

## Pivotal Cloud Cache 1.6

**Note:** The support period for Pivotal Cloud Cache (PCC) 1.6 has expired, and this version is no longer supported. To stay up to date with the latest software and security updates, upgrade to a supported version.

## Table of Contents

Table of Contents	2
Pivotal Cloud Cache	3
Product Snapshot	5
PCC and Other PCF Services	7
PCC Architecture	8
Workflow to Set Up a PCC Service	10
Networking for On-Demand Services	11
Recommended Usage and Limitations	16
Security	17
Pivotal Cloud Cache Operator Guide	19
Requirements for Pivotal Cloud Cache	20
Preparing for TLS	21
Installing and Configuring Pivotal Cloud Cache	25
Setting Service Instance Quotas	39
Monitoring Pivotal Cloud Cache Service Instances	44
Upgrading Pivotal Cloud Cache	57
Migrating to a TLS-Enabled Cluster	61
Updating Pivotal Cloud Cache Plans	67
Uninstalling Pivotal Cloud Cache	68
Troubleshooting for Operators	69
Rotating Certificates	72
Pivotal Cloud Cache Developer Guide	73
Viewing All Plans Available for Pivotal Cloud Cache	75
Creating a Pivotal Cloud Cache Service Instance	76
Set Up WAN-Separated Service Instances	79
Set Up a Bidirectional System	80
Set Up a Unidirectional System	87
Set Up an Additional Bidirectional Interaction	94
Set Up an Additional Unidirectional Interaction	99
Setting Up Servers for an Inline Cache	104
Deleting a Service Instance	111
Updating a Pivotal Cloud Cache Service Instance	112
gfs Command Restrictions	114
Accessing a Service Instance	116
Using Pivotal Cloud Cache	122
Developing an App Under TLS	129
Connecting a Spring Boot App to Pivotal Cloud Cache with Session State Caching	132
Creating Continuous Queries Using Spring Data GemFire	137
Application Development	139
Design Patterns	140
Region Design	149
Handling Events	156
Example Applications	157
A Simple Java App	158
A Spring Boot App	160
Troubleshooting	164
Pivotal Cloud Cache Release Notes	165

## Pivotal Cloud Cache

In this topic:

- [Product Snapshot](#)
- [PCC and Other PCF Services](#)
- [PCC Architecture](#)
  - [GemFire Basics](#)
  - [The PCC Cluster](#)
  - [Member Communications](#)
- [Workflow to Set Up a PCC Service](#)
- [Networking for On-Demand Services](#)
- [Recommended Usage and Limitations](#)
- [Security](#)

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.

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, creating, and deleting PCC service instances
- Provides app developers instructions for binding apps



## Product Snapshot

The following table provides version and version-support information about PCC:

Element	Details
Version	v1.6.3
Release date	May 8, 2019
Software component version	GemFire v9.6.2
Pivotal GemFire Native Client version	v10.0.2
Compatible Ops Manager version(s)	v2.4.x and v2.3.x
Compatible Pivotal Application Service (PAS)* version(s)	v2.4.x and v2.3.x
IaaS support	AWS, Azure, GCP, OpenStack, and vSphere
IPsec support	Yes
Required BOSH stemcell version	Xenial 250.9 or a more recent version
Minimum Java buildpack version required for apps	v3.13

The following table identifies the VMware GemFire version incorporated into the VMware Tanzu Gemfire version.

Tanzu Gemfire for VMs version	Tanzu GemFire version	Apache Geode version
1.13.0 Beta		Build 312 for v1.13
1.12.0	9.10.2	
1.11.2	9.9.3	
1.11.1	9.9.2	
1.11.0	9.9.1	
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	

1.8.2	9.8.6	
1.8.1	9.8.3	
1.8.0*	9.8.1	
1.7.1	9.7.2	
1.7.0	9.7.1	
1.6.3	9.6.2	
1.6.2	9.6.1	
1.6.1	9.6.0	
1.6.0	9.6.0	

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

## PCC and Other PCF 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.

# PCC Architecture

## In this topic

[GemFire Basics](#)

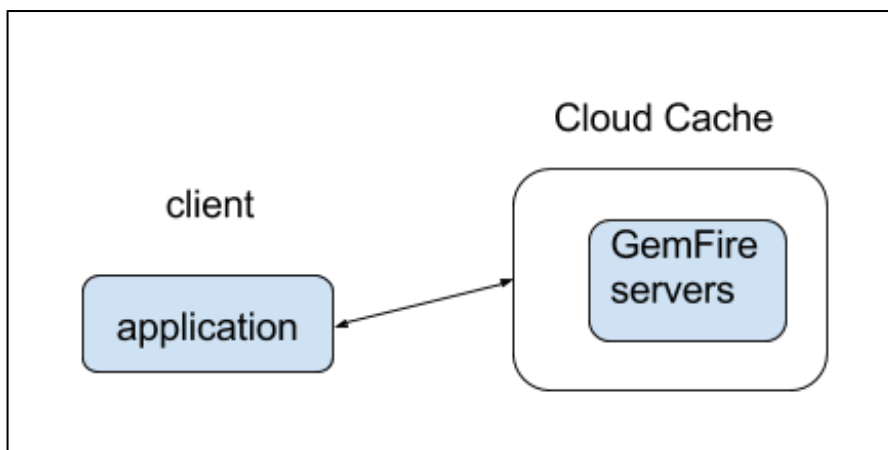
[The PCC Cluster](#)

[Member Communication](#)

## GemFire Basics

Pivotal GemFire is the data store within Pivotal Cloud Cache (PCC). A small amount of administrative GemFire setup is required for a PCC service instance, 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.

```
graph TD; Client subgraph P-CloudCache Cluster subgraph locators Locator1 Locator2 Locator3 end subgraph servers Server1 Server2 Server3 Server4 end end Client==>Locator1 Client-->Server1 Client-->Server2 Client-->Server3 Client-->Server4
```

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

```
graph TD; Client subgraph P-CloudCache Cluster subgraph locators Locator1 Locator2 Locator3 end subgraph servers Server1 Server2 Server3 Server4 Server5 Server6 Server7 end end Client==>Locator1 Client-->Server1 Client-->Server2 Client-->Server3 Client-->Server4 Client-->Server5 Client-->Server6 Client-->Server7
```

## 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.

```
sequenceDiagram participant Client participant Locator participant Server1 Client->>+Locator: What servers can I talk to? Locator->>-Client: Server1 Client->>Server1: Hello!
```

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

```
sequenceDiagram participant Client participant Server1 Client->>+Server1: What's the value for KEY? Server1->>-Client: VALUE
```

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

```
sequenceDiagram participant Client participant Server1 participant Server2 Client->>+Server1: What's the value for KEY? Server1->>+Server2: What's the value for KEY? Server2->>-Server1: VALUE Server1->>-Client: VALUE
```

## Workflow to Set Up a PCC Service

The workflow for the PCF admin setting up a PCC service plan:

```
graph TD; subgraph PCF_Admin_Actions s1 s2 end; subgraph Developer_Actions s4 end; s1[1. Upload P-CloudCache.pivotal to Ops Manager] --> s2[2. Configure CloudCache Service Plans, i.e. caching-small]; s2 --> s3[3. Ops Manager deploys CloudCache Service Broker]; s3 --> s4[4. Developer calls `cf create-service p-cloudcache caching-small test`]; s4 --> s5[5. Ops Manager creates a CloudCache cluster following the caching-small specifications];
```

## Networking for On-Demand Services

### In this topic

[Service Network Requirement](#)

[Default Network and Service Network](#)

[Required Networking Rules for On-Demand Services](#)

[PCC Instances Across WAN](#)

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

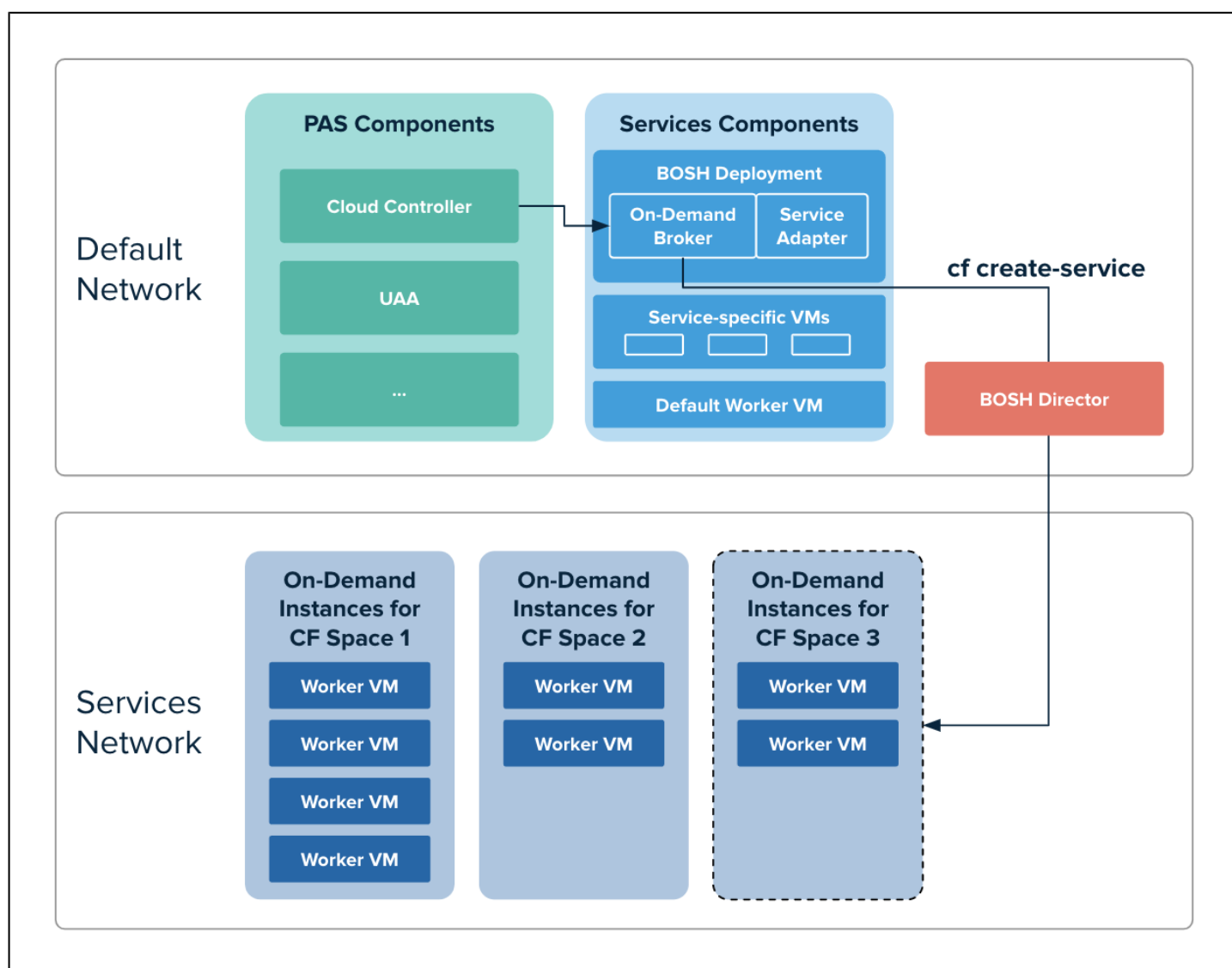
On-demand Pivotal Cloud Cache services use BOSH to dynamically deploy VMs and create single-tenant service instances in a dedicated network. On-demand services use the dynamically-provisioned service network to host single-tenant worker VMs. These worker VMs run as service instances within development 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
<b>BOSH Director</b>	Creates and updates service instances as instructed by ODB.
<b>BOSH Agent</b>	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.
<b>BOSH UAA</b>	Issues OAuth2 tokens for clients to use when they act on behalf of BOSH users.
<b>Pivotal Application Service</b>	Contains the apps that consume services.
<b>ODB</b>	Instructs BOSH to create and update services. Connects to services to create bindings.
<b>Deployed service instance</b>	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
<b>ODB</b>	<ul style="list-style-type: none"> <li><b>BOSH Director</b></li> <li><b>BOSH UAA</b></li> </ul>	<ul style="list-style-type: none"> <li>25555 (BOSH Director)</li> <li>8443 (UAA)</li> <li>8844 (CredHub)</li> </ul>	One-way	The BOSH Director and BOSH UAA default ports are not configurable. The CredHub default port is configurable.
		Specific to the		This connection is for

ODB	Deployed service instances	service (such as RabbitMQ for PCF). May be one or more ports.	One-way	administrative tasks. Avoid opening general use, app-specific ports for this connection.
ODB	PAS (or Elastic Runtime)	8443	One-way	The default port is not configurable.
Errand VMs	<ul style="list-style-type: none"> <li>PAS (or Elastic Runtime)</li> <li>ODB</li> <li>Deployed Service Instances</li> </ul>	<ul style="list-style-type: none"> <li>8443</li> <li>8080</li> <li>Specific to the service. May be one or more ports.</li> </ul>	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 (or Elastic Runtime)	Deployed service instances	Specific to the service. May be one or more ports.	One-way	This connection is for general use, app-specific tasks. Avoid opening administrative ports for this connection.
PAS (or Elastic Runtime)	ODB	8080	One-way	This port may be different for individual services. This port may also be configurable by the operator if allowed by the tile developer.

## 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](#) [↗](#).

## Recommended Usage and Limitations

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

## Limitations

- Scale down of the cluster is not supported.
- Plan migrations, for example, `-p` flag with the `cf update-service` command, are not supported.



## Security


Pivotal recommends that you do the following:

- Run PCC in its own network
- Use a load balancer to block direct, outside access to the Gorouter

To allow PCC network access from apps, you must create application security groups that allow access on the following ports:

- 1099
- 8080
- 40404
- 55221

For more information, see the PCF [Application Security Groups](#)  topic.

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

## Authentication

PCC service instances are created with three default GemFire user roles for interacting with clusters:

- A cluster operator manages the GemFire cluster and can access region data.
- A developer can access region data.
- A gateway sender propagates region data to another PCC service instance.

All client apps, gfsH, and JMX clients must authenticate as one of these user roles to access the cluster.

The identifiers assigned for these roles are detailed in [Create Service Keys](#).

## Authorization

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 the permission authorized to do the operation. Here are the permissions that each user role has:

- The cluster operator role has `CLUSTER:MANAGE`, `CLUSTER:WRITE`, `CLUSTER:READ`, `DATA:MANAGE`, `DATA:WRITE`, and `DATA:READ` permissions.
- The developer role has `CLUSTER:READ`, `DATA:WRITE`, and `DATA:READ` permissions.
- The gateway sender role has `DATA:WRITE` permission.

More details about these permissions are in the Pivotal GemFire manual under [Implementing Authorization](#) [↗](#).

## 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:

- **Requirements for Pivotal Cloud Cache**
- **Preparing for TLS**
  - **Overview**
  - **Provide or Generate a CA Certificate**
- **Installing and Configuring Pivotal Cloud Cache**
  - **Configure Tile Properties**
- **Setting Service Instance Quotas**
- **Monitoring Pivotal Cloud Cache Service Instances**
  - **Service Instance Metrics**
  - **Per Member Metrics**
  - **Gateway Sender and Gateway Receiver Metrics**
  - **Disk Metrics**
  - **Experimental Metrics**
  - **Total Memory Consumption**
- **Upgrading Pivotal Cloud Cache**
- **Migrating to a TLS-Enabled Cluster**
- **Updating Pivotal Cloud Cache Plans**
- **Uninstalling Pivotal Cloud Cache**
- **Troubleshooting for Operators**
- **Managing Certificates**

## Requirements for Pivotal Cloud Cache

- The [Networking for On-Demand Services](#) section describes networking requirements for PCC.
- As of PCC v1.6.2, PCC increases security by requiring TLS encryption for gfsd and Pulse. Follow the instructions in [Preparing for TLS](#) prior to installing the tile.


## Preparing for TLS

### In this topic


[Overview](#)

[Provide or Generate a CA Certificate](#)

[Add the CA Certificate](#)

 **warning:** As of PCC v1.6.2, PCC increases security by requiring TLS encryption for gfsh and Pulse. Complete the procedures in this topic before installing the PCC tile as part of an upgrade.

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 your 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 an encrypted connection 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.

- Enter the following command:

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

- 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.

- 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"
```

- 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**.



## Installing and Configuring Pivotal Cloud Cache

### In this topic

[Configure Tile Properties](#)

[Configure a Small Footprint Plan](#)

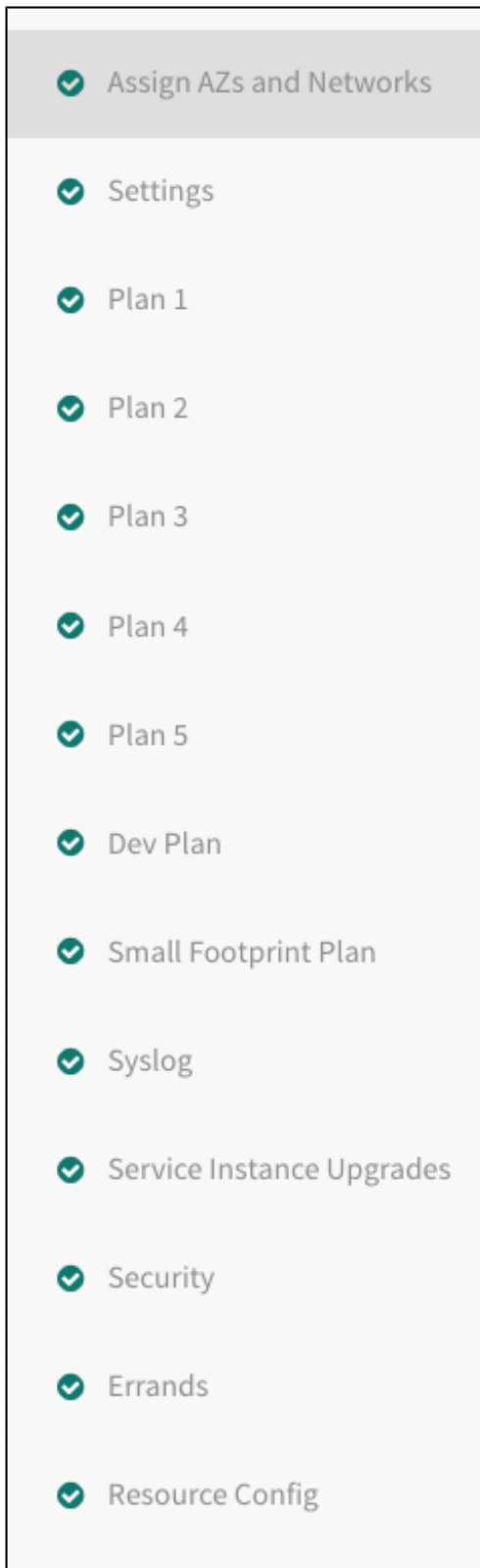
[Configure a Dev Plan](#)

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 the [Pivotal 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

### Smoke Test Settings

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)

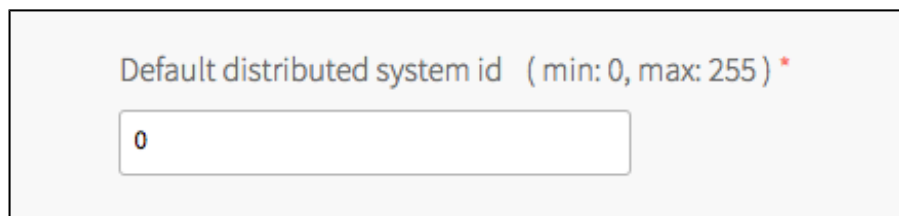
Please refer to the Ops Manager documentation for IaaS specific behavior. Log forwarding and backups may require internet access.

 **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.

## Default Distributed System ID Setting

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.

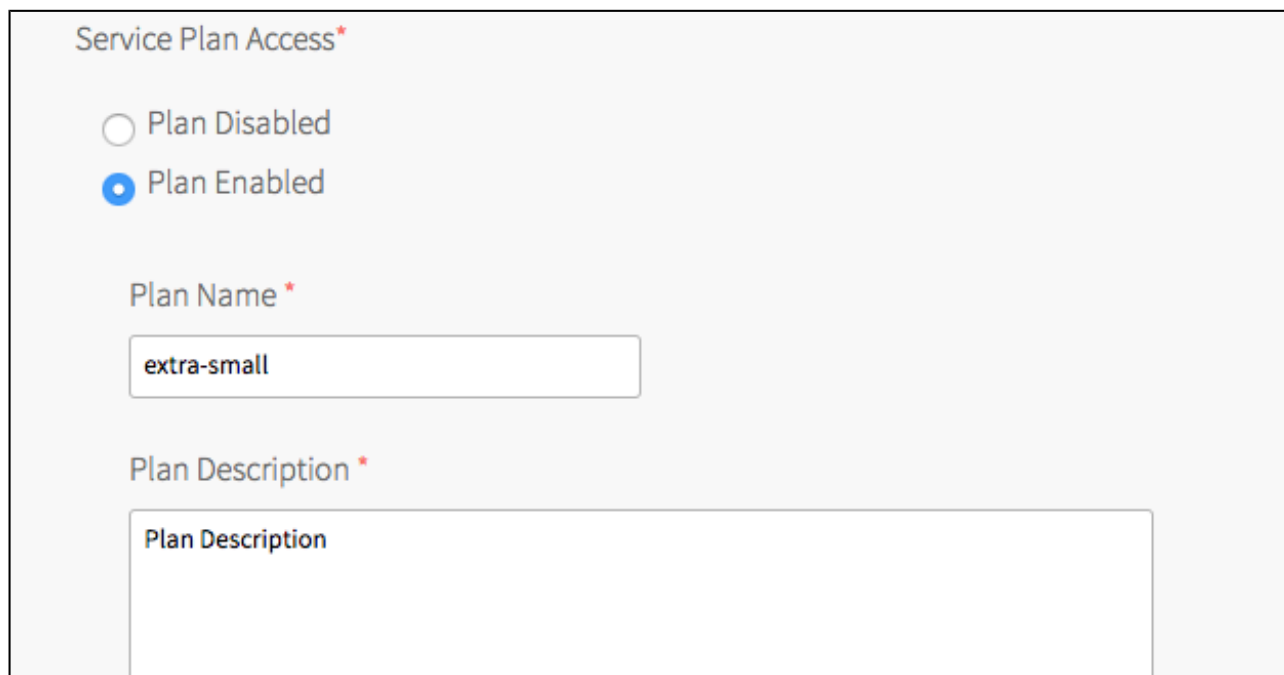
To change the default distributed system ID value, replace the default value of 0 with your new default value. Acceptable values are integers greater than or equal to 0 and less than or equal to 255.



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 \*

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)

Persistent disk type for the Locator VMs\*

10 GB

VM type for the Server VMs\*

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

Persistent disk type for the Server VMs\*

Automatic: 10 GB

Save

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 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 **Service Instance Quota** 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.

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.

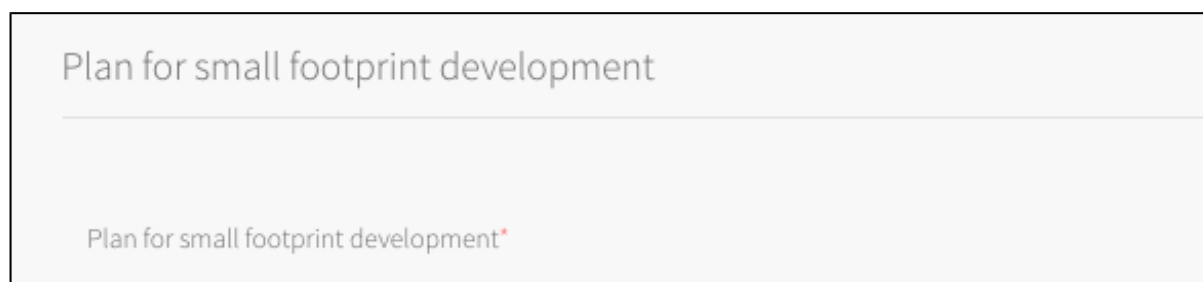
Scale up an existing service instance by increasing the number of servers, as specified in [Updating a Pivotal Cloud Cache Service Instance](#).

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 Disabled

☒ Plan Enabled

Plan Name \*

Plan Description \*

Plan Description

☐ Enable metrics for service instances

CF Service Access \*

Enable Service Access

Maximum service instances \*

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 ⬆ ⬇ ⬆

Save

## Syslog

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**.

External Syslog Host

logs.example.com ⓘ

External Syslog Port

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 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) \*

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) \*

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

Save

## Security

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:

1. Click **Security**.
2. Click on the box labeled **Enable Secure Service Instance Credentials** to enable use of CredHub.
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. See [Preparing for TLS](#) for how to prepare the PCF environment.
4. Click **Save**.

## Security properties

☐ Enable Secure Service Instance Credentials

When checked, service instance credentials will be stored in Credhub. Enable only when installing with PCF 2.0 or greater and this feature is also enabled in the PAS tile.

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. \*

Save

## 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**.

## Setting Service Instance Quotas

### In this topic

[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:

```
cf create-quota myquota -m 1024mb -i 16gb -r 30 -s 50 --allow-paid-service-plans
```

2. 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:



```
cf create-space-quota myspacequota -m 1024mb -i 16gb -r 30 -s 50 --allow-paid-service-plans
```

2. 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\*

Automatic: micro.cpu (cpu: 2, ram: 2 GB, disk: 8 GB)

Persistent disk type for the Locator VMs\*


Automatic: 10 GB

VM type for the Server VMs\*

micro (cpu: 1, ram: 1 GB, disk: 8 GB)

Persistent disk type for the Server VMs\*

20 GB

 **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**

## Monitoring Pivotal Cloud Cache Service Instances

### In this topic

#### Service Instance Metrics

- Member Count
- Total Available Heap Size
- Total Used Heap Size
- Total Available Heap Size as a Percentage

#### Per Member Metrics

- Memory Used as a Percentage
- Count of Java Garbage Collections
- CPU Utilization Percentage
- Average Latency of Get Operations
- Average Latency of Put Operations
- JVM pauses
- File Descriptor Limit
- Open File Descriptors
- Quantity of Remaining File Descriptors
- Threads Waiting for a Reply

#### Gateway Sender and Gateway Receiver Metrics

- Events Received at the Gateway Sender
- Events Queued by the Gateway Sender
- Events Received by the Gateway Receiver

#### Disk Metrics

- Average Latency of Disk Writes
- Quantity of Bytes on Disk
- Quantity of Available Bytes on Disk

#### Experimental Metrics

- Total Memory Consumption

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.

## 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

<b>Suggested measurement</b>	Every second
<b>Measurement Type</b>	pulse
<b>Why a KPI?</b>	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

<b>serviceinstance.UsedHeapSize</b>	
<b>Description</b>	Returns the total heap used across all instance members, in megabytes.
<b>Metric Type</b>	number
<b>Suggested measurement</b>	Every second
<b>Measurement Type</b>	pulse
<b>Why a KPI?</b>	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

<b>serviceinstance.UnusedHeapSizePercentage</b>	
<b>Description</b>	Returns the proportion of total available heap across all instance members, expressed as a percentage.
<b>Metric Type</b>	percent
<b>Suggested measurement</b>	Every second
<b>Measurement Type</b>	compound metric
<b>Warning Threshold</b>	40%
<b>Critical Threshold</b>	10%
<b>Suggested Actions</b>	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.
	If the total heap size and used heap size are too close, the system might see

## Why a KPI?

thrashing due to GC activity. This increases latency.

## Per Member Metrics

### Memory Used as a Percentage

<code>member.UsedMemoryPercentage</code>	
<b>Description</b>	RAM being consumed.
<b>Metric Type</b>	percent
<b>Suggested measurement</b>	Average over last 10 minutes
<b>Measurement Type</b>	average
<b>Warning Threshold</b>	75%
<b>Critical Threshold</b>	85%

### Count of Java Garbage Collections

<code>member.GarbageCollectionCount</code>	
<b>Description</b>	The number of times that garbage has been collected.
<b>Metric Type</b>	number
<b>Suggested measurement</b>	Sum over last 10 minutes
<b>Measurement Type</b>	count
<b>Warning Threshold</b>	Dependent on the IaaS and app use case.
<b>Critical Threshold</b>	Dependent on the IaaS and app use case.
<b>Suggested Actions</b>	Check the number of queries run against the system, which increases the deserialization of objects and increases garbage.
<b>Why a KPI?</b>	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	
<b>Description</b>	This member's process CPU utilization, expressed as a percentage.
<b>Metric Type</b>	percent
<b>Suggested measurement</b>	Average over last 10 minutes
<b>Measurement Type</b>	average
<b>Warning Threshold</b>	85%
<b>Critical Threshold</b>	95%
<b>Suggested Actions</b>	If this is not happening with high GC activity, the system is reaching its limits. Horizontal scaling might help.
<b>Why a KPI?</b>	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	
<b>Description</b>	The average latency of cache get operations, in nanoseconds.
<b>Metric Type</b>	number
<b>Suggested measurement</b>	Average over last 10 minutes
<b>Measurement Type</b>	average
<b>Warning Threshold</b>	Dependent on the IaaS and app use case.
<b>Critical Threshold</b>	Dependent on the IaaS and app use case.
<b>Suggested Actions</b>	If this is not happening with high GC activity, the system is reaching its limit. Horizontal scaling might help.
<b>Why a KPI?</b>	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	
<b>Description</b>	The average latency of cache put operations, in nanoseconds.
<b>Metric Type</b>	number
<b>Suggested measurement</b>	Average over last 10 minutes
<b>Measurement Type</b>	average
<b>Warning Threshold</b>	Dependent on the IaaS and app use case.
<b>Critical Threshold</b>	Dependent on the IaaS and app use case.
<b>Suggested Actions</b>	If this is not happening with high GC activity, the system is reaching its limit. Horizontal scaling might help.
<b>Why a KPI?</b>	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	
<b>Description</b>	The quantity of JVM pauses.
<b>Metric Type</b>	number
<b>Suggested measurement</b>	Sum over 2 seconds
<b>Measurement Type</b>	count
<b>Warning Threshold</b>	Dependent on the IaaS and app use case.
<b>Critical Threshold</b>	Dependent on the IaaS and app use case.
<b>Suggested Actions</b>	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.
<b>Why a KPI?</b>	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

<code>member.ReplyWaitsInProgress</code>	
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

<code>gatewaySender.&lt;sender-id&gt;.EventQueueSize</code>	
Description	The current size of the gateway sender queue.
Metric Type	number

<b>Measurement Type</b>	count
-------------------------	-------

## Events Received at the Gateway Sender

<code>gatewaySender.&lt;sender-id&gt;.EventsReceivedRate</code>	
<b>Description</b>	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.
<b>Metric Type</b>	number
<b>Measurement Type</b>	count

## Events Queued by the Gateway Sender

<code>gatewaySender.&lt;sender-id&gt;.EventsQueuedRate</code>	
<b>Description</b>	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.
<b>Metric Type</b>	number
<b>Measurement Type</b>	count

## Events Received by the Gateway Receiver

<code>gatewayReceiver.EventsReceivedRate</code>	
<b>Description</b>	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.
<b>Metric Type</b>	number
<b>Measurement Type</b>	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.DestroyRate`
- `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-xxxxxxxxxxxx1...  
JVM heap usage for service instance xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx1  
Used Total = 1204 MB  
Max Total = 3201 MB

Analyzing deployment xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx2...  
JVM heap usage for service instance xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx2  
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




## Upgrading Pivotal Cloud Cache

### In this topic

[Upgrade Procedure](#)

[Special Upgrade Procedure for Dev and Small Footprint Plans](#)

Upgrade minor release versions from your currently deployed version to the target version in sequential order. For example, 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](#) [↗](#).

 **warning:** To work around a race condition, follow a different procedure as detailed in [Special Upgrade Procedure for Dev and Small Footprint Plans](#) to upgrade to PCC v1.6.2 for plans that colocate system components on a single VM.

## Upgrade Procedure

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 Cloud Cache tile and configure the upgrade options.
  - 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](#).
  - Make sure that under the **Errands** section, the **Upgrade All Service Instances Post-Deploy Errand** is Default (On). **Save** the change.
5. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#) [↗](#)).


6. Click **Apply Changes**.

## Special Upgrade Procedure for Dev and Small Footprint Plans

This upgrade procedure applies to upgrades to PCC v1.6.2 for plans that colocate system components on a single VM. Those plans are the dev plan and the small footprint plan. This special upgrade procedure exists to work around a race condition that exists in the timing of bringing down and restarting the two components that communicate with each other and are both located within a single VM.

Use this procedure for upgrades from 1.5.x, 1.6.0, or 1.6.1 to 1.6.2.


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 Cloud Cache tile and configure the upgrade options.
  - 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](#). **Save** the change.
5. Click **Review Pending Changes** (see [Reviewing Pending Product Changes](#) .
6. At the bottom of the **Review Pending Changes** page within the **Pivotal CloudCache** specification, uncheck the box labeled **Upgrade All Service Instances**, as in this view:

## Review Pending Changes

☒ Select All Products

☒




**BOSH Director**  
 Version 2.5.0-build.135

Staged  
[SEE CHANGES](#)

**Depends on**  
 No Dependencies

☒




**Small Footprint PAS**  
 Version 2.5.0

Staged  
[SEE CHANGES](#)

**Depends on**  
 No Dependencies

[ERRANDS](#)

☒



**Pivotal CloudCache**  
 Version 1.7.0-build.80

Staged  
[SEE CHANGES](#)

**Depends on**  
 Small Footprint PAS (cf) >= 2.3.0

[ERRANDS](#)

**Select errands to run during the deploy**

☒ Register Broker  
☒ Smoke Tests  
☐ Upgrade All Service Instances

[APPLY CHANGES](#)

7. Click **Apply Changes**.

8. Once the upgrade is applied (without the upgrade of all service instances), the BOSH `upgrade-all-service-instances` errand must be successfully completed to finish the upgrade. This is the errand that is subject to the race condition. Therefore, the errand must be run an indeterminate number of times, as there is no way to predict whether or not the errand will hit the race condition. Sequentially run this errand until it succeeds:

```
bosh -e ENV -d DEPLOYMENT run-errand upgrade-all-service-instances
```

Acquire the broker's `DEPLOYMENT` name:

```
$ bosh -e my-pcf deployments | grep p-cloudcache
```

This is an example of running the errand:

```
$ bosh -e my-pcf -d p-cloudcache_1fd2850e-b754-4c5e-aa5c-ddb54ee301e6 run-errand upgrade-all-service-instances
```

## Migrating to a TLS-Enabled Cluster

An existing PCC service instance that does not use TLS encryption may be migrated to become a PCC service instance with TLS encryption enabled.

**⚠ warning!** This procedure will require downtime for the service instance during the migration.

Follow the procedure given here *after* these prerequisites have been met:

- All steps within **Preparing for TLS** have been completed.
- The service instance has been upgraded to PCC v1.5.2 or a more recent PCC version. There will be no PCC version change during the migration.

Follow this procedure to migrate the existing PCC service instance:

1. As a PCF operator, stop all apps. First, list all apps to identify the `APP_NAME`.

```
$ cf apps
```

Then, stop each app with:

```
$ cf stop APP_NAME
```

2. For all non-persistent regions, use the `gfsh` command line tool to export the data.

**⚠ warning!** Without an export, all non-persistent region entries will be irretrievably lost.

- Complete the steps within **Accessing a Service Instance** to acquire the correct version of `gfsh`, run it, and connect to the cluster using the cluster operator role/credentials from the service key.
- List the regions.

```
gfsh>list regions
```

- For each region, use `gfsh describe` to determine if the region is persistent or not and to acquire a server name.

```
gfsh>describe region --name=REGION_NAME
```

- For each non-persistent region, use this single `gfsh` command to export all the data within the region. The `SERVER_NAME` identifies which GemFire server receives the `export` command and propagates the command to all other GemFire servers within the cluster.

```
gfsh>export data --parallel --region=REGION_NAME --member=SERVER_NAME --dir=/var/vcap/store/gemfire-server
```

3. Your PCF operator needs to target the BOSH Director in order to acquire the `DEPLOYMENT_NAME`.

- Run

```
$ cf service SERVICE_INSTANCE_NAME
```

to acquire the digits that uniquely identify the service instance. The digits (`xxx-xxx` in the following instructions) are those between `cloudcache-` and the period `.`.

- Log in to the BOSH Director.

```
$ bosh log-in
```

- The `DEPLOYMENT_NAME` will appear in the output of

```
$ bosh deployments | grep XXX-XXX
```

#### 4. Using PCF operator credentials, stop the BOSH deployment:

```
$ bosh -d DEPLOYMENT_NAME stop
```

and type “y” when prompted.

#### 5. Acquire the BOSH manifest with:

```
$ bosh -d DEPLOYMENT_NAME manifest > DEPLOYMENT_NAME-manifest.yml
```

6. Edit the acquired BOSH manifest. There are three locations within the manifest file that will require additions. These three locations are identified within this anonymized portion of the manifest file with the symbols ①, ②, and ③. The first part of the manifest file is omitted, as its listed values change based on the PCC version. Real passwords have been replaced with the placeholder `password`, and user names have been replaced with the placeholder `userX` within this example.

```
instance_groups:
- name: locator
  instances: 3
  jobs:
  - name: gemfire-locator
    release: gemfire
    properties:
      gemfire: ①
        distributed-system-id: 0
        locator:
          bpm_enabled: true
          port: '55221'
          properties:
            enable-time-statistics: true
        persist-pdx: true
        security:
          internal_cluster_password: password
          internal_cluster_username: userX
        roles:
          cluster_operator:
            - CLUSTER:WRITE
            - CLUSTER:READ
            - DATA:MANAGE
            - DATA:WRITE
            - DATA:READ
            - CLUSTER:MANAGE:DEPLOY
            - CLUSTER:MANAGE
            - CLUSTER:MANAGE:GATEWAY
          developer:
            - CLUSTER:READ
            - DATA:WRITE
            - DATA:READ
          gateway:
            - DATA:WRITE
        users:
          cluster_operator_userX:
            password: password
            roles:
```

```

        - cluster_operator
        developer_userX:
          password: password
          roles:
            - developer
- name: route_registrar
  release: routing
  consumes:
    nats:
      deployment: cf-NNNNNNNNNNNN
      from: nats
  properties:
    route_registrar:
      routes:
        - name: cloudcache
          port: 8080 ②
          registration_interval: 20s
          uris:
            - cloudcache-XXX-XXX.example.com
- name: bpm
  release: bpm
  vm_type: micro.cpu
  stemcell: stemcell
  persistent_disk_type: '10240'
  azs:
    - us-centrall-f
  networks:
    - name: example-services-subnet
- name: server
  instances: 4
  jobs:
    - name: gemfire-server
      release: gemfire
      properties:
        gemfire:
          server:
            bpm_enabled: true
            create-gateway-receiver: true
            development-mode: false
            properties:
              enable-time-statistics: true
              jmx-manager-start: true
            security:
              gateway_password: password
              gateway_username: gateway_sender_userX
    - name: prime-cluster-for-pcc
      release: gemfire
    - name: bpm
      release: bpm
  vm_type: medium.cpu
  stemcell: stemcell
  persistent_disk_type: '10240'
  azs:
    - us-centrall-f
  networks:
    - name: example-services-subnet
update:
  canaries: 1
  canary_watch_time: 1000-600000
  update_watch_time: 1000-600000
  max_in_flight: 32
  serial: true
features:
  converge_variables: true ③

```

Add lines to the BOSH manifest, using the lines as shown in red in the following modified version of the manifest. Substitute your digits that uniquely identify your service instance for `xxx-xxx` within the *added* lines.

```
instance_groups:
- name: locator
  instances: 3
  jobs:
  - name: gemfire-locator
    release: gemfire
    properties:
      gemfire: ①
      tls: true
      truststore_password: ((trust-store-password))
      keystore_password: ((key-store-password))
      certificate: ((gemfire-certificate))
      trusted_certs:
      - ((/cf/diego-instance-identity-root-ca))
      - ((/services/tls_ca))
      distributed-system-id: 0
      locator:
        bpm_enabled: true
        port: '55221'
        properties:
          enable-time-statistics: true
      persist-pdx: true
      security:
        internal_cluster_password: password
        internal_cluster_username: userX
      roles:
        cluster_operator:
        - CLUSTER:WRITE
        - CLUSTER:READ
        - DATA:MANAGE
        - DATA:WRITE
        - DATA:READ
        - CLUSTER:MANAGE:DEPLOY
        - CLUSTER:MANAGE
        - CLUSTER:MANAGE:GATEWAY
        developer:
        - CLUSTER:READ
        - DATA:WRITE
        - DATA:READ
        gateway:
        - DATA:WRITE
      users:
        cluster_operator_userX:
          password: password
          roles:
          - cluster_operator
        developer_userX:
          password: password
          roles:
          - developer
- name: route_registrar
  release: routing
  consumes:
    nats:
      deployment: cf-NNNNNNNNNN
      from: nats
  properties:
    route_registrar:
      routes:
      - name: cloudcache
        port: 8080 ②
        tls_port: 8080
        server_cert_domain_san: cloudcache-XXX-XXX.example.com
        registration_interval: 20s
        uris:
        - cloudcache-XXX-XXX.example.com
- name: bpm
  release: bpm
  vm_type: micro.cpu
```



```

stemcell: stemcell
persistent_disk_type: '10240'
azs:
- us-central1-f
networks:
- name: example-services-subnet
- name: server
instances: 4
jobs:
- name: gemfire-server
  release: gemfire
  properties:
    gemfire:
      server:
        bpm_enabled: true
        create-gateway-receiver: true
        development-mode: false
        properties:
          enable-time-statistics: true
          jmx-manager-start: true
        security:
          gateway_password: password
          gateway_username: gateway_sender_userX
- name: prime-cluster-for-pcc
  release: gemfire
- name: bpm
  release: bpm
vm_type: medium.cpu
stemcell: stemcell
persistent_disk_type: '10240'
azs:
- us-central1-f
networks:
- name: example-services-subnet
update:
  canaries: 1
  canary_watch_time: 1000-600000
  update_watch_time: 1000-600000
  max_in_flight: 32
  serial: true
features:
  converge_variables: true ③
variables:
- name: trust-store-password
  type: password
- name: key-store-password
  type: password
- name: gemfire-certificate
  type: certificate
options:
  ca: /services/tls_ca
  common_name: gemfire-ssl
  alternative_names:
  - gemfire-ssl
  - cloudcache-XXX-XXX.example.com

```

7. Redeploy the BOSH manifest. Do a BOSH deploy using the edited BOSH manifest:

```
$ bosh -d SERVICE-INSTANCE-NAME deploy SERVICE-INSTANCE-NAME-manifest.yml
```

and type “y” when prompted.

8. Restart the cluster with a sequential BOSH start:

```
$ bosh start -d SERVICE-INSTANCE-NAME --max-in-flight=1
```

and type “y” when prompted.


9. Run `gfsh` and follow the directions in [Connect with gfsh over HTTPS](#) to connect to the TLS-enabled cluster.
10. Use `gfsh` to import all region data that was exported earlier in this procedure. For each earlier-exported region, do:

```
gfsh>import data --parallel --region=REGION_NAME --member=SERVER_NAME --dir=/var/vcap/store/gemfire-server
```

11. Revise the app such that it works with a TLS-enabled PCC service instance by following the instructions within [Developing an App Under TLS](#). Re-build, re-deploy, and start the app.

## 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.

## 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**.

## Troubleshooting for Operators

### In this topic

[View Statistics Files](#)

[Smoke Test Failures](#)

[General Connectivity](#)

### View Statistics Files

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 following command:

```
`bosh2 -e BOSH-ENVIRONMENT -d DEPLOYMENT-NAME scp server/0:/var/vcap/store/gemfire-server/statistics.gfs /tmp`
```

See the Pivotal GemFire [Installing and Running VSD](#) [↗](#) topic for information about loading the statistics files into Pivotal GemFire VSD.

### Smoke Test Failures

Error: “Creating p-cloudcache SERVICE-NAME failed”

The smoke tests could not create an instance of GemFire. 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: “Deleting SERVICE-NAME failed”

The smoke test attempted to clean up a service instance it created and failed to delete the service using the `cf delete-service` command. To troubleshoot this issue, run BOSH `logs` to view the logs on the broker or the service instance to see why the deletion may have failed.

Error: Cannot connect to the cluster SERVICE-NAME

The smoke test was unable to connect to the cluster.

To troubleshoot the issue, 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 gfs CLI. This requires creating a service key.

Error: “Could not perform create/put on Cloud Cache cluster”

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: “Could not retrieve value from Cloud Cache cluster”

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. 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

### Client-to-Server Communication

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 (or Elastic Runtime) network to service the network.

### Membership Port Range

PCC servers and locators communicate with each other using UDP and TCP. The current port range for this communication is `49152-65535`.

If you have a firewall between VMs, ensure this port range is open.

### Port Range Usage Across a WAN


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.

Ensure this port range is open when WAN-separated service instances will communicate.

## 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, follow the procedure in [Rotating the Services TLS CA and Its Leaf Certificates](#) .



## Pivotal Cloud Cache Developer Guide

This document describes how a Pivotal Cloud Foundry (PCF) app developer can choose a service plan, create and delete Pivotal Cloud Cache (PCC) 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:

- [Viewing All Plans Available for Pivotal Cloud Cache](#)
- [Creating a Pivotal Cloud Cache Service Instance](#)
  - [Provide Optional Parameters](#)
  - [Enable Session State Caching with the Java Buildpack](#)
  - [Enable Session State Caching Using Spring Session](#)
  - [Dev Plans](#)
- [Set Up WAN-Separated Service Instances](#)
  - [Set Up a Bidirectional System](#)
  - [Set Up a Unidirectional System](#)
- [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](#)
- [Deleting a Service Instance](#)
- [Updating a Pivotal Cloud Cache Service Instance](#)
  - [Rebalancing a Cluster](#)
  - [Restarting a Cluster](#)
  - [About Changes to the Service Plan](#)
- [gfsh Command Restrictions](#)
- [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 Pivotal Cloud Cache
  - Create Regions with gfsh
  - Working with Disk Stores
  - Java Build Pack Requirements
  - Bind an App to a Service Instance
  - 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
- Developing an App Under TLS
- 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

## Viewing All Plans Available for Pivotal Cloud Cache

Run `cf marketplace -s p-cloudcache` to view all plans available for PCC. 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

## Creating a Pivotal Cloud Cache Service Instance

### In this topic

[Provide Optional Parameters](#)

[Enable Session State Caching with the Java Buildpack](#)

[Enable Session State Caching Using Spring Session](#)

[Dev Plans](#)

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.

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}'
```

## 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 gfsh using the `cluster_operator_XXX` credentials:

```
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](#).

## Set Up WAN-Separated Service Instances

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 within 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

`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.

 **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.

## Set Up a Bidirectional System

This sequence of steps sets up a bidirectional transfer, as will be needed for an active-active pattern, as described in **Bidirectional Replication Across a WAN**.

1. Create the cluster A service instance using the cluster A Cloud Foundry credentials. This example explicitly sets the `distributed_system_id` of cluster A using a `-c` option with a command of the form:

```
cf create-service p-clouddcache PLAN-NAME SERVICE-INSTANCE-NAME -c '{
"distributed_system_id" : ID-VALUE }'
```

Here is a cluster A example of the `create-service` command:

```
$ cf create-service p-clouddcache wan-cluster wan1 -c '{
"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.

2. Create a service key for cluster A. The service key will contain generated credentials that this example will use in the creation of the cluster B service instance:

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

Within the service key, each `username` is generated with a unique string appended so there will be unique user names for the different roles. The user names in this example have been modified to be easy to understand, and they are not representative of the user names that will be generated upon service key creation. Passwords generated for the service key are output in clear text. The passwords shown in this example have been modified to be easy to understand, and they are not representative of the passwords that will be generated upon service key creation. Here is sample output from `cf service-key wan1 k1`:



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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]"
    "10.0.16.22[55221]"
    "10.0.16.23[55221]"
  ],
  "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": {
    "sender_credentials": {
      "active": {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    }
  }
}
```

3. Communicate the cluster A locators' IP and port addresses and `sender_credentials` to the cluster B Cloud Foundry administrator.
4. Create the cluster B service instance using the cluster B Cloud Foundry credentials. This example explicitly sets the `distributed_system_id`. Use a `-c` option with the command to specify the `distributed_system_id`, the cluster A service instance's locators, and the cluster A `sender_credentials`:

```
$ cf create-service p-cloudeache wan-cluster wan2 -c '
{
  "distributed_system_id":2,
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.16.21[55221]",
        "10.0.16.22[55221]",
        "10.0.16.23[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.

## 5. Create the service key of cluster B:

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

Here is sample output from `cf service-key wan2 k2`, which outputs details of the cluster B service key:

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

```
{
  "distributed_system_id": "2",
  "locators": [
    "10.0.24.21[55221]"
    "10.0.24.22[55221]"
    "10.0.24.23[55221]"
  ],
  "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": [
          "10.0.16.21[55221]",
          "10.0.16.21[55221]",
          "10.0.16.21[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    }
  }
}
```

6. Communicate the cluster B locators' IP and port addresses and `sender_credentials` to the cluster A Cloud Foundry administrator.
7. Update the cluster A service instance using the cluster A Cloud Foundry credentials to include the cluster B locators and the cluster B `sender_credentials` :

```
$ cf update-service wan1 -c '
{
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.24.21[55221]",
        "10.0.24.22[55221]",
        "10.0.24.23[55221]",
        "trusted_sender_credentials":[
          {
            "username":"gateway_sender_PQR",
            "password":"gws-PQR-password"
          }
        ]
      }
    }
  ]
}'
Updating service instance wan1 as admin
```

8. To observe and verify that the cluster A service instance has been correctly updated, it is necessary to delete and recreate the cluster A service key. As designed, the recreated service key will have the same user identifiers and passwords; new unique strings and passwords are not generated. Use the cluster A Cloud Foundry credentials in these commands:

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

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

The cluster A service key will now appear as:

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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]",
    "10.0.16.22[55221]",
    "10.0.16.23[55221]"
  ],
  "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": [
          "10.0.24.21[55221]",
          "10.0.24.22[55221]",
          "10.0.24.23[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_PQR"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    }
  }
}
```

9. Use gfsh to create the cluster A gateway sender and the region. Any region operations that occur after the region is created on cluster A, but before the region is created on cluster B will be lost.

- Connect using gfsh and the cluster A `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-1.example.com/gemfire/v1 --use-http --user=cluster_operator_ABC --password=cl-op-ABC-password
```

- 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
```

## 10. Use gfsh to create the cluster B gateway sender and region.

- Connect using gfsh and the cluster B `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-2.example.com/gemfire/v1 --use-http --user=cluster_operator_JKL --password=cl-op-JKL-password
```

- 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
```

## Set Up a Unidirectional System

This sequence of steps sets up a unidirectional transfer, 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](#).

1. Create the cluster A service instance using the cluster A Cloud Foundry credentials. This example explicitly sets the `distributed_system_id` of cluster A using a `-c` option with a command of the form:

```
cf create-service p-cloudcache PLAN-NAME SERVICE-INSTANCE-NAME -c '{
"distributed_system_id" : ID-VALUE }'
```

Here is a cluster A example of the `create-service` command:

```
$ cf create-service p-cloudcache wan-cluster wan1 -c '{
"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 completes.

2. Create a service key for cluster A. The service key will contain generated credentials that this example will use in the creation of the cluster B service instance:

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

Within the service key, each `username` is generated with a unique string appended so there will be unique user names for the different roles. The user names in this example have been modified to be easy to understand, and they are not representative of the user names that will be generated upon service key creation. Passwords generated for the service key are output in clear text. The passwords shown in this example have been modified to be easy to understand, and they are not representative of the passwords that will be generated upon service key creation. Here is sample output from `cf service-key wan1 k1`:

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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]"
    "10.0.16.22[55221]"
    "10.0.16.23[55221]"
  ],
  "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": {
    "sender_credentials": {
      "active": {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    }
  }
}
```

3. Communicate the cluster A locators' IP and port addresses and `sender_credentials` to the cluster B Cloud Foundry administrator.
4. Create the cluster B service instance using the cluster B Cloud Foundry credentials. This example explicitly sets the `distributed_system_id`. Use a `-c` option with the command to specify the `distributed_system_id`, the cluster A service instance's locators, and the cluster A `sender_credentials`:



```
$ cf create-service p-cloudeache wan-cluster wan2 -c '
{
  "distributed_system_id":2,
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.16.21[55221]",
        "10.0.16.22[55221]",
        "10.0.16.23[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.

## 5. Create the service key of cluster B:

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

Note that the cluster B service key will contain unneeded (for the unidirectional setup) but automatically created `sender_credentials`. Here is sample output from `cf service-key wan2 k2`, which outputs details of the cluster B service key:

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

```
{
  "distributed_system_id": "2",
  "locators": [
    "10.0.24.21[55221]"
    "10.0.24.22[55221]"
    "10.0.24.23[55221]"
  ],
  "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": [
          "10.0.16.21[55221]",
          "10.0.16.21[55221]",
          "10.0.16.21[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-PQR-password",
        "username": "gateway_sender_PQR"
      }
    }
  }
}
```

6. Communicate the cluster B locators' IP and port addresses to the cluster A Cloud Foundry administrator.
7. Update the cluster A service instance using the cluster A Cloud Foundry credentials to include the cluster B locators:

```
$ cf update-service wan1 -c '
{
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.24.21[55221]",
        "10.0.24.22[55221]",
        "10.0.24.23[55221]"
      ]
    }
  ]
}'
Updating service instance wan1 as admin
```

8. To observe and verify that the cluster A service instance has been correctly updated, it is necessary to delete and recreate the cluster A service key. As designed, the recreated service key will have the same user identifiers and passwords; new unique strings and passwords are not generated. Use the cluster A Cloud Foundry credentials in these commands:

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

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

The cluster A service key will now appear as:

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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]",
    "10.0.16.22[55221]",
    "10.0.16.23[55221]"
  ],
  "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": [
          "10.0.24.21[55221]",
          "10.0.24.22[55221]",
          "10.0.24.23[55221]"
        ]
      }
    ]
  },
  "sender_credentials": {
    "active": {
      "password": "gws-GHI-password",
      "username": "gateway_sender_GHI"
    }
  }
}
```

9. Use gfsh to create the cluster A gateway sender and the region. Any region operations that occur after the region is created on cluster A, but before the region is created on cluster B will be lost.

- Connect using gfsh and the cluster A `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-1.example.com/gemfire/v1 --use-http --user=cluster_operator_ABC --password=cl-op-ABC-password
```

- 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
```

## 10. Use gfsh to create the cluster B region.

- Connect using gfsh and the cluster B `cluster_operator` credentials, which are needed to be authorized for the create operation:

```
gfsh>connect --url=https://clouddcache-2.example.com/gemfire/v1 --use-http --user=cluster_operator_JKL --password=cl-op-JKL-password
```

- Create the cluster B region:

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

## Set Up an Additional Bidirectional Interaction

Follow this sequence of steps to set up a bidirectional transfer over WAN between two PCC service instances, once an initial setup is in place for a first pair of PCC service instances.

Call the first pair of PCC service instances A and B. This set of directions sets up an interaction between service instance A and service instance C. Service instance A is already created and has a service key.

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. Communicate the cluster A locators' IP and port addresses and `sender_credentials` to the cluster C Cloud Foundry administrator.
2. Create the cluster C service instance using the cluster C Cloud Foundry credentials. This example explicitly sets the `distributed_system_id`. Use a `-c` option with the command to specify the `distributed_system_id`, the cluster A service instance's locators, and the cluster A `sender_credentials`:

```
$ cf create-service p-cloudcache wan-cluster wan3 -c '{
  "distributed_system_id":3,
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.16.21[55221]",
        "10.0.16.22[55221]",
        "10.0.16.23[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 completes.

3. Create the service key of cluster C:

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

Here is sample output from `cf service-key wan3 k3`, which outputs details of the cluster C service key:

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

```
{
  "distributed_system_id": "3",
  "locators": [
    "10.0.32.21[55221]"
    "10.0.32.22[55221]"
    "10.0.32.23[55221]"
  ],
  "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": [
          "10.0.16.21[55221]",
          "10.0.16.21[55221]",
          "10.0.16.21[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-YZA-password",
        "username": "gateway_sender_YZA"
      }
    }
  }
}
```

4. Communicate the cluster C locators' IP and port addresses and `sender_credentials` to the cluster A Cloud Foundry administrator.
5. Update the cluster A service instance using the cluster A Cloud Foundry credentials to include the cluster C locators and the cluster C `sender_credentials`. The cluster A service instance must specify as `remote_locators` and `trusted_sender_credentials` the details for all clusters it interacts with. For this example, that is both clusters B and C:

```
$ cf update-service wan1 -c '
{
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.24.21[55221]",
        "10.0.24.22[55221]",
        "10.0.24.23[55221]",
        "10.0.32.21[55221]",
        "10.0.32.22[55221]",
        "10.0.32.23[55221]",
      ],
      "trusted_sender_credentials":[
        {
          "username":"gateway_sender_PQR",
          "password":"gws-PQR-password"
        },
        {
          "username":"gateway_sender_YZA",
          "password":"gws-YZA-password"
        }
      ]
    }
  ]
}'
Updating service instance wan1 as admin
```

6. To observe and verify that the cluster A service instance has been correctly updated, it is necessary to delete and recreate the cluster A service key. As designed, the recreated service key will have the same user identifiers and passwords; new unique strings and passwords are not generated. Use the cluster A Cloud Foundry credentials in these commands:

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

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

The cluster A service key will now appear as:



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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]",
    "10.0.16.22[55221]",
    "10.0.16.23[55221]"
  ],
  "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": [
          "10.0.24.21[55221]",
          "10.0.24.22[55221]",
          "10.0.24.23[55221]",
          "10.0.32.21[55221]",
          "10.0.32.22[55221]",
          "10.0.32.23[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_PQR",
          "gateway_sender_YZA"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-GHI-password",
        "username": "gateway_sender_GHI"
      }
    }
  }
}
```

7. Use gfsh to create the cluster A gateway sender and alter the existing region.

- Connect using gfsh and the cluster A `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-1.example.com/gemfire/v1 --use-http --user=cluster_operator_ABC --password=cl-op-ABC-password
```

- 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
```

## 8. Use gfsh to create the cluster C gateway sender and region.

- Connect using gfsh and the cluster C `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-3.example.com/gemfire/v1 --use-http --user=cluster_operator_STU --password=cl-op-STU-password
```

- 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
```

## Set Up an Additional Unidirectional Interaction

Follow this sequence of steps to set up an additional unidirectional transfer over WAN between two PCC service instances, once an initial setup is in place for a first pair of PCC service instances.

Call the first pair of PCC service instances A and B. This set of directions sets up a unidirectional interaction from service instance A to service instance C. Service instance A is already created and has a service key.

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. Communicate the cluster A locators' IP and port addresses and `sender_credentials` to the cluster C Cloud Foundry administrator.
2. Create the cluster C service instance using the cluster C Cloud Foundry credentials. This example explicitly sets the `distributed_system_id`. Use a `-c` option with the command to specify the `distributed_system_id`, the cluster A service instance's locators, and the cluster A `sender_credentials`:

```
$ cf create-service p-cloudcache wan-cluster wan3 -c '{
  "distributed_system_id":3,
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.16.21[55221]",
        "10.0.16.22[55221]",
        "10.0.16.23[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 completes.

3. Create the service key of cluster C:

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

Note that the cluster C service key will contain unneeded (for the unidirectional setup) but automatically created `sender_credentials`. Here is sample output from `cf service-key wan3 k3`, which outputs details of the cluster C service key:

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

```
{
  "distributed_system_id": "3",
  "locators": [
    "10.0.32.21[55221]"
    "10.0.32.22[55221]"
    "10.0.32.23[55221]"
  ],
  "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": [
          "10.0.16.21[55221]",
          "10.0.16.21[55221]",
          "10.0.16.21[55221]"
        ],
        "trusted_sender_credentials": [
          "gateway_sender_GHI"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-YZA-password",
        "username": "gateway_sender_YZA"
      }
    }
  }
}
```

4. Communicate the cluster C locators' IP and port addresses to the cluster A Cloud Foundry administrator.
5. Update the cluster A service instance using the cluster A Cloud Foundry credentials to include the cluster C locators. The cluster A service instance must specify as `remote_locators` the details for all clusters it interacts with. For this example, that is both clusters B and C:

```
$ cf update-service wan1 -c '
{
  "remote_clusters":[
    {
      "remote_locators":[
        "10.0.24.21[55221]",
        "10.0.24.22[55221]",
        "10.0.24.23[55221]",
        "10.0.32.21[55221]",
        "10.0.32.22[55221]",
        "10.0.32.23[55221]"
      ]
    }
  ]
}'
Updating service instance wan1 as admin
```

6. To observe and verify that the cluster A service instance has been correctly updated, it is necessary to delete and recreate the cluster A service key. As designed, the recreated service key will have the same user identifiers and passwords; new unique strings and passwords are not generated. Use the cluster A Cloud Foundry credentials in these commands:

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

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

The cluster A service key will now appear as:

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

```
{
  "distributed_system_id": "1",
  "locators": [
    "10.0.16.21[55221]",
    "10.0.16.22[55221]",
    "10.0.16.23[55221]"
  ],
  "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": [
          "10.0.24.21[55221]",
          "10.0.24.22[55221]",
          "10.0.24.23[55221]",
          "10.0.32.21[55221]",
          "10.0.32.22[55221]",
          "10.0.32.23[55221]"
        ]
      }
    ]
  },
  "sender_credentials": {
    "active": {
      "password": "gws-GHI-password",
      "username": "gateway_sender_GHI"
    }
  }
}
```

7. Use gfsh to create the cluster A gateway sender and alter the existing region.

- Connect using gfsh and the cluster A `cluster_operator` credentials, which are needed to be authorized for the gateway sender creation operation:

```
gfsh>connect --url=https://cloudcache-1.example.com/gemfire/v1 --use-http --user=cluster_operator_ABC --password=cl-op-ABC-password
```

- 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
```

## 8. Use gfsh to create the cluster C region.

- Connect using gfsh and the cluster C `cluster_operator` credentials, which are needed to be authorized for the create operation:

```
gfsh>connect --url=https://cloudcache-3.example.com/gemfire/v1 --use-http --user=cluster_operator_STU --password=cl-op-STU-password
```

- Create the cluster C region:

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

## Setting Up Servers for an Inline Cache

### In this topic

[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](#)

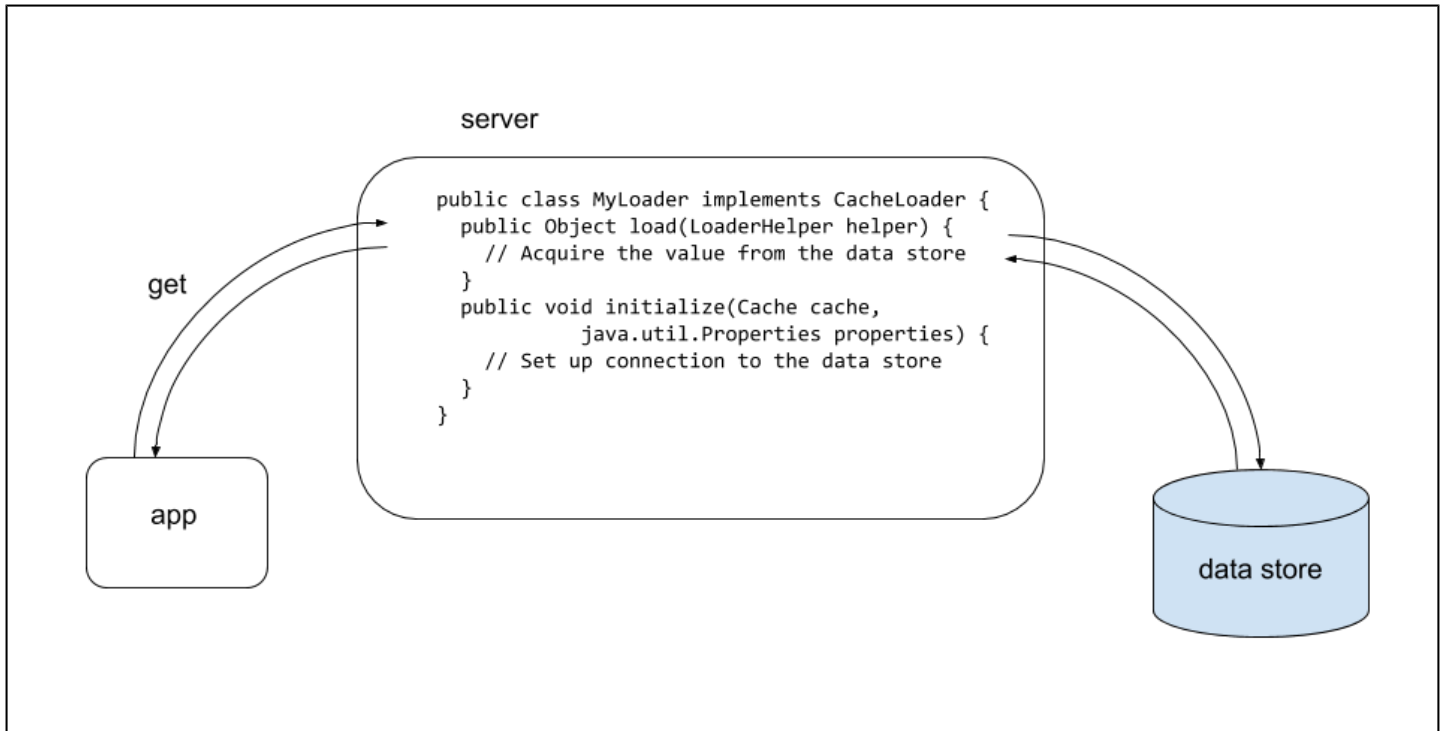
See [The Inline Cache](#) for an introductory description of an inline cache. The implementation of an inline cache requires custom code deployed on the GemFire servers to interact with the backend data store for read misses and for writes.

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 [Pivotal 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.

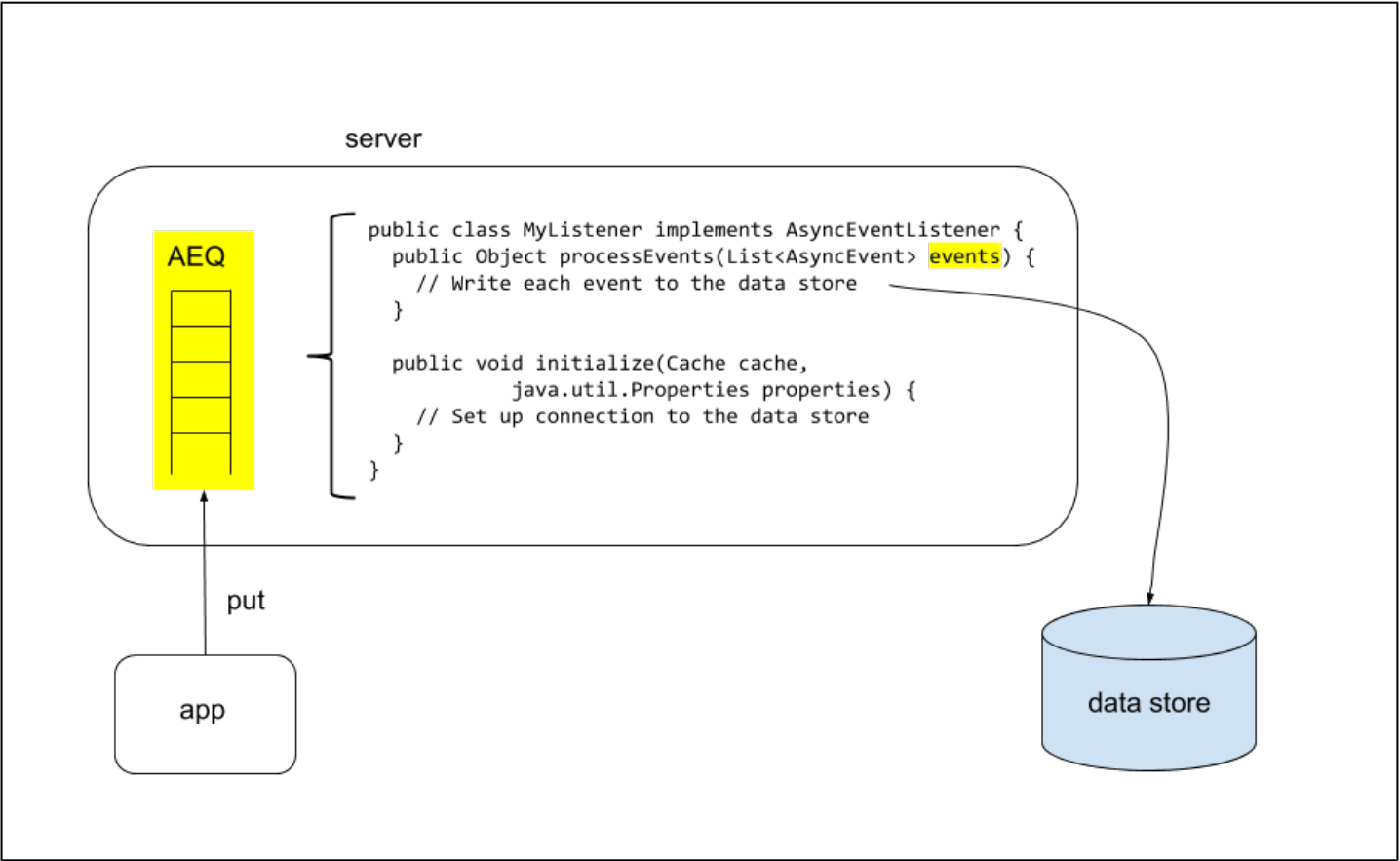
The `CacheLoader.load` method queries the backend data store for the desired entry. That communication between the server process and the backend data store requires a connection, and establishing a connection is likely to use a set of credentials. You provide a custom implementation of the `CacheLoader.initialize` 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 [Pivotal 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.

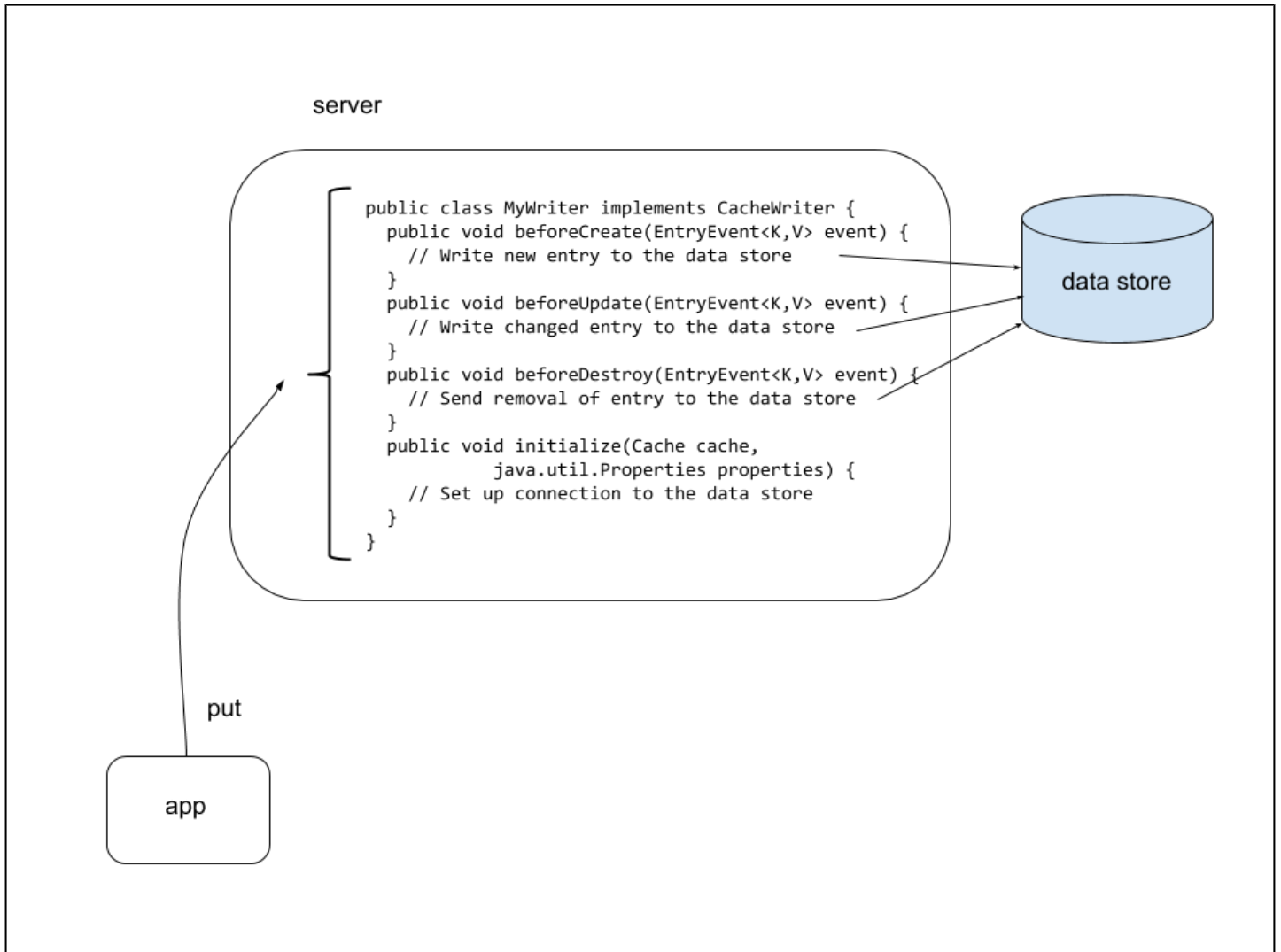
Your configuration will specify the AEQ as persistent, such that it does not lose queued backend-data-store writes across unexpected server restarts.

## 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 [Pivotal 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 `gfs` `create region` command during configuration as described in [Configure Using gfs 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 gfs 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 gfs over HTTPS](#) to connect to the cluster with the cluster-operator credentials from the service key.

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, the AEQ listener, 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'} \
--cache-listener=com.myCompany.MyListener \
--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 the cluster-operator credentials from the service key.
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'}
```



## Deleting 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.

## Updating a Pivotal Cloud Cache Service Instance

### In this topic

[Rebalancing a Cluster](#)

[Restarting a Cluster](#)

[About Changes to the Service Plan](#)

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 authenticated as a cluster operator.



**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 cluster 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}'
```

## About 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.

## 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 PCC 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 PCC Cluster

While the expectation is that configuration and data can be exported from a GemFire cluster and then imported into a PCC 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 PCC cluster. The PCC 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`

 **Note:** The restriction here on the use of the `gfsh import data` does not apply to the procedure for migrating from an existing PCC cluster that does not use TLS for encryption to a PCC cluster that does use TLS for encryption. See [Migrating to a TLS-Enabled Cluster](#) for that procedure.

# Accessing a Service Instance

## In this topic

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

Determine Your TLS Termination

After you have created a service instance, you can start accessing it. Usually, you set up cache regions before using your service instance from a deployed CF app. You can do this with the gfsh command line tool. To connect, you must set up a service key.

## Create Service Keys

Service keys provide 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 `cf service-key` returns output in the following format:

```
{
  "distributed_system_id": "0",
  "locators": [
    "10.244.0.66[55221]",
    "10.244.0.4[55221]",
    "10.244.0.3[55221]"
  ],
  "urls": {
    "gfs": "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": {
    "sender_credentials": {
      "active": {
        "password": "gws-XXX-password",
        "username": "gateway_sender_XXX"
      }
    }
  }
}
```

The service key specifies the user roles and URLs 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.

- The gateway sender writes data that is sent to another cluster. The identifier assigned for this role is of the form `gateway_sender_XXX`, where `XXX` is a unique string generated upon service instance creation and incorporated in this user role's name.
- A URL used to connect the gfsh client to the service instance
- A URL used to view the Pulse dashboard in a web browser, which allows monitoring of the service instance status. Use the developer credentials to authenticate.

## 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](#). The correct version of Pivotal GemFire to download is any patch version of the Pivotal GemFire version listed in the PCC release notes. A link to the PCC release notes is on Pivotal Network in the Release Details for your PCC version. Note that a JDK or JRE will also be required, as specified in the release notes.



**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 that specifies an HTTPS gfsh URL of the form:

```
connect --use-http=true --url=HTTPS-gfsh-URL
--trust-store=TRUSTSTORE-FILE-PATH --trust-store-password=PASSWORD
--user=CLUSTER-OPERATOR-XXX --password=CLUSTER-OPERATOR-PASSWORD
```

The HTTPS-gfsh-URL, the cluster operator user name, and its password are in 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 truststore password you created. If you omit the `--trust-store-password` option from the command line, you will be prompted to enter the password.

## 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](#).  
The correct version of Pivotal GemFire to download is any patch version of the Pivotal GemFire version listed in the PCC release notes. A link to the PCC release notes is on Pivotal Network in the Release Details for your PCC 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. Run `gfsh`, and then issue a `connect` command that specifies an HTTPS URL of the form:

```
connect --use-http=true --use-ssl --skip-ssl-validation=true
--url=<HTTPS-gfsh-URL> --user=<cluster_operator_XXX>
--password=<cluster_operator-password>
```

The cluster operator user name and password are in the service key. See [Create Service Keys](#) for instructions on how to view the service key.



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.

## Using Pivotal Cloud Cache

### In this topic

- Create Regions with gfsh
- Working with Disk Stores
  - Create a Disk Store
  - Destroy a Disk Store
- Java Build Pack Requirements
- Bind an App to a Service Instance
- Use the Pulse Dashboard
- Access Service Instance Metrics
- Access Service Broker Metrics
- Export gfsh Logs
- Deploy an App JAR File to the Servers
- Use the GemFire-Greenplum Connector

## Create Regions with gfsh

After connecting with gfsh as a `cluster_operator_XXX`, 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 PCC service instance using the cluster operator credentials.
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 PCC service instance using the cluster operator credentials.
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](#).

## 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.

The following is a sample `VCAP_SERVICES` environment variable:

```
{
  "p-cloudcache": [
    {
      "credentials": {
        "locators": [
          "10.244.0.4[55221]",
          "10.244.1.2[55221]",
          "10.244.0.130[55221]"
        ],
        "urls": {
          "gfish": "https://cloudcache-1.example.com/gemfire/v1",
          "pulse": "https://cloudcache-1.example.com/pulse"
        },
        "users": [
          {
            "password": "some_developer_password",
            "username": "developer_XXX"
          },
          {
            "password": "some_password",
            "username": "cluster_operator_XXX"
          }
        ]
      },
      "label": "p-cloudcache",
      "name": "test-service",
      "plan": "caching-small",
      "provider": null,
      "syslog_drain_url": null,
      "tags": [],
      "volume_mounts": []
    }
  ]
}
```

## Use the Pulse Dashboard

You can access the Pulse dashboard for a service instance by accessing the pulse-url you **obtained from a service key** in a web browser.

Use either the `cluster_operator_XXX` or `developer_XX` credentials to authenticate.

## 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.)

PCC service instances output metrics to the Loggregator Firehose. You can use the [Firehose plugin](#) to view metrics output on the cf CLI directly or connect the output to any other [Firehose nozzle](#); for example, the nozzle for [Datadog](#).

PCC 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 [Firehose nozzle plugin](#). For more information on broker metrics, see [On Demand Broker Metrics](#).

## Export gfsh Logs

You can get logs and `.gfs` stats files from your PCC 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.
2. Run `export logs`.
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 the cluster operator credentials, 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 the cluster operator role, 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 the cluster operator role to have the necessary permissions to use the connector.



## Developing an App Under TLS

Apps that connect to a TLS-enabled PCC service instance require a truststore containing the Services CA certificate from CredHub, and they must set properties to configure the communication with the Pivotal GemFire components within the PCC service instance.

Have your PCF operator follow this procedure to acquire and provide you with the Services CA certificate:

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=credhub --client-secret=CLIENT-SECRET
```

where `CLIENT-SECRET` is the client secret set in the creation of the UAA Client.

For example:

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

3. Run this command to print the Services CA certificate:

```
$ credhub get --name="/services/tls_ca" -j | jq -r .value.certificate
```

Follow this procedure to set up the truststore:

1. Add the Services CA certificate to an existing truststore or create a new truststore in the app's `src/main/resources` folder. See [Create a Truststore](#) for instructions. The location for this truststore is fixed. If the app uses a truststore located in a different spot than the `src/main/resources` folder, create a new truststore in this required location.

Four GemFire properties must be set to configure communication:

1. Set `ssl-enabled-components` to `all`.
2. Set `ssl-truststore` to the absolute path and file name of the truststore as it will exist within the expanded JAR file of the deployed app.
3. Set `ssl-truststore-password` to the password chosen when the truststore was created.
4. Set `ssl-require-authentication` to `false`, such that there will be one-way authentication of the GemFire component to the app.

For a Spring Data GemFire app that places its truststore in `src/main/resources`, these properties map to

```
spring.data.gemfire.security.ssl.truststore=/home/vcap/app/BOOT-INF/classes/truststore.jks
spring.data.gemfire.security.ssl.truststore.password=TRUSTSTORE-PASSWORD
spring.data.gemfire.security.ssl.require.authentication=false
```

where `TRUSTSTORE-PASSWORD` is the password chosen during truststore creation.

For other apps, the GemFire properties should be

```
ssl-enabled-components=all
ssl-truststore=/home/vcap/app/BOOT-INF/classes/truststore.jks
ssl-truststore-password=TRUSTSTORE-PASSWORD
ssl-require-authentication=false
```

where `TRUSTSTORE-PASSWORD` is the password chosen during truststore creation. 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.



## Connecting a Spring Boot App to Pivotal Cloud Cache with Session State Caching

### In this topic

[Use the Tomcat App](#)

[Use a Spring Session Data GemFire App](#)

[Upgrade PCC and Spring Session Data GemFire](#)

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

In PCC v1.1 and later, 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 this 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>1.5.3.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 v1.3.0 and later versions are compatible with Spring Session Data GemFire v2.0.0.M2 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 = '2.0.0.M3'
    }
    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:2.0.0.M3")
    compile("org.springframework.session:spring-session-data-gemfire:2.0.0.M2")
    compile("io.pivotal.spring.cloud:spring-cloud-gemfire-spring-connector:1.0.0.RELEASE")
    compile("io.pivotal.spring.cloud:spring-cloud-gemfire-cloudfoundry-connector:1.0.0.RELEASE")
}

```

4. Clear the session state region.

5. Start the rebuilt app.





## 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 v2.0.1 release
- Spring Boot v2.0.0+

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](#) .

## 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 Look-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**

## Design Patterns

### In this topic

[The Inline Cache](#)

[The Look-Aside Cache](#)

[Bidirectional Replication Across a WAN](#)

[Blue-Green Disaster Recovery](#)

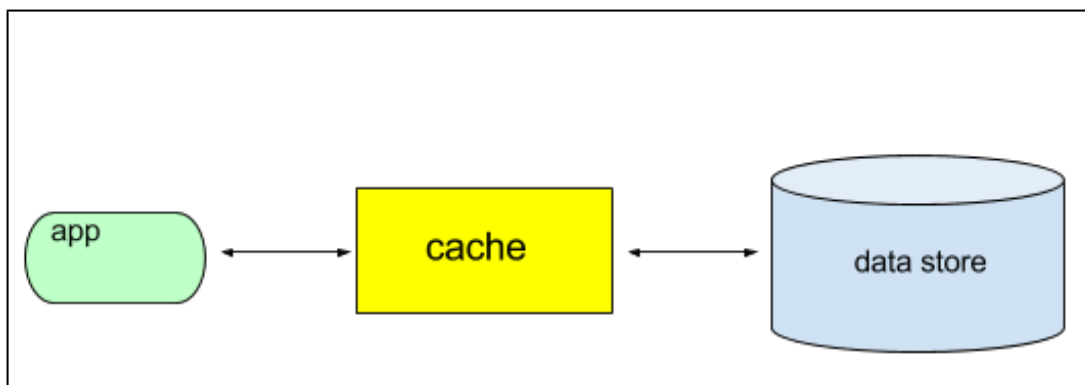
[CQRS Pattern Across a WAN](#)

[Hub-and-Spoke Topology with WAN Replication](#)

[Follow-the-Sun Pattern](#)

## 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

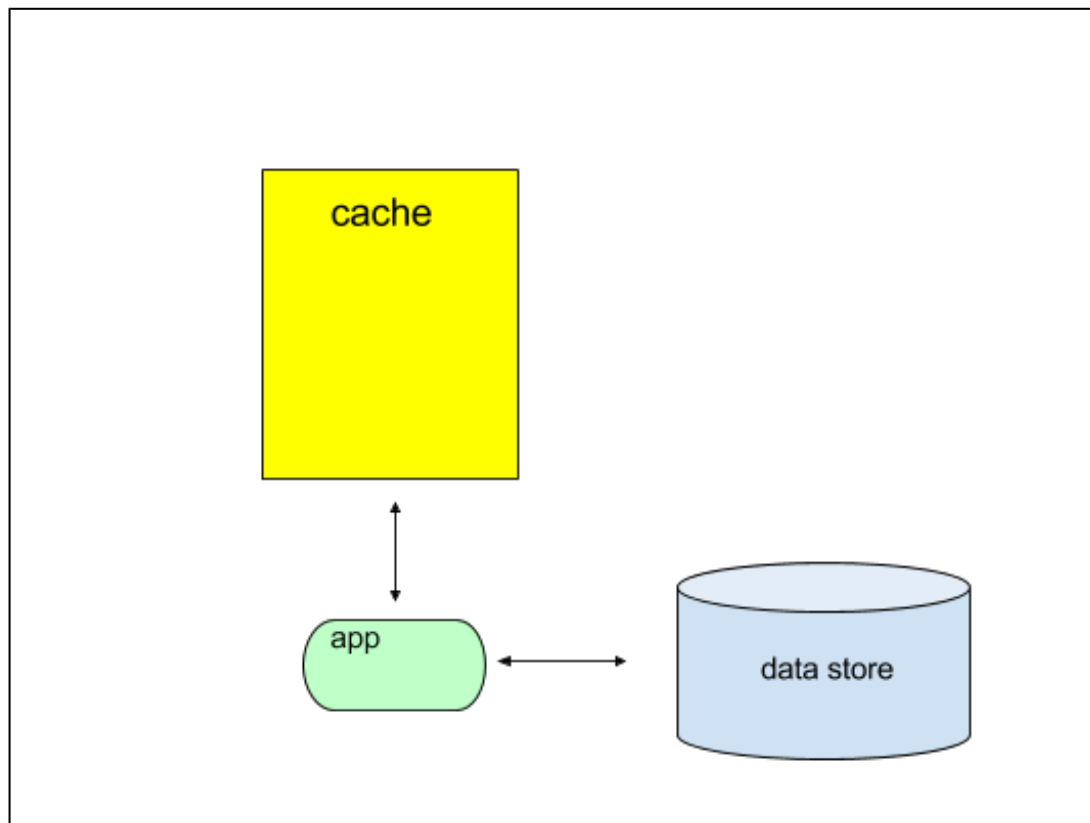
creates or updates the data store in one of the following ways:

- 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 Look-Aside Cache

The look-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 look-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.

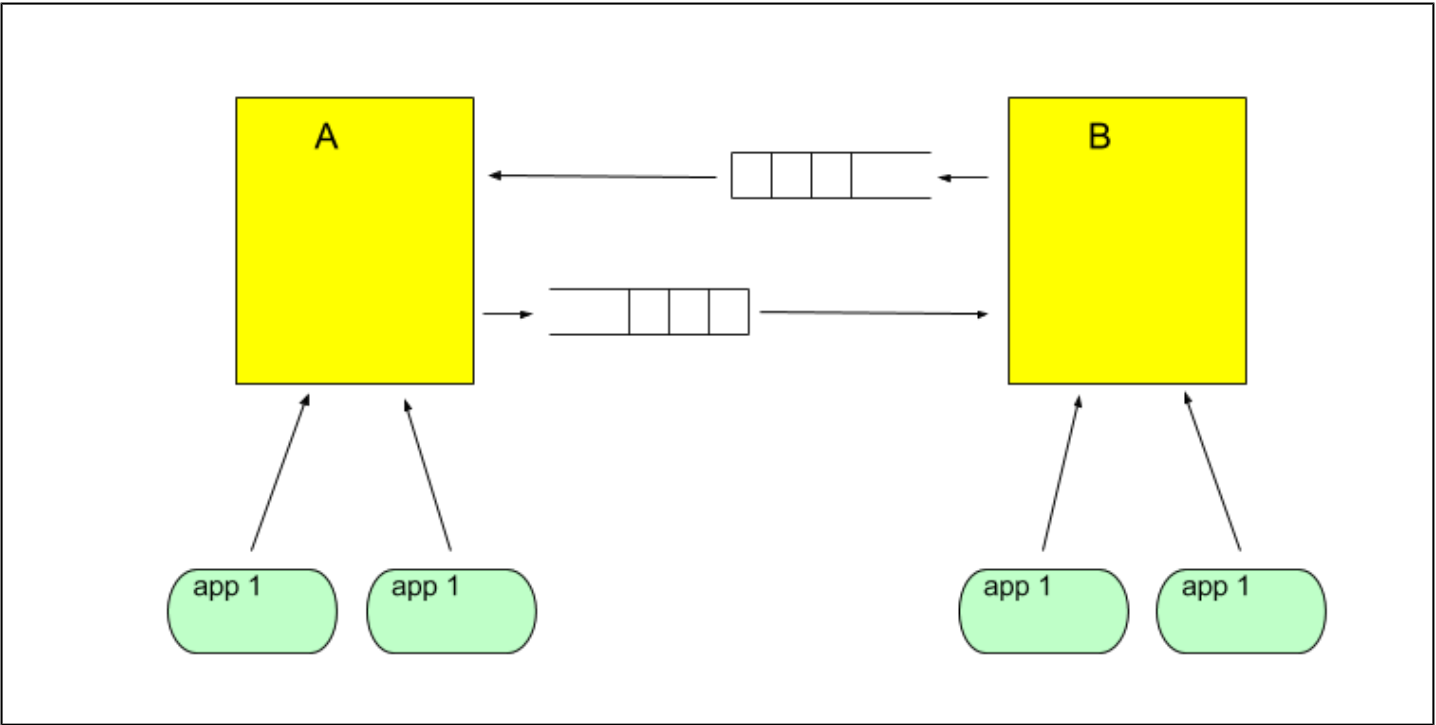
 **Note:** SDG (Spring Data GemFire) supports the look-aside pattern, as detailed at [Configuring Spring's Cache Abstraction](#) .

## 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.

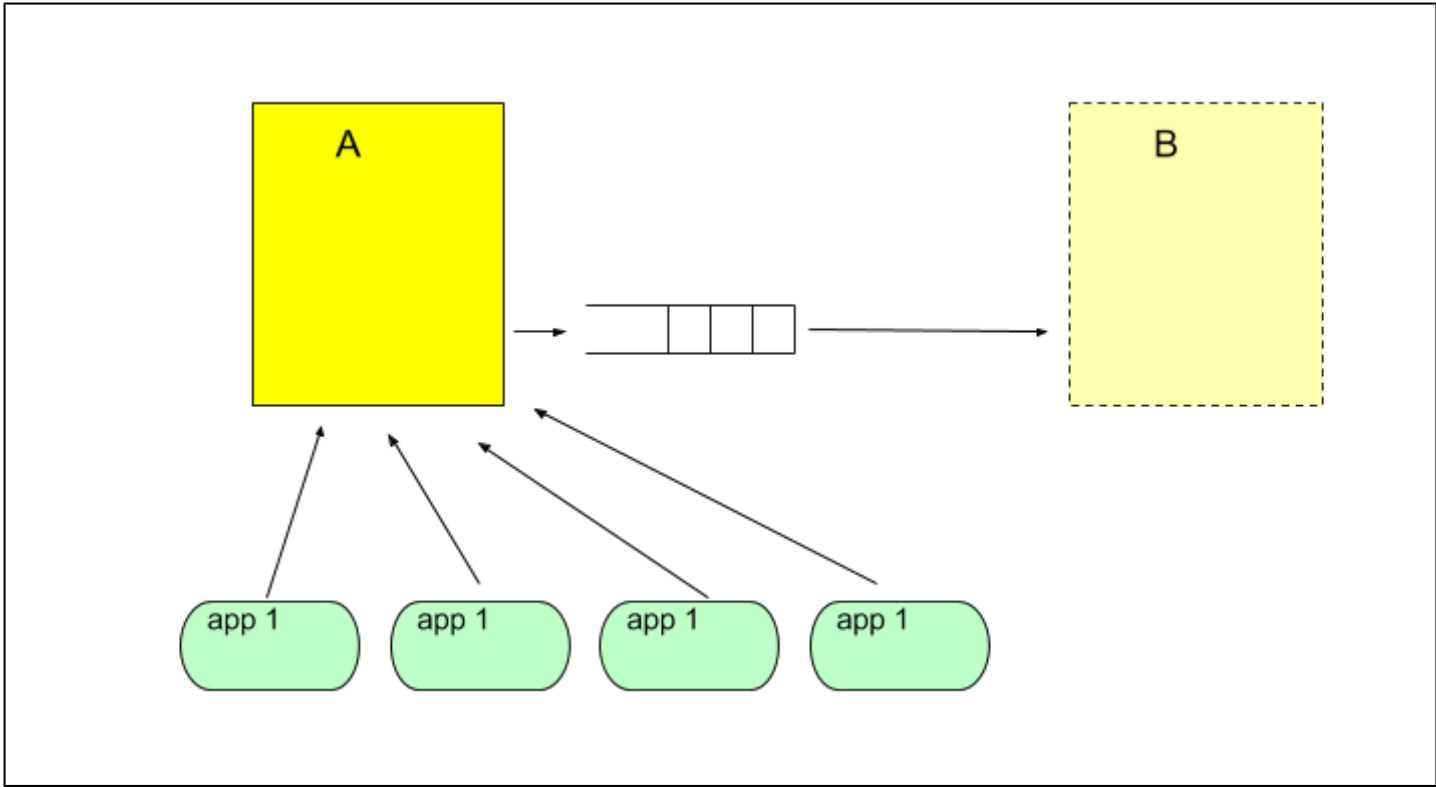
This diagram shows multiple instances of an app interacting with one of the two PCC service instances, cluster A and cluster B. Any change made in cluster A is sent to cluster B, and any change made in cluster B is sent to cluster A.



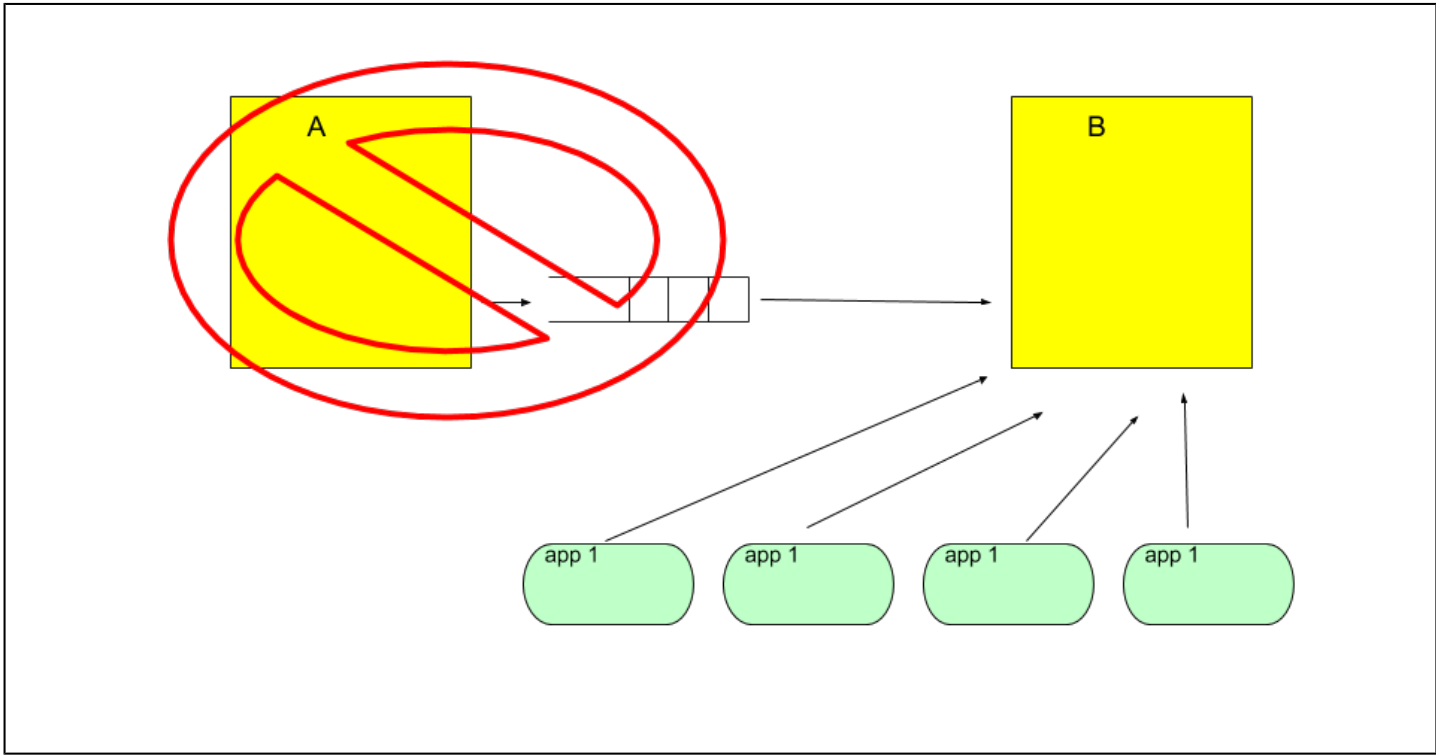
## Blue-Green Disaster Recovery

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. 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, apps connected to the failed site can be redirected by an external load-balancing entity to the replicate, which 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, cluster B takes over.



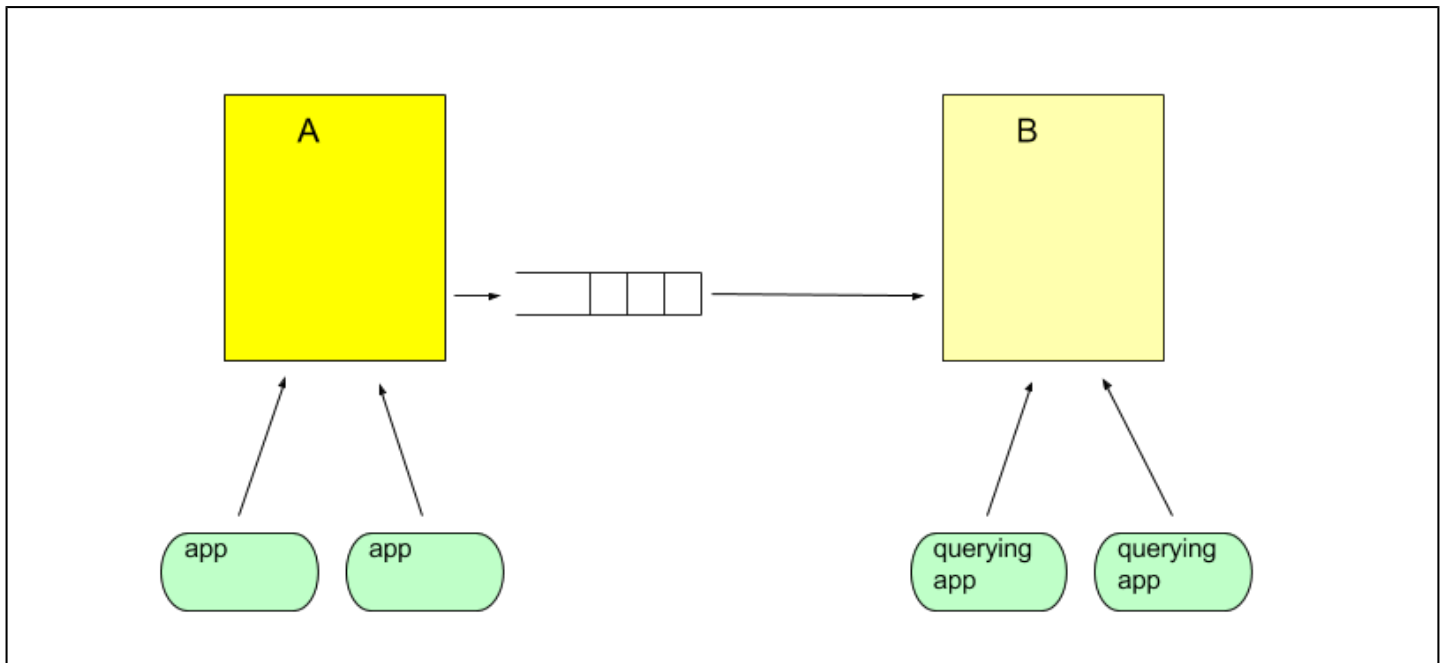
## CQRS Pattern Across a WAN

Two PCC service instances may be connected across a WAN to form a single distributed system that



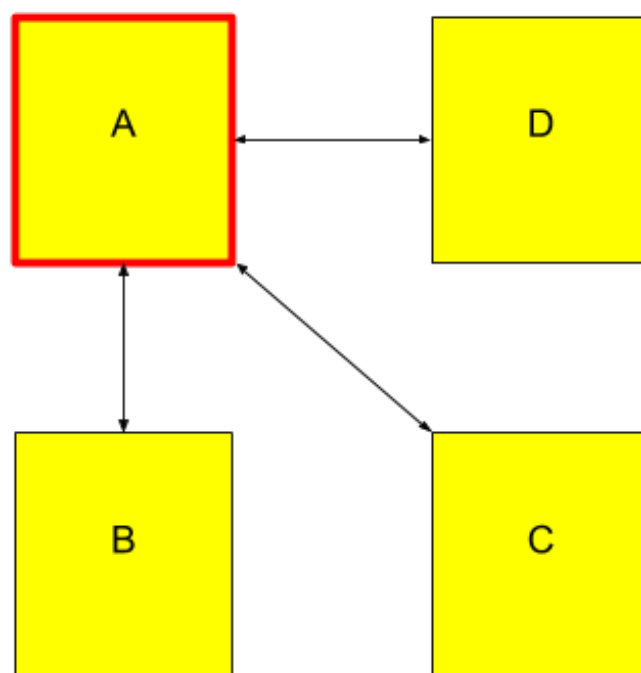
implements a CQRS (Command Query Responsibility Segregation) pattern. Within this pattern, commands are those that change the state, where state is represented by region contents. All region operations that change state are directed to the cluster within one PCC service instance. The changes are propagated 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.



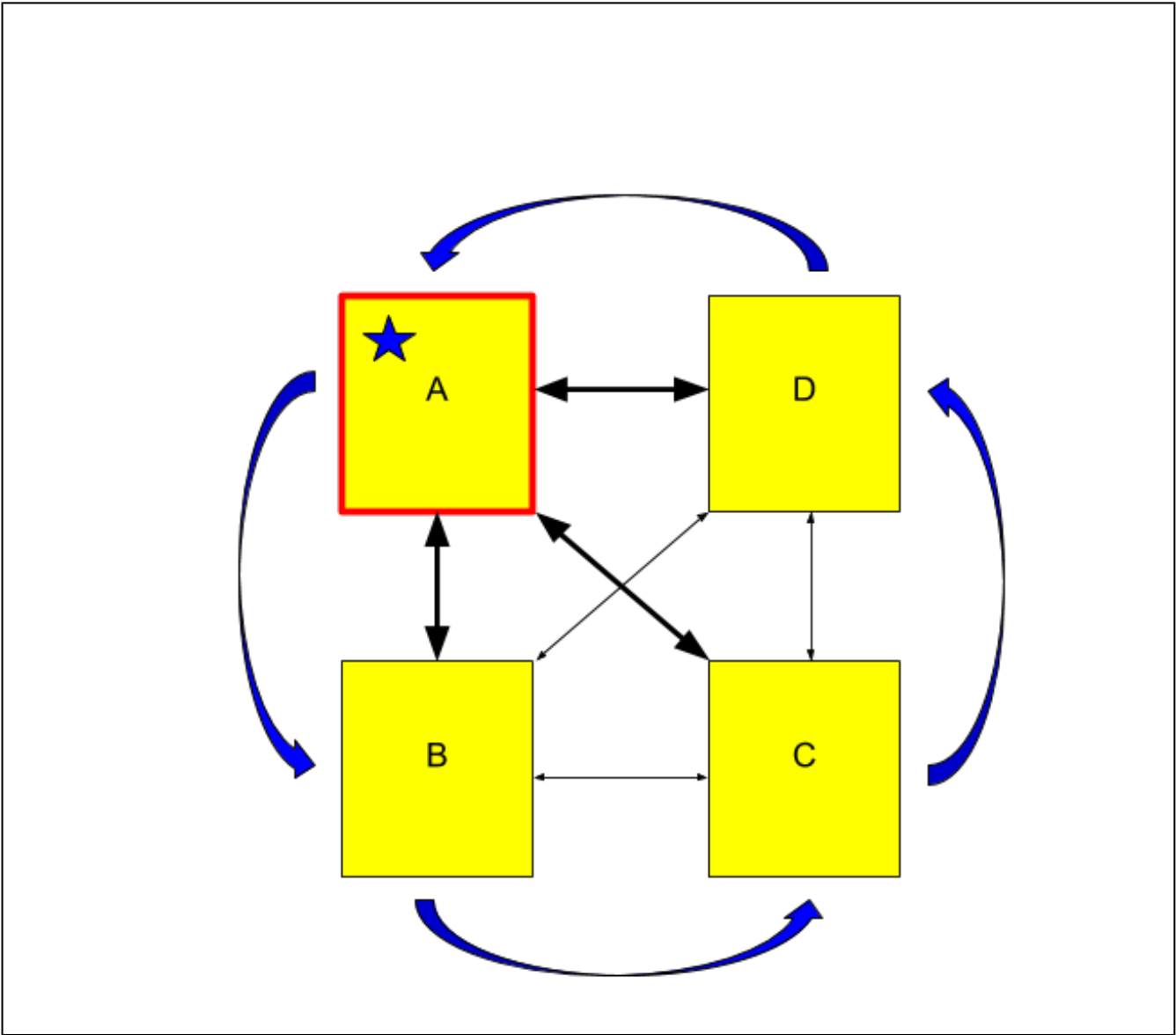
A common implementation that uses this topology directs all app operations that write or update region contents to the hub. Writes and updates are then propagated asynchronously across the WAN from the hub to the spokes.

## Follow-the-Sun Pattern

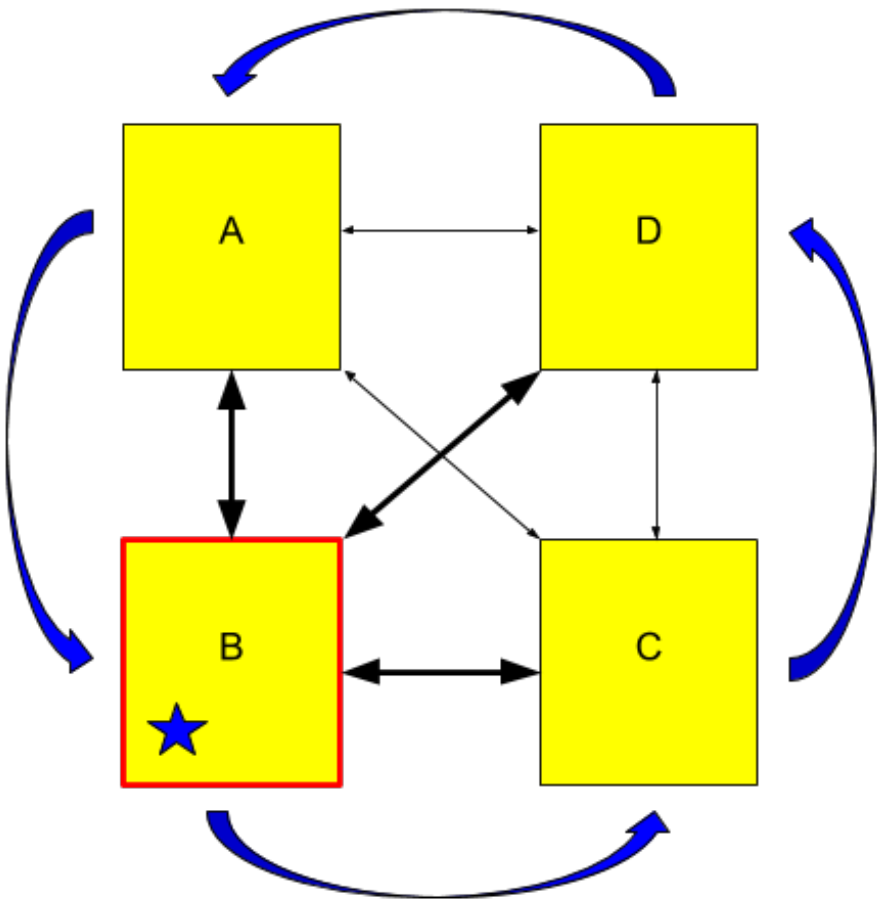
Performance improves when operation requests originate in close proximity to the service instance that handles those requests. Yet many data sets are relevant and used all over the world. If the most active location for write and update operations moves over the course of a day, then a performant design 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.



## Region Design

### In this topic

Keys

Partitioned Regions

Partitioned Region Types for Creating Regions on the Server

Replicated Regions

Replicated Region Types for Creating Regions on the Server

Persistence

Overflow

Regions as Used by the App

An Example to Demonstrate 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.

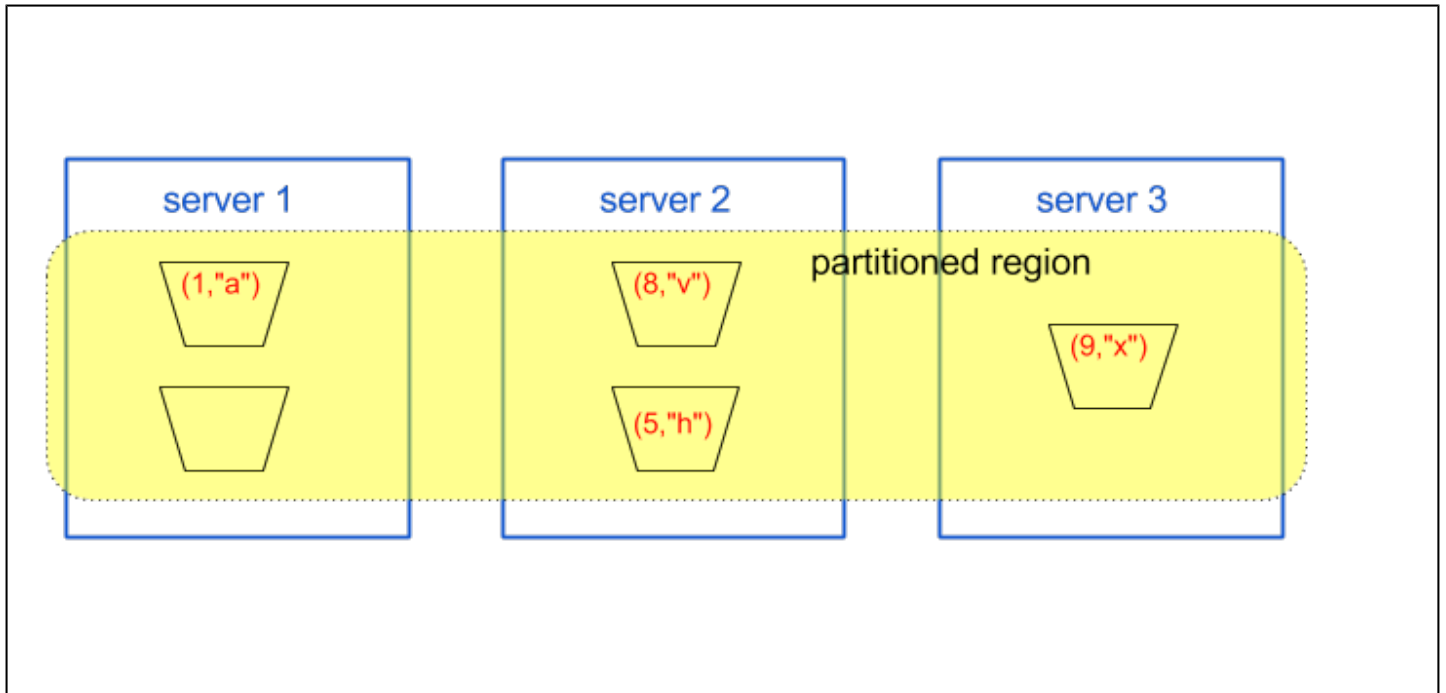
## Keys

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.

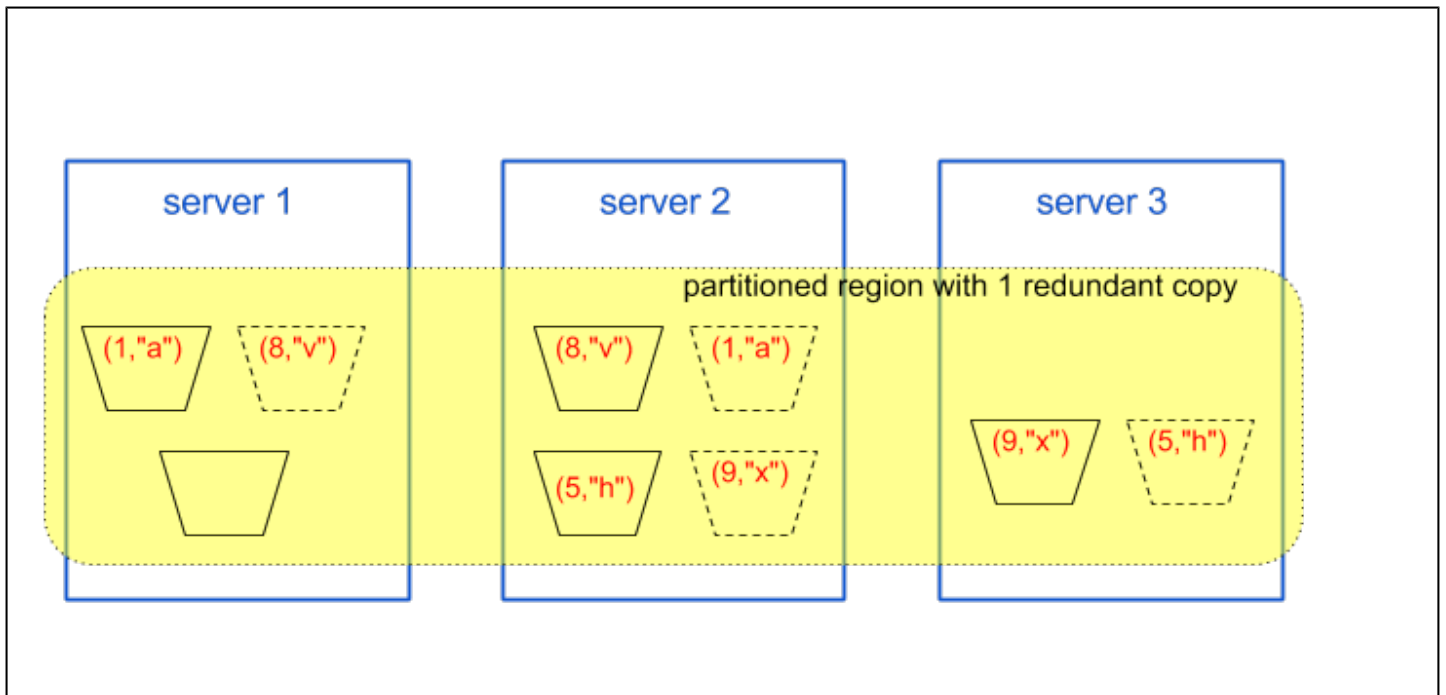
Here is a diagram that shows a single partitioned region (highlighted) with very few entries to illustrate partitioning.



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:

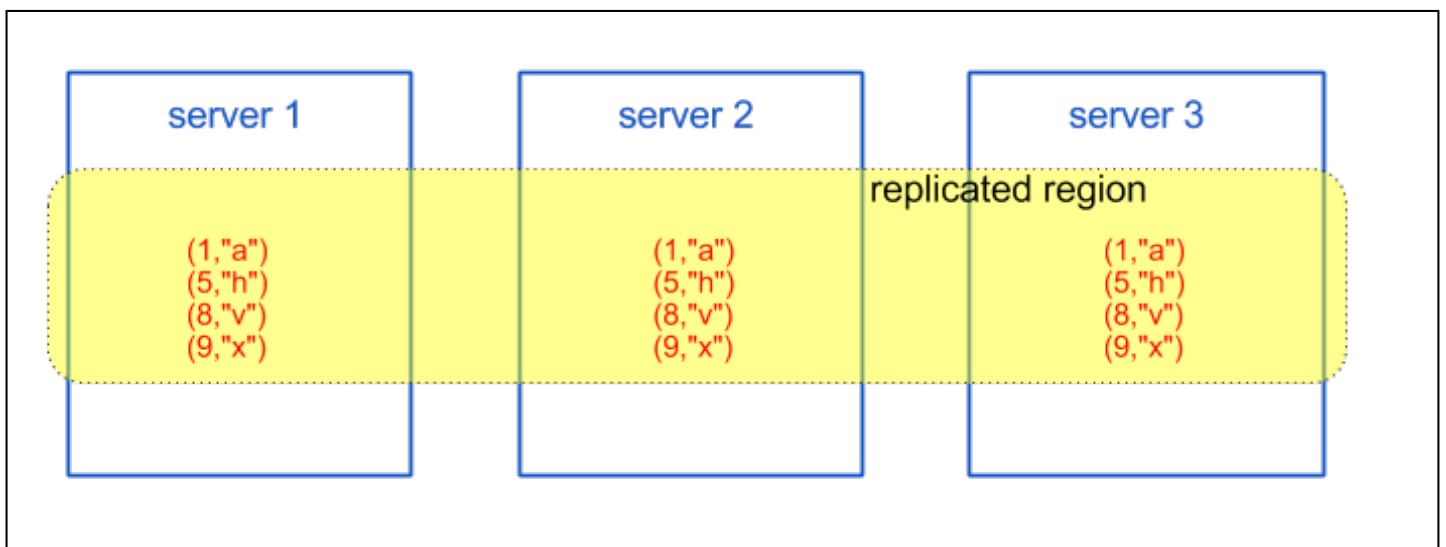
- **PARTITION\_REDUNDANT** Region entries are placed into the buckets that are distributed across all servers hosting the region. In addition, GemFire keeps and maintains a declared number of redundant copies of all entries. The default number of redundant copies is 1.
- **PARTITION\_REDUNDANT\_HEAP\_LRU** 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 65% 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 65% 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 65% 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 suffers 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 65% 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 65% 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 65% 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.

## Handling Events

### In this topic

#### Continuous Queries

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.

## 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](#)

## 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 PCC service instance is running.
- You have Cloud Foundry credentials for accessing the PCC service instance.
- You have a service key for the PCC 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 PCC 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 PCC service instance:
  - Modify the manifest in `manifest.yml` by replacing `service0` with the name of your PCC 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 PCC service instance.

3. Build the app with

```
$ ./gradlew clean build
```

4. In a second shell, run `gfsh`.
5. Use `gfsh` to connect to the PCC service instance as described in [Connect with gfsh over HTTPS](#).
6. Use `gfsh` to create a region named `test` as described in [Create Regions with gfsh](#). 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'
```

## 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 v1.6, use the branch named `release/1.6`.

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.



## Troubleshooting

Here are problems and fixes related to using PCC.

- **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.

## Pivotal Cloud Cache Release Notes

### In this topic

- [v1.6.3](#)
- [v1.6.2](#)
- [v1.6.1](#)
- [v1.6.0](#)
- [Known Issues](#)

### v1.6.3

**Release Date:** May 8, 2019



Features included in this release:

- PCC 1.6.3 uses [Pivotal GemFire 9.6.2](#) .

### v1.6.2

**Release Date:** April 10, 2019

Features included in this release:

-  **Breaking Change:** This patch release increases system security by requiring TLS encryption for using gfsh and Pulse. Follow the steps within [Preparing for TLS](#) prior to installing the PCC tile.
-  **warning:** Follow the procedure detailed in [Special Upgrade Procedure for Dev and Small Footprint Plans](#) when upgrading to PCC v1.6.2 for plans that colocate system components on a single VM. This applies to the dev plan and to the small footprint plan.
- If a ClamAV or File Integrity Monitor is detected, available memory for GemFire servers is reduced to allow enough memory for these PCF add-ons. This prevents a failure during PCC service instance creation.

- Multiple PCC service instances are now spread across availability zones.
- Increased the operating system limit for the number of files that can be open concurrently.
- PCC uses Pivotal GemFire 9.6.1.

## v1.6.1

**Release Date:** January 9, 2019

Features included in this release:

- PCC now ships with OpenJDK 1.8\_192 instead of the equivalent Oracle JDK.
- PCC uses Pivotal GemFire 9.6.
- PCC supports Pivotal Application Service (PAS) 2.4.

## v1.6.0

**Release Date:** December 19, 2018

Features included in this release:

- The new Small Footprint Plan, which uses fewer VMs than other plans, is available for use in production.
- PCC now ships with JDK 1.8\_191.
- PCC now runs Pivotal GemFire 9.6.
- PCC now supports Pivotal Application Service (PAS) 2.4.

## Known Issues

- Installations using HTTP session state replication have a known issue and workaround to correct the issue. The HTTP session module creates its region that holds metadata on only one server within a cluster. The region needs to be hosted on all the servers.  
To correct the issue on a running cluster, connect to the cluster using the GemFire cluster operator credentials, and run a single `gfsh` command to create the metadata region on all servers. The command has the form:

```
create region --name=REGION-NAME --type=REGION-SHORTCUT \  
--enable-statistics \  
--entry-idle-time-custom-expiry=org.apache.geode.modules.util.SessionCustomExpiry
```

If the metadata region's name or type have not been changed from their default, use this `gfsh` command:

```
gfsh>create region --name=gemfire_modules_sessions --type=PARTITION_REDUNDANT \  
--enable-statistics \  
--entry-idle-time-custom-expiry=org.apache.geode.modules.util.SessionCustomExpiry
```

For installations that have changed the metadata region's name or type, substitute the changed values for `REGION-NAME` and `REGION-SHORTCUT` in the command.

You can verify that the region is hosted on all servers with the `gfsh` command:

```
gfsh>describe region --name=gemfire_modules_sessions
```

- Current versions of the Cloud Foundry Command Line Interface (CLI) tool have a known bug that omits the documentation URL when using the `cf service` command.