication guarantees, and eventual consistency.

When you first spin up a cluster, you have three locators and at least four servers.

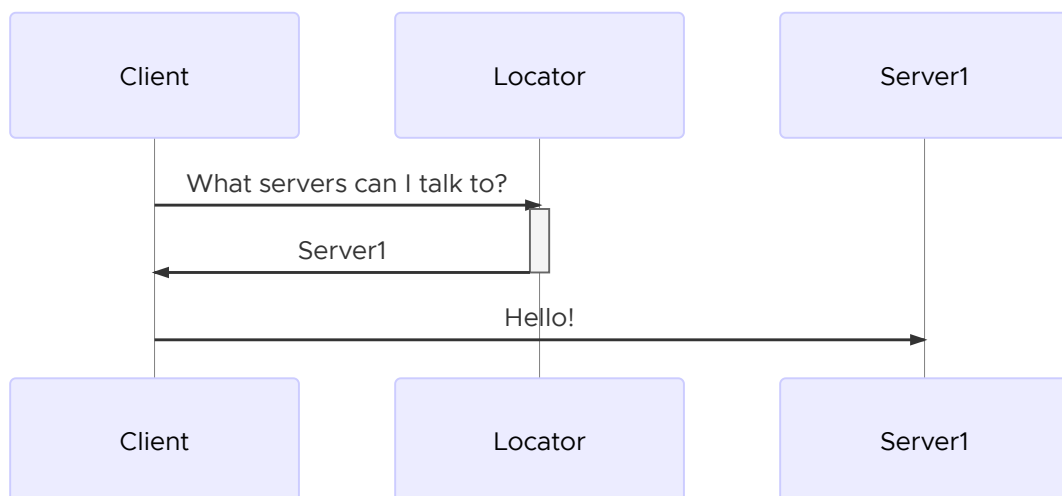


When you scale up the cluster, you have more servers, increasing the capacity of the cache. There are always three locators.

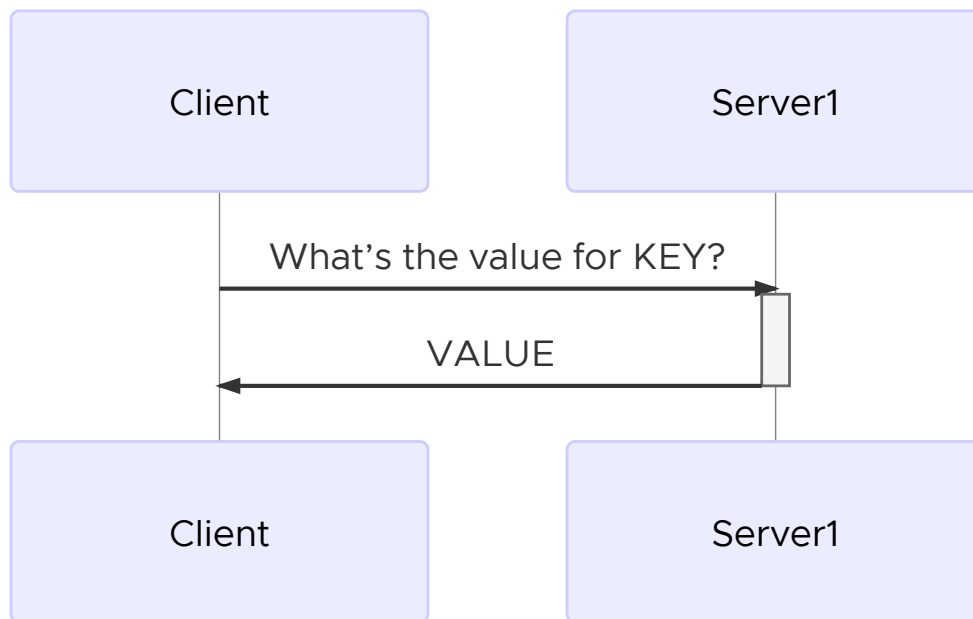


Member Communication

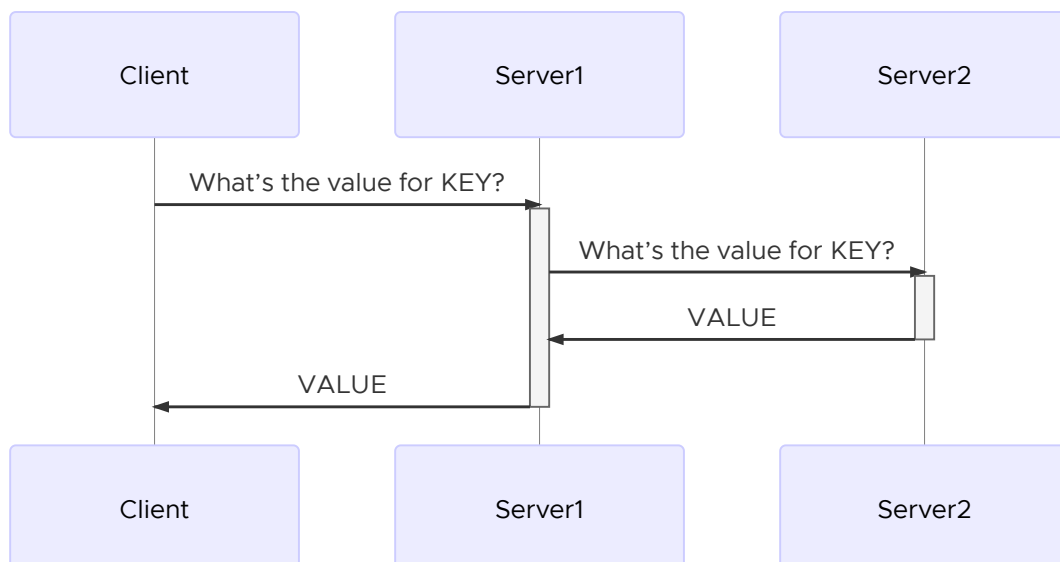
When a client connects to the cluster, it first connects to a locator. The locator replies with the IP address of a server for it to talk to. The client then connects to that server.



When the client wants to read or write data, it sends a request directly to the server.



If the server doesn't have the data locally, it fetches it from another server.



[Create a pull request or raise an issue on the source for this page in GitHub](#)

Recommended Usage and Limitations

- See [Design Patterns](#) for descriptions of the variety of design patterns that Pivotal Cloud Cache supports.
- Pivotal Cloud Cache stores objects in key/value format, where the value can be any object.
- See [gfsH Command Restrictions](#) for limitations on the use of gfsH commands.

Limitations

- Cloud Cache does not support scaling down of the cluster.
- Pivotal Cloud Cache does not support plan migrations. The `-p` option to the `cf update-service` command is not supported.

- Pivotal Cloud Cache does not publish locator metrics for service instances created as Co-located Single VM plans or Co-located Multi VM plans.

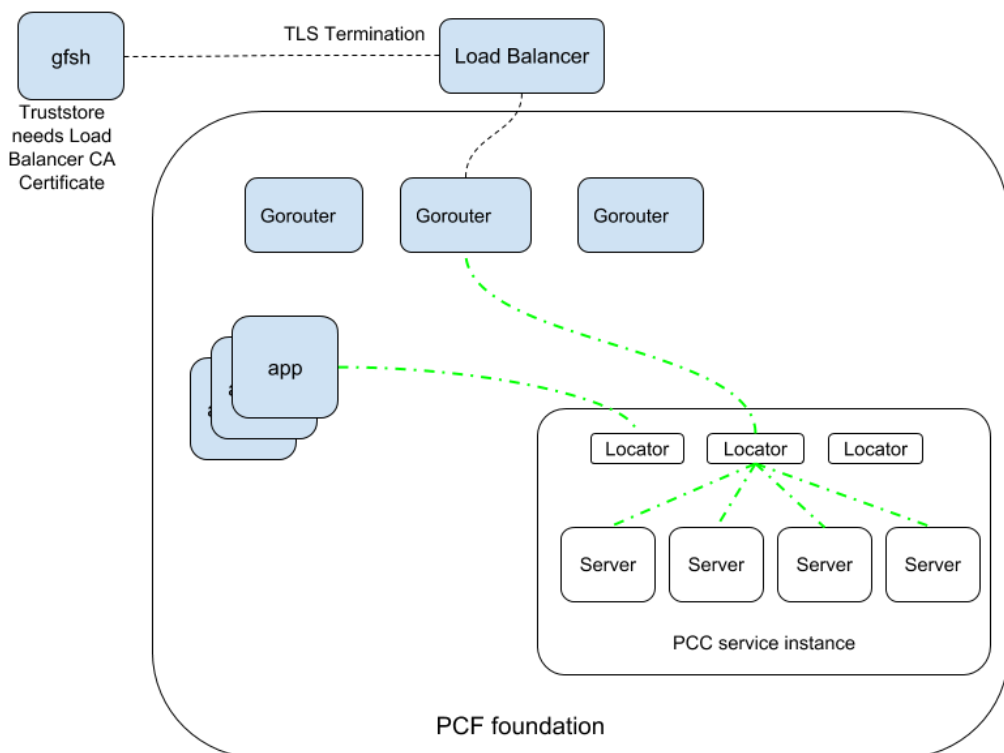
[Create a pull request or raise an issue on the source for this page in GitHub](#)

Security

The security measures implemented for a Pivotal Cloud Foundry (PCF) foundation and for Pivotal Cloud Cache (PCC) service instances within that foundation attempt to reduce harm from agents with foundation access. See [Cloud Foundry Security](#) for a general presentation on PCF security. Transport-Layer Security (TLS) encryption prevents easy access to communication between components, and role-based authentication and authorization limits access to data.

TLS Encryption for the PCC Service Instance

Without TLS encryption with and within the PCC service instance, The diagram below identifies via green dotted-and-dashed lines the unencrypted, plaintext communication that a bad agent with PCF foundation access could listen to without TLS encryption.



Each of these unencrypted communication paths may be TLS-encrypted. Enabling TLS encryption implements a one-way authentication of apps, verifying the identity of GemFire cluster members.

You must also secure gfsH communication. Follow directions in [Connect with gfsH over HTTPS](#).

Networking

To allow app access to the PCC network, create application security groups. Allow access on the following ports:

- 1099
- 8080
- 40404
- 55221

For more information, see Pivotal Platform documentation on [Restricting App Access to Internal PAS Components](#).

PCC works with the IPsec Add-on for PCF (see [Securing Data in Transit with the IPsec Add-on](#)).

Security within the GemFire Cluster

The GemFire cluster within a PCC service instance implements role-based authentication and authorizes cluster operations based upon the roles.

There are two sets of roles:

- One set has four roles for users that integrate an authentication and enterprise single-sign-on (SSO) system, such as LDAP. See [Configuring UAA Roles](#) for a description of the roles and the configuration that completes the integration.
- The other set of roles defaults when there is no authentication and enterprise SSO system integrated during the PCC tile installation. The identifiers assigned for these roles are detailed in [Create Service Keys](#). PCC service instances are created with three default GemFire user roles for interacting with clusters:
 - **Cluster operator:** manages the GemFire cluster and can access region data. Has the permissions `CLUSTER:MANAGE`, `CLUSTER:WRITE`, `CLUSTER:READ`, `DATA:MANAGE`, `DATA:WRITE`, and `DATA:READ`.
 - **Developer:** can access region data. Has the permissions `CLUSTER:READ`, `DATA:WRITE`, and `DATA:READ`.
 - **Gateway sender:** propagates region data to another PCC service instance. Has the permission `DATA:WRITE`.

Which set is used for a PCC service instance depends on the options chosen during PCC tile installation.

All gfsh and JMX clients must authenticate as one of these user roles to access the cluster. To authorize operations, each user role is given predefined permissions for cluster operations. To accomplish a cluster operation, the user authenticates using one of the roles. Prior to initiating the requested operation, there is a verification that the authenticated user role has permission to do the operation. Read more about these permissions in [GemFire: Implementing Authorization](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Pivotal Cloud Cache Operator Guide

This document describes how a Pivotal Cloud Foundry (PCF) operator can install, configure, and maintain Pivotal Cloud Cache (PCC).

In this topic:

- [Preparing for Installation](#)
 - [Networking for On-Demand-Services](#)
 - [Preparing for TLS](#)
 - [Create a UAA Client](#)
- [Installing and Configuring Pivotal Cloud Cache](#)
 - [Configure Tile Properties](#)
- [Configuring User Account and Authentication \(UAA\) Roles](#)
- [Setting Service Instance Quotas](#)
- [Upgrading Pivotal Cloud Cache](#)
 - [Enable Individual Service Instance Upgrades](#)
- [Updating Pivotal Cloud Cache Plans](#)
- [Uninstalling Pivotal Cloud Cache](#)
- [Backing Up and Restoring Service Instances](#)
- [Enable Service Instance Sharing](#)
- [Managing Certificates](#)
- [Restoring a WAN Connection](#)

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Preparing for Installation

Work through these items prior to PCC tile installation.

- The [Networking for On-Demand Services](#) section describes networking requirements for PCC.
- PCC requires TLS encryption for using gfsh and Pulse. Follow the instructions in [Preparing for TLS](#) to set up TLS encryption.
- For systems that implement an authentication and enterprise SSO system such as LDAP, follow the instructions within [Create a UAA Client](#). There are also additional configuration steps to set up the SSO system, as given within [Configuring UAA Roles](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Networking for On-Demand Services

This section describes networking considerations for Pivotal Cloud Cache.

Service Network Requirement

When you deploy Pivotal Application Service (PAS), you must create a statically defined network to host the component VMs that make up the infrastructure. Components, such as Cloud Controller and UAA, run on this infrastructure network.

On-demand services might require you to host them on a separate network from the default network. You can also deploy on-demand services on a separate service networks to meet your own security requirements.

PAS supports dynamic networking. Operators can use dynamic networking with asynchronous service provisioning to define dynamically-provisioned service networks. For more information, see [Default Network and Service Network](#) below.

On-demand services are enabled by default on all networks. Operators can optionally create separate networks to host services in BOSH Director. Operators can select which network hosts on-demand service instances when they configure the tile for that service.

Default Network and Service Network

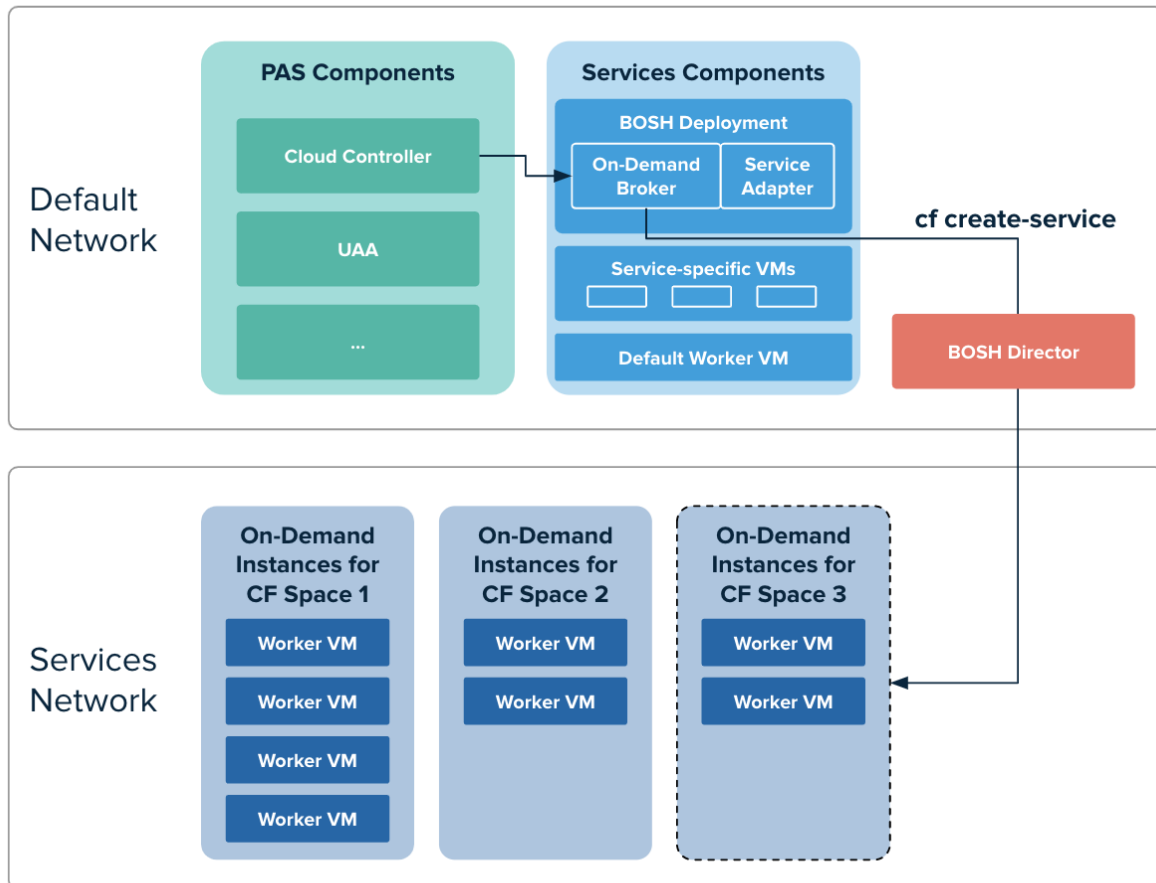
On-demand Pivotal Cloud Cache services use BOSH to dynamically deploy VMs and create single-tenant service instances in a dedicated network. On-demand services use the dynamically-provisioned service network to host single-tenant worker VMs. These worker VMs run as service instances within development spaces.

This on-demand architecture has the following advantages:

- Developers can provision IaaS resources for their services instances when the instances are created. This removes the need for operators to pre-provision a fixed amount of IaaS resources when they deploy the service broker.
- Service instances run on a dedicated VM and do not share VMs with unrelated processes. This removes the “noisy neighbor” problem, where an app monopolizes resources on a shared cluster.
- Single-tenant services can support regulatory compliances where sensitive data must be separated across different machines.

An on-demand service separates operations between the default network and the service network. Shared service components, such as executive controllers and databases, Cloud Controller, UAA, and other on-demand components, run on the default network. Worker pools deployed to specific spaces run on the service network.

The diagram below shows worker VMs in an on-demand service instance running on a separate services network, while other components run on the default network.



[View a larger version of this image](#)

Required Networking Rules for On-Demand Services

Before deploying a service tile that uses the on-demand service broker (ODB), you must create networking rules to enable components to communicate with ODB. For instructions for creating networking rules, see the documentation for your IaaS.

The following table lists key components and their responsibilities in the on-demand architecture.

Key Components	Component Responsibilities
BOSH Director	Creates and updates service instances as instructed by ODB.
BOSH Agent	Adds an agent on every VM that it deploys. The agent listens for instructions from the BOSH Director and executes those instructions. The agent receives job specifications from the BOSH Director and uses them to assign a role or job to the VM.
BOSH UAA	Issues OAuth2 tokens for clients to use when they act on behalf of BOSH users.
Pivotal Application Service	Contains the apps that consume services.
ODB	Instructs BOSH to create and update services. Connects to services to create bindings.

Deployed service instance	Runs the given service. For example, a deployed Pivotal Cloud Cache service instance runs the Pivotal Cloud Cache service.
----------------------------------	--

Regardless of the specific network layout, the operator must ensure network rules are set up so that connections are open as described in the table below.

This component...	Must communicate with...	Default TCP Port	Communication direction (s)	Notes
PCC cluster members	PCC cluster members	49152-65535	Two-way	Inclusive range. PCC servers and locators communicate with each other using UDP and TCP.
PCC Service Instance 1	PCC Service Instance 2	5000-5499	Two-way	Inclusive range. Gateway receivers and gateway senders communicate across WAN-separated service instances. Each PCC service instance uses GemFire defaults for the gateway receiver ports.
ODB	<ul style="list-style-type: none"> BOSH Director BOSH UAA 	<ul style="list-style-type: none"> 25555 (BOSH Director) 8443 (UAA) 8844 (CredHub) 	One-way	The BOSH Director and BOSH UAA default ports are not configurable. The CredHub default port is configurable.
ODB	PAS	8443	One-way	The default port is not configurable.
Errand VMs	<ul style="list-style-type: none"> PAS ODB 	<ul style="list-style-type: none"> 443 8080 	One-way	The default port is not configurable.
BOSH Agent	BOSH Director	4222	Two-way	The BOSH Agent runs on every VM in the system, including the BOSH Director VM. The BOSH Agent initiates the connection with the BOSH Director. The default port is not configurable.
Deployed apps on PAS	Deployed service instances	<ul style="list-style-type: none"> 55221 (GemFire locators) 40404 (GemFire servers) 	Two-way	These port numbers are not configurable.

PAS	<ul style="list-style-type: none"> • ODB • Deployed service instances 	8080	One-way	PAS communicates with service instances because the Gorouter proxies gfsH requests to GemFire clusters.
PCC	PCC	1053	Two-way	Allows DNS resolution for clusters communicating across a WAN-connected system.

PCC Instances Across WAN

PCC service instances running within distinct PCF foundations may communicate with each other across a WAN. In a topology such as this, the members within one service instance use their own private address space, as defined in [RFC1918](#).

A VPN may be used to connect the private network spaces that lay across the WAN. The steps required to enable the connectivity by VPN are dependent on the IaaS provider(s).

The private address space for each service instance's network must be configured with non-overlapping CIDR blocks. Configure the network prior to creating service instances. Locate directions for creating a network on the appropriate IaaS provider within the section titled [Architecture and Installation Overview](#).

Open port 1053 to allow DNS resolution of other WAN-connected clusters.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Preparing for Transport Layer Security (TLS)

This topic describes how to provide an existing Certificate Authority (CA) certificate to [BOSH CredHub](#) and how to generate a new CA certificate with BOSH CredHub, if you do not already have one.



WARNING: This procedure involves restarting all of the VMs in an existing PCF deployment in order to propagate a CA certificate. The operation can take a long time to complete.

Overview

Enabling TLS provisions PCC service instances with a certificate, so that apps, gfsH, and Pulse can establish encrypted connections with the PCC service instance.

The certificate deployed on the PCC service instance is a **server certificate**. The server certificate is generated by **CredHub**, a component designed for centralized credential management in PCF. **CredHub** is deployed on the same VM as the BOSH Director.

CredHub generates the server certificate using a **Certificate Authority (CA) certificate**. The CA certificate must be provided to CredHub by the operator or generated by CredHub.

Apps use the CA certificate to authenticate components of PCC service instances. Apps that communicate with PCC must have access to the CA certificate in order to validate that the server certificate can be trusted.



WARNING: An operator must rotate the CA certificate if it expires or if it becomes compromised. To rotate your CA certificate, see [Managing Certificates](#). Do not attempt to rotate a CA certificate on your own. Contact [Pivotal Support](#) and perform the procedure with their assistance.

Provide or Generate a CA Certificate

TLS authorization requires a credential generated by CredHub. You do not need to create a new User Account and Authentication (UAA) client for CredHub specifically to support TLS, as long as a UAA Client exists with `credhub.write` and `credhub.read` permissions. The client used in this section is one that was created during the OpsManager installation process: the `ops_manager` client.

Add the CA Certificate

Perform the following steps to log in to CredHub, provide or generate a CA certificate, and add the certificate to Ops Manager:

1. From the Ops Manager VM, set the API target of the CredHub CLI to your CredHub server.

Run the following command:

```
credhub api https://BOSH-DIRECTOR:8844 --ca-cert=/var/tempest/workspaces/default/root_ca_certificate
```

where `BOSH-DIRECTOR` is the IP address of the BOSH Director VM.

For example:

```
$ credhub api https://10.0.0.5:8844 --ca-cert=/var/tempest/workspaces/default/root_ca_certificate
```

2. Log in to CredHub.

Run the following command:

```
credhub login --client-name=CLIENT-NAME --client-secret=CLIENT-SECRET
```

where `CLIENT-NAME` is the client name, usually `ops_manager` or a CredHub-specific UAA client of your own creation, and `CLIENT-SECRET` is the client secret which can be found under the Credentials tab of the OpsManager tile with the name `Bosh Commandline Credentials`.

For example:

```
$ credhub login \
  --client-name=ops_manager \
  --client-secret=abcdefghijklm123456789
```

3. Use the CredHub CLI to check whether a services CA certificate already is present.

- o Enter the following command:

```
$ credhub get \
  --name="/services/tls_ca"
```

- o If you already have a certificate at the `services/tls_ca` path, skip to step 5.

4. Use the CredHub CLI to generate a CA certificate or provide an existing one.



Note: Your PCF deployment may have multiple CA certificates. Pivotal recommends a dedicated CA certificate for services.

- o If you do not have a CA certificate, use the CredHub CLI to generate one. Enter the following command:

```
$ credhub generate \
  --name="/services/tls_ca" \
  --type="certificate" \
  --no-overwrite \
  --is-ca \
  --common-name="rootCA"
```

- o If you have an existing CA certificate that you want to use, create a new file called `root.pem` with the contents of the certificate. Then enter the following command, specifying the path to `root.pem` and the private key for the certificate:

```
$ credhub set \
  --name="/services/tls_ca" \
  --type="certificate" \
  --certificate=./root.pem \
  --private=ERKSOSMFF...
```

5. Use the BOSH CLI v2 to extract the `certificate` portion from the CA certificate and print it. Enter the following command:

```
$ bosh2 interpolate <(credhub get --name=/services/tls_ca) \
  --path=/value/certificate
```

6. Record the output of the `bosh2 interpolate` command from step 5.
7. Navigate to the Ops Manager **Installation Dashboard** and select the Ops Manager Director tile. Click **Security**.

8. Paste the contents of the CA certificate into **Trusted Certificates**. Append to existing **Trusted Certificates**, if there are already certificates listed. Click **Save**.
9. The CA certificate must also be added for the Gorouter. Navigate to the PAS **Settings** tab. Click on **Networking**. Add the CA certificate to the box labeled **Certificate Authorities Trusted by Router and HAProxy** and click **Save**.
10. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#)).
11. Click **Apply Changes**.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Create a User Account and Authentication (UAA) Client

Extra configuration is required when defining users that have specific security roles with an authentication and enterprise single sign-on (SSO) such as LDAP.

Authenticating using an authentication and enterprise SSO system such as LDAP uses the User Account and Authentication (UAA) server. Access the UAA server through its command-line interface, UAAC.

When enabling the use of an authentication and enterprise single sign-on (SSO) such as LDAP during PCC tile configuration, you will specify a UAA client, as described in [Security](#). This procedure creates that UAA client.

Create a UAA client for use in tile configuration with the command:

```
uaac client add cloudcache_gfsh --scope="PCC_*" --secret="THE-SECRET"
--authorized_grant_types=password
```

Replace **THE-SECRET** with your chosen password (secret) for this UAA client. The remainder of the command is exactly as shown. The UAA client name is **cloudcache_gfsh**.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

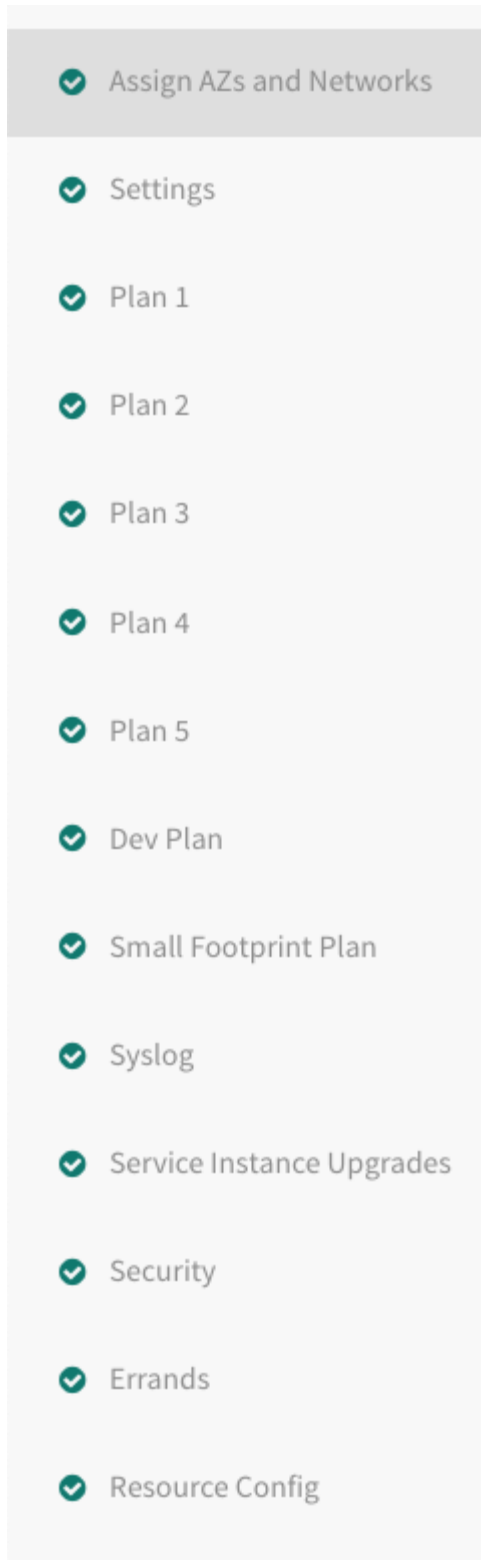
Installing and Configuring Pivotal Cloud Cache

With an Ops Manager role (detailed in [Understand Roles in Ops Manager](#)) that has the proper permissions to install and configure, follow these steps to install PCC on PCF:

1. Download the tile from [Tanzu Network](#).
2. Click **Import a Product** to import the tile into Ops Manager.
3. Click the **+** symbol next to the uploaded product description.
4. Click on the Cloud Cache tile.
5. Complete all the configuration steps in the [Configure Tile Properties](#) section below.
6. Return to the Ops Manager Installation Dashboard. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#)).
7. Click **Apply Changes** to complete the installation of the PCC tile.

Configure Tile Properties

Configure the sections listed on the left side of the page.



As you complete a section, save it. A green check mark appears next to the section name. Each section name must show this green check mark before you can complete your installation.

- [Assign AZs and Networks](#)
- [Settings](#)
- [Service Plans](#), including the Dev Plan and the Small Footprint Plan

- [Syslog](#)
- [Service Instance Upgrades](#)
- [Security](#)
- [Errands](#)

Assign Availability Zones and Networks

To select AZs and networks for VMs used by PCC, do the following:

1. Click **Assign AZs and Networks**.
2. Configure the fields on the **Assign AZs and Networks** pane as follows:

Field	Instructions
Place singleton jobs in	Select the region that you want for singleton VMs.
Balance other jobs in	Select the AZ(s) you want to use for distributing other GemFire VMs. Pivotal recommends selecting all of them.
Network	Select your PAS (or Elastic Runtime) network.
Service Network	Select the network to be used for GemFire VMs.

3. Click **Save**.

Settings

Settings: Smoke Test Plan

The smoke-tests errand that runs after tile installation. The errand verifies that your installation was successful. By default, the `smoke-test` errand runs on the `system` org and the `p-cloudcache-smoke-test` space.



Note: Smoke tests will fail unless you enable global default application security groups (ASGs). You can enable global default ASGs by binding the ASG to the `system` org without specifying a space. To enable global default ASGs, use `cf bind-running-security-group`.

To select which plan you want to use for smoke tests, do the following:

Select a plan to use when the `smoke-tests` errand runs.

Ensure the selected plan is enabled and configured. For information about configuring plans, see [Configure Service Plans](#) below. If the selected plan is not enabled, the `smoke-tests` errand fails.

Pivotal recommends that you use the smallest four-server plan for smoke tests. Because smoke tests create and later destroy this plan, using a very small plan reduces installation time.

Configure properties for Pivotal Cloud Cache tile

Plan to use for smoke test*

☒ Plan 1
☐ Plan 2
☐ Plan 3
☐ Plan 4
☐ Plan 5
☐ Dev Plan
☐ Small Footprint Plan

The selected plan will be used to run a smoke test to verify the tile works correctly after installation. You must make sure this plan is enabled, or the errand will fail.

Settings: Allow Outbound Internet Access Settings

By default, outbound internet access is not allowed from service instances.

If BOSH is configured to use an external blob store, you need allow outbound internet access from service instances. Log forwarding and backups, which require external endpoints, might also require internet access.

To allow outbound internet access from service instance, do the following:

Select **Allow outbound internet access from service instances (IaaS-dependent)**.

VM options

☐

Allow outbound internet access from service instances (IaaS-dependent)

Please refer to the Ops Manager documentation for IaaS specific behavior. Log forwarding and backups may require internet access.

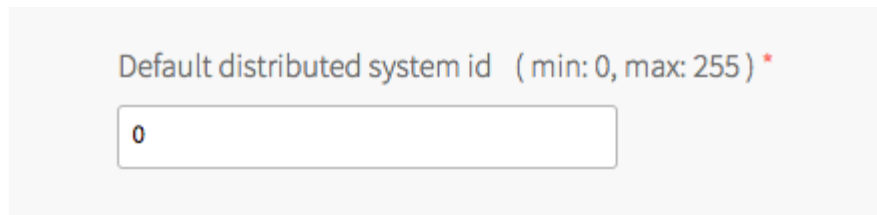


Note: Outbound network traffic rules also depend on your IaaS settings. Consult your network or IaaS administrator to ensure that your IaaS allows outbound traffic to the external networks you need.

Settings: Default Distributed System ID

Every service instance has an integer identifier called a distributed system ID. The ID defaults to the value 0. Service instances that form a distributed system that communicates across a WAN will need distinct IDs. Those distinct ID values are set when creating the service instance.

To change the default distributed system ID value, replace the default value of 0 with your new default value. Acceptable values are integers greater than or equal to 0 and less than or equal to 255.

A screenshot of a configuration interface. It features a label "Default distributed system id (min: 0, max: 255) *" in a dark grey font. Below the label is a text input field with a thin grey border, containing the number "0". The entire configuration area is set against a light grey background.

Default distributed system id (min: 0, max: 255) *

Configure Service Plans

You can configure five individual plans for your developers. Select the **Plan 1** through **Plan 5** tabs to configure each of them.

Service Plan Access*

- ☐ Plan Disabled
- ☒ Plan Enabled

Plan Name *

extra-small

Plan Description *

Plan Description

☐ Enable metrics for service instances

CF Service Access*

Enable Service Access

Maximum service instances *

10

Maximum servers per cluster (min: 4, max: 32) *

32

Default Number of Servers (min: 4, max: 32) *

4

Availability zones for service instances *

- ☒ us-central1-a
- ☒ us-central1-b
- ☒ us-central1-c

VM type for the Locator VMs*

medium (cpu: 2, ram: 4 GB, disk: 8 GB)

The **Plan Enabled** option is selected by default. If you do not want to add this plan to the CF service catalog, select **Plan Disabled**. You must enable at least one plan.

The **Plan Name** text field allows you to customize the name of the plan. This plan name is displayed to developers when they view the service in the Marketplace.

The **Plan Description** text field allows you to supply a plan description. The description is displayed to developers when they view the service in the Marketplace.

The **Enable metrics for service instances** checkbox enables metrics for service instances created using the plan. Once enabled, the metrics are sent to the Loggregator Firehose.

The **CF Service Access** drop-down menu gives you the option to display or not display the service plan in the Marketplace. **Enable Service Access** displays the service plan in the Marketplace. **Disable Service Access** makes the plan unavailable in the Marketplace. If you choose this option, you cannot make the plan available at a later time. **Leave Service Access Unchanged** makes the plan unavailable in the Marketplace by default, but allows you to make it available at a later time.

The **Maximum service instances** field sets the maximum number of PCC clusters that can exist simultaneously.

When developers create or update a service instance, they can specify the number of servers in the cluster. The **Maximum servers per cluster** field allows operators to set an upper bound on the number of servers developers can request. If developers do not explicitly specify the number of servers in a service instance, a new cluster has the number of servers specified in the **Default Number of Servers** field.

The **Availability zones for service instances** setting determines which AZs are used for a particular cluster. The members of a cluster are distributed evenly across AZs.



WARNING! After you've selected AZs for your service network, you cannot add additional AZs; doing so causes existing service instances to lose data on update.

The remaining fields control the VM type and persistent disk type for servers and locators. The total size of the cache is directly related to the number of servers and the amount of memory of the selected server VM type. We recommend the following configuration:

- For the **VM type for the Locator VMs** field, select a VM that has at least 2 CPUs, 1 GB of RAM and 4 GB of disk space.
- For the **Persistent disk type for the Locator VMs** field, select 10 GB or higher.
- For the **VM type for the Server VMs** field, select a VM that has at least 2 CPUs, 4 GB of RAM and 8 GB of disk space.
- For the **Persistent disk type for the server VMs** field, select 10 GB or higher.

For more information on how VM memory is allocated, see [VM Memory Allocation](#).

When you finish configuring the plan, click **Save** to save your configuration options.

Configure a Small Footprint Plan

The Small Footprint Plan uses three VMs to implement a default plan with three locators and three servers. Each of the three VMs has one locator and one server.

This plan is appropriate for production installations that can tolerate the diminished load-handling capacity and reduced resilience that would result from the loss of a VM. Installations needing resilience in the face of a VM loss should use a plan in which each PCC cluster member resides within its own VM.

A PCC service instance using the Small Footprint Plan may be created with more servers than the default three servers. Follow the instructions within [Creating a Pivotal Cloud Cache Service Instance](#) to increase the number of servers.

Scale up an existing service instance by increasing the number of servers, as specified in [Updating a Pivotal Cloud Cache Service Instance](#).

Whether creating a new service instance or updating an existing service instance to have more servers, each additional server will be within its own VM.

Scaling down the quantity of servers within an existing service instance is accomplished one server at a time to ensure redundancy is correctly maintained. To scale down, use the `cf update-service` command as described in [Updating a Pivotal Cloud Cache Service Instance](#) to specify one server fewer than the total quantity of servers currently running.

An error message will be issued if an attempt is made to scale down by more than one server at a time, or if an attempt is made to reduce the number of servers below the minimum size of the default Small Footprint Plan size of three servers.

To configure a Small Footprint Plan, enable the plan with the Small Footprint Plan tab.

Plan for small footprint development

Plan for small footprint development*

- ☐ Plan Disabled
- ☒ Plan Enabled

Plan Name *

small-footprint

Plan Description *

Plan Description

☐ Enable metrics for service instances

CF Service Access *

Enable Service Access

Maximum service instances *

10

Availability zones for service instances *

- ☐ us-central1-b
- ☐ us-central1-f
- ☐ us-central1-c

VM type for the locator-server VM *

Automatic: toolsmiths.custom-1-4.32 (cpu: 1, ram: 4 GB, disk: 32 GB)

Persistent disk type for the locator-server VM *

Automatic: 10 GB

Save

Configure a Dev Plan

A Dev Plan is a type of service plan. Use a Dev Plan for development and testing. The plan provides a single locator and server, which are colocated within a single VM. The Dev Plan automatically creates a single region called `example_partition_region`. The region type is `PARTITION`, a partitioned region without redundancy or persistence of entries. You can create other regions as desired with a Dev Plan service instance.

The page for configuring a Dev Plan is similar to the page for configuring other service plans. To configure the Dev Plan, input information in the fields and make selections from the options on the **Plan for test development** page.

Plan for test development

Plan for test development*

- ☐ Plan Disabled
- ☒ Plan Enabled

Plan Name *

dev-plan

Plan Description *

Plan Description

☒ Enable metrics for service instances

CF Service Access*

Enable Service Access

Maximum service instances *

10

Availability zones for service instances *

☒ default

VM type for the locator-server VM*

medium.cpu (cpu: 4, ram: 2 GB, disk: 8 GB)

Persistent disk type for the locator-server VM*

20 GB

SyslogSave

By default, syslog forwarding is not enabled in PCC. However, PCC supports forwarding syslog to an external log management service (for example, Papertrail, Splunk, or your custom enterprise log sink). The broker logs are useful for debugging problems creating, updating, and binding service instances.

To enable remote syslog for the service broker, do the following:

1. Click **Syslog**.



The screenshot shows a configuration window titled 'External Syslog Host'. It contains two input fields: 'External Syslog Host' with the value 'logs.example.com' and a file icon to its right, and 'External Syslog Port' with the value '12345'.

2. Configure the fields on the **Syslog** pane as follows:

Field	Instructions
Enable Remote Syslog	Select to enable.
External Syslog Address	Enter the address or host of the syslog server for sending logs, for example, <code>logs.example.com</code> .
External Syslog Port	Enter the port of the syslog server for sending logs, for example, <code>29279</code> .
Enable TLS for Syslog	Select to enable secure log transmission through TLS. Without this, remote syslog sends unencrypted logs. We recommend enabling TLS, as most syslog endpoints such as Papertrail and Logsearch require TLS.
Permitted Peer for TLS Communication. This is required if TLS is enabled.	If there are several peer servers that can respond to remote syslog connections, then provide a regex, such as <code>*.example.com</code> .
CA Certificate for TLS Communication	If the server certificate is not signed by a known authority, for example, an internal syslog server, provide the CA certificate of the log management service endpoint.
Send service instance logs to external	By default, only the broker logs are forwarded to your configured log management service. If you want to forward server and locator logs from all service instances, select this. This lets you monitor the health of the clusters, although it generates a large volume of logs. If you don't enable this, you get only the broker logs which include information about service instance creation, but not about on-going cluster health.

3. Click **Save**.

Service Instance TLS Upgrades

A configurable number of service instances may be upgraded concurrently by entering a new value that is greater than one and less than the BOSH worker count for the **Number of**

simultaneous upgrades.

Specify a set of service instances to act as canaries for the upgrade process by changing the **Number of upgrade canary instances** to a value greater than 0. If all canary instances successfully upgrade, the remaining instances are upgraded. If any canary instance fails to upgrade, the upgrade fails and no further instances are upgraded.

Click **Save** after changing values.

Configuration for the upgrade-all-service-instances errand

Number of simultaneous upgrades (min: 1) *

1

The maximum number of service instances that will be in an upgrading state at the same time. This should be set to a maximum of 1 less than the BOSH worker count.

Number of upgrade canary instances (min: 0) *

0

Number of service instances to upgrade first before going on to upgrade the rest of the instances.

Save

Security

There are four configuration aspects to the Security settings. Each of the four must be considered and appropriately handled within these settings. Once the Security properties settings have been set, click **Save**. The Security properties Settings page appears as:

Security properties

☐ Enable Secure Service Instance Credentials *

☒ TLS Supported *

Type "X" to acknowledge that the PCF environment must be prepared to use TLS in order to create PCC service instances with TLS enabled. See the section in the documentation titled Preparing for TLS. *

x

External UAA authentication *

☐ UAA Auth disabled

☒ UAA Auth enabled

PCC Client Creds *

Username

Password

Creds for the client id/secret you've created for PCC (see docs on how)

Save

The four items in this page to handle are:

1. The environment may be configured to more securely store service keys within CredHub, instead of within the cloud controller's data store. To enable this functionality, click on the box labeled **Enable Secure Service Instance Credentials** to enable use of CredHub.
2. To permit the creation of a PCC service instance that will TLS-encrypt communications within the cluster, check the box labeled **TLS Supported**. Checking the box does not cause

new service instances to encrypt communications; follow the directions in [Provide Optional Parameters](#) to create a new service instance with TLS encryption.

3. An 'X' is required in the text box to promote the understanding that a TLS-enabled service instance cannot be created if the PCF environment is not set up to handle TLS. A TLS-enabled environment is required independent of whether a TLS-enabled cluster will be created; see [Preparing for TLS](#) for how to prepare the PCF environment.
4. If User Account and Authentication (UAA) of PCC users will go through an external system, enable that with the radio button labeled **UAA Auth enabled**. With UAA enabled, create a UAA client before doing this tile installation as explained in [Create a UAA Client](#). Also follow the instructions in [Configuring UAA Roles](#) to complete the configuration by setting up the user roles. With UAA enabled, two text boxes will appear. Fill the boxes with the UAA client name and that client's secret, which were set when the client was created.

Errands

By default, post-deploy and pre-delete errands always run. Pivotal recommends keeping these defaults. However, if necessary, you can change these defaults as follows.

For general information about errands in PCF, see [Managing Errands in Ops Manager](#)

1. Click **Errands**.
2. Change the setting for the errands.
3. Click **Save**.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

VM Memory Allocation

The service plans you choose determines how memory is allocated to the GemFire processes. One consideration is how much memory is available in each VM.

Memory allocation depends on whether a plan is *colocated* or *non-colocated*:

- A *colocated plan* is one in which each VM hosts one locator and one server. For example, the default small footprint plan comprises three locators and three servers hosted on three VMs.
- A *non-colocated plan* is one in which each locator or server runs in its own VM. For example, the default medium plan comprises three locators and four servers, each running in its own VM, for a total of seven VMs.

The following sections describe how PCC allocates memory based on the total memory of each VM.

Memory Allocation for Colocated Plans

- If total VM memory is 5GB or less, 512 MB is allocated for the locator Java heap; 90% of the remaining memory is allocated to the GemFire server's Java heap, and the remainder is allocated to the OS.

- If total VM memory is greater than 5GB, 10% of total memory is allocated for the locator Java heap; 90% of the remaining memory is allocated to the GemFire server's Java heap, and the remainder is allocated to the OS.

Server heap is reduced to accommodate the **Pivotal Anti-Virus** and **File Integrity Monitoring (FIM)** PCF add-ons, if they are present.

Memory Allocation for Non-colocated Plans

Under non-colocated plans, each VM hosts one GemFire member, either a locator or a server.

- If total VM memory is 2GB or less, 1GB is allocated for the locator or server Java heap, and the remainder to the OS.
- If total VM memory is between 2G and 8GB, 2GB is allocated to the OS, and the remainder to the locator or server Java heap.
- If total VM total memory is greater than 8G, 4GB is allocated to the OS, and the remainder to locator or server Java heap.

Server heap is reduced to accommodate the **Pivotal Anti-Virus** and **File Integrity Monitoring (FIM)** PCF add-ons, if they are present.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Configuring User Account and Authentication (UAA) Roles

Extra configuration is required when defining users that have specific security roles with an authentication and enterprise single sign-on (SSO) such as LDAP.

Before PCC installation, create a UAA client as described in [Create a UAA Client](#).

Complete the remainder of configuration in conjunction with choosing the **UAA Auth enabled** radio button on the Security properties page when installing the PCC tile, as detailed in [Security](#).

The Roles

A PCC service instance internally defines four security roles. Each role is given predefined permissions for cluster operations. Each user is assigned security roles. As that user invokes a PCC cluster operation using gfsh, the PCC service's security manager verifies that the permission required for the cluster operation is one that the user's security role has.

The permissions assigned for each of the security roles:

- **PCC_ADMIN** has all permissions needed to manage the cluster and can access region data. This role has the GemFire permissions `CLUSTER:MANAGE`, `CLUSTER:WRITE`, `CLUSTER:READ`, `DATA:MANAGE`, `DATA:WRITE`, and `DATA:READ`.
- **PCC_OPERATOR** has all permissions needed to manage the cluster. This role may not handle or see region data. This role has the GemFire permissions `CLUSTER:MANAGE`, `CLUSTER:WRITE`, and `CLUSTER:READ`.
- **PCC_DATA-ACCESS** has all permissions needed to handle and see region data. This role has the GemFire permissions `CLUSTER:READ`, `DATA:WRITE`, and `DATA:READ`.

- **PCC_READ-ONLY** has the single GemFire permission **DATA:READ** to see region data.

Configuring the Roles

Configure the UAA server and your Enterprise SSO system (such as LDAP) with the space-specific roles.

1. Acquire the Globally Unique Identifier (GUID) for the CF space that will host your PCC service instance:

```
cf login -a REST-OF-ARGS-HERE
cf target -o NAME-OF-ORG
cf space --guid NAME-OF-SPACE
```

The form of the output GUID will be similar to the example:

```
03badc2a-4243-4251-84b5-c9bfba276f04
```

2. Create space-specific groups for each role within your Enterprise SSO system. The group name will take the form **ROLE_GUID**. For example, in the following group name:

```
PCC_ADMIN_03badc2a-4243-4251-84b5-c9bfba276f04
```

PCC_ADMIN is the role, and **03badc2a-4243-4251-84b5-c9bfba276f04** is the GUID.

3. Place users into the created space-specific groups you created within your Enterprise SSO system.
4. Use the UAA Command Line Interface (UAAC) to log in as **admin client** to your UAA server.
5. Use the UAAC to add each group name with:

```
$ uaac group add ROLE_GUID
```

Using the example GUID, the commands to add the four group names will be similar to:

```
$ uaac group add PCC_ADMIN_03badc2a-4243-4251-84b5-c9bfba276f04
$ uaac group add PCC_OPERATOR_03badc2a-4243-4251-84b5-c9bfba276f04
$ uaac group add PCC_DATA-ACCESS_03badc2a-4243-4251-84b5-c9bfba276f04
$ uaac group add PCC_READ-ONLY_03badc2a-4243-4251-84b5-c9bfba276f04
```

6. Use the UAAC to map each group name with **uaac group map** commands. The commands are described in [Grant Admin Permissions to an External Group \(SAML or LDAP\)](#). For LDAP:

```
$ uaac group map --name ROLE_GUID "GROUP-DISTINGUISHED-NAME"
```

where **GROUP-DISTINGUISHED-NAME** is the LDAP distinguished name of each space-specific group created in step 2. For example:

```
$ uaac group map --name PCC_DATA-ACCESS_03badc2a-4243-4251-84b5-c9bfba276f04 "CN=PCC_DATA-ACCESS_03badc2a-4243-4251-84b5-c9bfba276f04,OU=Groups,DC=pivotal,DC=io"
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Setting Service Instance Quotas

- [Create Global-level Quotas](#)
- [Create Plan-level Quotas](#)
- [Create and Set Org-level Quotas](#)
- [Create and Set Space-level Quotas](#)
- [View Current Org and Space-level Quotas](#)
- [Monitor Quota Use and Service Instance Count](#)
- [Calculate Resource Costs for On-Demand Plans](#)
 - [Calculate Maximum Resource Cost Per On-Demand Plan](#)
 - [Calculate Maximum Resource Cost for All On-Demand Plans](#)
 - [Calculate Actual Resource Cost of all On-Demand Plans](#)

On-demand provisioning is intended to accelerate app development by eliminating the need for development teams to request and wait for operators to create a service instance. However, to control costs, operations teams and administrators must ensure responsible use of resources.

There are several ways to control the provisioning of on-demand service instances by setting various **quotas** at these levels:

- [Global](#)
- [Plan](#)
- [Org](#)
- [Space](#)

After you set quotas, you can:

- [View Current Org and Space-level Quotas](#)
- [Monitor Quota Use and Service Instance Count](#)
- [Calculate Resource Costs for On-Demand Plans](#)

Create Global-level Quotas

Each on-demand service has a separate service broker. A global quota at the service level sets the maximum number of service instances that can be created by a given service broker. If a service has more than one plan, then the number of service instances for all plans combined cannot exceed the global quota for the service.

The operator sets a global quota for each service tile independently. For example, if you have two service tiles, you must set a separate global service quota for each of them.

When the global quota is reached for a service, no more instances of that service can be created unless the quota is increased, or some instances of that service are deleted.

Create Plan-level Quotas

A service may offer one or more plans. You can set a separate quota per plan so that instances of that plan cannot exceed the plan quota. For a service with multiple plans, the total number of instances created for all plans combined cannot exceed the [global quota](#) for the service.

When the plan quota is reached, no more instances of that plan can be created unless the plan quota is increased or some instances of that plan are deleted.

Create and Set Org-level Quotas

An org-level quota applies to all on-demand services and sets the maximum number of service instances an organization can create within their foundation. For example, if you set your org-level quota to 100, developers can create up to 100 service instances in that org using any combination of on-demand services.

When this quota is met, no more service instances of any kind can be created in the org unless the quota is increased or some service instances are deleted.

To create and set an org-level quota, do the following:

1. Run this command to create a quota for service instances at the org level:

```
cf create-quota QUOTA-NAME -m TOTAL-MEMORY -i INSTANCE-MEMORY -r ROUTES -s SERVICE-INSTANCES --allow-paid-service-plans
```

Where:

```
* `QUOTA-NAME`---A name for this quota
* `TOTAL-MEMORY`---Maximum memory used by all service instances combined
* `INSTANCE-MEMORY`---Maximum memory used by any single service instance
* `ROUTES`---Maximum number of routes allowed for all service instances combined
* `SERVICE-INSTANCES`---Maximum number of service instances allowed for the org
```

For example:

```
<pre class="terminal">$ cf create-quota myquota -m 1024mb -i 16gb -r 30 -s 50 --allow-paid-service-plans</pre>
```

1. Associate the quota you created above with a specific org by running the following command:

```
cf set-quota ORG-NAME QUOTA-NAME
```

For example:

```
$ cf set-quota dev_org myquota
```

For more information on managing org-level quotas, see [Creating and Modifying Quota Plans](#).

Create and Set Space-level Quotas

A space-level service quota applies to all on-demand services and sets the maximum number of service instances that can be created within a given space in a foundation. For example, if you set

your space-level quota to 100, developers can create up to 100 service instances in that space using any combination of on-demand services.

When this quota is met, no more service instances of any kind can be created in the space unless the quota is updated or some service instances are deleted.

To create and set a space-level quota, do the following:

1. Run the following command to create the quota:

```
cf create-space-quota QUOTA-NAME -m TOTAL-MEMORY -i INSTANCE-MEMORY -r ROUTES -s SERVICE-INSTANCES --allow-paid-service-plans
```

Where:

```
* `QUOTA-NAME`---A name for this quota
* `TOTAL-MEMORY`---Maximum memory used by all service instances combined
* `INSTANCE-MEMORY`---Maximum memory used by any single service instance
* `ROUTES`---Maximum number of routes allowed for all service instances combined
* `SERVICE-INSTANCES`---Maximum number of service instances allowed for the org
```

For example:

```
<pre class="terminal">$ cf create-space-quota myspacequota -m 1024mb -i 16gb -r 30 -s 50 --allow-paid-service-plans</pre>
```

1. Associate the quota you created above with a specific space by running the following command:

```
cf set-space-quota SPACE-NAME QUOTA-NAME
```

For example:

```
$ cf set-space-quota myspace myspacequota
```

For more information on managing space-level quotas, see [Creating and Modifying Quota Plans](#).

View Current Org and Space-level Quotas

To view **org** quotas, run the following command.

```
cf org ORG-NAME
```

To view **space** quotas, run the following command:

```
cf space SPACE-NAME
```

For more information on managing org and space-level quotas, see the [Creating and Modifying Quota Plans](#).

Monitor Quota Use and Service Instance Count

Service-level and plan-level quota use, and total number of service instances, are available through the on-demand broker metrics emitted to Loggregator. These metrics are listed below:

Metric Name	Description
<code>on-demand-broker/SERVICE-NAME/quota_remaining</code>	Quota remaining for all instances across all plans
<code>on-demand-broker/SERVICE-NAME/PLAN-NAME/quota_remaining</code>	Quota remaining for a specific plan
<code>on-demand-broker/SERVICE-NAME/total_instances</code>	Total instances created across all plans
<code>on-demand-broker/SERVICE-NAME/PLAN-NAME/total_instances</code>	Total instances created for a specific plan



Note: Quota metrics are not emitted if no quota has been set.

You can also view service instance usage information in Apps Manager. For more information, see [Reporting Instance Usage with Apps Manager](#).

Calculate Resource Costs for On-Demand Plans

On-demand plans use dedicated VMs, disks, and various other resources from an IaaS, such as AWS. To calculate maximum resource cost for plans individually or combined, you multiply the quota by the cost of the resources selected in the plan configuration(s). The specific costs depend on your IaaS.

To view configurations for your Pivotal Cloud Cache on-demand plan, do the following:

1. Navigate to **Ops Manager Installation Dashboard > Cloud Cache > Settings**.
2. Click the section for the plan you want to view. For example, **Plan 1**.

The image below shows an example that includes the VM type and persistent disk selected for the server VMs, as well as the quota for this plan. The quota is shown in the **Maximum service instances** field.

Maximum service instances *

Maximum servers per cluster (min: 4, max: 32) *

Default Number of Servers (min: 4, max: 32) *

Availability zones for service instances *

- ☒ us-central1-f
- ☒ us-central1-c
- ☒ us-central1-b

VM type for the Locator VMs *

Persistent disk type for the Locator VMs *

VM type for the Server VMs *

Persistent disk type for the Server VMs *



Note: Although operators can limit on-demand instances with plan quotas and a global quota, as described in the above topics, IaaS resource usage still varies based on the number of on-demand instances provisioned.

Calculate Maximum Resource Cost Per On-Demand Plan

To calculate the maximum cost of VMs and persistent disk for each plan, do the following calculation:

plan quota x cost of selected resources

For example, if you selected the options in the above image, you have selected a VM type **micro** and a persistent disk type **20 GB**, and the plan quota is **15**. The VM and persistent disk types have an associated cost for the IaaS you are using. Therefore, to calculate the maximum cost of resources for this plan, multiply the cost of the resources selected by the plan quota:

(15 x cost of micro VM type) + (15 x cost of 20 GB persistent disk) = max cost per plan

Calculate Maximum Resource Cost for All On-Demand Plans

To calculate the maximum cost for all plans combined, add together the maximum costs for each plan. This assumes that the sum of your individual plan quotas is less than the global quota.

Here is an example:

(plan1 quota x plan1 resource cost) + (plan2 quota x plan2 resource cost) = max cost for all plans

Calculate Actual Resource Cost of all On-Demand Plans

To calculate the current actual resource cost across all your on-demand plans:

1. Find the number of instances currently provisioned for each active plan by looking at the `total_instance` metric for that plan.
2. Multiply the `total_instance` count for each plan by that plan's resource costs. Record the costs for each plan.
3. Add up the costs noted in Step 2 to get your total current resource costs.

For example:

(plan1 total_instances x plan1 resource cost) + (plan2 total_instances x plan2 resource cost) = current cost for all plans

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Upgrading Pivotal Cloud Cache

Upgrade minor release versions from your currently deployed version to the target version in sequential order. For example, Pivotal Cloud Cache (PCC) v1.2 must be upgraded to PCC v1.3 prior to upgrading to PCC v1.4. Note that each PCC release is compatible with two Pivotal Application Service (PAS) and Ops Manager versions, as specified in the [Product Snapshot](#). Incorporate those upgrades to PAS and Ops Manager in your upgrade process as required to maintain compatibility, as described in [Upgrading Pivotal Cloud Foundry](#).

Follow the steps below to upgrade PCC:

1. Download the new version of the tile from the Pivotal Network.
2. Upload the product to Ops Manager.
3. Click **Add** next to the uploaded product.
4. Click on the PCC tile and configure the upgrade options.
 - o To try the upgrade on a small number of service instances first, set the quantity of canary service instances as described in [Service Instance Upgrades](#).

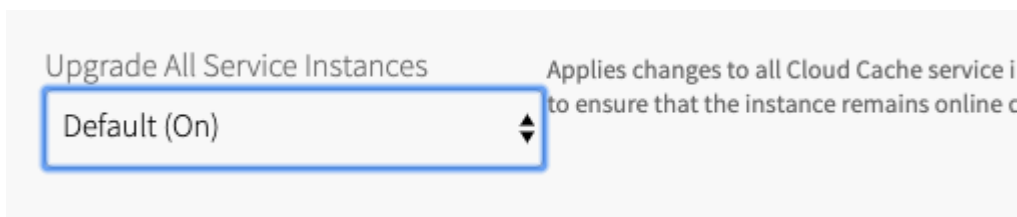
- Set the number of instances that are to be upgraded in parallel as described in [Service Instance Upgrades](#).
 - Under the **Errands** section, choose the **Default (On)** value for the **Upgrade All Service Instances** post-deploy errand. **Save** the change.
5. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#)).
 6. Click **Apply Changes**.

Enable Individual Service Instance Upgrades

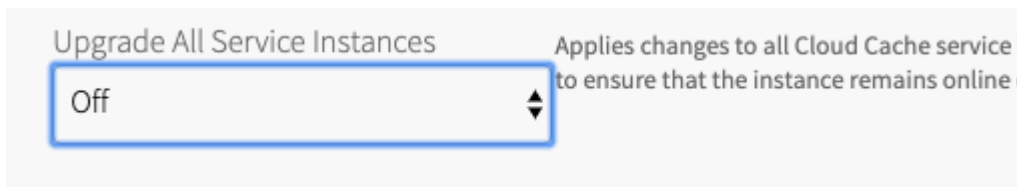
The default upgrade path upgrades all service instances as a result of a tile upgrade. This operation can take a lengthy amount of time. To expedite upgrades, an operator can permit developers to upgrade their own service instances once the tile has been upgraded.

An operator enables individual service instance upgrades during tile installation. This feature requires PAS/Ops Manager 2.7 or a higher version and works for upgrading from PCC 1.9.0 to a higher version.

Within the PCC tile, in the **Errands** section, the default for the **Upgrade All Service Instances** errand, which upgrades all service instances, appears as:



To change the state of this errand such that individual service instance upgrades are enabled, choose **Off** for this errand:



Click **Save**.

Once individual service instance upgrades are enabled, the developer upgrades an individual service instance following the instructions in [Upgrade a Single Service Instance](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Updating Pivotal Cloud Cache Plans

Follow the steps below to update plans in Ops Manager.

1. Click on the Cloud Cache tile.
2. Click on the plan you want to update under the **Information** section.
3. Edit the fields with the changes you want to make to the plan.
4. Click **Save** button on the bottom of the page.

5. Click on the **PCF Ops Manager** to navigate to the **Installation Dashboard**.
6. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#)).
7. Click **Apply Changes**.

Plan changes are not applied to existing services instances until you run the `upgrade-all-service-instances` BOSH errand. You must use the BOSH CLI to run this errand. Until you run this errand, developers cannot update service instances.

Changes to fields that can be overridden by optional parameters, for example `num_servers` or `new_size_percentage`, change the default value of these instance properties, but do not affect existing service instances.

If you change the allowed limits of an optional parameter, for example the maximum number of servers per cluster, existing service instances in violation of the new limits are not modified.

When existing instances are upgraded, all plan changes are applied to them. Upgrades and updates to service instances can cause a rolling restart of GemFire servers. Be aware that the rebalancing of data to maintain redundancy may impact the performance of the remainder of the servers within the service instance.



WARNING: Data loss may result from the restart of a cluster. See [Restarting a Cluster](#) for the conditions under which data loss occurs.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Uninstalling Pivotal Cloud Cache

To uninstall PCC, follow the steps from below from the **Installation Dashboard**:

1. Click the trash can icon in the bottom-right-hand corner of the tile.
2. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#)).
3. Click **Apply Changes**.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Backing Up and Restoring Service Instances

Both disaster recovery and system validation depend upon backups. A PCC service instance backup consists of the configuration for a PCC service instance together with all region data that has been persisted to disk. A PCC service instance backup may be used to restore the service on a new, but unconfigured PCC service instance.

Before You Begin



WARNING: Consider each of these important items when preparing to backup and restore a PCC service instance. Since recovery depends on a correct implementation, address each item during planning.

- The backup consists of data from regions that have been persisted to disk. All non-persistent region data will be lost. A persistent region is one that writes its region entries to disk.
- The backup and restoration process does not restore WAN replication configuration. A PCC service instance that replicates data via WAN may be backed up. A restore of that PCC service instance will contain all the persistent data. That PCC service instance will be able to communicate with other WAN-connected service instances after further configuration.
- The disk size on the jumpbox VM must be large enough to hold the backup artifacts. The quantity of disk space needs to exceed the sum total of all disk space used within the PCC service instance to be backed up. An approximation of that quantity of disk space may be calculated by looking at the fully populated PCC service instance. Sum the disk space (in `/var/vcap/store`) used by each locator and server.
- The PCC service instance needs to be quiescent when the backup is taken. An active cluster could have disk writes in progress, leading to a backup for which region data may or may not have been written to disk.
- The fresh PCC service instance that is to be used for a restore must have same quantity of locators and servers as the PCC service instance had when its backup was taken. The fresh PCC service instance must have sufficient resources to hold the data that is in the backup.
- Do not configure the fresh PCC service instance that is to be used for a restore after creation. It must be empty. Do not create any regions or start any gateway senders. The restore will create all regions, both persistent and not persistent. The non-persistent regions will be empty.
- Service keys are not part of the backup. Service keys will need to be created anew on a restored service instance.

Backing Up a PCC Service Instance

1. Optional: Back up the Pivotal Cloud Foundry environment. For instructions, see [Backing Up Pivotal Cloud Foundry with BBR](#).
2. Optional: Compact the disk stores of the cluster to be backed up. Use the `gfsh compact disk-store` command after connecting to the cluster with a role that is authorized with the `CLUSTER:MANAGE` operation permission. See the documentation for `gfsh compact disk-store`.
3. Optional: Acquire a region statistic that may be used for a minimal validation upon using this backup for subsequent restoration. Use the `gfsh describe region` command on each persistent region, after connecting to the cluster with a role that is authorized with the `CLUSTER:READ` operation permission.

```
describe region --name=REGION-NAME
```

For each persistent region (the `data-policy` will contain the string `PERSISTENT`), record the `size` of region. This size is the quantity of entries in the region. Assuming that activity on

the region is quiescent, when this backup is used in a restoration, the quantity of region entries in the restored region forms a minimal validation of the region.

4. Ssh to the jumpbox. Assuming that the Ops Manager VM is being used as the jumpbox, follow directions at [Log in to the Ops Manager VM with SSH](#).
5. Determine all needed credentials and parameters for the command that makes the backup:
 - `BOSH-DIRECTOR-IP`: Obtain the BOSH Director IP address by following the instructions at [Step 4:Retrieve BOSH Director Address](#).
 - `SERVICE-INSTANCE-DEPLOYMENT-NAME`: Obtain the deployment name by following the instructions at [Acquire the Deployment Name](#).
 - `BOSH-CLIENT`, `BOSH-CLIENT-PASSWORD`, and `PATH-TO-BOSH-SERVER-CERTIFICATE`: To learn all three of these values, open the Credentials tab of the BOSH Director tile in Ops Manager. Find and open `Bosh Commandline Credentials`. The path is the path to the root Certificate Authority (CA) certificate, and its value will be `/var/tempest/workspaces/default/root_ca_certificate` if Ops Manager and the jumpbox are on the same VM.
6. Make the backup. From the jumpbox, issue this command, substituting values acquired in the previous step:

```
BOSH_CLIENT_SECRET=BOSH-CLIENT-PASSWORD \
bbr deployment \
--target BOSH-DIRECTOR-IP \
--username BOSH-CLIENT \
--ca-cert PATH-TO-BOSH-SERVER-CERTIFICATE \
--deployment SERVICE-INSTANCE-DEPLOYMENT-NAME \
backup
```

`BOSH_CLIENT_SECRET` is an environment variable that is set only for the duration of the command.

The backup will be within a directory on the jumpbox named by the deployment name and a timestamp.

7. Copy the backup to a permanent home for archiving. Pivotal recommends compressing and encrypting the files. Disaster recovery plans often recommend archiving multiple copies of each backup.

Restoring a PCC Service Instance

1. Use SSH to connect to the jumpbox. Assuming that the Ops Manager VM is being used as the jumpbox, follow directions at [Log in to the Ops Manager VM with SSH](#).
2. Transfer the archived backup to the jumpbox. Expand if compressed, and decrypt if encrypted.
3. Make a new, but empty service instance. See directions in [Creating a Pivotal Cloud Cache Service Instance](#). If the service instance to be restored was connected to a WAN, be sure to create the new instance with the same `distributed_system_id` as the old one.

4. Determine all needed credentials and parameters for the command that does the restore by following step 5 instructions from making the backup.
5. Do the restore. From the jumpbox, issue this command, substituting values acquired in the previous step:

```
$ BOSH_CLIENT_SECRET=BOSH-CLIENT-PASSWORD \
bbr deployment \
--target BOSH-DIRECTOR-IP \
--username BOSH-CLIENT \
--ca-cert PATH-TO-BOSH-SERVER-CERTIFICATE \
--deployment SERVICE-INSTANCE-DEPLOYMENT-NAME \
restore --artifact-path PATH-TO-SERVICE-INSTANCE-ARTIFACT
```

`PATH-TO-SERVICE-INSTANCE-ARTIFACT` is the path to the artifact for the instance that you are currently restoring.

6. Create a new service key, as the service key was not an artifact that the backup created. For a stand-alone service instance (one that is not part of a WAN), follow the directions at [Create a Service Key](#). For a WAN-connected service instance, see [Restoring a WAN Connection](#).
7. If the region size was captured prior to making the backup, in order to do a minimal validation of the restored cluster, use the `gfs describe region` command on each persistent region, after connecting to the cluster with a role that is authorized with the `CLUSTER:READ` operation permission.

```
describe region --name=REGION-NAME
```

For each persistent region, the `size` of the restored region should be the same as the value captured when the backup was made.

8. Rebind or restage all apps.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Enable Service-Instance Sharing

The Cloud Cache implementation of instance sharing permits read-only access to a Cloud Cache service instance within one space by an app within another space. A Cloud Foundry `admin` operator, or one with administrative privileges, may enable service-instance sharing for Cloud Cache service instances.

Enable service-instance sharing across a foundation with:

```
$ cf enable-feature-flag service_instance_sharing
```

A developer will need to set up Cloud Cache service-instance sharing as detailed in [Service-Instance Sharing](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Rotating Certificates

This section specifies the procedure to follow to check expiration dates and rotate certificates.

To rotate the Services TLS CA and its leaf certificates, for service instances that are *not* connected by WAN gateways, follow the procedure in [Rotating the Services TLS CA and Its Leaf Certificates](#).

WAN-Connected Service Instances

Follow this procedure for rotating the certificates of foundations, when service instances connected via WAN gateways reside on distinct foundations.

The procedure assumes two foundations have service instances, one called foundation A and the other called foundation B. The procedure may be followed by installations that have more than two foundations. To expand the procedure to cover more foundations, in each place where foundation A and foundation B are identified, also apply the directions for a third foundation C, a fourth foundation D, and so on.

The procedure is broken into parts. Do the parts in the order listed. The parts are:

- [Generate New TLS CA Certificates](#)
- [Collect All the Certificates](#)
- [Add All the Certificates to the Tiles](#)
- [Apply Changes for the First Time](#)
- [Set New Services TLS CA Certificate](#)
- [Apply Changes for the Second Time](#)
- [Remove the Old Services TLS CA Certificates](#)
- [Apply Changes for the Third Time](#)

Generate New TLS CA Certificates

Follow these steps twice to obtain a new TLS CA certificate, once to generate a new TLS CA certificate for foundation A, and once to generate a new TLS CA certificate for foundation B.

1. Check if CredHub has a new temporary certificate from a previous rotation attempt. Run:

```
credhub get -n /services/new_ca
```



Note: This procedure assumes that `/services/new_ca` is the CredHub location used when generating the temporary certificate. If you used a different location, specify that location instead.

2. If an older temporary certificate already exists, delete that certificate by running:

```
credhub delete -n /services/new_ca
```

3. Do one of the following:
 - **If you get your certificates from your CA:** If you generate certificates from your own private or public CA, obtain a new certificate from that CA. You add this to the

Ops Manager settings in the next high-level step.

- o **If you use self-signed certificates:** Generate a new self-signed certificate with a new name or path in CredHub:

```
credhub generate \
--name="/services/new_ca" \
--type="certificate" \
--no-overwrite \
--is-ca \
--duration=1825 \
--common-name="opsmgr-services-tls-ca"
```

where

- **name** is the CredHub path or name of the new certificate. You can use `/services/new_ca` or substitute a different name as long as you use the value consistently in the rest of the procedure.
 - **common-name** is the common name of the generated certificate. You can use `opsmgr-services-tls-ca` or substitute a different common name as long as you use the value consistently in the rest of the procedure.
 - **duration** is set to 1825, which is a recommended value of five years. If you do not specify a duration, the default value is 365 or one year.
- o **If you use an intermediate certificate signed by a root CA in CredHub:**Generate a new certificate signed by the CredHub root CA by running:

```
credhub generate \
--name="/services/new_ca" \
--type="certificate" \
--no-overwrite \
--is-ca \
--duration=1825 \
--common-name="opsmgr-services-tls-ca" \
--ca=/PATH-TO-ROOT-CA
```

where

- **name** is the CredHub path or name of the new certificate. You can use `/services/new_ca` or substitute a different name as long as you use the value consistently in the rest of the procedure.
- **common-name** is the common name of the generated certificate. You can use `opsmgr-services-tls-ca` or substitute a different common name as long as you use the value consistently in the rest of the procedure.
- **duration** is set to 1825, which is a recommended value of five years. If you do not specify a duration, the default value is 365 or one year.
- **ca** is the path to the root CA for the new certificate. Replace `PATH-TO-ROOT-CA` with the appropriate path for the root CA.

Collect All the Certificates

Each of foundation A and foundation B has a current certificate and the newly generated one from the previous step, totalling four certificates across two foundations.

1. On foundation A, log in to CredHub.
2. On foundation A, get the *current* Services TLS CA certificate by capturing the certificate in the output of:

```
credhub get --name=/services/tls_ca -k ca
```

3. On foundation A, get the *new* Services TLS CA certificate either from a pre-existing file, or from your new CredHub location by capturing the certificate in the output of:

```
credhub get --name=/services/new_ca -k ca
```

4. On foundation B, log in to CredHub.
5. On foundation B, get the *current* Services TLS CA certificate by capturing the certificate in the output of:

```
credhub get --name=/services/tls_ca -k ca
```

6. On foundation B, get the *new* Services TLS CA certificate either from a pre-existing file, or from your new CredHub location by capturing the certificate in the output of:

```
credhub get --name=/services/new_ca -k ca
```

Add All the Certificates to the Tiles

This part adds all the certificates (current and new) to the tiles. For a distributed system with two foundations, the four certificates are:

- The current TLS CA certificate for foundation A (called `/services/tls_ca`)
 - The current TLS CA certificate for foundation B (also called `/services/tls_ca`)
 - The new TLS CA certificate for foundation A (called `/services/new_ca`)
 - The new TLS CA certificate for foundation B (also called `/services/new_ca`)
1. On foundation A, paste all certificates (ordering of the certificates does not matter) into the **BOSH Director > Security > Trusted Certificates** field. Click **Save**.
 2. On foundation A, paste all certificates (ordering of the certificates does not matter) into two fields of the VMware Tanzu Application Service tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. Click **Save**.
 3. On foundation A, paste all certificates (ordering of the certificates does not matter) into two fields of the Isolation Segment tile, if the environment has this optional tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. Click **Save**.
 4. On foundation B, paste all certificates (ordering of the certificates does not matter) into the **BOSH Director > Security > Trusted Certificates** field. Click **Save**.

5. On foundation B, paste all certificates (ordering of the certificates does not matter) into two fields of the VMware Tanzu Application Service tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. Click **Save**.
6. On foundation B, paste all certificates (ordering of the certificates does not matter) into two fields of the Isolation Segment tile, if the environment has this optional tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. Click **Save**.

Apply Changes for the First Time



Warning: This procedure involves redeploying all of the VMs in your Ops Manager deployment to apply a CA certificate. The operation can take a long time to complete.

You may apply the changes to both foundation A and foundation B at the same time.

Follow these steps twice, once to apply the changes to foundation A, and once to apply the changes to foundation B.

1. Navigate back to the **Installation Dashboard**.
2. Click **Review Pending Changes**.
3. Ensure that all product tiles, including PAS, PASW, Isolation Segment, and partner tiles, are selected.
4. Identify which on-demand service tiles are using the Services TLS CA certificate. Run:

```
credhub curl -p /api/v1/certificates?name=%2Fservices%2Ftls_ca
```

From the returned list, identify which on-demand service tiles use TLS in your deployment, such as MySQL for VMware Tanzu, Tanzu GemFire for VMs, Redis for VMware Tanzu, and RabbitMQ for VMware Tanzu [VMs].

5. For each on-demand service tile that uses TLS:
 1. Expand the errands.
 2. Enable the errand to upgrade all service instances.



Note: The name of the upgrade all service instances errand may differ slightly between services.

6. Click **Apply Changes**.

Set New Services TLS CA Certificate

Set the new Services TLS CA certificate in CredHub. Do this on foundation A with foundation A's new TLS CA certificate, and do this on foundation B with foundation B's new TLS CA certificate.

Do one of the following:

- **If you have an existing certificate:** Obtain the CA certificate and private key file corresponding to the certificate that you got in the [Get New TLS CA Certificates](#) part of the procedure. Then, run:

```
credhub set \
--name="/services/tls_ca" \
--type="certificate" \
--certificate=PEM-PATH/root.pem \
--private=CERT-KEY
```

Where:

- `certificate` is the location of the `root.pem` file for the certificate. Replace `PEM-PATH` with the path of your certificate's `root.pem` file.
 - `private` is the private key for the new certificate. Replace `CERT-KEY` with the value of the private key for your certificate.
- **If you created a new self-signed or intermediary certificate:** Set the `/services/new_ca` that you generated in the [Get New TLS CA Certificates](#) part of the procedure as the Services TLS CA:

```
credhub get -n /services/new_ca -k ca > new_ca.ca
credhub get -n /services/new_ca -k certificate > new_ca.certificate
credhub get -n /services/new_ca -k private_key > new_ca.private_key
credhub set -n /services/tls_ca \
--type=certificate \
--root=new_ca.ca \
--certificate=new_ca.certificate \
--private=new_ca.private_key
```

Apply Changes for the Second Time



Warning: This procedure involves redeploying all of the VMs in your Ops Manager deployment to apply a CA certificate. The operation can take a long time to complete.

You may apply the changes to both foundation A and foundation B at the same time.

In this step, you apply changes to ensure that all on-demand service instances generate new leaf certificates from the new Services TLS CA.

Follow these steps twice, once to apply the changes to foundation A, and once to apply the changes to foundation B.

1. Navigate back to the **Installation Dashboard**.
2. Click **Review Pending Changes**.
3. Ensure that all product tiles, including PAS, PASW, Isolation Segment, and partner tiles, are deselected in order to reduce deployment time.

4. Select only the on-demand services tiles that use TLS, such as MySQL for VMware Tanzu, Tanzu GemFire for VMs, Redis for VMware Tanzu, and RabbitMQ for VMware Tanzu [VMs].
5. For each on-demand service tile that uses TLS:
 1. Expand the errands.
 2. Enable the errand to upgrade all service instances.



Note: The name of the upgrade all service instances errand may differ slightly between services.

6. Click **Apply Changes**.

Remove the Old Services TLS CA Certificates

After your apps have reconnected to service instances with the certificates generated by the new CA, both of the old CA certificates may be removed. There will be one old CA certificate for foundation A and one old certificate for foundation B.

Follow these steps twice, once for foundation A, and once for foundation B.

1. Delete the two old CA certificates from the BOSH Director tile **Security > Trusted Certificates** field. You will remove one old CA certificate for foundation A and one old certificate for foundation B. Click **Save**.
2. Delete the two old CA certificates from the two fields of the VMware Tanzu Application Service tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. You will remove one old CA certificate for foundation A and one old certificate for foundation B. Click **Save**.
3. Delete the two old CA certificates from the two fields of the Isolation Segment tile, if the environment has this optional tile: **Networking > Certificate Authorities trusted by the Gorouter** and **Networking > Certificate Authorities trusted by the HAProxy**. You will remove one old CA certificate for foundation A and one old certificate for foundation B. Click **Save**.

Apply Changes for the Third Time

As a security best practice, you should remove outdated certificates as soon as possible from your deployment. You can schedule this step to a convenient time, because for most deployments you will not lose any deployment functionality if you do not perform this step immediately.

For some deployments, you may encounter an error if you create a service instance in a deployment that contains an expired Services TLS CA certificate. For more information, see this [Knowledgebase Article](#).



Warning: This procedure involves redeploying all of the VMs in your Ops Manager deployment to apply a CA certificate. The operation can take a long time to complete.

You may apply the changes to both foundation A and foundation B at the same time.

Follow these steps twice, once to apply the changes to foundation A, and once to apply the changes to foundation B.

1. Navigate back to the **Installation Dashboard**.
2. Click **Review Pending Changes**.
3. Ensure that all product tiles, including PAS, PASW, Isolation Segment, and partner tiles, are selected.
4. Select only the on-demand services tiles that use TLS, such as MySQL for VMware Tanzu, Tanzu GemFire for VMs, Redis for VMware Tanzu, and RabbitMQ for VMware Tanzu [VMs].
5. For each on-demand service tile that uses TLS:
 1. Expand the errands.
 2. Enable the errand to upgrade all service instances.



Note: The name of the upgrade all service instances errand may differ slightly between services.

6. Click **Apply Changes**.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Restoring a WAN Connection

This sequence of steps restores a bidirectional WAN connection to a service instance when the service instance has been restored from backup.

In this description, cluster A is an existing, running service instance that was formerly connected to cluster B over a WAN connection, and cluster B is the service that has been freshly restored from backup, and needs to be reconnected.

At this point in the process, cluster B should have the following characteristics:

- All data restored from backup
- The same `distributed_system_id` as its predecessor
- A freshly-generated service key

If the restored `distributed_system_id` matches that in the backup, then there is no need to recreate gateway senders and receivers; they are already defined. This procedure manually updates gateway senders with pointers to remote locators and credentials, re-enabling their ability to connect across the WAN without authentication errors.

1. Obtain the service key for cluster A. The service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster B in this example) to communicate with Cluster A:

```
$ cf service-key A k1
```

The contents of the service key differ based upon the cluster configuration, such as whether an authentication and enterprise single sign-on (SSO) system such as LDAP has been configured. Here is sample output from `cf service-key A k1`:

```
Getting key k1 for service instance A as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-ABC-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_ABC"
      },
      {
        "password": "dev-DEF-password",
        "roles": [
          "developer"
        ],
        "username": "developer_DEF"
      }
    ]
  },
}
```

```
"wan": {}
}
```

2. Obtain the service key of cluster B:

```
$ cf service-key B k2
```

As above, the service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster A in this example) to communicate with Cluster B. Here is sample output from `cf service-key B k2`:

```
Getting key k2 for service instance destination as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
      "pulse": "https://cloudcache-2.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-JKL-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_JKL"
      },
      {
        "password": "dev-MNO-password",
```

```

    "roles": [
      "developer"
    ],
    "username": "developer_MNO"
  }
],
"wan": {}
}

```

3. Update the Cluster A service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster B service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster A to communicate with Cluster B, and to accept data from Cluster B:

```

$ cf update-service A -c '
{
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet-2.service-instance-id-2.bosh": [
          "10.1.16.7:1053",
          "10.1.16.6:1053",
          "10.1.16.8:1053"
        ]
      },
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        {
          "password": "gws-PQR-password",
          "username": "gateway_sender_PQR"
        }
      ]
    }
  ]
}'
Updating service instance A as admin

```

4. To verify that a service instance has been correctly updated, delete and recreate the cluster service key. The recreated service key will have the same user identifiers and passwords as its predecessor, and will reflect the changes you specified in the recent `cf update-service` commands. In particular, the `wan{}` element at the end of a cluster's service key should be populated with the other cluster's remote connection information. For example, to verify that the Cluster A service key was updated correctly, log in as Cluster A administrator and issue these commands to delete and recreate the Cluster A service key:

```

$ cf delete-service-key A k1
...
$ cf create-service-key A k1

```

Verify that the `wan{}` field of the Cluster A service key contains a `remote_clusters` element which specifies contact information for Cluster B, including Cluster B's `recursors` array, `remote_locators` array, and `trusted_sender_credentials`:

```
Getting key k1 for service instance A as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ]
  },
  "urls": {
    "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
    "pulse": "https://cloudcache-1.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-ABC-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABC"
    },
    {
      "password": "dev-DEF-password",
      "roles": [
        "developer"
      ],
      "username": "developer_DEF"
    }
  ],
  "wan": {
    "remote_clusters": [
```



```
{
  "recursors": {
    "services-subnet-2.service-instance-id-2.bosh": [
      "10.1.16.7:1053",
      "10.1.16.6:1053",
      "10.1.16.8:1053"
    ]
  },
  "remote_locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "password": "gws-PQR-password",
      "username": "gateway_sender_PQR"
    }
  ]
}
}
```

5. Update the Cluster B service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster A service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster B to communicate with Cluster A, and to accept data from Cluster A:

```
$ cf update-service B -c '
{
  "remote_clusters":[
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.5:1053",
          "10.0.8.7:1053",
          "10.0.8.6:1053"
        ]
      }
    }
  ]
  "remote_locators":[
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ]
  "trusted_sender_credentials":[
    {
      "username": "gateway_sender_GHI",
      "password":"gws-GHI-password"
    }
  ]
}
]'
Updating service instance B as admin
```

6. To verify that the Cluster B service key was updated correctly, log in as Cluster B administrator and issue these commands to delete and recreate the Cluster B service key:

```
$ cf delete-service-key B k2
...
$ cf create-service-key B k2
```

Verify that the `wan{}` field of the Cluster B service key contains a `remote_clusters` element which specifies contact information for Cluster A, including Cluster A's `recursors` array, `remote_locators` array, and `trusted_sender_credentials`:

```
Getting key k1 for service instance B as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    }
  },
  "remote_locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "password": "gws-PQR-password",
      "username": "gateway_sender_PQR"
    }
  ],
  "urls": {
    "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
    "pulse": "https://cloudcache-2.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-JKL-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_JKL"
    },
    {
      "password": "dev-MNO-password",
      "roles": [
        "developer"
      ]
    }
  ],
```

```

    "username": "developer_MNO"
  }
],
"wan": {
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.6:1053",
          "10.0.8.7:1053",
          "10.0.8.5:1053"
        ]
      },
      "remote_locators": [
        "id1.locator.services-subnet.service-instance-id.bosh[55221]",
        "id2.locator.services-subnet.service-instance-id.bosh[55221]",
        "id3.locator.services-subnet.service-instance-id.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        {
          "password": "gws-GHI-password",
          "username": "gateway_sender_GHI"
        }
      ]
    }
  ]
}
}
}

```

7. To verify that the connection has been successfully restored, see [Verify Bidirectional WAN Setup](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Working with Service Instances

This document describes how a Pivotal Cloud Foundry (PCF) app developer can choose a service plan, create and delete Pivotal Cloud Cache service instances, and bind an app.

You must install the [Cloud Foundry Command Line Interface \(cf CLI\)](#) to run the commands in this topic.

In this topic:

- [View Available Plans](#)
- [Create or Delete a Service Instance](#)
 - [Create a Service Instance](#)
 - [Delete a Service Instance](#)
- [Upgrade a Single Service Instance](#)
- [Updating a Pivotal Cloud Cache Service Instance](#)
 - [Rebalancing a Cluster](#)
 - [Restarting a Cluster](#)
 - [Changes to the Service Plan](#)
- [Monitoring Pivotal Cloud Cache Service Instances](#)
 - [Service Instance Metrics](#)
 - [Per Member Metrics](#)
 - [Gateway Sender and Gateway Receiver Metrics](#)
 - [Disk Metrics](#)
 - [Total Memory Consumption](#)
- [Service Instance Sharing](#)
- [Set Up Service Instances Across a Wide Area Network \(WAN\)](#)
 - [Set Up a Bidirectional System](#)
 - [Set Up a Unidirectional System](#)
 - [Set Up an Additional Bidirectional Interaction](#)
 - [Set Up an Additional Unidirectional Interaction](#)
- [Setting Up Servers for an Inline Cache](#)
 - [Implement a Cache Loader for Read Misses](#)
 - [Implement an Asynchronous Event Queue and Cache Listener for Write Behind](#)

- Implement a Cache Writer for Write Through
 - Configure Using gfsh for Write Behind
 - Configure Using gfsh for Write Through
- Accessing a Service Instance
 - Create Service Keys
 - Connect with gfsh over HTTPS
 - Create a Truststore
 - Establish the Connection with HTTPS
 - Establish the Connection with HTTPS in a Development Environment
- Using gfsh
 - gfsh Command Restrictions
 - Create Regions
 - Working with Disk Stores
 - Use the Pulse Dashboard
 - Access Service Metrics
 - Access Service Broker Metrics
 - Export gfsh logs
 - Deploy an App JAR File to the Servers
 - Use the GemFire-Greenplum Connector
 - Scripting Common Operations
- Connecting a Spring Boot App to Pivotal Cloud Cache with Session State Caching
 - Use the Tomcat App
 - Use a Spring Session Data GemFire App
- Creating Continuous Queries Using Spring Data GemFire

Create a pull request or raise an issue on the source for this page in GitHub

View Available Plans

Run `cf marketplace` to view the installed Cloud Cache tile version and its associated Pivotal GemFire version. For example:

```
$ cf marketplace
Getting services from marketplace in org system / space system as admin...
OK
```

service	plans	description	broker
p-cloudcache	small-footprint	Pivotal Cloud Cache (PCC v1.9.1-build.12, GemFire v9.8.4)	cloud-cache-broker

Run `cf marketplace -s p-cloudcache` to view all plans available for Cloud Cache. The plan names displayed are configured by the operator on tile installation.

```
$ cf marketplace -s p-cloudcache

Getting service plan information for service p-cloudcache as admin...
OK
```

service plan	description	free or paid
extra-small	Caching Plan 1	free
small	Caching Plan 2	free
medium	Caching Plan 3	free
large	Caching Plan 4	free
extra-large	Caching Plan 5	free

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Create or Delete a Service Instance

Create a Service Instance

Run `cf create-service p-cloudcache PLAN-NAME SERVICE-INSTANCE-NAME` to create a service instance. Replace `PLAN-NAME` with the name from the list of available plans. Replace `SERVICE-INSTANCE-NAME` with a name of your choice. Use this name to refer to your service instance with other commands. Service instance names can include alpha-numeric characters, hyphens, and underscores.

```
$ cf create-service p-cloudcache extra-large my-cloudcache
```

Service instances are created asynchronously. Run the `cf services` command to view the current status of the service creation, and of other service instances in the current org and space:

```
$ cf services
Getting services in org my-org / space my-space as user...
OK
```

name	service	plan	bound apps	last operation
my-cloudcache	p-cloudcache	small		create in progress

When completed, the status changes from `create in progress` to `create succeeded`.

Provide Optional Parameters

You can create a customized service instance by passing optional parameters to `cf create-service` using the `-c` flag. The `-c` flag accepts a valid JSON object containing service-specific configuration parameters, provided either in-line or in a file.

The PCC service broker supports the following parameters:

- `tls`: A boolean, that when true, enables TLS for all communication within the cluster.
- `num_servers`: An integer that specifies the number of server instances in the cluster. The minimum value is 4. The maximum and default values are configured by the operator.

- `new_size_percentage`: An integer that specifies the percentage of the heap to allocate to young generation. This value must be between 5 and 83. By default, the new size is 2 GB or 10% of heap, whichever is smaller.
- `distributed_system_id`: An integer that provides a unique identifier for a cluster that participates in a WAN.

This example enables TLS within the cluster:

```
$ cf create-service p-cloudcache small TLS-cluster -c '{"tls": true}'
```

This example creates the service with five service instances in the cluster:

```
$ cf create-service p-cloudcache small my-cloudcache -c '{"num_servers": 5}'
```

This example assigns the service a specific distributed system ID, which is important when restoring a WAN participant from backups:

```
$ cf create-service p-cloudcache small my-cloudcache -c '{"distributed_system_id": 2}'
```

Enable Session State Caching with the Java Buildpack

When the `session-replication` tag is specified, the Java buildpack downloads all the required resources for session state caching. This feature is available in Java buildpack version 3.19 and higher, up to but not including version 4. It is then available again in version 4.3.

To enable session state caching, do *one* of the following items:

- Option 1: When creating your service instance name, specify the `session-replication` tag. For example:

```
$ cf create-service p-cloudcache small-plan my-service-instance -t session-replication
```

- Option 2: Update your service instance, specifying the `session-replication` tag:

```
$ cf update-service new-service-instance -t session-replication
```

- Option 3: When creating the service, name the service instance name by appending it with the string `-session-replication`, for example `my-service-instance-session-replication`.

Enable Session State Caching Using Spring Session

To use [Spring Session](#) for session state caching for apps with PCC, follow the steps below:

1. Make the following changes to the app:
 - Replace existing Spring Session `@EnableXXXHttpSession` annotation with `@EnableGemFireHttpSession(maxInactiveIntervalInSeconds = N)` where `N` is seconds.
 - Add the `spring-session-data-geode` and `spring-data-geode` dependencies to the build.

- Add beans to the Spring app config.

For more information, see the [spring-session-data-gemfire-example](#) repository.

2. Create a region named `ClusteredSpringSessions` in gfsch when connected with a security role that can manage cluster data: `create region --name=ClusteredSpringSessions --type=PARTITION_HEAP_LRU`

Dev Plans

The Dev Plan is a type of service plan that is useful for development and testing. This example creates a Dev Plan service instance:

```
$ cf create-service p-cloudcache dev-plan my-dev-cloudcache
```

The plan provides a single locator and a single server colocated within a single VM. Because the VM is recycled when the service instance is updated or upgraded, all data within the region is lost upon update or upgrade.

When post-deploy scripts are enabled for Ops Manager, the service instance is created with a single sample region called `example_partition_region`. The region is of type `PARTITION_REDUNDANT_HEAP_LRU`, as described in [Partitioned Region Types for Creating Regions on the Server](#).

If `example_partition_region` has **not** been created, it is probably because post-deploy scripts are not enabled for Ops Manager, as described in [Configure a Dev Plan](#).

Delete a Service Instance

You can delete service instances using the cf CLI. Before doing so, you must remove any existing service keys and app bindings.

1. Run `cf delete-service-key SERVICE-INSTANCE-NAME KEY-NAME` to delete the service key.
2. Run `cf unbind-service APP-NAME SERVICE-INSTANCE-NAME` to unbind your app from the service instance.
3. Run `cf delete-service SERVICE-INSTANCE-NAME` to delete the service instance.

```
$ cf delete-service-key my-cloudcache my-service-key
$ cf unbind-service my-app my-cloudcache
$ cf delete-service my-cloudcache
```

Deletions are asynchronous. Run `cf services` to view the current status of the service instance deletion.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Upgrade a Single Service Instance

If individual service instance upgrades are permitted by the operator, and a newer tile has been made available, then developers may upgrade their own service instances by following the

procedure here. [Enable Individual Service Instance Upgrades](#) describes how an operator enables developer upgrades of their own service instances.

The cf CLI, v6.46.0 or a more recent version, is a prerequisite for upgrading a Pivotal Cloud Cache (PCC) service instance.

To upgrade a single PCC service instance:

1. Confirm that an upgrade is available for the service instance; the upgrade is available when the `upgrade available` column of the output says `yes`, as in the sample output:

```
$ cf services
Getting services in org system / space system as admin...

name      service      plan            last operation  broker            upgrade available
testSI    p-cloudcache  small-footprint create succeeded  cloudcache-broker yes
```

1. Upgrade the service instance with a command of the form:

```
cf update-service SERVICE-INSTANCE-NAME --upgrade
```

Replace `SERVICE-INSTANCE-NAME` with the name of the instance to be upgraded. Confirm that you want to update when prompted.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Updating a Pivotal Cloud Cache Service Instance

You can apply all optional parameters to an existing service instance using the `cf update-service` command. You can, for example, scale up a cluster by increasing the number of servers.

Previously specified optional parameters are persisted through subsequent updates. To return the service instance to default values, you must explicitly specify the defaults as optional parameters.

For example, if you create a service instance with five servers using a plan that has a default value of four servers:

```
$ cf create-service p-cloudcache small my-cloudcache -c '{"num_servers": 5}'
```

And you set the `new_size_percentage` to 50%:

```
$ cf update-service my-cloudcache -c '{"new_size_percentage": 50}'
```

Then the resulting service instance has 5 servers and `new_size_percentage` of 50% of heap.

Rebalancing a Cluster

When updating a cluster to increase the number of servers, the available heap size is increased. When this happens, PCC automatically rebalances data in the cache to distribute data across the cluster.

This automatic rebalancing does not occur when a server leaves the cluster and later rejoins, for example when a VM is re-created, or network connectivity lost and restored. In this case, you must

manually rebalance the cluster using the `gfsh rebalance` command while connected to the cluster under a role that can manage a cluster's data.



Note: You must first [Connect with gfsh over HTTPS](#) before you can use the `rebalance` command.

Restarting a Cluster

Restarting a cluster stops and restarts each cluster member in turn, issuing a rebalance as each restarted server joins the cluster.



WARNING: Restart of a cluster may cause data loss.

There is a potential for data loss when restarting a cluster; the region type and number of servers in the cluster determine whether or not data is lost.

- **All data is lost when restarting a cluster with these region types and number of servers:**
 - Partitioned regions without redundancy or persistence. As each server is stopped, the region entries hosted in buckets on that stopped server are permanently lost.
 - Replicated regions without persistence on a cluster that has a single server.
 - A Dev Plan cluster loses all data, as there is a single server and no region persistence.
- **No data is lost when restarting the cluster with these region types and number of servers:**
 - Replicated regions for clusters with more than one server.
 - Persistent regions will not lose data, as all data is on the disk and available upon restart of a server.
 - Partitioned regions with redundancy. When the server with the primary copy of an entry is stopped, the redundant copy still exists on a running server.

To restart a cluster, use the PCF operator credentials to run the command:

```
cf update-service SERVICE-INSTANCE-NAME -c '{"restart": true}'
```

For example:

```
$ cf update-service my-cluster -c '{"restart": true}'
```

Changes to the Service Plan

Your PCF operator can change details of the service plan available on the Marketplace. If your operator changes the default value of one of the optional parameters, this does not affect existing service instances.

However, if your operator changes the allowed values of one of the optional parameters, existing instances that exceed the new limits are not affected, but any subsequent service updates that

change the optional parameter must adhere to the new limits.

For example, if the PCF operator changes the plan by decreasing the maximum value for `num_servers`, any future service updates must adhere to the new `num_servers` value limit.

You might see the following error message when attempting to update a service instance:

```
$ cf update-service my-cloudcache -c '{"num_servers": 5}'
Updating service instance my-cloudcache as admin...
FAILED
Server error, status code: 502, error code: 10001, message: Service broker error: Service cannot be updated at this time, please try again later or contact your operator for more information
```

This error message indicates that the operator has made an update to the plan used by this service instance. You must wait for the operator to apply plan changes to all service instances before you can make further service instance updates.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Monitoring Pivotal Cloud Cache Service Instances

PCC clusters and brokers emit service metrics. You can use any tool that has a corresponding Cloud Foundry nozzle to read and monitor these metrics in real time.

As an app developer, when you opt to use a data service, you should be prepared to:

- monitor the state of that service
- triage issues that occur with that service
- be notified of any concerns

If you believe an issue relates to the underlying infrastructure (network, CPU, memory, or disk), you will need to capture evidence and notify your platform team. The metrics described in this section can help in characterizing the performance and resource consumption of your service instance.

Pivotal Cloud Cache does not publish locator metrics for service instances created as Co-located Single VM plans or Co-located Multi VM plans.

Service Instance Metrics

In the descriptions of the metrics, KPI stands for Key Performance Indicator.

Member Count

`serviceinstance.MemberCount`

Description	Returns the number of members in the distributed system.
Metric Type	number
Suggested measurement	Every second

Measurement Type	count
Warning Threshold	less than the manifest member count
Suggested Actions	This depends on the expected member count, which is available in the BOSH manifest. If the number expected is different from the number emitted, this is a critical situation that may lead to data loss, and the reasons for node failure should be investigated by examining the service logs.
Why a KPI?	Member loss due to any reason can potentially cause data loss.

Total Available Heap Size

```
serviceinstance.TotalHeapSize
```

Description	Returns the total available heap, in megabytes, across all instance members.
Metric Type	number
Suggested measurement	Every second
Measurement Type	pulse
Why a KPI?	If the total heap size and used heap size are too close, the system might see thrashing due to GC activity. This increases latency.

Total Used Heap Size

```
serviceinstance.UsedHeapSize
```

Description	Returns the total heap used across all instance members, in megabytes.
Metric Type	number
Suggested measurement	Every second
Measurement Type	pulse
Why a KPI?	If the total heap size and used heap size are too close, the system might see thrashing due to GC activity. This increases latency.

Total Available Heap Size as a Percentage

```
serviceinstance.UnusedHeapSizePercentage
```

Description	Returns the proportion of total available heap across all instance members, expressed as a percentage.
Metric Type	percent
Suggested measurement	Every second
Measurement Type	compound metric

Warning Threshold	40%
Critical Threshold	10%
Suggested Actions	If this is a spike due to eviction catching up with insert frequency, then customers need to keep a close watch that it should not hit the RED marker. If there is no eviction, then horizontal scaling is suggested.
Why a KPI?	If the total heap size and used heap size are too close, the system might see thrashing due to GC activity. This increases latency.

Per Member Metrics

Memory Used as a Percentage

`member.UsedMemoryPercentage`

Description	RAM being consumed.
Metric Type	percent
Suggested measurement	Average over last 10 minutes
Measurement Type	average
Warning Threshold	75%
Critical Threshold	85%

Count of Java Garbage Collections

`member.GarbageCollectionCount`

Description	The number of times that garbage has been collected.
Metric Type	number
Suggested measurement	Sum over last 10 minutes
Measurement Type	count
Warning Threshold	Dependent on the IaaS and app use case.
Critical Threshold	Dependent on the IaaS and app use case.
Suggested Actions	Check the number of queries run against the system, which increases the deserialization of objects and increases garbage.
Why a KPI?	If the frequency of garbage collection is high, the system might see high CPU usage, which causes delays in the cluster.

CPU Utilization Percentage

`member.HostCpuUsage`

Description	This member's process CPU utilization, expressed as a percentage.
Metric Type	percent
Suggested measurement	Average over last 10 minutes
Measurement Type	average
Warning Threshold	85%
Critical Threshold	95%
Suggested Actions	If this is not happening with high GC activity, the system is reaching its limits. Horizontal scaling might help.
Why a KPI?	High CPU usage causes delayed responses and can also make the member non-responsive. This can cause the member to be kicked out of the cluster, potentially leading to data loss.

Average Latency of Get Operations

`member.GetsAvgLatency`

Description	The average latency of cache get operations, in nanoseconds.
Metric Type	number
Suggested measurement	Average over last 10 minutes
Measurement Type	average
Warning Threshold	Dependent on the IaaS and app use case.
Critical Threshold	Dependent on the IaaS and app use case.
Suggested Actions	If this is not happening with high GC activity, the system is reaching its limit. Horizontal scaling might help.
Why a KPI?	It is a good indicator of the overall responsiveness of the system. If this number is high, the service administrator should diagnose the root cause.

Average Latency of Put Operations

`member.PutsAvgLatency`

Description	The average latency of cache put operations, in nanoseconds.
Metric Type	number
Suggested measurement	Average over last 10 minutes
Measurement Type	average

Warning Threshold	Dependent on the IaaS and app use case.
Critical Threshold	Dependent on the IaaS and app use case.
Suggested Actions	If this is not happening with high GC activity, the system is reaching its limit. Horizontal scaling might help.
Why a KPI?	It is a good indicator of the overall responsiveness of the system. If this number is high, the service administrator should diagnose the root cause.

JVM pauses

```
member.JVMPauses
```

Description	The quantity of JVM pauses.
Metric Type	number
Suggested measurement	Sum over 2 seconds
Measurement Type	count
Warning Threshold	Dependent on the IaaS and app use case.
Critical Threshold	Dependent on the IaaS and app use case.
Suggested Actions	Check the cached object size; if it is greater than 1 MB, you may be hitting the limitation on JVM to garbage collect this object. Otherwise, you may be hitting the utilization limit on the cluster, and will need to scale up to add more memory to the cluster.
Why a KPI?	Due to a JVM pause, the member stops responding to “are-you-alive” messages, which may cause this member to be kicked out of the cluster.

File Descriptor Limit

```
member.FileDescriptorLimit
```

Description	The maximum number of open file descriptors allowed for the member’s host operating system.
Metric Type	number
Suggested measurement	Every second
Measurement Type	pulse
Why a KPI?	If the number of open file descriptors exceeds number available, it causes the member to stop responding and crash.

Open File Descriptors

```
member.TotalFileDescriptorOpen
```

Description	The current number of open file descriptors.
Metric Type	number
Suggested measurement	Every second
Measurement Type	pulse
Why a KPI?	If the number of open file descriptors exceeds number available, it causes the member to stop responding and crash.

Quantity of Remaining File Descriptors

```
member.FileDescriptorRemaining
```

Description	The number of available file descriptors.
Metric Type	number
Suggested measurement	Every second
Measurement Type	compound metric
Warning Threshold	1000
Critical Threshold	100
Suggested Actions	Scale horizontally to increase capacity.
Why a KPI?	If the number of open file descriptors exceeds number available, it causes the member to stop responding and crash.

Threads Waiting for a Reply

```
member.ReplyWaitsInProgress
```

Description	The quantity of threads currently waiting for a reply.
Metric Type	number
Suggested measurement	Average over the past 10 seconds
Measurement Type	pulse
Warning Threshold	1
Critical Threshold	10
Suggested Actions	If the value does not average to zero over the sample interval, then the member is waiting for responses from other members. There are two possible explanations: either another member is unhealthy, or the network is dropping packets. Check other members' health, and check for network issues.
Why a KPI?	Unhealthy members are excluded from the cluster, possibly leading to data loss.

Gateway Sender and Gateway Receiver Metrics

These are metrics emitted through the CF Nozzle for gateway senders and gateway receivers.

Queue Size for the Gateway Sender

```
gatewaySender.<sender-id>.EventQueueSize
```

Description	The current size of the gateway sender queue.
Metric Type	number
Measurement Type	count

Events Received at the Gateway Sender

```
gatewaySender.<sender-id>.EventsReceivedRate
```

Description	A count of the events coming from the region to which the gateway sender is attached. It is the count since the last time the metric was checked. The first time it is checked, the count is of the number of events since the gateway sender was created.
Metric Type	number
Measurement Type	count

Events Queued by the Gateway Sender

```
gatewaySender.<sender-id>.EventsQueuedRate
```

Description	A count of the events queued on the gateway sender from the region. This quantity of events might be lower than the quantity of events received, as not all received events are queued. It is a count since the last time the metric was checked. The first time it is checked, the count is of the number of events since the gateway sender was created.
Metric Type	number
Measurement Type	count

Events Received by the Gateway Receiver

```
gatewayReceiver.EventsReceivedRate
```

Description	A count of the events received from the gateway sender which will be applied to the region on the gateway receiver's site. It is the count since the last time the metric was checked. The first time it is checked, the count is of the number of events since the gateway receiver was created.
Metric Type	number
Measurement Type	count

Disk Metrics

These are metrics emitted through the CF Nozzle for disks.

Average Latency of Disk Writes

`diskstore.DiskWritesAvgLatency`

Description	The average latency of disk writes in nanoseconds.
Metric Type	number
Measurement Type	time in nanoseconds

Quantity of Bytes on Disk

`diskstore.TotalSpace`

Description	The total number of bytes on the attached disk.
Metric Type	number
Measurement Type	count

Quantity of Available Bytes on Disk

`diskstore.UseableSpace`

Description	The total number of bytes of available space on the attached disk.
Metric Type	number
Measurement Type	count

Experimental Metrics

These metrics are experimental. Any of these metrics may be removed without advance notice, and the current name of any of these metrics may change. Expect changes to these metrics.

These experimental metrics are specific to a region (`REGION-NAME`):

- `region.REGION-NAME.BucketCount`
- `region.REGION-NAME.CreatesRate`
- `region.REGION-NAME.DestroyRate`
- `region.REGION-NAME.EntrySize`
- `region.REGION-NAME.FullPath`
- `region.REGION-NAME.GetsRate`
- `region.REGION-NAME.NumRunningFunctions`
- `region.REGION-NAME.PrimaryBucketCount`
- `region.REGION-NAME.PutAllRate`
- `region.REGION-NAME.PutLocalRate`
- `region.REGION-NAME.PutRemoteRate`

- `region.REGION-NAME.PutsRate`
- `region.REGION-NAME.RegionType`
- `region.REGION-NAME.SystemRegionEntryCount`
- `region.REGION-NAME.TotalBucketSize`
- `region.REGION-NAME.TotalRegionCount`
- `region.REGION-NAME.TotalRegionEntryCount`

These experimental metrics are specific to a member:

- `member.AverageReads`
- `member.AverageWrites`
- `member.BytesReceivedRate`
- `member.BytesSentRate`
- `member.CacheServer`
- `member.ClientConnectionCount`
- `member.CreatesRate`
- `member.CurrentHeapSize`
- `member.DeserializationRate`
- `member DestroysRate`
- `member.FunctionExecutionRate`
- `member.GetRequestRate`
- `member.GetsRate`
- `member.MaxMemory`
- `member.MemberUpTime`
- `member.NumThreads`
- `member.PDXDeserializationRate`
- `member.PutAllRate`
- `member.PutRequestRate`
- `member.PutsRate`
- `member.TotalBucketCount`
- `member.TotalHitCount`
- `member.TotalMissCount`
- `member.TotalPrimaryBucketCount`

Total Memory Consumption

The BOSH `mem-check` errand calculates and outputs the quantity of memory used across all PCC service instances. This errand helps PCF operators monitor resource costs, which are based on memory usage.

From the director, run a BOSH command of the form:

```
bosh -d <service broker name> run-errand mem-check
```

With this command:

```
bosh -d cloudcache-service-broker run-errand mem-check
```

Here is an anonymized portion of example output from the `mem-check` errand for a two cluster deployment:

```
Analyzing deployment xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx1...
JVM heap usage for service instance xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx1
Used Total = 1204 MB
Max Total = 3201 MB

Analyzing deployment xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx2...
JVM heap usage for service instance xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx2
Used Total = 986 MB
Max Total = 3201 MB

JVM heap usage for all clusters everywhere:
Used Global Total = 2390 MB
Max Global Total = 6402 MB
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

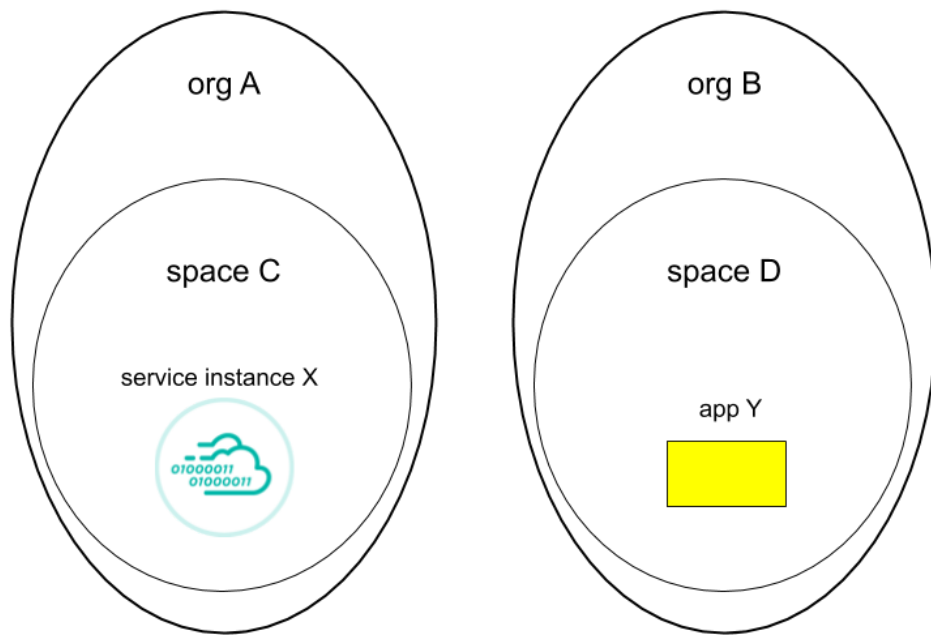
Service-Instance Sharing

Service-instance sharing for Cloud Cache permits read-only access to a Cloud Cache service instance by an app within a different space.

Setting up sharing has three steps:

1. A Cloud Foundry operator enables instance sharing as detailed in [Enable Service-Instance Sharing](#)
2. A developer shares a service instance
3. A developer binds the app to the shared service instance

These instructions require identification of the org and the space of both the service instance and the app. The following diagram names the components for use in the configuration instructions. Service instance X resides within space C, which is part of org A. App Y resides within space D, which is part of org B.



Share a Service Instance

The Cloud Cache service instance must be up and running prior to sharing.

To share the service instance:

1. An org A developer does a `cf login` with a space developer role. Target the space that contains the service to be shared: org A, space C.
2. The org A developer shares the the space with a command of the form

```
cf share-service SERVICE-X -s SPACE-D -o ORG-B
```

Replace `SERVICE-X` with the Cloud Cache service instance name. Replace `SPACE-D` with the space name where the app resides. Replace `ORG-B` with the org name where the app resides.

Bind an App to a Shared Service Instance

The app must be bound to the shared service instance prior to starting the app.

To bind the app to the shared service instance:

1. An org B developer does a `cf login` with a space developer role. Target the org and space that contains the app: org B, space D.
2. Verify that the Cloud Cache service instance is available and shared across the spaces in the output of the command:

```
$ cf services
```

3. The org B developer binds the app with a command of the form

```
cf bind APP-Y SERVICE-X
```

Replace `SERVICE-X` with the Cloud Cache service instance name. Replace `APP-Y` with the name of the app.

App Authentication

Apps that interact with a shared Cloud Cache service instance which resides in a different space will be given a set of read-only credentials. The app must acquire and use this set of credentials for authentication. Apps built with Spring Boot Data GemFire version 1.1.1 or a more recent version will automatically pick up the credentials, so these apps do not need to acquire the credentials.

The role of these credentials is authorized only for read access of region data.

The set of credentials are in the `VCAP_SERVICES` environment variable, with a role of `readonly`. The app must parse the `VCAP_SERVICES` environment variable to extract the credentials. The app uses the credentials to set a GemFire property that then gets passed to the `ClientCacheFactory` for the purpose of authentication prior to creating the cache.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Set Up Service Instances Across a Wide Area Network (WAN)

Two service instances may form a single distributed system across a WAN. The interaction of the two service instances may follow one of the patterns described in the section on [Design Patterns](#).

Call the two service instances A and B. The GemFire cluster within each service instance uses an identifier called a `distributed_system_id`. This example assigns `distributed_system_id = 1` to cluster A and `distributed_system_id = 2` to cluster B. GemFire gateway senders provide the communication path and construct that propagates region operations from one cluster to another. On the receiving end are GemFire gateway receivers. Creating a service instance also creates gateway receivers.

The procedures in this section assume that service instances communicate using TLS encryption.



Note: To set up more than two service instances across a WAN, set up the interaction between the first two service instances A and B following the directions in either [Set Up a Bidirectional System](#) or [Set Up a Unidirectional System](#), as appropriate. After that, set up the interaction between service instance A and another service instance (called C) following the directions in either [Set Up an Additional Bidirectional Interaction](#) or [Set Up an Additional Unidirectional Interaction](#), as appropriate.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Establishing Mutually Trusted TLS Credentials

In order for services instances in two foundations to communicate using TLS encryption, the CredHub “/services/tls_ca” certificate must be trusted in both foundations; otherwise, WAN connections with TLS will fail.

Assumptions

- You have two PCF foundations, A and B, with a network connection between them.
- You wish to establish a TLS-encrypted WAN connection between a service instance on Foundation A and a service instance on Foundation B.
- The [Preparing for TLS](#) procedure has been followed for each foundation, establishing a CredHub “/services/tls_ca” certificate for each.

Establish Mutual Trust

In order for services instances in two foundations to communicate with TLS, the CredHub “/services/tls_ca” certificate must be trusted in both foundations; otherwise, WAN connections requiring TLS will fail.

To be trusted in both foundations, their certificates must be signed by:

- a single CA that is identical in both foundations, or
- two CAs, one in each foundation, which is trusted by the other foundation.

If one of these conditions is satisfied, mutually trusted credentials are already in place; there is no need to execute the following procedure.

If the two foundations have different “/services/tls_ca” certificates that are not already mutually trusted, follow these steps to establish mutual trust.

Assuming you have two different PCF foundations, A and B, connected by a WAN.

1. Access the CredHub of Foundation A using instructions from [Access BOSH CredHub](#)
2. Fetch the certificate from Foundation A using CredHub:

```
credhub get -n /services/tls_ca -k certificate
```

3. Record the output.
4. Navigate to the **Ops Manager Installation Dashboard** of **Foundation B** and click the **BOSH Director** tile.
5. Click **Security**.
6. Append the contents of the new CA certificate to the old CA certificate under **Trusted Certificates**. Do not remove the old CA certificate.
7. Click **Save**.
8. Distribute the new CA certificate to your PCC VMs and regenerate each server certificate using the new CA:
 - Navigate back to the **Installation Dashboard**.
 - Click **Review Pending Changes**.

- Click **ERRANDS**.
- Select **Upgrade All Service Instances**.

9. Return to the **Installation Dashboard** in Ops Manager and click **Apply Changes**.

Repeat Steps 2 - 9 for Foundation B to put its services CA into Foundation A.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Set Up a Bidirectional System

This section describes how to set up a bidirectional WAN connection, using TLS encryption, between two PCC service instances in two different foundations. A bidirectional connection like this is needed if you wish to implement an active-active pattern, as described in [Bidirectional Replication Across a WAN](#).

Assumptions

- You have two PCF foundations, A and B, with a network connection between them.
- You wish to establish a TLS-encrypted WAN connection between a service instance on Foundation A and a service instance on Foundation B.
- The connection will be bidirectional, such that operations in each cluster can be replicated in the other.
- The [Preparing for TLS](#) procedure has been followed for each foundation, establishing a CredHub “/services/tls_ca” certificate for each.
- The [Establishing Mutually Trusted TLS Credentials](#) procedure has been followed, establishing mutually trusted TLS credentials between Foundations A and B.

Create Clusters

1. Log into Foundation A using Foundation A Cloud Foundry credentials.
2. Use the `cf create-service` command to create a service instance, which we will call Cluster A in this example. In the `cf create-service` command, use the `-c` option with the argument `"tls" : true` to enable TLS. This example also uses the `-c` option to set the `distributed_system_id` of Cluster A to “1”:

```
$ cf create-service p-cloudcache wan-cluster wan1 -c '{
  "tls" : true,
  "distributed_system_id" : 1 }'
```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation as create succeeded` when service creation is completed.

3. Log into Foundation B using Foundation B Cloud Foundry credentials.
4. Use the `cf create-service` command to create a service instance, which we will call Cluster B. In the `cf create-service` command, use the `-c` option with the argument `"tls"`

: `true` to enable TLS. This example also uses the `-c` option to set the `distributed_system_id` of Cluster B to “2”:

```
$ cf create-service p-cloudcache wan-cluster wan2 -c '{
  "tls" : true,
  "distributed_system_id" : 2 }'
```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation as create succeeded` when service creation is completed.

Create Service Keys

1. While logged in to Foundation A, create a service key for Cluster A. The contents of the service key differ based upon the cluster configuration, such as whether an authentication and enterprise single sign-on (SSO) system such as LDAP has been configured.

```
$ cf create-service-key wan1 k1
```

The service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster B in this example) to communicate with Cluster A:

```
Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ]
  },
}
```

```

"urls": {
  "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
  "pulse": "https://cloudcache-1.example.com/pulse"
},
"users": [
  {
    "password": "cl-op-ABC-password",
    "roles": [
      "cluster_operator"
    ],
    "username": "cluster_operator_ABC"
  },
  {
    "password": "dev-DEF-password",
    "roles": [
      "developer"
    ],
    "username": "developer_DEF"
  }
],
"wan": {}
}

```

2. While logged in to Foundation B, create a service key for Cluster B:

```
$ cf create-service-key wan2 k2
```

As above, the service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster A in this example) to communicate with Cluster B:

```

Getting key k2 for service instance destination as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    }
  },
  "remote_locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {

```

```

    "password": "gws-PQR-password",
    "username": "gateway_sender_PQR"
  }
],
},
"urls": {
  "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
  "pulse": "https://cloudcache-2.example.com/pulse"
},
"users": [
  {
    "password": "cl-op-JKL-password",
    "roles": [
      "cluster_operator"
    ],
    "username": "cluster_operator_JKL"
  },
  {
    "password": "dev-MNO-password",
    "roles": [
      "developer"
    ],
    "username": "developer_MNO"
  }
],
"wan": {}
}

```

Establish a Bidirectional Connection

To establish bidirectional communication between Clusters A and B, The `remote_clusters` element in each must be updated with the contents of the other's `remote_cluster_info`, which contains three elements:

- The `recursors` array and the `remote_locators` array enable communication between clusters.
- The `trusted_sender_credentials` element identifies a cluster from which data can be accepted for replication. For example, when Cluster B has been updated with Cluster A's `trusted_sender_credentials`, then Cluster B can accept and store data sent from Cluster A.

In order to implement a bidirectional WAN connection between two clusters, each must have the `trusted_sender_credentials` of the other.

1. While logged in to Foundation A, update the Cluster A service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster B service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster A to communicate with Cluster B, and to accept data from Cluster B:

```

$ cf update-service wan1 -c '
{
  "remote_clusters": [
    {

```

```

    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    ]
  }
}
}'
Updating service instance wan1 as admin

```

2. While logged in to Foundation B, update the Cluster B service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster A service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster B to communicate with Cluster A, and to accept data from Cluster A:

```

$ cf update-service wan2 -c '
{
  "remote_clusters":[
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.5:1053",
          "10.0.8.7:1053",
          "10.0.8.6:1053"
        ]
      }
    }
  ]
  "remote_locators":[
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ]
  "trusted_sender_credentials":[
    {
      "username": "gateway_sender_GHI",
      "password":"gws-GHI-password"
    }
  ]
}
}'
Updating service instance wan2 as admin

```

3. To verify that a service instance has been correctly updated, delete and recreate the cluster service key. The recreated service key will have the same user identifiers and passwords as its predecessor, and will reflect the changes you specified in the recent `cf update-service`

commands. In particular, the `wan{}` element at the end of a cluster's service key should be populated with the other cluster's remote connection information. For example, to verify that the Cluster A service key was updated correctly, log in as Cluster A administrator and issue these commands to delete and recreate the Cluster A service key:

```
$ cf delete-service-key wan1 k1
...
$ cf create-service-key wan1 k1
```

Verify that the `wan{}` field of the Cluster A service key contains a `remote_clusters` element which specifies contact information for Cluster B, including the `remote_locators` array, and `trusted_sender_credentials`:

```
Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-ABC-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_ABC"
      }
    ]
  }
}
```

```
{
  "password": "dev-DEF-password",
  "roles": [
    "developer"
  ],
  "username": "developer_DEF"
}
],
"wan": {
  "remote_clusters": [
    {
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        "gateway_sender_PQR"
      ]
    }
  ]
}
}
```

Similarly, to verify that the Cluster B service key was updated correctly, log in as Cluster B administrator and issue these commands to delete and recreate the Cluster B service key:

```
$ cf delete-service-key wan2 k2
...
$ cf create-service-key wan2 k2
```

Verify that the `wan{}` field of the Cluster B service key contains a `remote_clusters` element which specifies contact information for Cluster A, including the `remote_locators` array, and `trusted_sender_credentials`:

```
Getting key k2 for service instance destination as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    }
  },
  "remote_locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
```

```

    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "password": "gws-PQR-password",
      "username": "gateway_sender_PQR"
    }
  ],
  },
  "urls": {
    "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
    "pulse": "https://cloudcache-2.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-JKL-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_JKL"
    },
    {
      "password": "dev-MNO-password",
      "roles": [
        "developer"
      ],
      "username": "developer_MNO"
    }
  ],
  "wan": {
    "remote_clusters": [
      {
        "remote_locators": [
          "id1.locator.services-subnet.service-instance-id.bosh[55221]",
          "id2.locator.services-subnet.service-instance-id.bosh[55221]",
          "id3.locator.services-subnet.service-instance-id.bosh[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ]
  }
}

```

Create Gateway Senders and Regions

On each cluster, create a gateway sender that sends to the other cluster, then create regions on each cluster that have the same name and type. Note: the gateway sender must be created before the region that uses it, so there is a window in which region operations that occur after the region is created on Cluster A, but before the corresponding region is created on Cluster B, will be lost.

1. While logged in to Foundation A, use gfsh to create the Cluster A gateway sender and the region.

- Connect using gfsh following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
- Create the Cluster A gateway sender. The required `remote-distributed-system-id` option identifies the `distributed-system-id` of the destination cluster. It is 2 for this example:

```
gfsh>create gateway-sender --id=send_to_2 --remote-distributed-system-id=2 --enable-persistence=true
```

- Create the Cluster A region. The `gateway-sender-id` associates region operations with a specific gateway sender. The region must have an associated gateway sender in order to propagate region events across the WAN.

```
gfsh>create region --name=regionX --gateway-sender-id=send_to_2 --type=PARTITION_REDUNDANT
```

2. While logged in to Foundation B, use gfsh to create the Cluster B gateway sender and region.

- Connect using gfsh following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
- Create the Cluster B gateway sender:

```
gfsh>create gateway-sender --id=send_to_1 --remote-distributed-system-id=1 --enable-persistence=true
```

- Create the Cluster B region:

```
gfsh>create region --name=regionX --gateway-sender-id=send_to_1 --type=PARTITION_REDUNDANT
```

Verify Bidirectional WAN Setup

This verification uses gfsh to put a region entry into each of Cluster A and Cluster B, in order to observe that the entry appears in the other cluster.

1. While logged in to Foundation A, run gfsh and connect following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
2. Verify that Cluster A has a complete set of gateway senders and gateway receivers:

```
gfsh>list gateways
```

Also verify that there are no queued events in the gateway senders.

3. Log in to Foundation B in a separate terminal window, then run gfsh and connect following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
4. Verify that Cluster B has a complete set of gateway senders and gateway receivers:


```
gfsh>list gateways
```

Also verify that there are no queued events in the gateway senders.

- From the Cluster A gfsh connection, use gfsh to perform a `put` operation. This example assumes that both the key and the value for the entry are strings. The gfsh `help put` command shows the put command's options. Here is an example:

```
gfsh>put --region=regionX --key=mykey1 --value=stringvalue1
Result      : true
Key Class   : java.lang.String
Key         : mykey1
Value Class : java.lang.String
Old Value   : null
```

- Wait approximately 30 seconds, then use the Cluster B gfsh connection to perform a `get` of the same key that was put into the region on Cluster A.

```
gfsh>get --region=regionX --key=mykey1
Result      : true
Key Class   : java.lang.String
Key         : mykey1
Value Class : java.lang.String
Value       : "stringvalue1"
```

You should see that Cluster B has the value.

- Repeat steps 5 and 6 to perform a `put` operation operation on Cluster B and verify that the entry is sent to Cluster A.
- Remove both test entries from both clusters with two gfsh commands, issuing the commands on one of the clusters. WAN replication will cause the removal of the test entries on the other cluster.

```
gfsh>remove --region=regionX --key=mykey1
Result      : true
Key Class   : java.lang.String
Key         : mykey1

gfsh>remove --region=regionX --key=mykey2
Result      : true
Key Class   : java.lang.String
Key         : mykey2
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Set Up a Unidirectional System with TLS

This section describes how to set up a unidirectional WAN connection, using TLS encryption, between two PCC instances in two different foundations, such that all operations in Cluster A are replicated in Cluster B. Two design patterns that use unidirectional replication are described in [Blue-Green Disaster Recovery](#) and [CQRS Pattern Across a WAN](#).

Assumptions

- You have two PCF foundations, A and B, with a network connection between them.
- You wish to establish a TLS-encrypted WAN connection between a service instance on Foundation A and a service instance on Foundation B.
- The connection will be unidirectional, such that all operations in Cluster A are replicated in Cluster B, but Cluster B does not send operations to Cluster A.
- The [Preparing for TLS](#) procedure has been followed for each foundation, establishing a CredHub “/services/tls_ca” certificate for each.
- The [Establishing Mutually Trusted TLS Credentials](#) procedure has been followed, establishing mutually trusted TLS credentials between Foundations A and B.

Create Clusters

1. Log into Foundation A using Foundation A Cloud Foundry credentials.
2. Use the `cf create-service` command to create a service instance, which we will call Cluster A in this example. In the `cf create-service` command, use the `-c` option with the argument `"tls" : true` to enable TLS. This example also uses the `-c` option to set the `distributed_system_id` of Cluster A to “1”:

```
$ cf create-service p-cloudcache wan-cluster wan1 -c '{
  "tls" : true,
  "distributed_system_id" : 1 }'
```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation` as `create succeeded` when service creation is completed.

3. Log into Foundation B using Foundation B Cloud Foundry credentials.
4. Use the `cf create-service` command to create a service instance, which we will call Cluster B. In the `cf create-service` command, use the `-c` option with the argument `"tls" : true` to enable TLS. This example also uses the `-c` option to set the `distributed_system_id` of Cluster B to “2”:

```
$ cf create-service p-cloudcache wan-cluster wan2 -c '{
  "tls" : true,
  "distributed_system_id" : 2 }'
```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation` as `create succeeded` when service creation is completed.

Create Service Keys

1. While logged in to Foundation A, create a service key for Cluster A. The contents of the service key differ based upon the cluster configuration, such as whether an authentication and enterprise single sign-on (SSO) system such as LDAP has been configured.

```
$ cf create-service-key wan1 k1
```

The service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster B in this example) to communicate with Cluster A:

```
Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-ABC-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_ABC"
      },
      {
        "password": "dev-DEF-password",
        "roles": [
          "developer"
        ],
        "username": "developer_DEF"
      }
    ]
  }
}
```

```
"wan": {}
}
```

2. While logged in to Foundation B, create a service key for Cluster B:

```
$ cf create-service-key wan2 k2
```

As above, the service key contains generated credentials, in a JSON element called `remote_cluster_info`, that enable other clusters (Cluster A in this example) to communicate with Cluster B:

```
Getting key k2 for service instance destination as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
      "pulse": "https://cloudcache-2.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-JKL-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_JKL"
      },
      {
        "password": "dev-MNO-password",
        "roles": [
```

```

    "developer"
  ],
  "username": "developer_MNO"
}
],
"wan": {}
}

```

Establish a Unidirectional TLS Connection

To establish unidirectional communication between Clusters A and B, The `remote_clusters` element in each must be updated with selected contents of the other's `remote_cluster_info`, which contains three elements:

- The `recursors` array and the `remote_locators` array enable communication between clusters.
- The `trusted_sender_credentials` element identifies a cluster from which data can be accepted for replication. For example, when Cluster B has been updated with Cluster A's `trusted_sender_credentials`, then Cluster B can accept and store data sent from Cluster A.

In order to implement a unidirectional WAN connection in which Cluster A sends data to Cluster B:

- Each cluster must be updated with the `recursors` array and the `remote_locators` array of the other.
- Cluster B must be updated with the `trusted_sender_credentials` of Cluster A.
- Cluster A must *NOT* be updated with the `trusted_sender_credentials` of Cluster B.

Follow these steps to configure a unidirectional connection in which Cluster A is the active member, and Cluster B accepts data for replication, but does not send operations to Cluster A:

1. While logged into Foundation A, update the Cluster A service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster B service key `remote_cluster_info` element, including Cluster B's `recursors` array and the `remote_locators` array, but **NOT** Cluster B's `trusted_sender_credentials`. This allows Cluster A to send messages to Cluster B, but blocks data flow from Cluster B to Cluster A.

```

$ cf update-service wan1 -c '
{
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet-2.service-instance-id-2.bosh": [
          "10.1.16.7:1053",
          "10.1.16.6:1053",
          "10.1.16.8:1053"
        ]
      },
    },
    "remote_locators": [
      "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
    ],
  ],
}
'

```

```
} '
Updating service instance wan1 as admin
```

2. While logged in to Foundation B, update the Cluster B service instance, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster A service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster B to communicate with Cluster A, and to accept data from Cluster A:

```
$ cf update-service wan2 -c '
{
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.5:1053",
          "10.0.8.7:1053",
          "10.0.8.6:1053"
        ]
      }
    }
  ]
  "remote_locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ]
  "trusted_sender_credentials": [
    {
      "username": "gateway_sender_GHI",
      "password": "gws-GHI-password"
    }
  ]
}
]'
Updating service instance wan2 as admin
```

3. To verify that each service instance has been correctly updated, delete and recreate the cluster service key. The recreated service key will have the same user identifiers and passwords as its predecessor, and will reflect the changes you specified in the recent `cf update-service` commands. In particular the `wan{}` elements at the end of each cluster's service key should be populated with the other cluster's remote connection information. For example, to verify that the Cluster A service key was updated correctly, log in as Cluster A administrator and issue these commands to delete and recreate the Cluster A service key:

```
$ cf delete-service-key wan1 k1
...
$ cf create-service-key wan1 k1
```

Verify that the `wan{}` field of the Cluster A service key contains a `remote_clusters` element which specifies contact information for Cluster B, including the `remote_locators` array, but **NOT** Cluster B's `trusted_sender_credentials`:

```
Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
```

```

"gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
"locators": [
  "id1.locator.services-subnet.service-instance-id.bosh[55221]",
  "id2.locator.services-subnet.service-instance-id.bosh[55221]",
  "id3.locator.services-subnet.service-instance-id.bosh[55221]"
],
"remote_cluster_info": {
  "recursors": {
    "services-subnet.service-instance-id.bosh": [
      "10.0.8.6:1053",
      "10.0.8.7:1053",
      "10.0.8.5:1053"
    ]
  },
  "remote_locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "password": "gws-GHI-password",
      "username": "gateway_sender_GHI"
    }
  ],
  "urls": {
    "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
    "pulse": "https://cloudcache-1.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-ABC-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABC"
    },
    {
      "password": "dev-DEF-password",
      "roles": [
        "developer"
      ],
      "username": "developer_DEF"
    }
  ],
  "wan": {
    "remote_clusters": [
      {
        "remote_locators": [
          "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
          "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
          "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
        ]
      }
    ]
  }
]

```

```
}
}
```

Similarly, to verify that the Cluster B service key was updated correctly, log in as Cluster B administrator and issue these commands to delete and recreate the Cluster B service key:

```
$ cf delete-service-key wan2 k2
...
$ cf create-service-key wan2 k2
```

Verify that the `wan{}` field of the Cluster B service key contains a `remote_clusters` element which specifies contact information for Cluster A, including the `remote_locators` array, and `trusted_sender_credentials`:

```
Getting key k2 for service instance destination as admin...

{
  "distributed_system_id": "2",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-2.service-instance-id-2.bosh": [
        "10.1.16.7:1053",
        "10.1.16.6:1053",
        "10.1.16.8:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
      "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-2.example.com/gemfire/v1",
      "pulse": "https://cloudcache-2.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-JKL-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_JKL"
      }
    ]
  }
}
```



```

    },
    {
      "password": "dev-MNO-password",
      "roles": [
        "developer"
      ],
      "username": "developer_MNO"
    }
  ],
  "wan": {
    "remote_clusters": [
      {
        "remote_locators": [
          "id1.locator.services-subnet.service-instance-id.bosh[55221]",
          "id2.locator.services-subnet.service-instance-id.bosh[55221]",
          "id3.locator.services-subnet.service-instance-id.bosh[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ]
  }
}

```

Create a Gateway Sender and Regions

Create a gateway sender on the active cluster that sends to the passive cluster, then create regions on each cluster that have the same name and type. Note: the gateway sender must be created before the region that uses it, so there is a window in which region operations that occur after the region is created on Cluster A, but before the corresponding region is created on Cluster B, will be lost.

1. While logged in to Foundation A, use `gfsh` to create the Cluster A gateway sender and the region.
 - Connect using `gfsh` following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the Cluster and the data.
 - Create the Cluster A gateway sender. The required `remote-distributed-system-id` option identifies the `distributed-system-id` of the destination cluster. It is 2 for this example:

```
gfsh>create gateway-sender --id=send_to_2 --remote-distributed-system-id=2 --enable-persistence=true
```

- Create the Cluster A region. The `gateway-sender-id` associates region operations with a specific gateway sender. The region must have an associated gateway sender in order to propagate region events across the WAN.

```
gfsh>create region --name=regionX --gateway-sender-id=send_to_2 --type=PARTITION_REDUNDANT
```

2. While logged in to Foundation B, use gfsh to create the Cluster B region. (Because Cluster B is the passive member in this unidirectional connection, it does not need a gateway sender.)
 - o Connect using gfsh following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
 - o Create the Cluster B region:

```
gfsh>create region --name=regionX --type=PARTITION_REDUNDANT
```

Verify Unidirectional WAN Setup

This verification uses gfsh to put a region entry into Cluster A, in order to observe that the entry appears in Cluster B.

1. While logged in to Foundation A, run gfsh and connect following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
2. Verify that Cluster A has a complete set of gateway senders and gateway receivers:

```
gfsh>list gateways
```

Also verify that there are no queued events in the gateway senders.

3. Log in to Foundation B in a separate terminal window, then run gfsh and connect following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
4. Verify that Cluster B has a complete set of gateway receivers (Cluster B should have no gateway senders):

```
gfsh>list gateways
```

5. From the Cluster A gfsh connection, use gfsh to perform a `put` operation. This example assumes that both the key and the value for the entry are strings. The gfsh `help put` command shows the put command's options. Here is an example:

```
gfsh>put --region=regionX --key=mykey1 --value=stringvalue1
Result      : true
Key Class   : java.lang.String
Key         : mykey1
Value Class : java.lang.String
Old Value   : null
```

6. Wait approximately 30 seconds, then use the Cluster B gfsh connection to perform a `get` of the same key that was put into the region on Cluster A.

```
gfsh>get --region=regionX --key=mykey1
Result      : true
Key Class   : java.lang.String
Key         : mykey1
```

```
Value Class : java.lang.String
Value      : "stringvalue1"
```

You should see that Cluster B has the value.

7. From the Cluster A gfsh connection, remove the test entry from Cluster A. WAN replication will cause the removal of the test entry from Cluster B.

```
gfsh>remove --region=regionX --key=mykey1
Result      : true
Key Class   : java.lang.String
Key         : mykey1
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Set Up an Additional Bidirectional Interaction

This section describes how to add a bidirectional TLS-encrypted WAN connection to an additional PCC service instance, once an initial setup is in place for a first pair of PCC service instances.

Call the first pair of PCC service instances Cluster A and Cluster B, which reside on Foundation A and Foundation B, respectively. This set of directions sets up an interaction between Cluster A and a third service instance, Cluster C, which resides on third foundation, Foundation C.

Assumptions

- You have already established a TLS-encrypted WAN connection between a service instance on Foundation A, referred to here as Cluster A, and a service instance on Foundation B, which we'll call Cluster B, following the [Set Up a Bidirectional System](#) procedure.
- You wish to add an additional TLS-encrypted WAN connection between Cluster A and a third service instance on Foundation C, which we will call Cluster C.
- The connection will be bidirectional, such that an operation on either cluster can be replicated in the other.
- The [Preparing for TLS](#) procedure has been followed for Foundation C, establishing a CredHub “/services/tls_ca” certificate.
- The [Establishing Mutually Trusted TLS Credentials](#) procedure has been followed, establishing mutually trusted TLS credentials between Foundations A and C.

Get Cluster A's Remote Credentials

1. Log into Foundation A using Foundation A Cloud Foundry credentials.
2. View the Cluster A service key and save a copy of the `remote_cluster_info` element, which includes the `recursors` array, the `remote_locators` array, and `trusted_sender_credentials`. These are the credentials that enable other clusters to communicate with Cluster A.

```
$ cf service-key wan1 k1
Getting key k1 for service instance wan1 as admin...
```

```

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-ABC-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_ABC"
      },
      {
        "password": "dev-DEF-password",
        "roles": [
          "developer"
        ],
        "username": "developer_DEF"
      }
    ],
    "wan": {
      "remote_clusters": [
        {
          "remote_locators": [
            "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
            "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
            "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
          ],

```

```

    "trusted_sender_credentials": [
      "gateway_sender_PQR"
    ]
  }
]
}
}

```

Create the New Cluster

The GemFire cluster within each service instance uses an identifier called a `distributed_system_id`. This example assumes the assignment of `distributed_system_id = 1` for cluster A, `distributed_system_id = 2` for cluster B, and `distributed_system_id = 3` for cluster C.

1. Log into Foundation C using Foundation C Cloud Foundry credentials.
2. Use the `cf create-service` command to create a service instance, which we will call Cluster C in this example. In the `cf create-service` command, use the `-c` option to specify the following arguments:
 - `"tls" : true` to enable TLS
 - `"distributed_system_id" : 3` to set the distributed system id of Cluster C to “3”
 - `"remote_clusters" : [CLUSTER-A-CREDENTIALS]` to enable communication with Cluster A, where CLUSTER-A-CREDENTIALS are the contents of Cluster A’s `remote_cluster_info` element.

The following command specifies the Cluster A credentials you copied in an earlier step, enabling Cluster C to communicate with and receive data from Cluster A:

```

$ cf create-service p-cloudcache wan-cluster wan3 -c '{
  "tls" : true,
  "distributed_system_id" : 3
  "remote_clusters":[
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.5:1053",
          "10.0.8.7:1053",
          "10.0.8.6:1053"
        ]
      }
    }
  ],
  "remote_locators":[
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "trusted_sender_credentials":[
    {
      "username": "gateway_sender_GHI",
      "password":"gws-GHI-password"
    }
  ]
}]
}'

```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation as create succeeded` when service creation is completed.

3. While logged in to Foundation C, create a service key for Cluster C.

```
$ cf create-service-key wan3 k3
Creating service key wan3 for service instance k3 ...
OK
```

Display the service key and save a copy of the `remote_cluster_info` element, which includes the `recursors` array, the `remote_locators` array and `trusted_sender_credentials`. These are the credentials that enable other clusters to communicate with Cluster C.

```
$ cf create-service-key wan3 k3

Getting key k3 for service instance destination as admin...

{
  "distributed_system_id": "3",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-3.service-instance-id-3.bosh": [
        "10.2.32.7:1053",
        "10.2.32.6:1053",
        "10.2.32.8:1053"
      ]
    },
  },
  "remote_locators": [
    "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "username": "gateway_sender_XYZ",
      "password": "gws-XYZ-password"
    }
  ],
  "urls": {
    "gfsh": "https://cloudcache-3.example.com/gemfire/v1",
    "pulse": "https://cloudcache-3.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-STU-password",
      "roles": [
```

```

    "cluster_operator"
  ],
  "username": "cluster_operator_STU"
},
{
  "password": "dev-VWX-password",
  "roles": [
    "developer"
  ],
  "username": "developer_VWX"
}
],
"wan": {
  "remote_clusters": [
    {
      "remote_locators": [
        "id1.locator.services-subnet.service-instance-id.bosh[55221]",
        "id2.locator.services-subnet.service-instance-id.bosh[55221]",
        "id3.locator.services-subnet.service-instance-id.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        "gateway_sender_GHI"
      ]
    }
  ]
}
}
}

```

4. While logged in to Foundation A, update the Cluster A service instance to add the Cluster C credentials, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster C service key `remote_cluster_info` element, including the `recursors` array, `remote_locators` array, and `trusted_sender_credentials`. This allows Cluster A to communicate with Cluster C, and to accept data from Cluster C.

IMPORTANT: You must also retain the credentials for all clusters with which Cluster A interacts, in this example those of Cluster B. The existing credentials can be copied from the `remote_cluster_info` entry in Cluster B's service key, before updating the Cluster A service instance to append credentials for the additional cluster.

The following example adds Cluster C's credentials to Cluster A's `remote_clusters` element:

```

$ cf update-service wan1 -c '
{
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet-2.service-instance-id-2.bosh": [
          "10.1.16.7:1053",
          "10.1.16.6:1053",
          "10.1.16.8:1053"
        ]
      },
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",

```

```

    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
  ],
  "trusted_sender_credentials": [
    {
      "username": "gateway_sender_PQR",
      "password": "gws-PQR-password",
    }
  ],
},
{
  "recursors": {
    "services-subnet-3.service-instance-id-3.bosh": [
      "10.2.32.7:1053",
      "10.2.32.6:1053",
      "10.2.32.8:1053"
    ]
  },
  "remote_locators": [
    "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "username": "gateway_sender_XYZ",
      "password": "gws-XYZ-password"
    }
  ]
}
]
}'
Updating service instance wan1 as admin

```

Verify the completion of the service update prior to continuing to the next step. Output from the `cf services` command will show the `last operation` as `update succeeded` when service update is completed.

5. To verify that a service instance has been correctly updated, delete and recreate the cluster service key. The recreated service key will have the same user identifiers and passwords as its predecessor, and will reflect the changes you specified in the recent `cf update-service` commands. In particular, the `wan{}` element at the end of a cluster's service key should be populated with the other cluster's remote connection information. For example, to verify that the Cluster A service key was updated correctly, log in as Cluster A administrator and issue these commands to delete and recreate the Cluster A service key:

```

$ cf delete-service-key wan1 k1
...
$ cf create-service-key wan1 k1

```

Verify that the `wan{}` field of the Cluster A service key contains a `remote_clusters` element which specifies contact information for Cluster B, including the `remote_locators` array, and `trusted_sender_credentials`:


```
$ cf service-key wan1 k1
```

```
Getting key k1 for service instance wan1 as admin...
```

```
{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "tls-enabled": "true",
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "cl-op-ABC-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_ABC"
      },
      {
        "password": "dev-DEF-password",
        "roles": [
          "developer"
        ],
        "username": "developer_DEF"
      }
    ],
    "wan": {
      "remote_clusters": [
        {
          "remote_locators": [
```

```

    "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
    "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    "gateway_sender_PQR"
  ]
},
{
  "remote_locators": [
    "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    "gateway_sender_XYZ"
  ]
}
]
}
}
}

```

Create Gateway Senders and Regions

- While logged in to Foundation A, use gfsH to create the cluster A gateway sender and alter the existing region.
 - Connect using gfsH following the instructions in [Connect with gfsH over HTTPS](#) with a role that is able to manage both the cluster and the data.
 - Create the cluster A gateway sender. The required `remote-distributed-system-id` option identifies the `distributed-system-id` of the destination cluster. It is 3 for this example:

```
gfsH>create gateway-sender --id=send_to_3 --remote-distributed-system-id=3 --enable-persistence=true
```

- Alter the existing cluster A region so that it specifies all gateway senders associated with the region. There are two gateway senders in this example, one that goes to cluster B and a second that goes to cluster C.

```
gfsH>alter region --name=regionX --gateway-sender-id=send_to_2,send_to_3
```

- While logged in to Foundation C, use gfsH to create the cluster C gateway sender and region.

- Connect using gfsH following the instructions in [Connect with gfsH over HTTPS](#) with a role that is able to manage both the cluster and the data.
- Create the cluster C gateway sender:

```
gfsH>create gateway-sender --id=send_to_1 --remote-distributed-system-id=1 --enable-persistence=true
```

- Create the cluster C region:

```
gfsh>create region --name=regionX --gateway-sender-id=send_to_1 --type=PARTITION_REDUNDANT
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Set Up an Additional Unidirectional Interaction

This section describes how to add a unidirectional TLS-encrypted WAN connection to an additional PCC service instance, once an initial setup is in place for a first pair of PCC service instances.

Call the first pair of PCC service instances Cluster A and Cluster B, which reside on Foundation A and Foundation B, respectively. This set of directions sets up an interaction between Cluster A and a third service instance, Cluster C, which resides on third foundation, Foundation C.

Assumptions

- You have already established a TLS-encrypted WAN connection between a service instance on Foundation A, referred to here as Cluster A, and a service instance on Foundation B, which we'll call Cluster B, following the [Set Up a Unidirectional System](#) procedure.
- You wish to add an additional TLS-encrypted WAN connection between Cluster A and a third service instance on Foundation C, which we will call Cluster C.
- The connection will be unidirectional, such that all operations in Cluster A are replicated in Cluster C, but Cluster C does not send operations to Cluster A.
- The [Preparing for TLS](#) procedure has been followed for Foundation C, establishing a CredHub “/services/tls_ca” certificate.
- The [Establishing Mutually Trusted TLS Credentials](#) procedure has been followed, establishing mutually trusted TLS credentials between Foundations A and C.

Get Cluster A's Remote Credentials

1. Log into Foundation A using Foundation A Cloud Foundry credentials.
2. View the Cluster A service key and save a copy of the `remote_cluster_info` element, which includes the `recursors` array, the `remote_locators` array, and `trusted_sender_credentials`. These are the credentials that enable other clusters to communicate with Cluster A.

```
$ cf service-key wan1 k1
Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
```

```

"locators": [
  "id1.locator.services-subnet.service-instance-id.bosh[55221]",
  "id2.locator.services-subnet.service-instance-id.bosh[55221]",
  "id3.locator.services-subnet.service-instance-id.bosh[55221]"
],
"remote_cluster_info": {
  "recursors": {
    "services-subnet.service-instance-id.bosh": [
      "10.0.8.6:1053",
      "10.0.8.7:1053",
      "10.0.8.5:1053"
    ]
  },
  "remote_locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "trusted_sender_credentials": [
    {
      "password": "gws-GHI-password",
      "username": "gateway_sender_GHI"
    }
  ],
  "urls": {
    "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
    "pulse": "https://cloudcache-1.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-ABC-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABC"
    },
    {
      "password": "dev-DEF-password",
      "roles": [
        "developer"
      ],
      "username": "developer_DEF"
    }
  ],
  "wan": {
    "remote_clusters": [
      {
        "remote_locators": [
          "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
          "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
          "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
        ],
      }
    ]
  }
}

```

Create the New Cluster

The GemFire cluster within each service instance uses an identifier called a `distributed_system_id`. This example assumes the assignment of `distributed_system_id = 1` for cluster A, `distributed_system_id = 2` for cluster B, and `distributed_system_id = 3` for cluster C.

1. Log into Foundation C using Foundation C Cloud Foundry credentials.
2. Use the `cf create-service` command to create a service instance, which we will call Cluster C in this example. In the `cf create-service` command, use the `-c` option to specify the following arguments:
 - `"tls" : true` to enable TLS
 - `"distributed_system_id" : 3` to set the distributed system id of Cluster C to “3”
 - `remote_clusters" : [CLUSTER-A-CREDENTIALS]` to enable communication with Cluster A, where CLUSTER-A-CREDENTIALS are the contents of Cluster A's `remote_cluster_info` element.

The following command specifies the Cluster A credentials you copied in an earlier step, enabling Cluster C to communicate with and receive data from Cluster A:

```
$ cf create-service p-cloudcache wan-cluster wan3 -c '{
  "tls" : true,
  "distributed_system_id" : 3
  "remote_clusters":[
    {
      "recursors": {
        "services-subnet.service-instance-id.bosh": [
          "10.0.8.5:1053",
          "10.0.8.7:1053",
          "10.0.8.6:1053"
        ]
      }
    }
  ],
  "remote_locators":[
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "trusted_sender_credentials":[
    {
      "username": "gateway_sender_GHI",
      "password":"gws-GHI-password"
    }
  ]
}]
}'
```

Verify the completion of service creation prior to continuing to the next step. Output from the `cf services` command will show the `last operation as create succeeded` when service creation is completed.

3. While logged in to Foundation C, create a service key for Cluster C.

```
$ cf create-service-key wan3 k3
Creating service key wan3 for service instance k3 ...
```

OK

Display the service key and save a copy of the `remote_cluster_info` element, which includes the `recursors` array, the `remote_locators` array and `trusted_sender_credentials`. These are the credentials that enable other clusters to communicate with Cluster C.

```
$ cf create-service-key wan3 k3

Getting key k3 for service instance destination as admin...

{
  "distributed_system_id": "3",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
    "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet-3.service-instance-id-3.bosh": [
        "10.2.32.7:1053",
        "10.2.32.6:1053",
        "10.2.32.8:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
      "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
      "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "username": "gateway_sender_XYZ",
        "password": "gws-XYZ-password"
      }
    ]
  },
  "urls": {
    "gfsh": "https://cloudcache-3.example.com/gemfire/v1",
    "pulse": "https://cloudcache-3.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-STU-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_STU"
    },
    {
      "password": "dev-VWX-password",
      "roles": [
        "developer"
      ]
    }
  ],
}
```

```

    "username": "developer_VWX"
  }
],
"wan": {
  "remote_clusters": [
    {
      "remote_locators": [
        "id1.locator.services-subnet.service-instance-id.bosh[55221]",
        "id2.locator.services-subnet.service-instance-id.bosh[55221]",
        "id3.locator.services-subnet.service-instance-id.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        "gateway_sender_GHI"
      ]
    }
  ]
}
}
}

```

4. While logged in to Foundation A, update the Cluster A service instance to add the Cluster C credentials, using the `-c` option to specify a `remote_clusters` element that includes the contents of the Cluster C service key `remote_cluster_info` element, including the `recursors` array and the `remote_locators` array, but **NOT** the `trusted_sender_credentials`. This allows Cluster A to send messages to Cluster C, but blocks data flow from Cluster C to Cluster A.

IMPORTANT: You must also retain the credentials for all clusters with which Cluster A interacts, in this example those of Cluster B. The existing `remote_locators` and `recursors` credentials can be copied from the `remote_cluster_info` entry in Cluster B's service key, before updating the Cluster A service instance to append credentials for the additional cluster. Do not include Cluster B's `trusted_sender_credentials`.

The following example adds Cluster C's credentials to Cluster A's `remote_clusters` element:

```

$ cf update-service wan1 -c '
{
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet-2.service-instance-id-2.bosh": [
          "10.1.16.7:1053",
          "10.1.16.6:1053",
          "10.1.16.8:1053"
        ]
      },
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
      ]
    },
    {
      "recursors": {
        "services-subnet-3.service-instance-id-3.bosh": [

```

```

    "10.2.32.7:1053",
    "10.2.32.6:1053",
    "10.2.32.8:1053"
  ]
},
"remote_locators": [
  "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
  "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
  "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
]
}
}
]
}'
Updating service instance wan1 as admin

```

Verify the completion of the service update prior to continuing to the next step. Output from the `cf services` command will show the `last operation` as `update succeeded` when service update is completed.

5. To verify that a service instance has been correctly updated, delete and recreate the cluster service key. The recreated service key will have the same user identifiers and passwords as its predecessor, and will reflect the changes you specified in the recent `cf update-service` commands. In particular, the `wan{}` element at the end of a cluster's service key should be populated with the other cluster's remote connection information. For example, to verify that the Cluster A service key was updated correctly, log in as Cluster A administrator and issue these commands to delete and recreate the Cluster A service key:

```

$ cf delete-service-key wan1 k1
...
$ cf create-service-key wan1 k1

```

Verify that the `wan{}` field of the Cluster A service key contains a `remote_clusters` element which specifies contact information for Cluster B, including Cluster B's `remote_locators` array, but **NOT** `trusted_sender_credentials`:

```

$ cf service-key wan1 k1

Getting key k1 for service instance wan1 as admin...

{
  "distributed_system_id": "1",
  "gfsh_login_string": "connect
--url=https://cloudcache-url.com/gemfire/v1
--user=cluster_operator_user --password=pass --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",

```



```

    "10.0.8.5:1053"
  ]
},
"remote_locators": [
  "id1.locator.services-subnet.service-instance-id.bosh[55221]",
  "id2.locator.services-subnet.service-instance-id.bosh[55221]",
  "id3.locator.services-subnet.service-instance-id.bosh[55221]"
],
"trusted_sender_credentials": [
  {
    "password": "gws-GHI-password",
    "username": "gateway_sender_GHI"
  }
],
"tls-enabled": "true",
"urls": {
  "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
  "pulse": "https://cloudcache-1.example.com/pulse"
},
"users": [
  {
    "password": "cl-op-ABC-password",
    "roles": [
      "cluster_operator"
    ],
    "username": "cluster_operator_ABC"
  },
  {
    "password": "dev-DEF-password",
    "roles": [
      "developer"
    ],
    "username": "developer_DEF"
  }
],
"wan": {
  "remote_clusters": [
    {
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
      ]
    },
    {
      "remote_locators": [
        "id1.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
        "id2.locator.services-subnet-3.service-instance-id-3.bosh[55221]",
        "id3.locator.services-subnet-3.service-instance-id-3.bosh[55221]"
      ]
    }
  ]
}
}

```

Create Gateway Senders and Regions

1. While logged in to Foundation A, use gfsh to create the cluster A gateway sender and alter the existing region.

- Connect using gfsh following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
- Create the cluster A gateway sender. The required `remote-distributed-system-id` option identifies the `distributed-system-id` of the destination cluster. It is 3 for this example:

```
gfsh>create gateway-sender --id=send_to_3 --remote-distributed-system-id=3 --enable-persistence=true
```

- Alter the existing cluster A region so that it specifies all gateway senders associated with the region. There are two gateway senders in this example, one that goes to cluster B and a second that goes to cluster C.

```
gfsh>alter region --name=regionX --gateway-sender-id=send_to_2,send_to_3
```

2. While logged in to Foundation C, use gfsh to create the cluster C region. (Because Cluster C is a passive member in this unidirectional connection, it does not need a gateway sender.)
 - Connect using gfsh following the instructions in [Connect with gfsh over HTTPS](#) with a role that is able to manage both the cluster and the data.
 - Create the cluster C region:

```
gfsh>create region --name=regionX --gateway-sender-id=send_to_1 --type=PARTITION_REDUNDANT
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

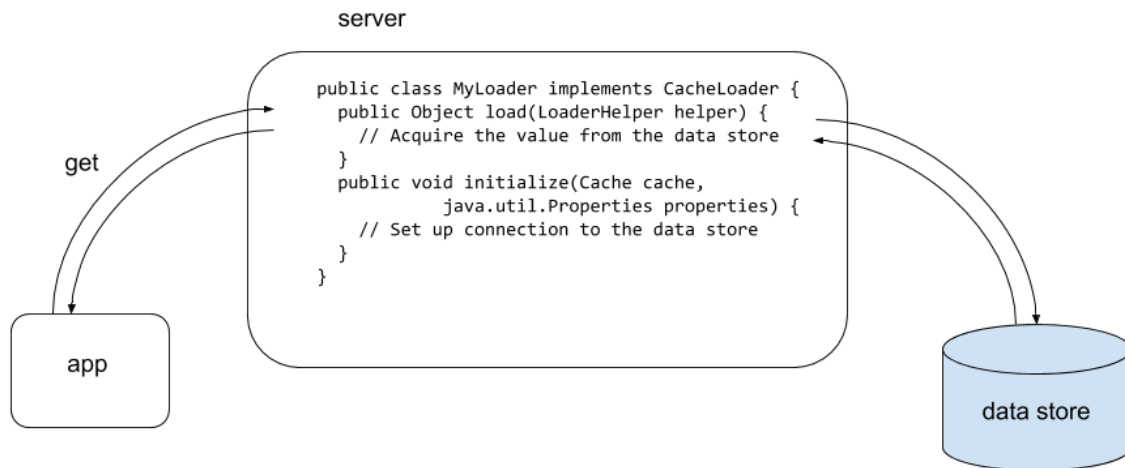
Setting Up Servers for an Inline Cache

See [The Inline Cache](#) for an introductory description of an inline cache. The implementation of an inline cache requires custom code deployed on the GemFire servers to interact with the backend data store for read misses and for writes.

The custom code always implements a cache loader for read misses. The custom code and configuration setup differs for writes. A write-behind implementation uses an asynchronous event queue (AEQ) and an AEQ listener. A write-through implementation uses a cache writer.

Implement a Cache Loader for Read Misses

An app's get operation is a cache read. If the desired entry is in the region, it is a cache hit, and the value is quickly returned to the app. If the desired entry is not in the region, it is a cache miss. For an inline cache, that value is acquired from the backend data store. You implement the `CacheLoader` interface to handle cache misses. Each cache miss invokes the `CacheLoader.load` method. The `CacheLoader.load` method must acquire and return the value for the specified key. See the [VMware Tanzu GemFire API Documentation](#) for the interface's details.



The value returned from the `CacheLoader.load` method will be put into the region and then returned to the waiting app, completing the app's get operation. Since the app blocks while waiting for the result of the get operation, design the `CacheLoader.load` method to acquire the value as quickly as possible.

The `CacheLoader` implementation must be thread-safe. You will deploy the implementation to the servers during configuration.

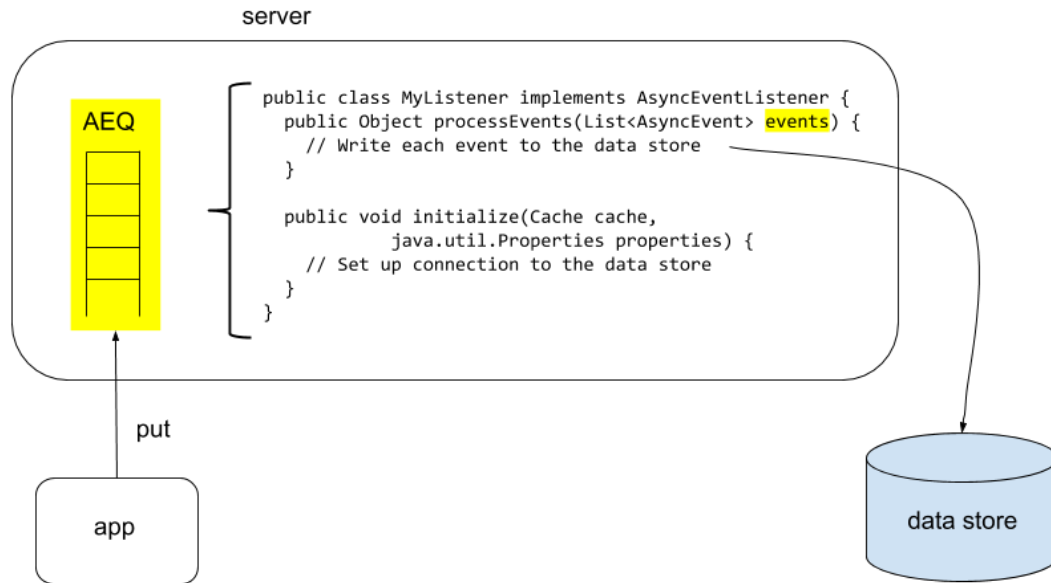
The `CacheLoader.load` method queries the backend data store for the desired entry. That communication between the server process and the backend data store requires a connection, and establishing a connection is likely to use a set of credentials. You provide a custom implementation of the `CacheLoader.initialize` method to establish the connection.

You specify the credentials during configuration with the `gfsh create region` command by adding the JSON description to the `--cache-loader` option. The credentials will be passed as parameters to the invoked `CacheLoader.initialize` method as part of the `CacheLoader` instance construction.

Implement an Asynchronous Event Queue and Cache Listener for Write Behind

An app's put operation is a cache write. For a write-behind implementation, the value is placed into the region, and it will also be asynchronously written to the backend data store, allowing the app's write operation to complete without waiting for the backend-data-store write to complete.

An asynchronous event queue (AEQ) to queue the write events together with an implementation of the `AsyncEventListener` interface provides the desired behavior. See the [VMware Tanzu GemFire API Documentation](#) for the interface's details.



With a configured AEQ, all put operations first create or update the entry in the hosted region on the server and then add the event to the AEQ.

You provide a custom implementation of the `AsyncEventListener` interface. Your `AsyncEventListener.processEvents` method's task is to iterate through the events in the AEQ, writing each newly created or updated entry in the AEQ to the backend data store. The `AsyncEventListener.processEvents` method is invoked when either the AEQ holds a configured quantity of events, or a configured quantity of time has elapsed since the earliest entry entered the AEQ.

The communication between the server process and the backend data store to do the writes requires a connection, and establishing a connection is likely to use a set of credentials. You provide a custom implementation of the `AsyncEventListener.initialize` method to establish the connection.

You specify the credentials during configuration in the `gfsh create async-event-queue` command with the `--listener-param` option as described in [Configure Using gfsh for Write Behind](#). The credentials will be passed as parameters to the invoked `AsyncEventListener.initialize` method as part of `AsyncEventListener` instance construction.

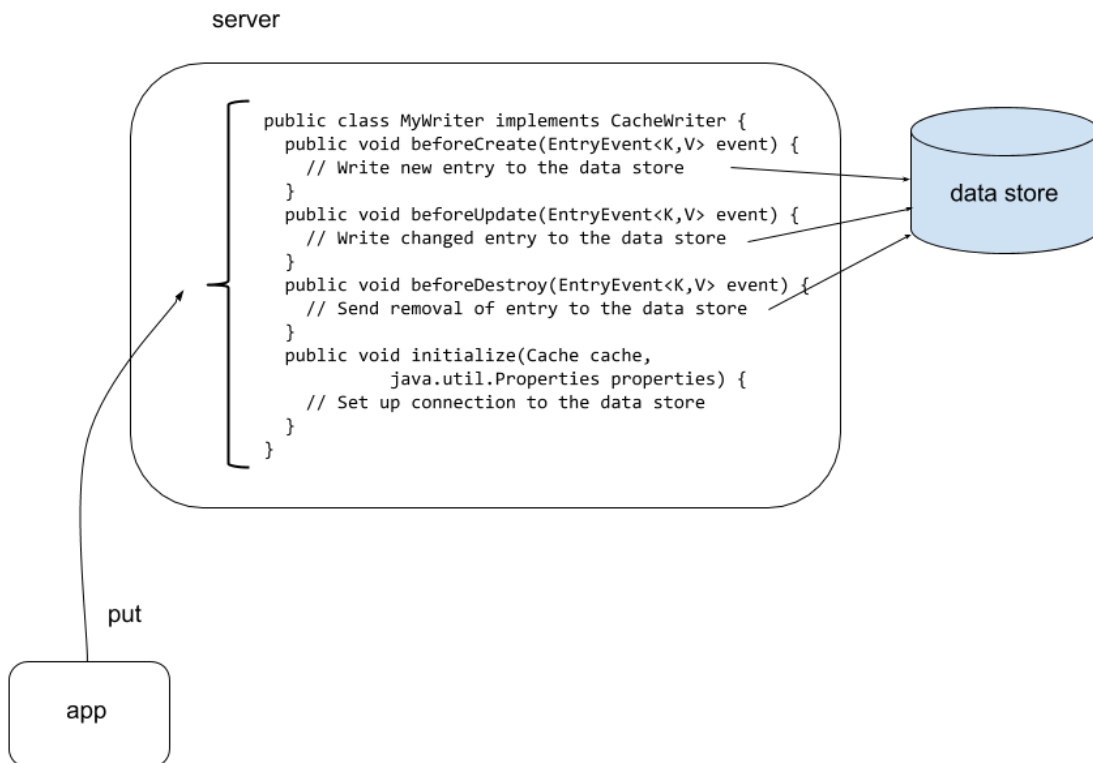
Your configuration will specify the AEQ as persistent, such that it does not lose queued backend-data-store writes across unexpected server restarts.

Implement a Cache Writer for Write Through

An app's put operation is a cache write. For a write-through implementation, the value will be written to the backend data store prior to being placed into the region. After both writes, the app's put operation completes.

An implementation of the `CacheWriter` interface implementation provides the correct behavior for write through. See the [VMware Tanzu GemFire API Documentation](#) for the interface's details. You provide a custom implementation of the `CacheWriter.beforeCreate` method to handle backend-data-store writes for put operations that add a new entry to the region. You provide a custom implementation of the `CacheWriter.beforeUpdate` method to handle backend-data-store writes for put operations that modify an existing entry in the region. You provide a custom implementation of `CacheWriter.beforeDestroy`, as appropriate, to handle an update of the backend data store for a region operation that removes an entry.

The `CacheWriter` implementation must be thread-safe. You will deploy the implementation to the servers during configuration.



Communication between the server process and the backend data store to do the writes requires a connection, and establishing a connection is likely to use a set of credentials. You provide a custom implementation of the `CacheWriter.initialize` method to establish the connection.

Specify the credentials in the `gfsh create region` command during configuration as described in [Configure Using gfsh for Write Through](#). Add the JSON description to the `--cache-writer` option. The credentials will be passed as parameters to the invoked `CacheWriter.initialize` method as part of the `CacheWriter` instance construction.

Configure Using gfsh for Write Behind

Follow this procedure to deploy your custom implementations of the interfaces to the servers, create the AEQ, and configure the region to use the AEQ and the deployed interface

implementations.

1. Follow the directions in [Connect with gfsh over HTTPS](#) to connect to the cluster with a role that is able to manage both the cluster and the data.
2. Deploy the cache loader and the AEQ listener code to the servers within the PCC service instance:

```
gfsh>deploy --jars=/path/to/MyLoader.jar,/path/to/MyListener.jar
```

3. Create the AEQ, assigning a name for the AEQ (called **WB-AEQ** in this example), specifying the AEQ listener, and specifying the AEQ listener's parameters:

```
gfsh>create async-event-queue --id=WB-AEQ \
  --parallel=true --persistent \
  --listener=com.myCompany.MyListener \
  --listener-param=url#jdbc:db2:SAMPLE,username#admin,password#gobbledeegook
```

The persistence of the AEQ uses the default disk store, since no disk store is specified in this command.

4. Create the region, specifying the cache loader and the assigned AEQ name.

```
gfsh>create region --name=myRegion --type=PARTITION_REDUNDANT \
  --cache-loader=com.myCompany.MyLoader{'url':'jdbc:db2:SAMPLE','username':'admin',password:'gobbledeegook'}
  --async-event-queue-id=WB-AEQ
```

Configure Using gfsh for Write Through

Follow this procedure to deploy your custom implementations of the interfaces to the servers, and configure the region to use the deployed interface implementations.

1. Follow the directions in [Connect with gfsh over HTTPS](#) to connect to the cluster with a role that is able to manage both the cluster and the data.
2. Deploy the cache loader and the cache writer code to the servers within the PCC service instance:

```
gfsh>deploy --jars=/path/to/MyLoader.jar,/path/to/MyWriter.jar
```

3. Create the region, specifying the cache loader and the cache writer:

```
gfsh>create region --name=myRegion --type=PARTITION_REDUNDANT \
  --cache-loader=com.myCompany.MyLoader{'url':'jdbc:db2:SAMPLE','username':'admin',password:'gobbledeegook'}
  --cache-writer=com.myCompany.MyWriter{'url':'jdbc:db2:SAMPLE','username':'admin',password:'gobbledeegook'}
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Accessing a Service Instance

After you have created a service instance, you can access it. The `gfsh` command line tool provides cluster maintenance and data access functionality. Many `gfsh` commands require an authenticated connection that can be set up with the `gfsh connect` command.

Connecting requires these one-time, setup steps:

1. Create a service key.
2. Create a truststore.
3. Acquire the correct version of `gfsh`.
4. Set a `JAVA_ARGS` environment variable.

Create a Service Key

A service key provides a way to access your service instance outside the scope of a deployed CF app. Run `cf create-service-key SERVICE-INSTANCE-NAME KEY-NAME` to create a service key. Replace `SERVICE-INSTANCE-NAME` with the name you chose for your service instance. Replace `KEY-NAME` with a name of your choice. You can use this name to refer to your service key with other commands.

```
$ cf create-service-key my-cloudcache my-service-key
```

Run `cf service-key SERVICE-INSTANCE-NAME KEY-NAME` to view the newly created service key.

```
$ cf service-key my-cloudcache my-service-key
```

The service key reveals the configuration of your service instance. It shows addresses and ports of its locators, and contains an element called `remote_cluster-info` that provides fields by which the service instance can be reached from other service instances. The service key specifies two URLs, one URL used to connect the `gfsh` client to the service instance, and another URL used to view the Pulse dashboard in a web browser, which allows monitoring of the service instance status.

If an authentication and enterprise single sign-on (SSO) system such as LDAP has *not* been configured, the service key shows role-based login credentials as username/password pairs. If an SSO system is in place, it handles user credentials, so they will not appear in the service key.

If an authentication and enterprise SSO system has been configured, then the `cf service-key` returns output in the following format, without role-based login credentials:

```
{
  "distributed_system_id": "0",
  "gfsh_login_string": "connect
--url=https://cloudcache-1.example.com/gemfire/v1
--user=cluster_operator_XXX --password=cluster_operator-password --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
```

```

    "10.0.8.7:1053",
    "10.0.8.5:1053"
  ],
},
"remote_locators": [
  "id1.locator.services-subnet.service-instance-id.bosh[55221]",
  "id2.locator.services-subnet.service-instance-id.bosh[55221]",
  "id3.locator.services-subnet.service-instance-id.bosh[55221]"
],
"trusted_sender_credentials": [
  {
    "password": "gws-GHI-password",
    "username": "gateway_sender_GHI"
  }
],
},
"urls": {
  "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
  "pulse": "https://cloudcache-1.example.com/pulse"
},
"wan": {
  "remote_clusters": [
    {
      "recursors": {
        "services-subnet-2.service-instance-id-2.bosh": [
          "10.1.16.7:1053",
          "10.1.16.6:1053",
          "10.1.16.8:1053"
        ]
      },
      "remote_locators": [
        "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
        "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
      ],
      "trusted_sender_credentials": [
        {
          "password": "gws-PQR-password",
          "username": "gateway_sender_PQR"
        }
      ]
    }
  ]
}
}

```

If an authentication and enterprise single sign-on (SSO) system such as LDAP has *not* been configured, then the `cf service-key` returns output in a format that includes role-based login credentials:

```

{
  "distributed_system_id": "0",
  "gfsh_login_string": "connect
--url=https://cloudcache-1.example.com/gemfire/v1
--user=cluster_operator_XXX --password=cluster_operator-password --skip-ssl-validation",
  "locators": [
    "id1.locator.services-subnet.service-instance-id.bosh[55221]",
    "id2.locator.services-subnet.service-instance-id.bosh[55221]",
  ]
}

```



```

    "id3.locator.services-subnet.service-instance-id.bosh[55221]"
  ],
  "remote_cluster_info": {
    "recursors": {
      "services-subnet.service-instance-id.bosh": [
        "10.0.8.6:1053",
        "10.0.8.7:1053",
        "10.0.8.5:1053"
      ]
    },
    "remote_locators": [
      "id1.locator.services-subnet.service-instance-id.bosh[55221]",
      "id2.locator.services-subnet.service-instance-id.bosh[55221]",
      "id3.locator.services-subnet.service-instance-id.bosh[55221]"
    ],
    "trusted_sender_credentials": [
      {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    ],
    "urls": {
      "gfsh": "https://cloudcache-1.example.com/gemfire/v1",
      "pulse": "https://cloudcache-1.example.com/pulse"
    },
    "users": [
      {
        "password": "developer-password",
        "roles": [
          "developer"
        ],
        "username": "developer_XXX"
      },
      {
        "password": "cluster_operator-password",
        "roles": [
          "cluster_operator"
        ],
        "username": "cluster_operator_XXX"
      }
    ],
    "wan": {
      "remote_clusters": [
        {
          "recursors": {
            "services-subnet-2.service-instance-id-2.bosh": [
              "10.1.16.7:1053",
              "10.1.16.6:1053",
              "10.1.16.8:1053"
            ]
          },
          "remote_locators": [
            "id1.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
            "id2.locator.services-subnet-2.service-instance-id-2.bosh[55221]",
            "id3.locator.services-subnet-2.service-instance-id-2.bosh[55221]"
          ],
          "trusted_sender_credentials": [
            {

```

```

    "password": "gws-PQR-password",
    "username": "gateway_sender_PQR"
  }
]
}
]
}
}
}

```

This service key for a PCC installation in which an authentication and enterprise single sign-on (SSO) system such as LDAP has *not* been configured specifies these user roles that are predefined, for interacting with and within the cluster:

- The cluster operator administers the pool, performing operations such as creating and destroying regions, and creating gateway senders. The identifier assigned for this role is of the form `cluster_operator_XXX`, where `XXX` is a unique string generated upon service instance creation and incorporated in this user role's name.
- The developer does limited cluster administration such as region creation, and the developer role is expected to be used by applications that are interacting with region entries. The developer does CRUD operations on regions. The identifier assigned for this role is of the form `developer_XXX`, where `XXX` is a unique string generated upon service instance creation and incorporated in this user role's name.

Connect with gfsh over HTTPS

When connecting over HTTPS, you must use the same certificate you use to secure traffic into Pivotal Application Service (PAS); that is, the certificate you use where your TLS termination occurs. See [Determine Your TLS Termination](#).

Before you can connect, you must create a truststore.

Create a Truststore

To create a truststore, use the same certificate you used to configure TLS termination. We suggest using the `keytool` command line utility to create a truststore file.

1. Locate the certificate you use to configure TLS termination. See [Determine Your TLS Termination](#).
2. Using your certificate, run the `keytool` command:

```
keytool -import -file CERTIFICATE.CER -keystore TRUSTSTORE-FILE-PATH -storetype JKS
```

where

- `CERTIFICATE.CER` is your certificate file
 - `TRUSTSTORE-FILE-PATH` is the path to the location where you want to create the truststore file, including the name you want to give the file
3. When you run this command, you are prompted to enter a keystore password. Create a password and remember it!
 4. When prompted for the certificate details, enter **yes** to trust the certificate.

The following example shows how to run `keytool` and what the output looks like:

```
$ keytool -import -file /tmp/loadbalancer.cer -keystore /tmp/truststore/prod.myTrustStore -storetype JKS
Enter keystore password:
Re-enter new password:
Owner: CN=*.url.example.com, OU=Cloud Foundry, O=Pivotal, L=New York, ST=New York, C=US
Issuer: CN=*.url.example.com, OU=Cloud Foundry, O=Pivotal, L=New York, ST=New York, C=US
Serial number: bd84912917b5b665
Valid from: Sat Jul 29 09:18:43 EDT 2017 until: Mon Apr 07 09:18:43 EDT 2031
Certificate fingerprints:
    MD5:  A9:17:B1:C9:6C:0A:F7:A3:56:51:6D:67:F8:3E:94:35
    SHA1: BA:DA:23:09:17:C0:DF:37:D9:6F:47:05:05:00:44:6B:24:A1:3D:77
    SHA256: A6:F3:4E:B8:FF:8F:72:92:0A:6D:55:6E:59:54:83:30:76:49:80:92:52:3D:91:4D:61:1C:A1:29:D3:BD:56:57
    Signature algorithm name: SHA256withRSA
    Version: 3

Extensions:

#1: ObjectId: 2.5.29.10 Criticality=true
BasicConstraints:[
    CA:true
    PathLen:0
]

#2: ObjectId: 2.5.29.11 Criticality=false
SubjectAlternativeName [
    DNSName: *.sys.url.example.com
    DNSName: *.apps.url.example.com
    DNSName: *.uaa.sys.url.example.com
    DNSName: *.login.sys.url.example.com
    DNSName: *.url.example.com
    DNSName: *.ws.url.example.com
]

Trust this certificate? [no]: yes
Certificate was added to keystore
```

Establish the Connection with HTTPS

After you have created the truststore, you can use the Pivotal GemFire command line interface, `gfsh`, to connect to the cluster over HTTPS.

1. Acquire the correct `gfsh` by downloading the correct Pivotal GemFire ZIP archive from [Pivotal Network](#). See [View Available Plans](#) to identify the correct Pivotal GemFire version. Note that a JDK or JRE will also be required.



Note: An attempt to use the wrong `gfsh` version will result in an error message indicating that there is a version mismatch.

2. Unzip the Pivotal GemFire ZIP archive. `gfsh` is within the `bin` directory in the expanded Pivotal GemFire. Use `gfsh` with Unix or `gfsh.bat` with Windows.
3. Run `gfsh`, and then issue a `connect` command of the form:

```
connect --use-http=true --url=HTTPS-gfsh-URL
--trust-store=TRUSTSTORE-FILE-PATH --trust-store-password=PASSWORD
--user=USER-NAME --password=USER-PASSWORD
```

The `HTTPS-gfsh-URL` is the gfsh URL from the service key. See [Create Service Keys](#) for instructions on how to view the service key. `TRUSTSTORE-FILE-PATH` is the path to the trustStore file you created in [Create a Truststore](#), and `PASSWORD` is the associated password you created. If you omit the `--trust-store-password` option from the command line, you will be prompted to enter the password.

For an installation configured *without* an authentication and enterprise single sign-on (SSO) system such as LDAP, the `USER-NAME` and `USER-PASSWORD` are taken from the service key, and they will either be for the developer role or for the cluster operator role, depending on the gfsh operations to be performed.

For an installation configured *with* an authentication and enterprise single sign-on (SSO) system such as LDAP, the `USER-NAME` and `USER-PASSWORD` are your credentials as issued and set up by your organization within your SSO system.

Establish the Connection with HTTPS in a Development Environment

When working in a non-production, development environment, a developer may choose to work in a less secure manner by eliminating the truststore and SSL mutual authentication.

The steps to establish the `gfsh` connection become:

1. Acquire `gfsh` by downloading the correct Pivotal GemFire ZIP archive from [Pivotal Network](#). See [View Available Plans](#) to identify the correct Pivotal GemFire version. Note that a JDK or JRE will also be required, as specified in the release notes.
2. Unzip the Pivotal GemFire ZIP archive. `gfsh` is within the `bin` directory in the expanded Pivotal GemFire. Use `gfsh` with Unix or `gfsh.bat` with Windows.
3. Acquire the `connect` command (the `gfsh_login_string`) from the service key. This command will connect as the `cluster_operator` role. This `connect` command assumes that the installation is configured *without* an authentication and enterprise single sign-on (SSO) system such as LDAP.
4. Run `gfsh`, and then issue the acquired `connect` command:

```
gfsh>connect --url=https://cloudcache-1.example.com/gemfire/v1
--user=cluster_operator_XXX --password=cluster_operator-password
--skip-ssl-validation
```

At each of the nine `gfsh` prompts that ask for keystore, truststore, and SSL details, hit `Enter` to step through the prompts and connect.

Determine Your TLS Termination

To connect your PCC service instance using `gfsh`, you will need the certificate from where your TLS termination occurs. The TLS termination may be at the Gorouter, at the HAProxy, or at your load balancer. Request the needed certificate from your Pivotal Cloud Foundry (PCF) operator.

The PCF operator determines the location of your TLS termination:

1. Bring up the Ops Manager dashboard.
2. Click on the PAS product tile.
3. Click on the Networking section under the Settings tab.

The choice of TLS termination is labeled with **Configure support for the X-Forwarded-Client-Cert**.

If the choice names the **Router** or **HAProxy**, the certificate is in the same section, labeled with **Certificate and Private Key for HAProxy and Router**.

If the choice names the **infrastructure load balancer**, then the PCF operator can retrieve the certificate from the load balancer.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Using gfsh

gfsh Command Restrictions

Developers may invoke all `gfsh` commands. Given credentials with sufficient permissions, those `gfsh` command will be executed. However, not all `gfsh` commands are supported. An invocation of an unsupported command may lead to incorrect results. Those results range from ineffective results to inconsistent region entries. **Do not use these listed `gfsh` commands; each has an explanation why it must not be used.**

These `gfsh start` commands will bring up members contrary to the configured plan. Their configuration will be wrong, and their existence is likely to contribute to data loss. Since they are not part of the configured plan, any upgrade will not include them, and if they were to stop or crash, the BOSH Director will not restart them.

- `gfsh start locator`
- `gfsh start server`

These cluster stop commands will temporarily stop the member or cluster. However, the BOSH Director will notice that members are not running and restart them. So, these commands will be ineffective:

- `gfsh stop locator`
- `gfsh stop server`
- `gfsh shutdown`

These Lucene-related commands are not supported:

- `gfsh create lucene index`
- `gfsh describe lucene index`
- `gfsh destroy lucene index`
- `gfsh list lucene indexes`
- `gfsh search lucene`

These JNDI binding-related commands are not supported:

- `gfsh create jndi-binding`
- `gfsh describe jndi-binding`
- `gfsh destroy jndi-binding`
- `gfsh list jndi-binding`

This `configure` command will instill configuration contrary to the already-configured plan. Since it is not part of the configured plan, any upgrade will not include it. Therefore, do not use:

- `gfsh configure pdx`

The create of a gateway receiver will never be appropriate for any situation. The Cloud Cache cluster will already have gateway receivers, and there is no situation in which the cluster can benefit from creating more. Therefore, do not use:

- `gfsh create gateway receiver`

Do Not Export from a GemFire Cluster to a Cloud Cache Cluster

While the expectation is that configuration and data can be exported from a GemFire cluster and then imported into a Cloud Cache cluster, this does **not** work. Using export and import commands will not have the desired effect of migration from one cluster to another. The import of cluster configuration requires a state that cannot be provided by a Cloud Cache cluster. The Cloud Cache cluster will already have its configuration, and upon restart or upgrade, that same configuration will be used. Given that the configuration cannot be imported, data import is problematic.

Therefore, do not use:

- `gfsh import cluster-configuration`
- `gfsh import data`

Create Regions

After connecting with `gfsh` with a role that is able to manage a cluster's data, you can define a new cache region.

The following command creates a partitioned region with a single redundant copy:

```
gfsh>create region --name=my-cache-region --type=PARTITION_REDUNDANT_HEAP_LRU
      Member      | Status
-----|-----
cacheserver-z2-1 | Region "/my-cache-region" created on "cacheserver-z2-1"
cacheserver-z3-2 | Region "/my-cache-region" created on "cacheserver-z3-2"
cacheserver-z1-0 | Region "/my-cache-region" created on "cacheserver-z1-0"
cacheserver-z1-3 | Region "/my-cache-region" created on "cacheserver-z1-3"
```

See [Region Design](#) for guidelines on choosing a region type.

You can test the newly created region by writing and reading values with `gfsh`:

```
gfsh>put --region=/my-cache-region --key=test --value=thevalue
Result      : true
Key Class   : java.lang.String
```

```

Key          : test
Value Class  : java.lang.String
Old Value    : NULL

gfsh>get --region=/my-cache-region --key=test
Result       : true
Key Class    : java.lang.String
Key          : test
Value Class  : java.lang.String
Value       : thevalue

```

In practice, you should perform these get/put operations from a deployed PCF app. To do this, you must bind the service instance to these apps.

Working with Disk Stores

Persistent regions and regions that overflow upon eviction use disk stores. Use `gfsh` to create or destroy a disk store.

Create a Disk Store

To create a disk store for use with a persistent or overflow type of region:

1. Use the directions in [Connect with gfsh over HTTPS](#) to connect to the Cloud Cache service instance with a role that is able to manage a cluster's data.
2. Create the disk store with a `gfsh` command of the form:

```
create disk-store --name=<name-of-disk-store>
--dir=<relative/path/to/diskstore/directory>
```

Specify a relative path for the disk store location. That relative path will be created within `/var/vcap/store/gemfire-server/`. For more details on further options, see the Pivotal GemFire [create disk-store Command Reference Page](#).

Destroy a Disk Store

To destroy a disk store:

1. Use the directions in [Connect with gfsh over HTTPS](#) to connect to the Cloud Cache service instance with a role that is able to manage a cluster's data.
2. Destroy the disk store with a `gfsh` command of the form:

```
destroy disk-store --name=<name-of-disk-store>
```

For more details on further options, see the Pivotal GemFire [destroy disk-store Command Reference Page](#).

Use the Pulse Dashboard

You can access the Pulse dashboard for a service instance in a web browser using the Pulse URL specified in the service key you created following the directions in [Create Service Keys](#).

Access Service Instance Metrics

To access service metrics, you must have **Enable Plan** selected under **Service Plan Access** on the page where you configure your tile properties. (For details, see the [Configure Service Plans](#) page.)

Cloud Cache service instances output metrics to the Loggregator Firehose. You can use the Firehose plugin to view metrics output on the cf CLI directly. For more information about using the Firehose plugin, see [Installing the Loggregator Firehose Plugin for CLI](#).

You can also connect the output to a Firehose nozzle. Nozzles are programs that consume data from the Loggregator Firehose. For example, to connect the output to the nozzle for Datadog, see [Datadog Firehose Nozzle](#) on GitHub.

Cloud Cache supports metrics for the whole cluster and metrics for each member. Each server and locator in the cluster outputs metrics.

Service Instance (Cluster-wide) Metrics

- `serviceinstance.MemberCount`: the number of VMs in the cluster
- `serviceinstance.TotalHeapSize`: the total MBs of heap available in the cluster
- `serviceinstance.UsedHeapSize`: the total MBs of heap in use in the cluster

Member (per-VM) Metrics

- `member.GarbageCollectionCount`: the number of JVM garbage collections that have occurred on this member since startup
- `member.CpuUsage`: the percentage of CPU time used by the GemFire process
- `member.GetsAvgLatency`: the avg latency of GET requests to this GemFire member
- `member.PutsAvgLatency`: the avg latency of PUT requests to this GemFire member
- `member.JVMPauses`: the number of JVM pauses that have occurred on this member since startup
- `member.FileDescriptorLimit`: the number of files this member allows to be open at once
- `member.TotalFileDescriptorOpen`: the number of files this member has open now
- `member.FileDescriptorRemaining`: the number of files that this member could open before hitting its limit
- `member.TotalHeapSize`: the number of megabytes allocated for the heap
- `member.UsedHeapSize`: the number of megabytes currently in use for the heap
- `member.UnusedHeapSizePercentage`: the percentage of the total heap size that is not currently being used

Access Service Broker Metrics

Service broker metrics are on by default and can be accessed through the Loggregator Firehose CLI plugin. For more information, see [Installing the Loggregator Firehose Plugin for CLI](#). For more information about broker metrics, see [On Demand Broker Metrics](#).

Export gfsh Logs

You can get logs and `.gfs` stats files from your Cloud Cache service instances using the `export logs` command in gfsh.

1. Use the [Connect with gfsh over HTTPS](#) procedure to connect to the service instance for which you want to see logs. Use a role that can read cluster information.
2. Run `export logs`. If you see a message of the form

```
Estimated exported logs expanded file size = 289927115, file-size-limit = 104857600. To deactivate exported logs file size check use option "--file-size-limit=0".
```

your logs exceed an expanded size of 100 megabytes, and they were not exported. To export in this case, use the gfsh command:

```
export logs --file-size-limit=0
```

3. Find the ZIP file in the directory where you started gfsh. This file contains a folder for each member of the cluster. The member folder contains the associated log files and stats files for that member.

For more information about the gfsh export command, see the [gfsh export documentation](#).

Deploy an App JAR File to the Servers

You can deploy or redeploy an app JAR file to the servers in the cluster.

To do an initial deploy of an app JAR file after connecting within gfsh using credentials that can manage the cluster, run this gfsh command:

```
deploy --jar=PATH-TO-JAR/FILENAME.jar
```

For example,

```
gfsh>deploy --jar=working-directory/myJar.jar
```

To redeploy an app JAR file after connecting within gfsh using credentials that can manage the cluster, do the following:

1. Run this gfsh command to deploy the updated JAR file:

```
deploy --jar=PATH-TO-UPDATED-JAR/FILENAME.jar
```

For example,

```
gfsh>deploy --jar=newer-jars/myJar.jar
```

2. Run this command to restart the cluster and load the updated JAR file:

```
cf update-service SERVICE-INSTANCE-NAME -c '{"restart": true}'
```

For example,

```
$ cf update-service my-service-instance -c '{"restart": true}'
```

Use the GemFire-Greenplum Connector

The GemFire-Greenplum connector permits the transfer of a PCC region out to a Greenplum database table or the transfer of a Greenplum database table into a PCC region. `gfsh` commands set up the configuration and initiate transfers. See the [GemFire-Greenplum Connector](#) documentation for details.

Connect in `gfsh` with a role that can both manage the cluster and read and write cluster data.

Scripting Common Operations

Gfsh commands may be placed into files, providing a way to script common maintenance operations. Scripting operations reduces errors that might occur when manually entering commands, and can be especially useful for running test cases and in deployment automation.

Running a gfsh Script

The `gfsh run` command invokes the `gfsh` commands that are in a file. Start up the `gfsh` command line interface first, and then invoke the command with:

```
gfsh>run --file=thecommands.gfsh
```

To see the `gfsh run` options, invoke:

```
gfsh>help run
```

File specification can be a relative or absolute path and file name. Scripted commands are not interactive; any command that would have prompted for input will instead use default values.

In order to eliminate placing cleartext passwords within a script, run `gfsh` and connect to the cluster prior to running a script with the `gfsh run` command. The password will appear on the command line, but it will not appear in the file that logs and captures `gfsh` commands.

Example Scripts

Tightly focussed scripts will ease cluster maintenance. Most of these scripts will do cluster management operations, so connect to the cluster with a role that is able to manage both the cluster and the data.

A common operation will be creating the regions hosted on the servers. This example `gfsh` script contents creates two regions:

```
create region --name=sessions --type=PARTITION_REDUNDANT
create region --name=customers --type=REPLICATE
```

This example `gfsh` script contents deploys a app JAR file to the cluster servers:

```
deploy --jar=/path/to/app-classes.jar
```

This example `gfsh` script contents creates the disk store needed for persistent regions:

```
create disk-store --name=all-regions-disk --dir=regions
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Connecting a Spring Boot App to Pivotal Cloud Cache with Session State Caching

This section describes the two ways in which you can connect a Spring Boot app to PCC:

- Using a Tomcat app with a WAR file. This is the default method for Tomcat apps.
- Using the `spring-session-data-gemfire` library. This method requires that you use the correct version of these libraries.

Use the Tomcat App

To get a Spring Boot app running with session state caching (SSC) on PCC, you must create a WAR file using the `spring-boot-starter-tomcat` plugin instead of the `spring-boot-maven` plugin to create a JAR file.

For example, if you want your app to use SSC, you cannot use `spring-boot-maven` to build a JAR file and push your app to PCF, because the Java buildpack does not pull in the necessary JAR files for SSC when it detects a Spring JAR file.

To build your WAR file, add this dependency to your `pom.xml`:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

For a full example of running a Spring Boot app that connects with SSC, [run this app](#) and use the following for your `pom.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>io.pivotal.gemfire.demo</groupId>
  <artifactId>HttpSessionCaching-Webapp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>HttpSessionCaching-Webapp</name>
  <description>Demo project for GemFire Http Session State caching</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.8.RELEASE</version>
```

```

    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>

```

Use a Spring Session Data GemFire App

You can connect your Spring app to PCC to do session state caching. Use the correct version of the [spring-session-data-gemfire](#) library; apps built for PCC v1.6.0 and later versions are compatible with Spring Session Data GemFire v2.1.5.RELEASE and later versions.

Upgrade PCC and Spring Session Data GemFire

1. Before your operator upgrades PCC, stop your app. This avoids breaking the app in this upgrade process.
2. Upgrade PCC. See [Upgrading Pivotal Cloud Cache](#) for details.
3. Rebuild your app using a [build.gradle](#) file that depends on the correct version of Pivotal GemFire. Here is an example [build.gradle](#) file:

```

version = '0.0.1-SNAPSHOT'

buildscript {
  ext {
    springBootVersion = '2.1.8.RELEASE'
  }
  repositories {
    mavenCentral()
    maven { url "https://repo.spring.io/snapshot" }
    maven { url "https://repo.spring.io/milestone" }
  }
}

```

```

    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'java'
apply plugin: 'org.springframework.boot'
apply plugin: 'idea'

idea{
    module{
        downloadSources = true
        downloadJavadoc = true
    }
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
    mavenCentral()
    maven { url "https://repo.spring.io/libs-milestone" }
    maven { url "https://repo.spring.io/milestone" }
    maven { url "http://repo.springsource.org/simple/ext-release-local" }
    maven { url "http://repo.spring.io/libs-release/" }
    maven { url "https://repository.apache.org/content/repositories/snapshots" }
}

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web:2.1.8.RELEASE")
    compile("org.springframework.session:spring-session-data-gemfire:2.1.5.RELEASE")
    compile("org.springframework.geode:spring-gemfire-starter:1.1.0.RELEASE")
}

```

4. Clear the session state region.
5. Start the rebuilt app.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Creating Continuous Queries Using Spring Data GemFire

To create continuous queries with the Spring Data GemFire library, you must have the following:

- Spring Data GemFire v2.1.0+
- Spring Boot v2.1.0+

To create continuous queries, do the following items:

- Specify attributes `subscriptionEnabled` and `readyForEvents` for the `ClientCacheApplication` annotation. Apply this annotation to the Spring Boot client application class:

```

@ClientCacheApplication(name = "GemFireSpringApplication", readyForEvents = true,
    subscriptionEnabled = true)

```

The annotation for a durable event queue for continuous queries also sets the `durableClientId` and `keepAlive` attributes. For example:

```
@ClientCacheApplication(name = "GemFireSpringApplication",
    durableClientId = "durable-client-id", keepAlive = true,
    readyForEvents = true, subscriptionEnabled = true)
```

- Annotate the method that handles the events to specify the query. To make the event queue durable across server failures and restarts, include the `durable = true` attribute in the annotation, as is done in the example:

```
@Component
public class ContinuousQuery {

    @ContinuousQuery(name = "yourQuery",
        query = "SELECT * FROM /yourRegion WHERE someAttribute == true",
        durable = true)
    public void handleChanges(CqEvent event) {
        //PERFORM SOME ACTION
    }
}
```

The class that contains the method with the `@ContinuousQuery` annotation must have the `@Component` annotation, such that the continuous query is wired up correctly for the server.

For more information, see the [Spring Data GemFire documentation](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Application Development

An app that interacts with a PCC service instance will use the Pivotal GemFire® cluster within that service instance. Architecting the data storage for the app requires some familiarity with GemFire.

This section introduces design patterns for structuring app design. It presents a minimal view of GemFire data organization to help with data architecture design. A complete presentation of GemFire's capabilities is in the [Pivotal GemFire Documentation](#).

In this topic:

- [Design Patterns](#)
 - [The Inline Cache](#)
 - [The Cache-Aside Cache](#)
 - [Bidirectional Replication Across a WAN](#)
 - [Blue-Green Disaster Recovery](#)
 - [CQRS Pattern Across a WAN](#)
 - [Hub-and-Spoke Topology with WAN Replication](#)
 - [Follow-the-Sun Pattern](#)
- [Region Design](#)
 - [Keys](#)
 - [Partitioned Regions](#)
 - [Replicated Regions](#)
 - [Persistence](#)
 - [Overflow](#)
 - [Regions as Used by the App](#)
 - [An Example to Demonstrate Region Design](#)
- [Handling Events](#)
- [Example Applications](#)
 - [A Simple Java App](#)
 - [A Spring Boot App](#)
- [Steps to Run an App](#)
- [Developing an App Under TLS](#)
- [HTTP Session State Caching](#)
 - [Deactivate Near Caching Within the App](#)

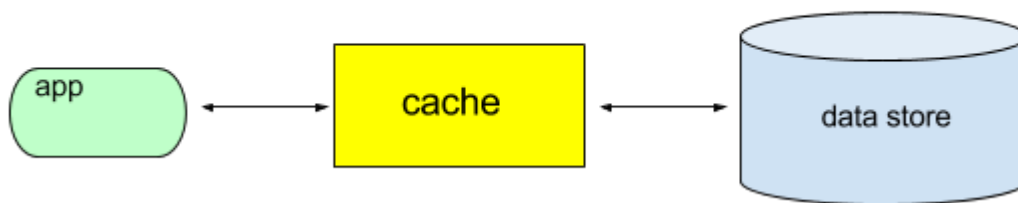
[Create a pull request or raise an issue on the source for this page in GitHub](#)

Design Patterns

The design patterns in this section are intended as overviews of common and useful configurations. Effective implementations will require planning—consult with a system architect to fill in the design details for your app.

The Inline Cache

An inline cache places the caching layer between the app and the backend data store.



The app will want to accomplish CRUD (create, read, update, delete) operations on its data. The app's implementation of the CRUD operations result in cache operations that break down into cache lookups (reads) and/or cache writes.

The algorithm for a cache lookup quickly returns the cache entry when the entry is in the cache. This is a cache hit. If the entry is not in the cache, it is a cache miss, and code on the cache server retrieves the entry from the backend data store. In the typical implementation, the entry returned from the backend data store on a cache miss is written to the cache, such that subsequent cache lookups of that same entry result in cache hits.

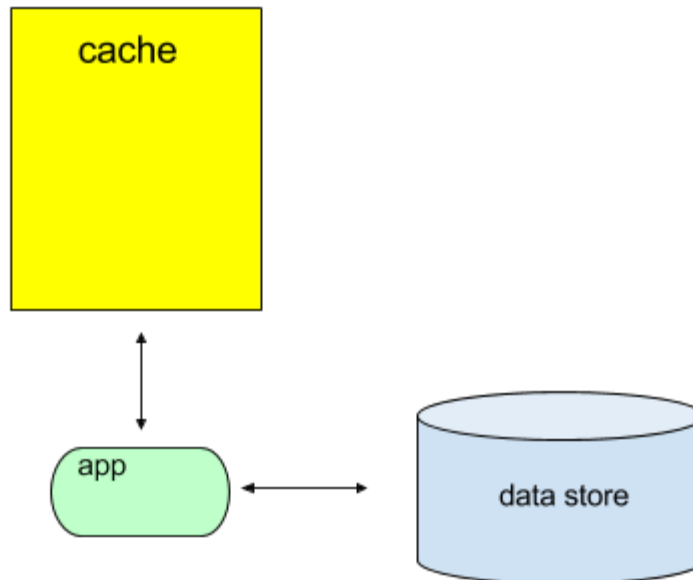
The implementation for a cache write typically creates or updates the entry within the cache. It also creates or updates the data store in one of the following ways:

- Synchronously, in a write-through manner. Each write operation from the app is sent on to be written to the backend data store. After the backend data store write finishes, the value is also written to the cache. The app blocks until the writes to both the backend data store and the cache complete.
- Asynchronously, in a write-behind manner. The cache gets updated, and the value to be written to the backend data store gets queued. Control then returns to the app, which continues independent of the write to the backend data store.

Developers design the server code to implement this inline-caching pattern. See [Setting Up Servers for an Inline Cache](#) for details about the custom server code and how to configure an inline cache.

The Cache-Aside Cache

The cache-aside pattern of caching places the app in charge of communication with both the cache and the backend data store.



The app will want to accomplish CRUD (CREATE, READ, UPDATE, DELETE) operations on its data. That data may be

- in both the data store and the cache
- in the data store, but not in the cache
- not in either the data store or the cache

The app's implementation of the CRUD operations result in cache operations that break down into cache lookups (reads) and/or cache writes.

The algorithm for a cache lookup returns the cache entry when the entry is in the cache. This is a cache hit. If the entry is not in the cache, it is a cache miss, and the app attempts to retrieve the entry from the data store. In the typical implementation, the entry returned from the backend data store is written to the cache, such that subsequent cache lookups of that same entry result in cache hits.

The cache-aside pattern of caching leaves the app free to implement whatever it chooses if the data store does not have the entry.

The algorithm for a cache write implements one of these:

- The entry is either updated or created within the data store, and the entry is updated within or written to the cache.
- The entry is either updated or created within the backend data store, and the copy currently within the cache is invalidated.



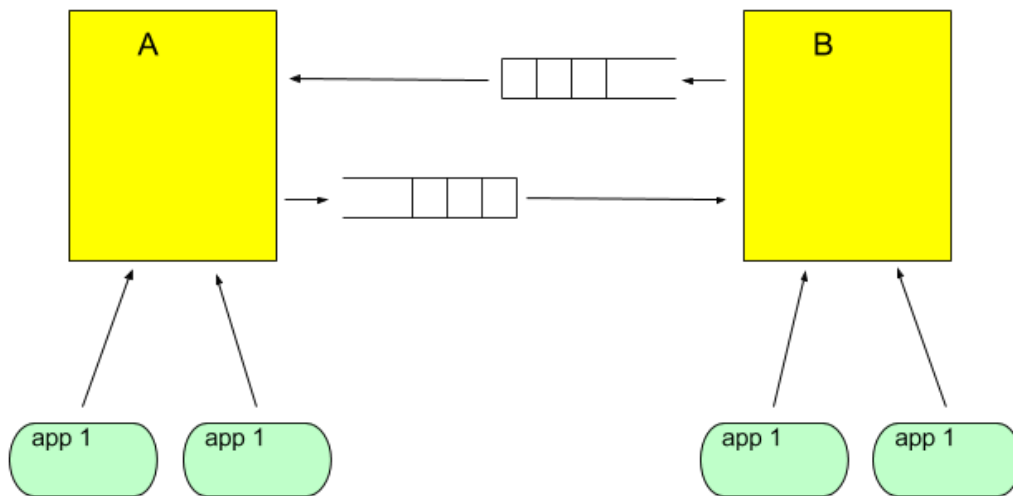
Note: SDG (Spring Data GemFire) supports the cache-aside pattern, as detailed at [Configuring Spring's Cache Abstraction](#).

Bidirectional Replication Across a WAN

Two PCC service instances may be connected across a WAN to form a single distributed system with asynchronous communication. The cluster within each of the PCC service instances will host the same region. Updates to either PCC service instance are propagated across the WAN to the other PCC service instance. The distributed system implements an eventual consistency of the region that also handles write conflicts which occur when a single region entry is modified in both PCC service instances at the same time.

In this active-active system, an external entity implements load-balancing by directing app connections to one of the two service instances. If one of the PCC service instances fails, apps may be redirected to the remaining service instance.

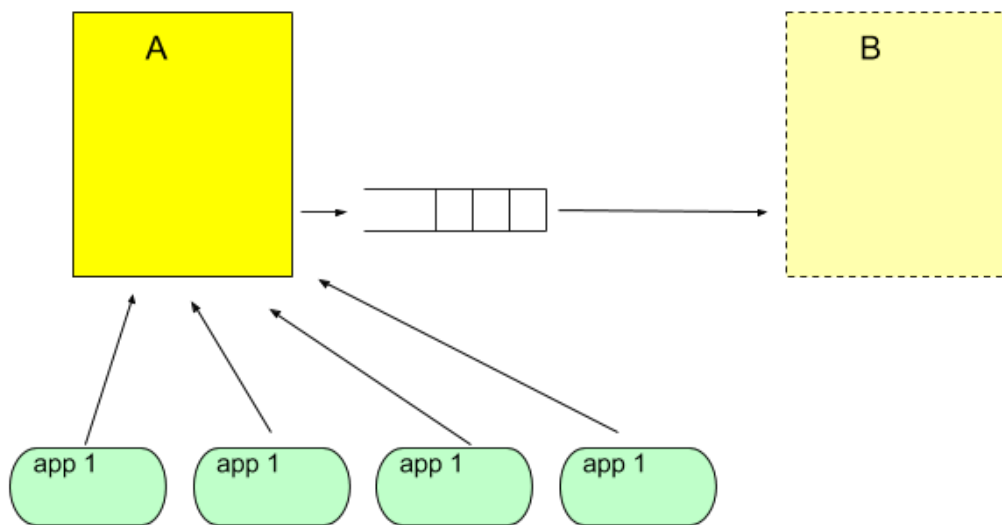
This diagram shows multiple instances of an app interacting with one of the two PCC service instances, cluster A and cluster B. Any change made in cluster A is sent to cluster B, and any change made in cluster B is sent to cluster A.



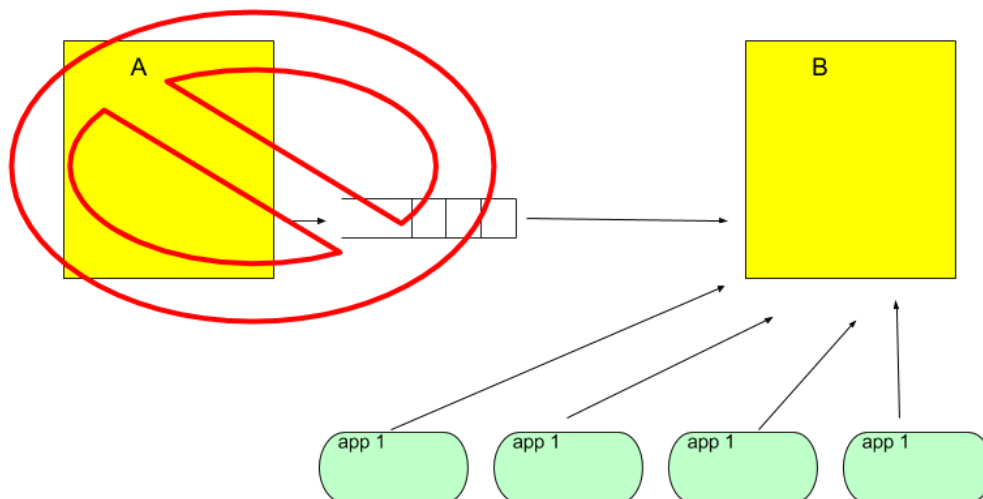
Blue-Green Disaster Recovery

Two PCC service instances may be connected across a WAN to form a single distributed system with asynchronous communication. An expected use case propagates all changes to a region's data from the cluster within one service instance (the primary) to the other, where both service instances reside in the same foundation. The replicate increases the fault tolerance of the system by acting as a "hot" spare. In the scenario of the failure of an entire data center or an availability zone, you can rebind apps to the replicate and restage them. The replicate then takes over as the primary.

In this diagram, cluster A is primary, and it replicates all data across a WAN to cluster B.



If cluster A fails, you can manually rebind and restage the apps so that cluster B takes over.

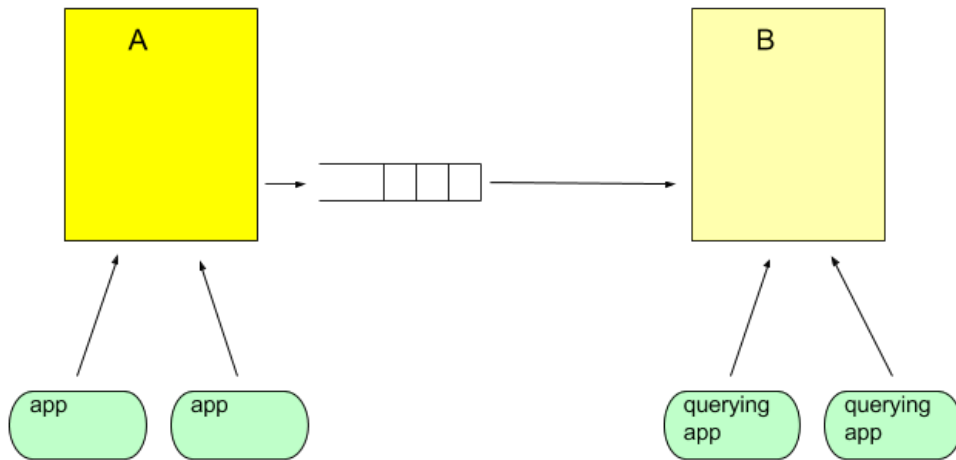


CQRS Pattern Across a WAN

Two PCC service instances may be connected across a WAN to form a single distributed system that implements a CQRS (Command Query Responsibility Segregation) pattern. Within this pattern, commands are those that change the state, where state is represented by region contents. All region operations that change state are directed to the cluster within one PCC service instance. The changes are propagated asynchronously to the cluster within the other PCC service instance via WAN replication, and that other cluster provides only query access to the region data.

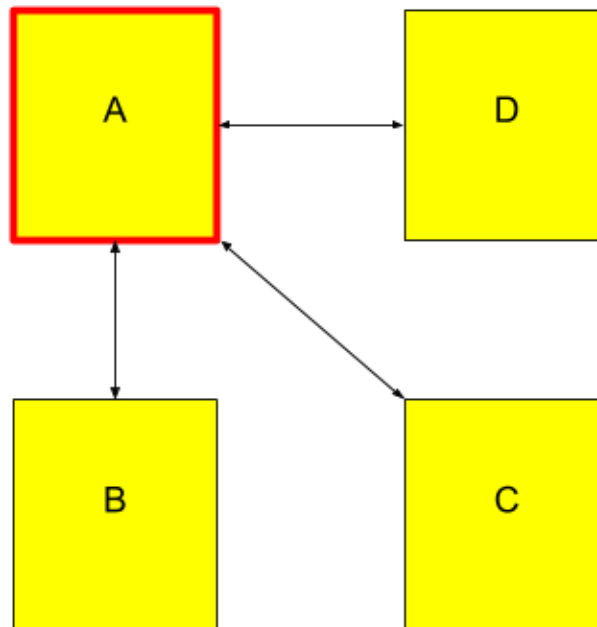
This diagram shows an app that may update the region within the PCC service instance of cluster A. Changes are propagated across the WAN to cluster B. The app bound to cluster B may only

query the region data; it will not create entries or update the region.



Hub-and-Spoke Topology with WAN Replication

Multiple PCC service instances connected across a WAN form a single hub and a set of spokes. This diagram shows PCC service instance A is the hub, and PCC service instances B, C, and D are spokes.



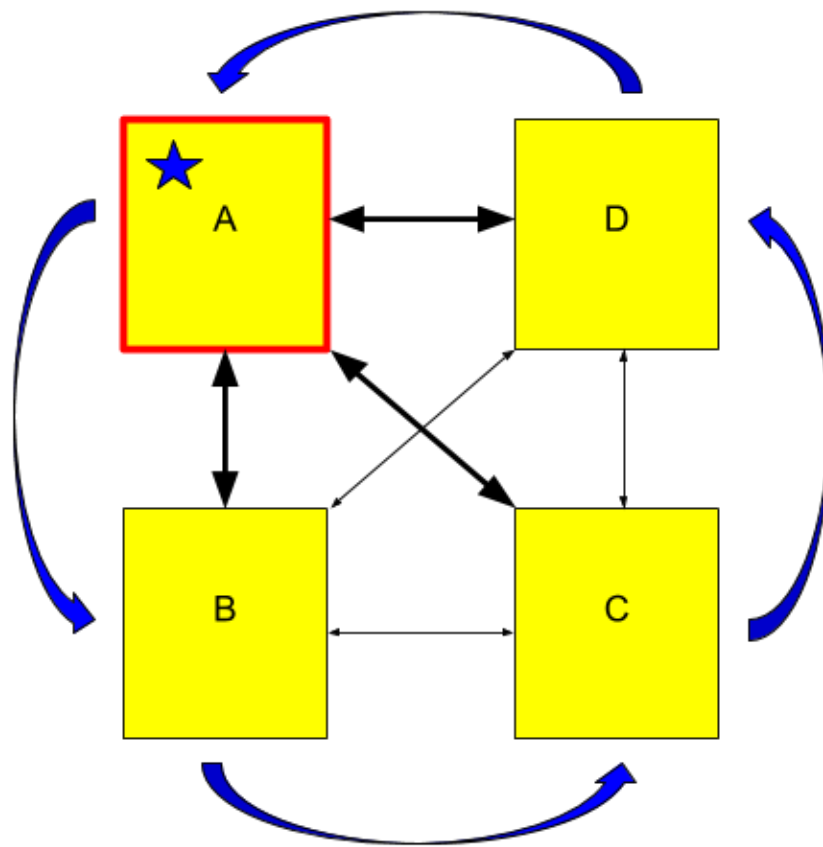
A common implementation that uses this topology directs all app operations that write or update region contents to the hub. Writes and updates are then propagated asynchronously across the WAN from the hub to the spokes.

Follow-the-Sun Pattern

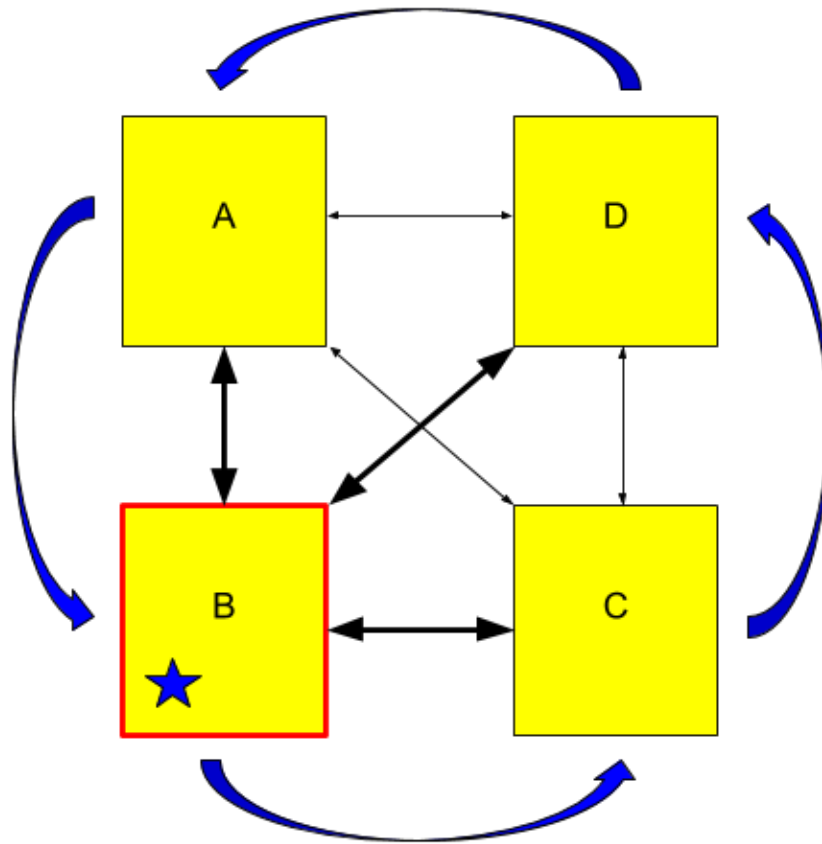
Performance improves when operation requests originate in close proximity to the service instance that handles those requests. Yet many data sets are relevant and used all over the world. If the most active location for write and update operations moves over the course of a day, then a performant design pattern is a variation on the hub-and-spoke implementation that changes which PCC service instance is the hub to the most active location.

Form a ring that contains each PCC service instance that will act as the hub. Define a token to identify the hub. Over time, pass the token from one PCC service instance to the next, around the ring.

This diagram shows PCC service instance A is the hub, as it has the token, represented in this diagram as a star. PCC service instances B, C, and D are spokes. Write and update operations are directed to the hub.



This diagram shows that the token has passed from A to B, and B has become the hub.



[Create a pull request or raise an issue on the source for this page in GitHub](#)

Region Design

Cached data are held in GemFire regions. Each entry within a region is a key/value pair. The choice of key and region type affect the performance of the design. There are two basic types of regions: partitioned and replicated. The distinction between the two types is based on how entries are distributed among servers that host the region.

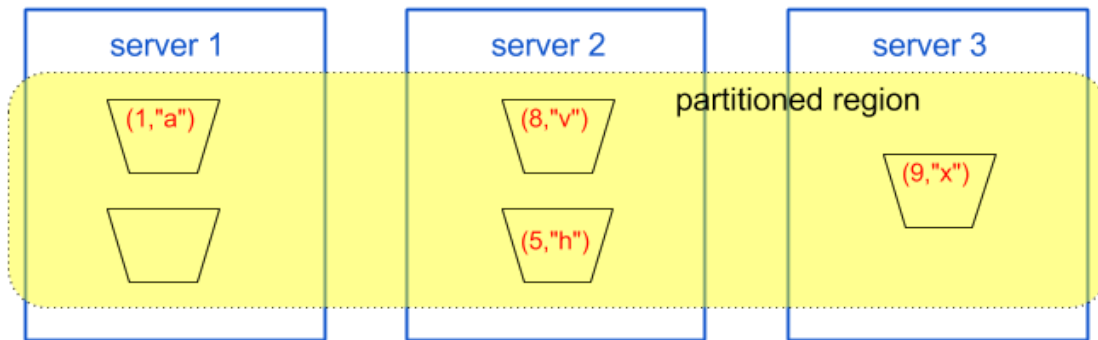
Keys

Each region entry must have a unique key. Use a wrapped primitive type of `String`, `Integer`, or `Long`. Experienced designers have a slight preference of `String` over `Integer` or `Long`. Using a `String` key enhances the development and debugging environment by permitting the use of a REST API (Swagger UI), as it only works with `String` types.

Partitioned Regions

A partitioned region distributes region entries across servers by using hashing. The hash of a key maps an entry to a bucket. A fixed number of buckets are distributed across the servers that host the region.

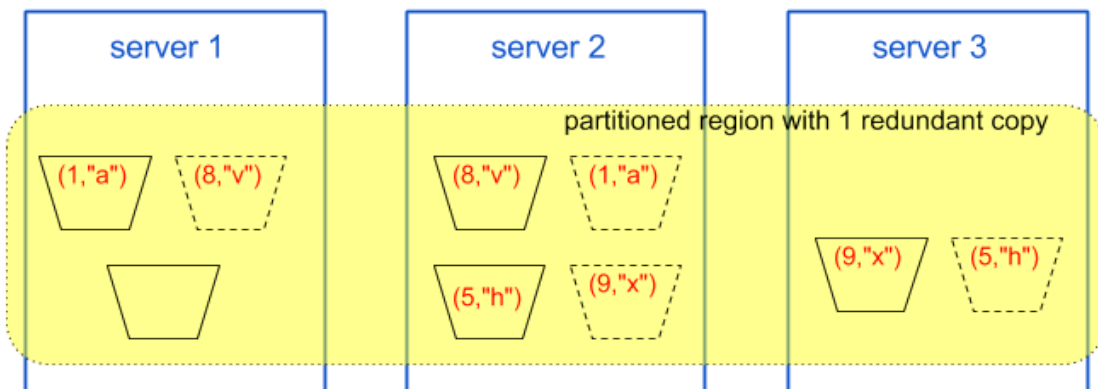
Here is a diagram that shows a single partitioned region (highlighted) with very few entries to illustrate partitioning.



A partitioned region is the proper type of region to use when one or both of these situations exist:

- The region holds vast quantities of data. There may be so much data that you need to add more servers to scale the system up. PCC can be scaled up without downtime; to learn more, see [Updating a Pivotal Cloud Cache Service Instance](#).
- Operations on the region are write-heavy, meaning that there are a lot of entry updates.

Redundancy adds fault tolerance to a partitioned region. Here is that same region, but with the addition of a single redundant copy of each each entry. The buckets drawn with dashed lines are redundant copies. Within the diagram, the partitioned region is highlighted.



With one redundant copy, the GemFire cluster can tolerate a single server failure or a service upgrade without losing any region data. With one less server, GemFire revises which server holds the primary copy of an entry.

A partitioned region without redundancy permanently loses data during a service upgrade or if a server goes down. All entries hosted in the buckets on the failed server are lost.

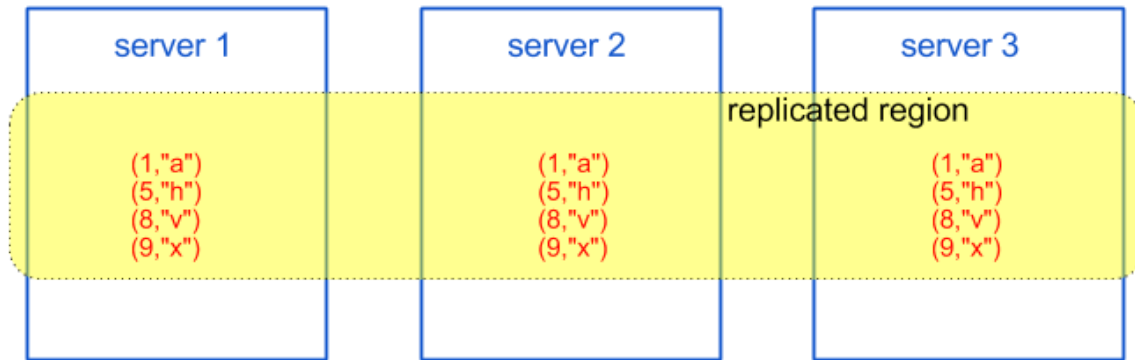
Partitioned Region Types for Creating Regions on the Server

Region types associate a name with a particular region configuration. The type is used when creating a region. Although more region types than these exist, use one of these types to ensure that no region data is lost during service upgrades or if a server fails. These partitioned region types are supported:

- **PARTITION_REDUNDANT** Region entries are placed into the buckets that are distributed across all servers hosting the region. In addition, GemFire keeps and maintains a declared number of redundant copies of all entries. The default number of redundant copies is 1.
- **PARTITION_REDUNDANT_HEAP_LRU** Region entries are placed into the buckets that are distributed across all servers hosting the region. GemFire keeps and maintains a declared number of redundant copies. The default number of redundant copies is 1. As a server (JVM) reaches a heap usage of 75% of available heap, the server destroys entries as space is needed for updates. The oldest entry in the bucket where a new entry lives is the one chosen for destruction.
- **PARTITION_PERSISTENT** Region entries are placed into the buckets that are distributed across all servers hosting the region, and all servers persist all entries to disk.
- **PARTITION_REDUNDANT_PERSISTENT** Region entries are placed into the buckets that are distributed across all servers hosting the region, and all servers persist all entries to disk. In addition, GemFire keeps and maintains a declared number of redundant copies of all entries. The default number of redundant copies is 1.
- **PARTITION_REDUNDANT_PERSISTENT_OVERFLOW** Region entries are placed into the buckets that are distributed across all servers hosting the region, and all servers persist all entries to disk. In addition, GemFire keeps and maintains a declared number of redundant copies of all entries. The default number of redundant copies is 1. As a server (JVM) reaches a heap usage of 75% of available heap, the server overflows entries to disk when it needs to make space for updates.
- **PARTITION_PERSISTENT_OVERFLOW** Region entries are placed into the buckets that are distributed across all servers hosting the region, and all servers persist all entries to disk. As a server (JVM) reaches a heap usage of 75% of available heap, the server overflows entries to disk when it needs to make space for updates.

Replicated Regions

Here is a replicated region with very few entries (four) to illustrate the distribution of entries across servers. For a replicated region, all servers that host the region have a copy of every entry.



GemFire maintains copies of all region entries on all servers. GemFire takes care of distribution and keeps the entries consistent across the servers.

A replicated region is the proper type of region to use when one or more of these situations exist:

- The region entries do not change often. Each write of an entry must be propagated to all servers that host the region. As a consequence, performance decreases when many concurrent write accesses cause subsequent writes to all other servers hosting the region.
- The overall quantity of entries is not so large as to push the limits of memory space for a single server. The PCF service plan sets the server memory size.
- The entries of a region are frequently accessed together with entries from other regions. The entries in the replicated region are always available on the server that receives the access request, leading to better performance.

Replicated Region Types for Creating Regions on the Server

Region types associate a name a particular region configuration. These replicated region types are supported:

- **REPLICATE** All servers hosting the region have a copy of all entries.
- **REPLICATE_HEAP_LRU** All servers hosting the region have a copy of all entries. As a server (JVM) reaches a heap usage of 75% of available heap, the server destroys entries as it needs to make space for updates.
- **REPLICATE_PERSISTENT** All servers hosting the region have a copy of all entries, and all servers persist all entries to disk.
- **REPLICATE_PERSISTENT_OVERFLOW** All servers hosting the region have a copy of all entries. As a server (JVM) reaches a heap usage of 75% of available heap, the server overflows entries to disk as it need to make space for updates.

Persistence

Persistence adds a level of fault tolerance to a PCC service instance by writing all region updates to local disk. Disk data, hence region data, is not lost upon cluster failures that exceed redundancy failure tolerances. Upon cluster restart, regions are reloaded from the disk, avoiding the slower method of restart that reacquires data using a database of record.

Creating a region with one of the region types that includes `PERSISTENT` in its name causes the instantiation of local disk resources with these default properties:

- Synchronous writes. All updates to the region generate operating system disk writes to update the disk store.
- The disk size is part of the instance configuration. See [Configure Service Plans](#) for details on setting the persistent disk types for the server. Choose a size that is at least twice as large as the expected maximum quantity of region data, with an absolute minimum size of 2 GB. Region data includes both the keys and their values.
- Warning messages are issued when a 90% disk usage threshold is crossed.

Overflow

Region overflow is an eviction action that keeps heap memory space usage below a fixed threshold of 75% of available heap memory space. For a region that pushes at the limits of memory usage, overflow reduces the number of or eliminates pauses for stop-the-world garbage collection.

The action of overflow writes one or more least recently used region entries to disk to make room in memory for another entry. The least recently used entry within the bucket to which new entry maps is the chosen overflow victim. The key of the victim remains in memory, but the value of the entry is written to disk. An operation on an entry that has overflowed to disk causes the entry to be swapped back into memory.

If using a region type with overflow, be sure to configure a plan with sufficient disk space for the Server VM, allocating at least the minimums given for the *Persistent disk type for the Server VMs*, as described in [Configure Tile Properties](#).

If no disk store is created, region creation with a region type that uses a disk store will cause the creation of one called `DEFAULT` with a default size (2 Gbyte). Alternatively, create the disk store using `gfsh`, as described in [Working with Disk Stores](#). Then, create the region using the `--disk-store` option to specify the created disk store. If the disk store has been created, but the `gfsh` region creation command neglects to specify a disk store, a new `DEFAULT` disk store will be created and used. For more details on region creation options, see the Pivotal GemFire [create region Command Reference Page](#).

Regions as Used by the App

The client accesses regions hosted on the servers by creating a cache and the regions. The type of the client region determines if data is only on the servers or if it is also cached locally by the client in addition to being on the servers. Locally cached data can introduce consistency issues, because region entries updated on a server are not automatically propagated to the client's local cache.

Client region types associate a name with a particular client region configuration.

- `PROXY` forwards all region operations to the servers. No entries are locally cached. Use this client region type unless there is a compelling reason to use one of the other types. Use this type for all Twelve-Factor apps in order to assure stateless processes are implemented. Not caching any entries locally prevents the app from accidentally caching state.

- `CACHING_PROXY` forwards all region operations to the servers, and entries are locally cached.
- `CACHING_PROXY_HEAP_LRU` forwards all region operations to the servers, and entries are locally cached. Locally cached entries are destroyed when the app's usage of cache space causes its JVM to hit the threshold of being low on memory.

An Example to Demonstrate Region Design

Assume that on servers, a region holds entries representing customer data. Each entry represents a single customer. With an ever-increasing number of customers, this region data is a good candidate for a partitioned region.

Perhaps another region holds customer orders. This data also naturally maps to a partitioned region. The same could apply to a region that holds order shipment data or customer payments. In each case, the number of region entries continues to grow over time, and updates are often made to those entries, making the data somewhat write heavy.

A good candidate for a replicated region would be the data associated with the company's products. Each region entry represents a single product. There are a limited number of products, and those products do not often change.

Consider that as the client app goes beyond the most simplistic of cases for data representation, the PCC instance hosts all of these regions such that the app can access all of these regions. Operations on customer orders, shipments, and payments all require product information. The product region benefits from access to all its entries available on all the cluster's servers, again pointing to a region type choice of a replicated region.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Handling Events

The GemFire cluster within a PCC service instance can handle events. An app registers interest in a particular event, and when the server detects the event, an app-defined callback (also called a handler or a listener) is invoked to handle the event.

There are three aspects to configuring and implementing an event:

- the app defines or specifies the event
- the app registers the event with or identifies the event to the system component that will detect the event
- the app defines the callback, which handles the event

Continuous Queries

Continuous queries use a GemFire OQL query on region data to define an event. A change in query results is the event. The app registers both the query and the callback to set up a continuous query. Each region operation invokes the query, leading to the naming of this type of event handling as continuous.

See [Querying with OQL](#) for details on the Object Query Language (OQL) and generating queries.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Example Applications

These example applications provide insight into aspects of app design for PCC.

In this topic:

- [A Simple Java App](#)
- [A Spring Boot App](#)

[Create a pull request or raise an issue on the source for this page in GitHub](#)

A Simple Java App

The sample Java client app at <https://github.com/cf-gemfire-org/cloudcache-sample-app.git> demonstrates how to connect an app to a service instance.

These instructions assume:

- A Cloud Cache service instance is running.
- You have Cloud Foundry credentials for accessing the Cloud Cache service instance.
- You have a service key for the Cloud Cache service instance.
- You have a login on the Pivotal Commercial Maven Repository at <https://commercial-repo.pivotal.io>.
- You have a `gfsh` client of the same version as is used within your Cloud Cache service instance.

Follow these instructions to run the app.

1. Clone the sample Java app from <https://github.com/cf-gemfire-org/cloudcache-sample-app.git>.
2. Update your clone of the sample Java app to work with your Cloud Cache service instance:
 - Modify the manifest in `manifest.yml` by replacing `service0` with the name of your Cloud Cache service instance.
 - Replace the username and password in the `gradle.properties` file with your username and password for the Pivotal Commercial Maven Repository.
 - Update the GemFire version in the dependencies section of the `build.gradle` file to be the same as the version within your Cloud Cache service instance.

3. Build the app with

```
$ ./gradlew clean build
```

4. In a second shell, run `gfsh`.
5. Use `gfsh` to connect to the Cloud Cache service instance as described in [Connect with gfsh over HTTPS](#).
6. Use `gfsh` to create a region named `test` as described in [Create Regions](#). This sample app places a single entry into the region, so the region type is not important. `PARTITION_REDUNDANT` is a good choice.

7. In the shell where the app was built, deploy and run the app with

```
cf push -f manifest.yml
```

8. After the app starts, there will be an entry of (“1”, “one”) in the `test` region. you can see that there is one entry in the region with the `gfsh` command:

```
gfsh>describe region --name=test
```

For this very small region, you can print the contents of the entire region with a `gfsh` query:

```
gfsh>query --query='SELECT * FROM /test'
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

A Spring Boot App

The versioned example Spring Boot Data GemFire app at [PCC-Sample-App-PizzaStore](#) uses the GemFire cluster within a PCC service instance as a system of record. Directions for running the app are in the GitHub repository’s `README.md` file. The app is versioned, and branches of the repository represent PCC versions. For PCC v1.9, use the branch named `release/1.9`.

The app uses Spring Boot Data GemFire. The documentation for this opinionated extension of Spring Data GemFire is at [Spring Boot for Apache Geode & Pivotal GemFire Reference Guide](#).

The app saves pizza orders within the GemFire servers of a PCC service instance. The app takes orders for pizza toppings and sauces with a REST interface.

In addition, the app demonstrates two distinct features of PCC:

- Running an app on a TLS-enabled cluster within PCC.
- GemFire continuous queries as used by a Spring Boot app.

Top Down Explanation

In this opinionated version of Spring Boot Data GemFire, at the topmost level of the app, the `@SpringBootApplication` annotation causes the app to have a `ClientCache` instance. Within the example app’s `CloudcachePizzaStoreApplication` class, the annotation belongs on the class header:

```
@SpringBootApplication
public class CloudcachePizzaStoreApplication
```

This app is the client within a standard client/server architecture. The PCC cluster contains the servers. The client cache is a driver for interactions with the servers, allowing eventual region creation within the client cache.

Annotate the configuration with `@EnableEntityDefinedRegions` to enable a runtime scan of classes that define the GemFire regions. Within `GemFireConfiguration.java` in this example:

```
@EnableEntityDefinedRegions(basePackageClasses = Pizza.class)
```

```
public class GemFireConfiguration
```

The Spring repository construct is the correct choice to use for the data storage, which will be a GemFire region. To implement it, annotate this example's `PizzaRepository` implementation with `@Repository`:

```
@Repository
public interface PizzaRepository extends CrudRepository<Pizza, String>
```

A GemFire region underlies the Spring repository, storing the ordered pizzas. Annotate the `Pizza` class model with `@Region`:

```
@Region("Pizza")
public class Pizza {
```

Within the `Pizza` class, identify the key of the GemFire region entries with the `@Id` annotation. It is the `name` field in this example:

```
@Getter @Id @NonNull
private final String name;
```

The `@SpringBootApplication` annotation results in a chain of opinionated defaults, all of which are appropriate for this app. It identifies the app as a client. The client receives a GemFire client cache. Any regions will default to type `PROXY`. A proxy type of region forwards all region operations to the Gemfire servers; no data is stored within the app's client cache.

See [Configuring Regions](#) for Spring details. See [Region Design](#) for PCC details on regions.

The App Controller

The `AppController` class implements the REST interface, by annotating the class with `@RestController`:

```
@RestController
public class AppController
```

As pizzas are ordered, a `CrudRepository.save()` operation causes a GemFire `put` operation that updates the region on the GemFire server.

TLS-Enabled Cluster Demonstration

The app operates in one of two ways:

1. Communication with and within the GemFire cluster uses TLS encryption.
2. Communication with and within the GemFire cluster *does not* use TLS encryption.

The operation of the running app is unchanged whether TLS encryption is enabled within the Gemfire cluster or not.

Setting up this app with TLS encryption enabled demonstrates one way to set the needed GemFire properties detailed in [Developing an App Under TLS](#). It uses Spring profiles. The manifest that sets

the `tls` profile incorporates the GemFire properties from file `/resources/application-tls.properties`.

Continuous Queries in the App

The app defines two continuous queries (CQ). See [Handling Events](#) for a brief introduction to continuous queries.

The first CQ queries for any (and every) pizza order. Its callback logs the order.

The second CQ queries for pizzas with a `PESTO` sauce. When a pesto-sauced pizza is ordered, the callback adds that pizza to a second region called `Name`.

Within a Spring Boot Data GemFire app, to specify a continuous query:

- annotate the class that defines the queries with `@Component`:

```
@Component
public class PizzaQueries
```

- annotate the callback method with `@ContinuousQuery` and define the query, as in the app's pesto pizza order example:

```
@ContinuousQuery(name = "PestoPizzaOrdersQuery", durable = true,
    query = "SELECT * FROM /Pizza p WHERE p.sauce.name = 'PESTO'")
public void handlePestoPizzaOrder(CqEvent event)
```

The annotation causes an implementation of all three needed items for this type of event handling. It defines the query, registers the query as a continuous query event, and it identifies the callback method to invoke when the event occurs.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Steps to Run an App

Java Build Pack Requirements

To ensure that your app can use all the features from PCC, use the latest buildpack. The buildpack is available on GitHub at [cloudfoundry/java-buildpack](#).

Bind an App to a Service Instance

Binding your apps to a service instance enables the apps to connect to the service instance and read or write data to the region. Run `cf bind-service APP-NAME SERVICE-INSTANCE-NAME` to bind an app to your service instance. Replace `APP-NAME` with the name of the app. Replace `SERVICE-INSTANCE-NAME` with the name you chose for your service instance.

```
$ cf bind-service my-app my-cloudcache
```

Binding an app to the service instance provides connection information through the `VCAP_SERVICES` environment variable. Your app can use this information to configure components, such as the

GemFire client cache, to use the service instance.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Developing an App Under TLS

Apps that connect to a TLS-enabled PCC service instance must set properties to configure the communication with the Pivotal GemFire components within the PCC service instance.

Ensure that the cluster-level prerequisite step of [Preparing for TLS](#) has been completed.

For a Spring Data GemFire app with a Spring Data GemFire library dependency of 2.2.0.BUILD-SNAPSHOT or a more recent version, attach the `@EnableSsl` annotation to your configuration class to enable the TLS encryption for all GemFire components. Also set these GemFire properties:

```
ssl-use-default-context=true
ssl-endpoint-identification-enabled=false
```

For other apps, the GemFire properties should be

```
ssl-enabled-components=all
ssl-use-default-context=true
ssl-endpoint-identification-enabled=false
```

An app may set these properties with the `ClientCacheFactory.set()` method, prior to creating a `ClientCache` instance.

The build and `cf push` of the app does not require any changes to work with a TLS-enabled PCC service instance.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

HTTP Session State Caching

Deactivate Near Caching Within the App

Near caching is when an app locally caches data. Near caching uses an embedded cache within the app. Web apps that deploy Tomcat with GemFire session state caching have near caching by default. To keep an app stateless, deactivate near caching.

There are two methods for deactivating near caching. Choose and implement one of these methods:

1. Create and use a custom buildpack (that deactivate near caching) that resides in an external repository. Modify the app setup to acquire the configuration from the external repository. This method facilitates having a single configuration that may be used by a variety of apps.
2. Create and use a custom buildpack (that deactivate near caching) for the app.

Deactivate Caching Using an External Repository for Configuration

There are two parts to this method for deactivating near caching within the app. The first part builds a repository to hold the configuration and custom buildpack, and then pushes the repository to the space such that the app will have access. The second part changes the app configuration such that it uses the custom buildpack.

The procedure to build the repository:

1. Make a directory to hold the configuration:

```
mkdir tomcat-config
```

2. Make other needed files and directories within the newly created directory:

```
cd tomcat-config
mkdir public
mkdir -p tomcat-1.0.0/conf
touch Staticfile
```

3. Edit `Staticfile` to contain:

```
root: public
directory: visible
```

4. Create `tomcat-1.0.0/conf/context.xml`, and edit the file such that it contains the following content:

```
<?xml version='1.0' encoding='UTF-8'?>
<!--
~ Copyright 2013-2019 the original author or authors.
~
~ Licensed under the Apache License, Version 2.0 (the "License");
~ you may not use this file except in compliance with the License.
~ You may obtain a copy of the License at
~
~     http://www.apache.org/licenses/LICENSE-2.0
~
~ Unless required by applicable law or agreed to in writing, software
~ distributed under the License is distributed on an "AS IS" BASIS,
~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
~ See the License for the specific language governing permissions and
~ limitations under the License.
-->
<Context>
    <Resources allowLinking='true'/>
    <Manager className='org.apache.geode.modules.session.catalina.Tomcat8DeltaSessionMa
nager' enableLocalCache='false' regionAttributesId='PARTITION_REDUNDANT_HEAP_LRU'/>
</Context>
```

5. Place a compressed TAR format version of the `tomcat-1.0.0` directory into the `public` directory:

```
tar -czf tomcat-1.0.0.tar.gz tomcat-1.0.0/
cp tomcat-1.0.0.tar.gz public/
```

- Use `cf push` to place the configuration into the Pivotal Cloud Foundry space that will host this configuration:

```
cf push tomcat-config
```

- Issue the command

```
cf apps
```

to acquire the URL of the `tomcat-config` app. Prepend the listed URL with `http://` to have the URL that will identify the location of the configuration needed by the app.

For example, your complete URL will look something like `http://tomcat-config.apps.yellow-green.cf-app.com`.

- Within the `tomcat-config` directory, edit `public/index.yml` to have URL-specific contents. Using the example URL as a guide, the contents of this `public/index.yml` file will contain:

```
---
1.0.0: http://tomcat-config.apps.yellow-green.cf-app.com/tomcat-1.0.0.tar.gz
```

- Push the app a second time with its completed configuration:

```
cf push tomcat-config
```

The second part of this procedure modifies the app such that it uses the custom buildpack:

- Edit the `manifest.yml` file by appending this URL-specific configuration to the `applications` section of the `manifest.yml` file:

```
env:
  JBP_CONFIG_TOMCAT: "{ tomcat: { external_configuration_enabled: true }, external_configuration: { repository_root: \"http://tomcat-config.apps.yellow-green.cf-app.com\" } }"
```

Substitute your complete URL for the URL in this example.

A complete `manifest.yml` file will appear similar to:

```
---
applications:
- name: http-session-caching
  path: build/libs/http-session-caching-0.0.1.war
  buildpack: java_buildpack_offline
  env:
    JBP_CONFIG_TOMCAT: "{ tomcat: { external_configuration_enabled: true }, external_configuration: { repository_root: \"http://tomcat-config.apps.yellow-green.cf-app.com\" } }"
```

- Push, bind, and start your app with a buildpack of version 4.18 or a more recent version:

```
cf push -f ./manifest.yml --no-start -b https://github.com/cloudfoundry/java-buildpack.git#v4.18
```

```
cf bind-service APP-NAME SERVICE-INSTANCE-NAME

cf start APP-NAME
```

3. To verify that local caching is deactivated for the app, use `cf ssh` to access the app and visually verify that `enableLocalCache='false'` appears within the `context.xml` file. Use this sequence of commands:

```
cf ssh APP-NAME

find ./ -name *.xml

cat ./app/.java-buildpack/tomcat/conf/context.xml
```

Deactivate Caching Using a Custom Java Buildpack

This procedure creates and uses a custom java buildpack that deactivates near caching in the app. Once created, the `cf push` specifies the custom Java buildpack.

1. Clone the git buildpack repository:

```
git clone git@github.com:cloudfoundry/java-buildpack.git
```

2. Change directories to the newly created repository:

```
cd java-buildpack
```

3. Edit the Geode ruby configuration file such that it deactivates caching within the app. The file to edit is `lib/java_buildpack/container/tomcat/tomcat_geode_store.rb`. The single change is the string `true` to instead be `false`. Before the change, here is the portion of the file to be changed, with the string to change highlighted:

```
def add_manager(context)
  context.add_element 'Manager',
    'className' => SESSION_MANAGER_CLASS_NAME,
    'enableLocalCache' => 'true',
    'regionAttributesId' => REGION_ATTRIBUTES_ID
end
```

After changing the highlighted string from `true` to `false`, here is the portion of the file with the change highlighted:

```
def add_manager(context)
  context.add_element 'Manager',
    'className' => SESSION_MANAGER_CLASS_NAME,
    'enableLocalCache' => 'false',
    'regionAttributesId' => REGION_ATTRIBUTES_ID
end
```

4. Create your custom Java buildpack on the platform with a command of the form:

```
cf create-buildpack BUILDPACK PATH POSITION --enable
```

where `BUILDPACK` is the name you choose for your buildpack.

5. After building your application, push it such that it uses your buildpack:

```
cf push -f ./manifest.yml -b BUILDPACK
```

6. Bind your app as described in [Bind an App to a Service Instance](#).
7. Restage the app to ensure proper configuration:

```
cf restage APP-NAME
```

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Troubleshooting

Here are problems and fixes related to using PCC.

Acquire Artifacts for Troubleshooting

Gather GemFire Logs

GemFire statistics and log files may be obtained by using the `gfsh export logs` command. See [Export gfsh Logs](#) for details.

View Statistics Files and Logs

You can visualize the performance of your cluster by downloading the statistics files from your servers. These files are located in the persistent store on each VM. To copy these files to your workstation, run the command:

```
bosh2 -e BOSH-ENVIRONMENT -d DEPLOYMENT-NAME scp server/0:/var/vcap/store/gemfire-server/statistics.gfs /tmp
```

Alternatively, use `bosh ssh` to access PCC service instance GemFire server VMs and directly obtain the GemFire logs that reside within the directory `/var/vcap/sys/log/gemfire-server`.

See the Pivotal GemFire [Installing and Running VSD](#) topic for information about loading the statistics files into Pivotal GemFire VSD.

Acquire Thread Dumps

Thread dumps may be useful for debugging. Take at least three thread dumps on each VM, separating them by about one second.

1. To list your VMs, run:

```
bosh -e ENV -d DEPLOYMENT vms
```

[Acquire the Deployment Name](#) instructs how to acquire the string to substitute for `DEPLOYMENT`.

2. Use the VM in a `bosh ssh` command to ssh in to the PCC VM where you want to produce the thread dumps. PCC VMs can be referenced using a 0-based index, for example `server/0`, or `locator/2`:

```
bosh -e ENV -d DEPLOYMENT ssh server/0
```

3. Get into the Bosh Process Manager (bpm) shell by running

```
sudo /var/vcap/packages/bpm/bin/bpm shell JOB-NAME
```

where JOB-NAME is either `gemfire-server` or `gemfire-locator`, depending on which PCC VM you are on.

- Find the process ID (PID) that is running the GemFire Java process by running

```
ps -aux | grep java
```

Typically, the PID is 1.

- As you take multiple thread dumps, redirect the output of each to a uniquely named file. This example uses the file name `threaddump1.txt`:

```
/var/vcap/packages/jdk8/bin/jcmd 1 Thread.print > /tmp/threaddump1.txt
```

Files in `/tmp` will be accessible on the VM in directory `/var/vcap/data/gemfire-server/tmp` or `/var/vcap/data/gemfire-locator/tmp`.

- Move the files to the `/tmp` directory on the VM by running

```
mv /var/vcap/data/gemfire-server/tmp/threaddump1.txt /tmp/
```

or

```
mv /var/vcap/data/gemfire-locator/tmp/threaddump1.txt /tmp/
```

- Files can be copied to your local machine using `bosh scp` command. From your local machine, run:

```
bosh -d DEPLOYMENT scp VM:/tmp/threaddump1.txt .
```

For example:

```
$ bosh -d service-instance_1fd2850e-b754-4c5e-aa5c-ddb54ee301e6 scp server/0:/tmp/threaddump1.txt .
```

Acquire the Deployment Name

The `DEPLOYMENT` name is needed in several troubleshooting procedures. To acquire the `DEPLOYMENT` name:

- Use the Pivotal Cloud Foundry CLI. Target the space where the service instance runs.
- Discover the globally unique identifier (GUID) for the service instance:

```
cf service INSTANCE-NAME --guid
```

The output is the GUID. For example:

```
$ cf service dev-instance --guid
1fd2850e-b754-4c5e-aa5c-ddb54ee301e6
```

- Prefix the GUID with the string `service-instance_` to obtain the `DEPLOYMENT` name. For the example GUID, the `DEPLOYMENT` name is `service-instance_1fd2850e-b754-4c5e-aa5c-ddb54ee301e6`.

Troubleshooting for Operators

Smoke Test Failures

- Error message:** “Creating p-cloudcache SERVICE-NAME failed”

Cause of the Problem: The smoke tests could not create an instance of GemFire.

Action: To troubleshoot why the deployment failed, use the CF CLI to create a new service instance using the same plan and download the logs of the service deployment from BOSH.

- Error message:** “Deleting SERVICE-NAME failed”

Cause of the Problem: The smoke test attempted to clean up a service instance it created and failed to delete the service using the `cf delete-service` command.

Action: Run BOSH `logs` to view the logs on the broker or the service instance to see why the deletion may have failed.

- Error message:** “Cannot connect to the cluster SERVICE-NAME”

Cause of the Problem: The smoke test was unable to connect to the cluster.

Action: Review the logs of your load balancer, and review the logs of your CF Router to ensure the route to your PCC cluster is properly registered.

You also can create a service instance and try to connect to it using the `gfsh` CLI. This requires creating a service key.

- Error message:** “Could not perform create/put on Cloud Cache cluster”

Cause of the Problem: The smoke test was unable to write data to the cluster. The user may not have permissions to create a region or write data.

- Error message:** “Could not retrieve value from Cloud Cache cluster”

Cause of the Problem: The smoke test was unable to read back the data it wrote. Data loss can happen if a cluster member improperly stops and starts again or if the member machine crashes and is resurrected by BOSH.

Action: Run BOSH `logs` to view the logs on the broker to see if there were any interruptions to the cluster by a service update.

General Connectivity

- Problem:** Client-to-server communication

Cause of the Problem: PCC Clients communicate to PCC servers on port 40404 and with locators on port 55221. Both of these ports must be reachable from the PAS network to service the network.

- Problem:** Membership port range

Cause of the Problem: PCC servers and locators communicate with each other using UDP and TCP. The current port range for this communication is 49152–65535.

Solution: If you have a firewall between VMs, ensure this port range is open.

- **Problem:** Port range usage across a WAN

Cause of the Problem: Gateway receivers and gateway senders communicate across WAN-separated service instances. Each PCC service instance uses GemFire defaults for the gateway receiver ports. The default is the inclusive range of port numbers 5000 to 5499.

Solution: Ensure this port range is open when WAN-separated service instances will communicate.

Troubleshooting for Developers

- **Problem:** An error occurs when creating a service instance or when running a smoke test. The service creation issues an error message that starts with

```
Instance provisioning failed: There was a problem completing your request.
```

GemFire server logs at `/var/vcap/sys/log/gemfire-server/gemfire/server-<N>.log` will contain a disk-access error with the string

```
A DiskAccessException has occurred
```

and a stack trace similar to this one that begins with

```
org.apache.geode.cache.persistence.ConflictingPersistentDataException
    at org.apache.geode.internal.cache.persistence.PersistenceAdvisorImpl.checkMyStateOnMembers(PersistenceAdvisorImpl.java:743)
    at org.apache.geode.internal.cache.persistence.PersistenceAdvisorImpl.getInitialImageAdvice(PersistenceAdvisorImpl.java:819)
    at org.apache.geode.internal.cache.persistence.CreatePersistentRegionProcessor.getInitialImageAdvice(CreatePersistentRegionProcessor.java:52)
    at org.apache.geode.internal.cache.DistributedRegion.getInitialImageAndRecovery(DistributedRegion.java:1178)
    at org.apache.geode.internal.cache.DistributedRegion.initialize(DistributedRegion.java:1059)
    at org.apache.geode.internal.cache.GemFireCacheImpl.createVMRegion(GemFireCacheImpl.java:3089)
```

Cause of the Problem: The PCC VMs are underprovisioned; the quantity of disk space is too small.

Solution: Use Ops Manager to provision VMs of at least the minimum size. See [Configure Service Plans](#) for minimum-size details.

Create a pull request or raise an issue on the source for this page in [GitHub](#)

Pivotal Cloud Cache Release Notes

v1.9.2

Release Date: January 31, 2020

Features included in this release:

- Cloud Cache supports CA certificate rotation. See [Managing Certificates](#) for details.
- Fixed a bug which caused installation to fail when uppercase letters were used in network names.
- Upgraded the underlying Go language version for various internal components.
- Cloud Cache 1.9.2 runs Pivotal GemFire 9.8.6.

v1.9.1

Release Date: October 24, 2019

Features included in this release:

- Cloud Cache 1.9.1 runs Pivotal GemFire 9.8.4.
- The `cf marketplace` command displays the installed Cloud Cache and GemFire versions.
- Fixed a bug which prevented successful upgrade from Cloud Cache 1.8 to Cloud Cache 1.9.0 for version 1.8 service instances that have TLS encryption enabled.
- Fixed a bug in the `mem-check` post deploy errand that occurred when more than 50 service instances were present.



Note: Letters within the service network name must be lowercase letters for v1.9.1. Uppercase letters in the service network name will cause the installation to fail, apps will not be able to connect to the service instance, and service instance creation will fail. Version 1.9.2 fixes this issue, so upgrade to the most recent 1.9 version to avoid this issue.

v1.9.0



Warning: Cloud Cache 1.9.0 is no longer available for download due to an issue encountered when upgrading to Cloud Cache 1.9.0 from version 1.8 Cloud Cache service instances that have TLS encryption enabled.

Release Date: September 27, 2019

Features included in this release:

- Cloud Cache 1.9 supports TLS encryption across WAN connections



Warning: This is a breaking change. Service key format for setting up WAN has changed. See [Set Up a Bidirectional System with TLS](#) for more information.

- Cloud Cache 1.9 supports service instance sharing
- Cloud Cache runs Pivotal GemFire 9.8.3.
- Cloud Cache supports Pivotal Application Service (PAS) 2.7.
- Cloud Cache ships with JDK 1.8_192.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

