

Pivotal Cloud Cache 1.3


Rev: v1.3.5

© Copyright Pivotal Software Inc, 2013-2019

Table of Contents

Table of Contents	2
Pivotal Cloud Cache	3
Pivotal Cloud Cache Operator Guide	9
Pivotal Cloud Cache Developer Guide	23
Application Development	44
Pivotal Cloud Cache Release Notes	51

Pivotal Cloud Cache

 **Note:** The support period for Pivotal Cloud Cache (PCC) v1.3.5 has expired, and this version no longer is supported. To stay up to date with the latest software and security updates, upgrade to a supported version.

Overview

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	v1.3.5
Release date	February 11, 2019
Software component version	Pivotal GemFire v9.3.3 (Limited-availability v1.3.2 runs Pivotal GemFire v9.2.0)
Compatible Ops Manager version(s)	v1.12.x and v2.0.x (For limited-availability v1.3.2: v1.12.x, v2.0.x, v2.1.x, and v2.2.x)
Compatible Elastic Runtime version(s)	v1.12.x (For limited-availability v1.3.2: v1.12.x)
Compatible Pivotal Application Service (PAS)* version(s)	v2.0.x (For limited-availability v1.3.2: v2.0.x, v2.1.x, and v2.2.x)
IaaS support	AWS, Azure, GCP, OpenStack, and vSphere
IPsec support	Yes
Required BOSH stemcell version	3586 (Limited-availability v1.3.2 requires 3468)
Minimum Java buildpack version required for apps	v3.13

* As of PCF v2.0, *Elastic Runtime* is renamed *Pivotal Application Service (PAS)*.

PCC and Other PCF Services

Some PCF services offer *on-demand* service plans. These plans let developers provision service instances when they want.

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

The following PCF services offer on-demand service plans:

- MySQL for PCF v2.0 and later
- RabbitMQ for PCF
- Redis for PCF
- Pivotal Cloud Cache (PCC)

These services package and deliver their on-demand service offerings differently. For example, some services, like Redis for PCF, have one tile, and you configure the tile differently depending on whether you want on-demand service plans or pre-provisioned service plans.

For other services, like PCC and MySQL for PCF, only on-demand service plans are available.

The following table lists and contrasts the different ways that PCF services package on-demand and pre-provisioned service offerings.

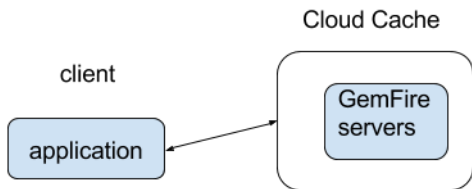
PCF service tile	Standalone product related to the service	Versions supporting on demand	Versions supporting pre-provisioned
RabbitMQ for PCF	Pivotal RabbitMQ	v1.8 and later	All versions
Redis for PCF	Redis	v1.8 and later	All versions
MySQL for PCF	MySQL	v2.x	NA
PCC	Pivotal GemFire	All versions	NA

PCC Architecture

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 `gfs`, facilitates region administration. Use `gfs` 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
    subgraph Client
        direction TB
        L1[Locator1]
        L2[Locator2]
        L3[Locator3]
    end
    subgraph Servers
        direction TB
        S1[Server1]
        S2[Server2]
        S3[Server3]
        S4[Server4]
    end
    Client --> Servers
```

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

```
graph TD
    subgraph Client
        direction TB
        L1[Locator1]
        L2[Locator2]
        L3[Locator3]
    end
    subgraph Servers
        direction TB
        S1[Server1]
        S2[Server2]
        S3[Server3]
        S4[Server4]
        S5[Server5]
        S6[Server6]
        S7[Server7]
    end
    Client --> Servers
```

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 [PCF Admin Actions]
        s1[s1]
        s2[s2]
    end
    subgraph Developer_Actions [Developer Actions]
        s4[s4]
    end
    s1-->s2
    s2-->s3
    s3-->s4
    s4-->s5
    s5-->s6
```

1. Upload P-CloudCache.pivotal to Ops Manager
 2. Configure CloudCache Service Plans, i.e. caching-small
 3. Ops Manager deploys CloudCache Service Broker
 4. Developer calls `cf create-service p-cloudcache caching-small test`
 5. Ops Manager creates a CloudCache cluster following the caching-small specifications
 6. Developer calls `cf push`

Networking for On-Demand Services

This section describes networking considerations for Pivotal Cloud Cache.

Service Network Requirement

When you deploy PCF, you must create a statically defined network to host the component virtual machines that constitute the PCF infrastructure.

PCF components, like the Cloud Controller and UAA, run on this infrastructure network. In PCF v2.0 and earlier, on-demand PCF services require that you host them on a network that runs separately from this network.

Cloud operators pre-provision service instances from Ops Manager. Then, for each service, Ops Manager allocates and recovers static IP addresses from a pre-defined block of addresses.

To enable on-demand services in PCF v2.0 and earlier, operators must create a service networks in BOSH Director and select the **Service Network** checkbox. Operators then can select the service network to host on-demand service instances when they configure the tile for that service.

Default Network and Service Network

On-demand PCF services rely on the BOSH 2.0 ability to dynamically deploy VMs in a dedicated network. The on-demand service broker uses this capability to create single-tenant service instances in a dedicated service network.

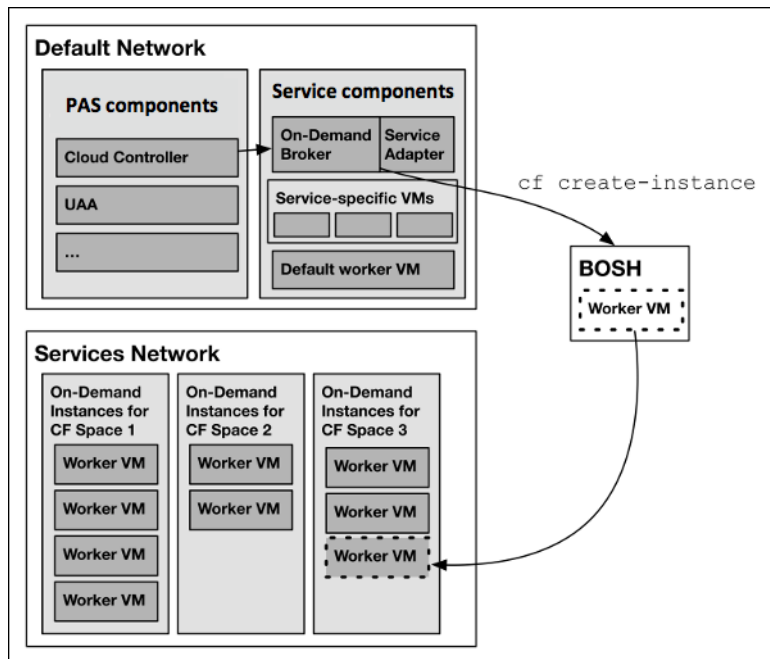
On-demand services use the dynamically-provisioned service network to host the single-tenant worker VMs that run as service instances within development spaces. This architecture lets developers provision IaaS resources for their service instances at creation time, rather than the operator pre-provisioning a fixed quantity of IaaS resources when they deploy the service broker.

By making services single-tenant, where each instance runs on a dedicated VM rather than sharing VMs with unrelated processes, on-demand services eliminate the “noisy neighbor” problem when one app hogs resources on a shared cluster. Single-tenant services can also support regulatory compliance where sensitive data must be compartmentalized across separate machines.

An on-demand service splits its operations between the default network and the service network. Shared components of the service, such as executive controllers and databases, run centrally on the default network along with the Cloud Controller, UAA, and other PCF components. The worker pool

deployed to specific spaces runs 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.



Required Networking Rules for On-Demand Services

Before deploying a service tile that uses the on-demand service broker (ODB), request the needed network connections to allow components of Pivotal Cloud Foundry (PCF) to communicate with ODB.

The specifics of how to open those connections varies for each IaaS.

See the following table for key components and their responsibilities in an on-demand architecture.

Key Components	Their Responsibilities
BOSH Director	Creates and updates service instances as instructed by ODB.
BOSH Agent	Includes an agent on every VM that it deploys. The agent listens for instructions from the BOSH Director and carries out those instructions. The agent receives job specifications from the BOSH Director and uses them to assign a role, or job, to the VM.
BOSH UAA	Issues OAuth2 tokens for clients to use when they act on behalf of BOSH users.
PAS	Contains the apps that are consuming services
ODB	Instructs BOSH to create and update services, and connects to services to create bindings.
Deployed service instance	Runs the given data service. For example, the deployed Redis for PCF service instance runs the Redis for PCF data 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
ODB	<ul style="list-style-type: none"> BOSH Director BOSH UAA 	<ul style="list-style-type: none"> 25555 8443 	One-way	The default ports are not configurable.
	Deployed	Specific to the service (such as		

ODB	service instances	RabbitMQ for PCF). May be one or more ports.	One-way	This connection is for 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"> PAS (or Elastic Runtime) ODB Deployed Service Instances 	<ul style="list-style-type: none"> 8443 8080 Specific to the service. May be one or more ports. 	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.
- Any `gfs` command not explained in the PCC documentation is **not supported**.
- PCC supports basic OQL queries, with no support for joins.

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`, `CLUSTER:MANAGE:DEPLOY`, `CLUSTER:MANAGE:GATEWAY`, `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](#).

Known Issues

Pulse Issue

The topology diagram might not be accurate and might show more members than are actually in the cluster. However, the numerical value displayed on the top bar is accurate.

Feedback

Please provide any bugs, feature requests, or questions to the [Pivotal Cloud Foundry Feedback list](#).

Pivotal Cloud Cache Operator Guide

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

Requirements for Pivotal Cloud Cache

Service Network

You must have access to a Service Network in order to install PCC.

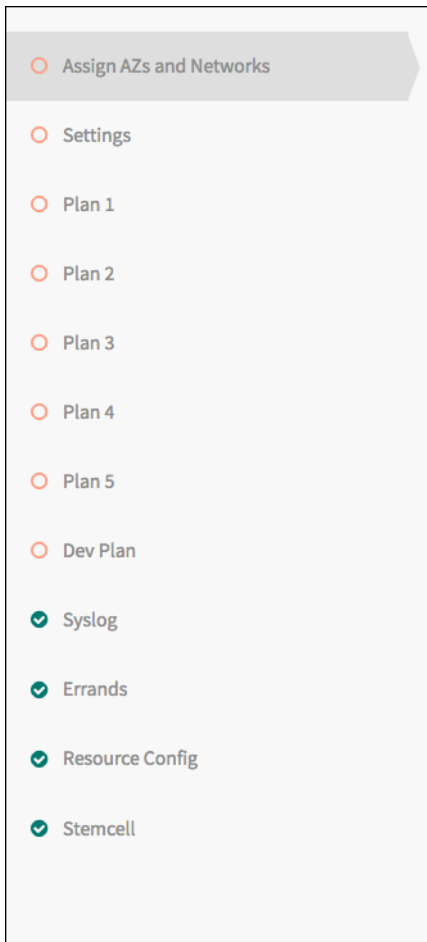
Installing and Configuring Pivotal Cloud Cache

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

1. Download the tile from 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 and 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.



After you complete a section, 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
- [Syslog](#)
- [Errands](#)
- [Resource Config](#)
- [Stemcell](#)

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**.

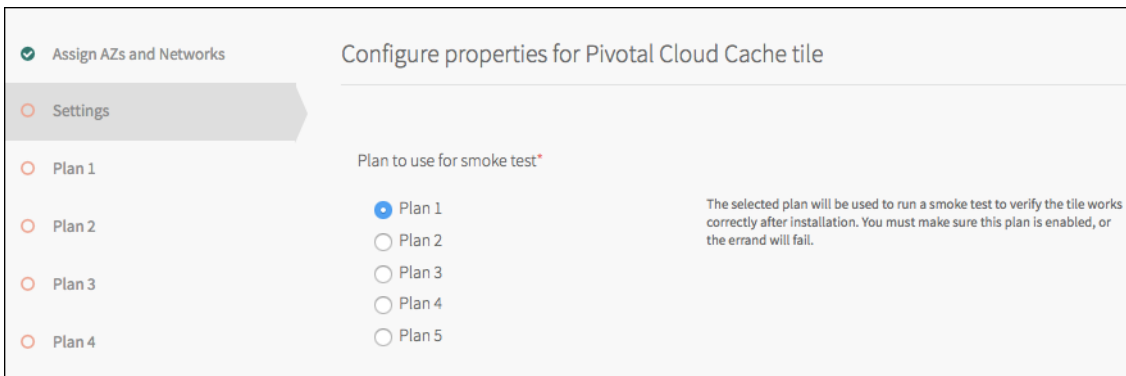
Settings: Smoke Tests

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:

1. Click **Settings**.
2. 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.



3. Click **Save**.

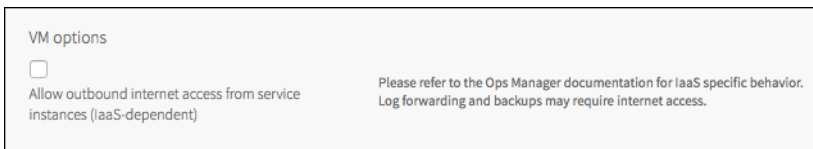
Settings: Allow Outbound Internet Access

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:

1. Click **Settings**.
2. Select **Allow outbound internet access from service instances (IaaS-dependent)**.



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.

3. Click **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:

External Syslog Host

logs.example.com

External Syslog Port

12345

1. Click **Syslog**.
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	<p>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.</p> <p>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.</p>

3. Click **Save**.

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 *

Plan Description

☐ Enable metrics for service instances

CF Service Access*

Enable Service Access

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-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 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 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

If you have enabled post-deploy scripts in your Ops Manager Director, a region is automatically created. To confirm that post-deploy scripts are enabled, navigate to the **Director Config** pane of Ops Manager Director and verify that **Enable Post Deploy Scripts** is selected.

Installation Dashboard

Ops Manager Director

Settings Status Credentials

- Google Config
- Director Config**
- Create Availability Zones
- Create Networks
- Assign AZs and Networks
- Security
- Syslog
- Resource Config

Director Config

NTP Servers (comma delimited)*
169.254.169.254

JMX Provider IP Address

Bosh HM Forwarder IP Address

☐ Enable VM Resurrector Plugin

☒ **Enable Post Deploy Scripts**

☐ Recreate all VMs

This will force BOSH to recreate all VMs on the next deploy. Persistent disk will be preserved

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**.

Stemcell

Ensure you import the correct type of stemcell indicated on this tab.

You can download the latest available stemcells from the [Pivotal Network](#).

Monitoring Pivotal Cloud Cache Service Instances

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

The table below shows the metrics emitted through the CF Nozzle and their corresponding Key Performance Indicator (KPI) guidelines.

Metric Name	Description	Metric Type	Suggested Measurement	Measurement Type	Warning Threshold	Critical Threshold	Suggested Actions	Why a KPI?
	RAM being consumed	percent	Average over last 10	avg	75%	85%		

Metric Name	Description	Metric Type	Suggested Measurement	Measurement Type	Warning Threshold	Critical Threshold	Suggested Actions	Why a KPI?
<code>member.MemoryPercentage</code>	Returns the number of members in the distributed system.	number	Every second	count	< manifest member count		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.	Member loss due to any reason can cause potential data loss.
<code>serviceins.tance.TotalHeapSize</code>	Returns the total available heap (in megabytes) across all distributed members.	number	Every second	pulse				If the total heap size and used heap size are too close, the system might see thrashing due to GC activity. This increases latency.
<code>serviceins.tance.UsedHeapSize</code>	Returns the total heap used on all members.	number	Every second	pulse				If the total heap size and used heap size are too close, the system might see thrashing due to GC activity. This increases latency.
<code>member.GarbageCollectionCount</code>	Returns the number of times garbage collection has occurred.	number	Sum over 10 minutes	sum	Dependent on IaaS and app use case.	Dependent on IaaS and app use case.	Check the number of queries run against the system, which increases the deserialization of objects and increases garbage.	If the frequency of GC is high, the system might see high CPU usage, which causes delays in the cluster.
<code>member.HostCpuUsage</code> (process CPU usage)	Amount of CPU utilization	percent	Average over 10 minutes	avg	85%	95%	If this is not happening with high GC activity, the system is reaching its limits. Horizontal scaling might help.	High CPU usage causes delayed responses and can also make the member non-responsive. This eventually causes it to be kicked out of the cluster, potentially leading to data loss.
<code>member.GetAvgLatency</code>	Returns the cache get average latency.	number	Average over 10 minutes	avg	Dependent on IaaS and app use case.	Dependent on IaaS and app use case.	If this is not happening with high GC activity, the system is reaching its limit. Horizontal scaling might help.	A good indicator of the overall responsiveness of the system. If this number is high, the service administrator should diagnose the root cause.
	Returns the cache put		Average over		Dependent on IaaS and	Dependent on IaaS	If this is not happening with high GC activity, the system	A good indicator of the overall responsiveness of the system. If

Metric Name	Description	Member Type	Suggested Measurement	Measurement Type	Warning Threshold	Critical Threshold	Suggested Actions	Why a KPI?
memberAvgLatency	average latency.						is reaching its limit. Horizontal scaling might help.	this number is high, the service administrator should diagnose the root cause.
member.JVM Pauses	Returns the number JVM pauses	number	Sum over 2 seconds	sum	Dependent on IaaS and app use case.	Dependent on IaaS and app use case.	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.	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.
member.FileDescriptorLimit	Returns the maximum number of open file descriptors allowed for the member's host operating system.	number	Every second	pulse				If total FD open is higher than total FD available, it causes the member to stop responding and crash.
member.TotalFileDescriptorOpen	Returns the current number of open file descriptors.	number	Every second	pulse				If total FD open is higher than total FD available, it causes the member to stop responding and crash.
serviceInstance.UnusedHeapSizePercentage	Returns the proportion of available heap size as a percentage	percent	Every second	compound metric	40%	10%	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, system might see thrashing due to GC activity. This increases the latency
member.FileDescriptorRemaining	Returns the number of available File Descriptors	number	Every second	compound metric	1000	100	You need to scale horizontally to increase capacity	If total FD open is higher than total FD available, it causes the member to stop responding and crash.

This table shows the metrics emitted through the CF Nozzle for gateway senders and gateway receivers.

Metric Name	Description	Metric Type	Measurement Type
gatewaySender.<sender-id>.EventQueueSize	The current size of the gateway sender queue.	number	count
gatewaySender.<sender-id>.EventsReceivedRate	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.	number	count
gatewaySender.<sender-id>.EventsQueuedRate	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.	number	count
	A count of the events received from the gateway sender which will be applied to the region on the		

Metric Name	Description	Metric Type	Measurement Type
gatewayReceiverEventsReceivedRate	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.	number	count

This table shows the metrics emitted through the CF Nozzle for disks.

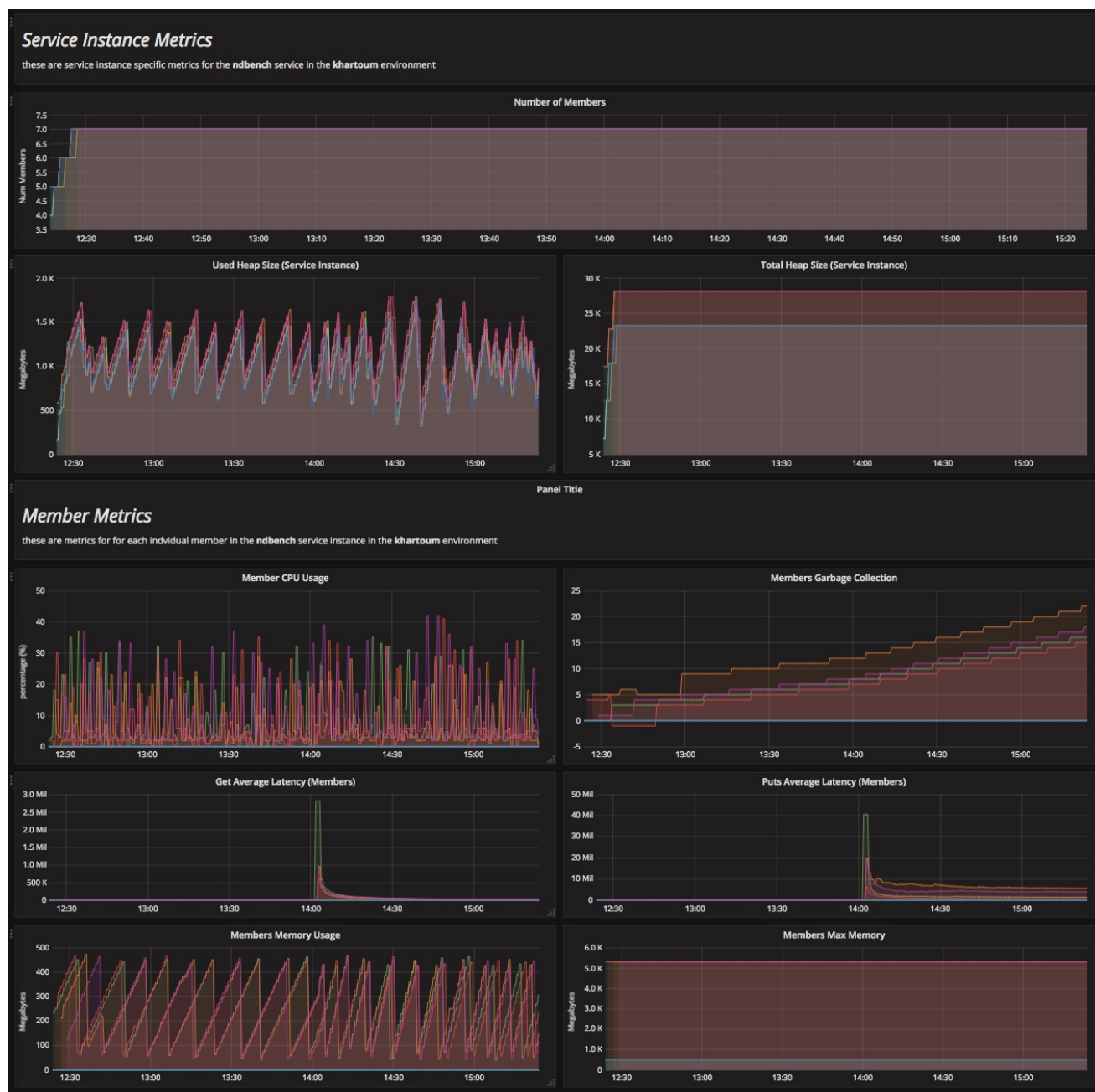
Metric Name	Description	Metric Type	Measurement Type
diskstore.DiskWritesAvgLatency	The average latency of disk writes in nanoseconds.	number	time in nanoseconds
diskstore.TotalSpace	The total number of bytes on the attached disk.	number	count
diskstore.UseableSpace	The total number of bytes of available space on the attached disk.	number	count

Monitoring PCC Service Instances with Prometheus

Prometheus is one of various tools you can use to monitor services instances. It is a monitoring and alerting toolkit that allows for metric scraping. You can use the [Firehose exporter](#) to export all the metrics from the Firehose, which you can then graph with [Grafana](#) to monitor your PCC cluster.

Follow the instructions [here](#) to deploy Prometheus alongside your PCF cluster.

Prometheus can be deployed on any IaaS. You need to verify that the Firehose exporter job can talk to your UAA VM. This might involve opening up firewall rules or enabling your VM to allow outgoing traffic.



You can run queries on, and build a custom dashboard of, specific metrics that are important to you.

Upgrading Pivotal Cloud Cache

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 review your configuration options.
5. Click **Apply Changes**.

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 **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.

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 **Apply Changes**.

Troubleshooting

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 `gfsh` 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.

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.

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

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:

- `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.

The following example creates the service with five service instances in the cluster:

```
$ cf create-service p-cloudeache small my-cloudeache -e '{"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-cloudeache 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:

- 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.

- Create a region named `ClusteredSpringSessions` in gfs 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-cloudeache dev-plan my-dev-cloudeache
```

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` to Cluster A and `distributed_system_id = 2` to Cluster B. GemFire gateway senders provide the communication path and

construct that propagates region operations from one cluster to another. On the receiving end are GemFire gateway receivers. Creating a service instance also creates gateway receivers.

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": {
    "gfs": "https://cloudcache-1.example.com/gemfire/v1",
    "pulse": "https://cloudcache-1.example.com/pulse"
  },
  "users": [
    {
      "password": "cl-op-ABCDE-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABCDE"
    },
    {
      "password": "dev-FGHIJ-password",
      "roles": [
        "developer"
      ],
      "username": "developer_FGHIJ"
    }
  ],
  "wan": {
    "sender_credentials": {
      "active": {
        "password": "gws-KLMNO-password",
        "username": "gateway_sender_KLMNO"
      }
    }
  }
}
```

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 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_KLMNO",
          "password":"gws-KLMNO-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-PQRST-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_PQRST"
    },
    {
      "password": "dev-UVWXY-password",
      "roles": [
        "developer"
      ],
      "username": "developer_UVWXY"
    }
  ],
  "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_KLMNO"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-ZABCD-password",
        "username": "gateway_sender_ZABCD"
      }
    }
  }
}
```

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_ZABCD",
          "password":"gws-ZABCD-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-ABCDE-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABCDE"
    },
    {
      "password": "dev-FGHIJ-password",
      "roles": [
        "developer"
      ],
      "username": "developer_FGHIJ"
    }
  ],
  "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_ZABCD"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-KLMNO-password",
        "username": "gateway_sender_KLMNO"
      }
    }
  }
}
```

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_ABCDE --password=cl-op-ABCDE-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:

```
gfsb>connect --url=https://cloudcache-2.example.com/gemfire/v1 --use-http --user=cluster_operator_PQRST --password=cl-op-PQRST-password
```

- Create the cluster B gateway sender:

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

- Create the cluster B region:

```
gfsb>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 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-ABCDE-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABCDE"
    },
    {
      "password": "dev-FGHIJ-password",
      "roles": [
        "developer"
      ],
      "username": "developer_FGHIJ"
    }
  ],
  "wan": {
    "sender_credentials": {
      "active": {
        "password": "gws-KLMNO-password",
        "username": "gateway_sender_KLMNO"
      }
    }
  }
}
```

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 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-cloudcache 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_KLMNO",
          "password":"gws-KLMNO-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-PQRST-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_PQRST"
    },
    {
      "password": "dev-UVWXY-password",
      "roles": [
        "developer"
      ],
      "username": "developer_UVWXY"
    }
  ],
  "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_KLMNO"
        ]
      }
    ],
    "sender_credentials": {
      "active": {
        "password": "gws-ZABCD-password",
        "username": "gateway_sender_ZABCD"
      }
    }
  }
}
```

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-ABCDE-password",
      "roles": [
        "cluster_operator"
      ],
      "username": "cluster_operator_ABCDE"
    },
    {
      "password": "dev-FGHIJ-password",
      "roles": [
        "developer"
      ],
      "username": "developer_FGHIJ"
    }
  ],
  "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-KLMNO-password",
      "username": "gateway_sender_KLMNO"
    }
  }
}
```

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_ABCDE --password=cl-op-ABCDE-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://cloudcache-2.example.com/gemfire/v1 --use-http --user=cluster_operator_PQRST --password=cl-op-PQRST-password
```

- Create the cluster B region:


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

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

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.

Cluster Rebalancing

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](#) before you can use the `rebalance` command.

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.

Accessing a Service Instance

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": {
    "gfsh": "gfsh-url",
    "pulse": "pulse-url"
  },
  "users": [
    {
      "password": "developer-password",
      "username": "developer_XXX",
      "roles": [
        "developer"
      ]
    },
    {
      "password": "cluster_operator-password",
      "username": "cluster_operator_XXX",
      "roles": [
        "cluster_operator"
      ]
    }
  ]
}
```

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) or Elastic Runtime; that is, the certificate you use where your TLS termination occurs. 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.

2. Using your certificate, run the `keytool` command.

For example:

```
$ keytool -import -alias ENV -file CERTIFICATE.CER -keystore TRUSTSTORE-FILE-PATH"
```

Where:

- `ENV` is your system environment.
- `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 -alias prod-ssl -file /tmp/loadbalancer.cer -keystore /tmp/truststore/prod.myTrustStore
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 `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. Set the `JAVA_ARGS` environment variable with the following command:
`export JAVA_ARGS="-Djavax.net.ssl.trustStore=TRUSTSTORE-FILE-PATH"`
 Where: `TRUSTSTORE-FILE-PATH` is the path to the TrustStore file you created in [Create a Truststore](#).

For example:

```
$ export JAVA_ARGS="-Djavax.net.ssl.trustStore=/tmp/truststore/prod.myTrustStore"
```

4. Run `gfsh`, and then issue a `connect` command that specifies an HTTPS URL of the form:

```
connect --use-http=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.

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.

Using Pivotal Cloud Cache

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.

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](https://github.com/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": {
          "gfsh": "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 v1.3.x 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 deploy an app JAR file after connecting within gfsh using the cluster operator credentials, do the following:

1. Run this gfsh command to deploy the JAR file:

```
deploy --jar=PATH-TO-JAR/FILENAME.jar
```

For example,

```
gfsh>deploy --jar=working-directory/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}'
```

To redeploy an app JAR file after connecting within gfsh using the cluster operator role, do the following:

1. Run this gfsh command to remove the existing JAR file:

```
undeploy --jar=PATH-TO-JAR/FILENAME.jar
```

For example,

```
gfsh>undeploy --jar=current-jars/myJar.jar
```

2. Run this gfsh command to deploy the updated JAR file:

```
gfsh>deploy --jar=PATH-TO-UPDATED-JAR/FILENAME.jar
```

For example,

```
gfsh>deploy --jar=newer-jars/myJar.jar
```

3. 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}'
```

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

This section describes the two ways in which you can connect a Spring Boot app to PCC:

- Using a Tomcat app with a WAR file. This is the default method for Tomcat apps.
- Using the spring-session-data-gemfire library. This method requires that you use the correct version of these libraries.

Use the Tomcat App

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 Spring Session Data 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.

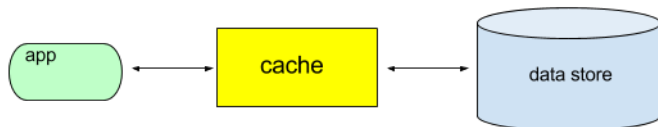
For more information, see the [Spring Data GemFire documentation](#).

Application Development

Design Patterns

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.

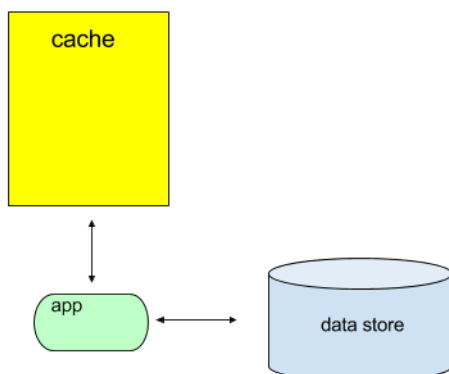
The implementation for a cache write typically creates or updates the entry within the cache. It also creates or updates the data store in one of the following ways:

- Synchronously, in a write-through manner
- Asynchronously, in a write-behind manner

Developers design the server code to implement this inline-caching pattern.

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.

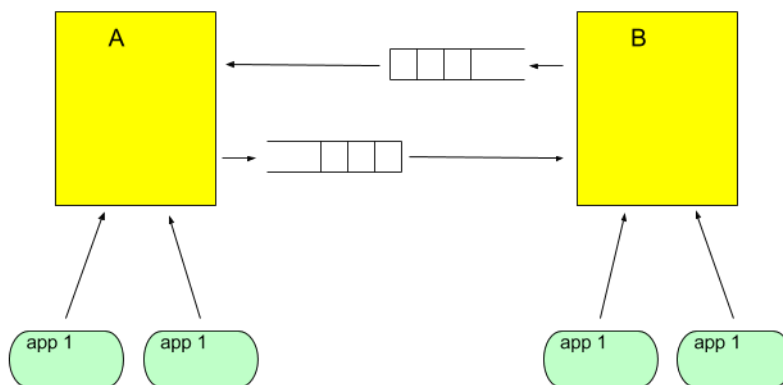
Note: SDG (Spring Data GemFire) supports the look-aside pattern, as detailed at [Configuring Spring's Cache Abstraction](#).

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.

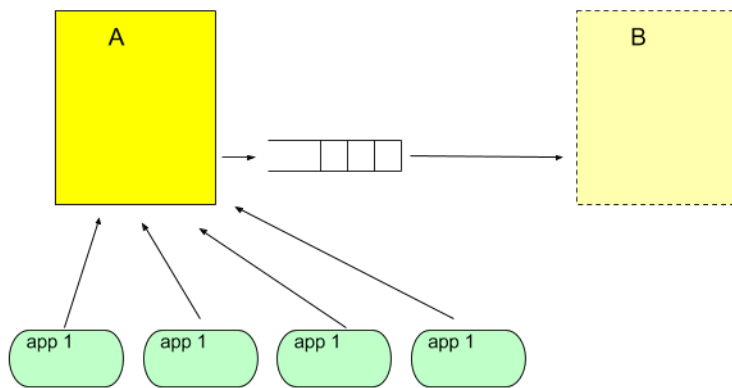
This diagram shows multiple instances of an app interacting with one of the two PCC service instances, cluster A and cluster B. Any change made in cluster A is sent to cluster B, and any change made in cluster B is sent to cluster A.



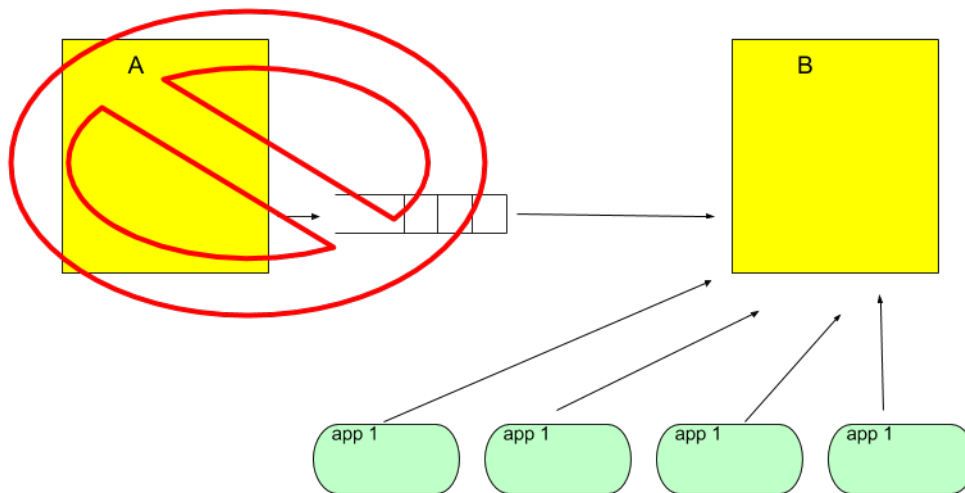
Blue-Green Disaster Recovery

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.



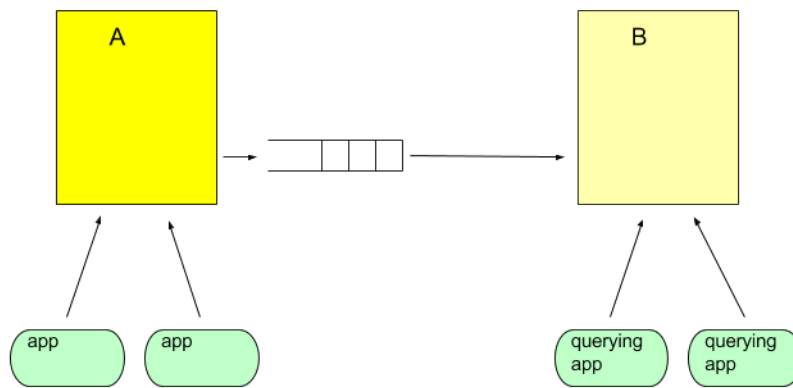
If cluster A fails, cluster B takes over.



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.



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.

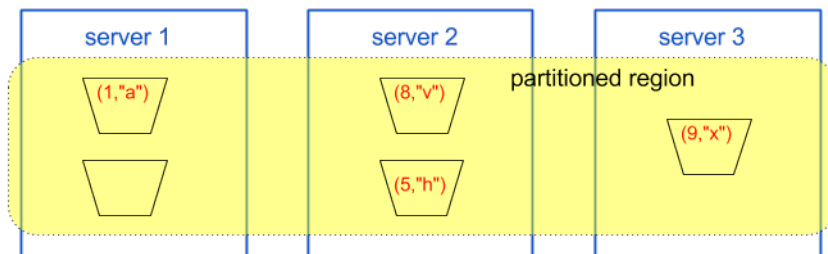
Keys

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.

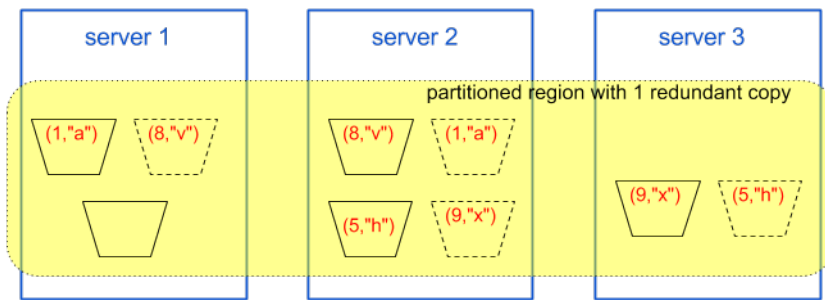
Here is a diagram that shows a single partitioned region (highlighted) with very few entries to illustrate partitioning.



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 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.

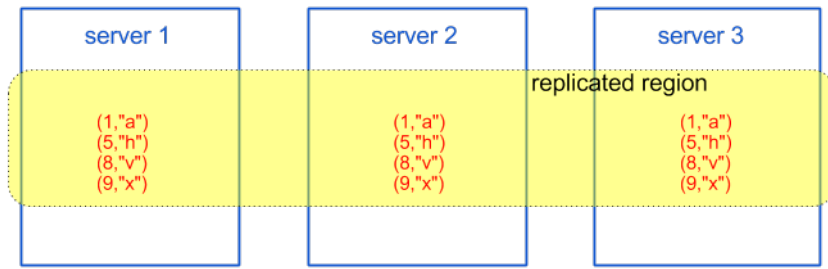
Partitioned Region Types for Creating Regions on the Server

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.

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.

Connect a Sample Java Client App to PCC

1. Clone the sample Java app at <https://github.com/cf-gemfire-org/cloudcache-sample-app.git> to use in this demonstration of how to connect an app to a service instance.
2. Create a region named `test` as described in [Create Regions with gfsH](#).
3. Update the manifest in `manifest.yml` by replacing `service0` with the name of your service instance.
4. Update the `gradle.properties` with your username and password for the Pivotal Commercial Maven Repository at <https://commercial-repo.pivotal.io>.
5. Run

```
cf push -f PATH-TO-MANIFEST
```

to deploy the app. Replace `PATH-TO-MANIFEST` with the path to your modified manifest file. 1. After the app starts, there will be an entry of ("1", "one") in the `test` region.

Pivotal Cloud Cache Release Notes

v1.3.5

Release Date: February 11, 2019

Features included in this release:

- PCC now ships with OpenJDK 1.8_192 instead of the equivalent Oracle JDK.

v1.3.4

Release Date: November 27, 2018

Features included in this release:

- **Security Vulnerability:** PCC depends upon the Pivotal Cloud Foundry On Demand Services Broker, which addressed the following security vulnerability:
 - [CVE-2018-15759](#): On Demand Services SDK timing attack vulnerability
- **Security Vulnerability:** PCC depends upon an included Java SE, which addressed the following security vulnerabilities:
 - [CVE-2018-3149](#)
 - [CVE-2018-3180](#)
 - [CVE-2018-3183](#)
 - [CVE-2018-3214](#)
- PCC is now running Pivotal GemFire v9.3.0.

v1.3.3

Release Date: July 11, 2018

Features included in this release:

- PCC is now running Pivotal GemFire v9.3.0. This release supports upgrades from PCC v1.3.1 to this 1.3.3 version. Using `gfs-h` with this GemFire version requires a JDK or JRE with Java 8 release 92 or a more recent version 8 update.
- The cluster operator role adds the `CLUSTER:MANAGE` permission to its already-defined permissions.
- This release improves performance and maintains high availability for WAN-separated clusters by creating a gateway receiver on each server, instead of creating only one gateway receiver per cluster.
- The JDK included with this PCC release is v1.8.0_170.

v1.3.2 (Limited availability)

Release Date: March 21, 2018

Features included in this release:

- This limited-availability patch release is running Pivotal GemFire v9.2.0. Inline caching is not supported, and no JAR files may be deployed to servers. Using `gfs-h` with this GemFire version requires a JDK or JRE with Java 8 release 92 or a more recent version 8 update.
- This limited-availability patch release exists *only* to support an upgrade from PCC v1.3.0 to this v1.3.2 version.
- To ensure upgrade paths, this limited-availability patch release works with many versions of Ops Manager and and Pivotal Application Service (PAS). This limited-availability patch release works with Ops Manager and and PAS versions v1.12.x, v2.0.x, v2.1.x, and 2.2.x.

v1.3.1

Release Date: March 5, 2018

Features included in this release:

- PCC is now running Pivotal GemFire v9.3.0. Using `gfsh` with this GemFire version requires a JDK or JRE with Java 8 release 92 or a more recent version 8 update.
- Inline caching is supported, which permits both write behind and write through designs for interaction with a backend data store.

Bug fixes:

- Eliminated a deadlock situation that occurred when recovering all locators, due to the use of `gfsh configure pdx`.

v1.3.0

Release Date: January 31, 2018

Features included in this release:

- Support for WAN, asynchronously replicating data across Pivotal Cloud Foundry (PCF) deployments
- Secure GemFire processes using BOSH Process Manager (BPM)
- Support for region persistence
- A single node cluster Dev Plan
- IPsec supported
- Support for viewing serialized objects in `gfsh` using query
- PCC is now running GemFire v9.2.0. Using `gfsh` with this GemFire version requires a JDK or JRE with Java 8 release 92 or a more recent version 8 update.
- Cluster users are now appended with a unique identifier and are assigned roles

Bug fixes:

- For slow IaaS, where the server takes a long time to start up, the cluster fails to start.
- Fixed issue where two different tiles could be installed with the same resource name, preventing creation of service instances.

Known Issues

- If you are upgrading to Pivotal Cloud Cache (PCC) v1.3.0 and you created service keys on PCC before you installed v1.3.0: delete and recreate the service keys so that users are assigned roles in the cluster. Then, rebind all your apps. For information about how to perform these tasks, see [Delete a Service Key](#), [Create Service Keys](#), and [Bind an App to a Service Instance](#).
- 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.