

Pivotal Cloud Cache for PCF® Documentation


Version 1.1.5

Published: 15 Nov 2018

Table of Contents

Table of Contents	2
Pivotal Cloud Cache	3
Operator Guide	8
Pivotal Cloud Cache Developer Guide	17
Pivotal Cloud Cache Release Notes	28

Pivotal Cloud Cache

 **Note:** Pivotal Cloud Cache (PCC) v1.1.5 is no longer supported. The support period for PCC has expired. 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, and binding apps

Product Snapshot

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

Element	Details
Version	v1.1.5
Release date	March 8, 2018
Software component version	GemFire v9.3.0
Compatible Ops Manager version(s)	v1.10.x and v1.11.x
Compatible Elastic Runtime version(s)	v1.10.x and v1.11.x
IaaS support	AWS, Azure, GCP, OpenStack, and vSphere
IPsec support	No
Required BOSH stemcell version	3468.21
Minimum Java buildpack version required for apps	v3.13

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, you install one tile for on-demand service plans and a different tile for pre-provisioned service plans.

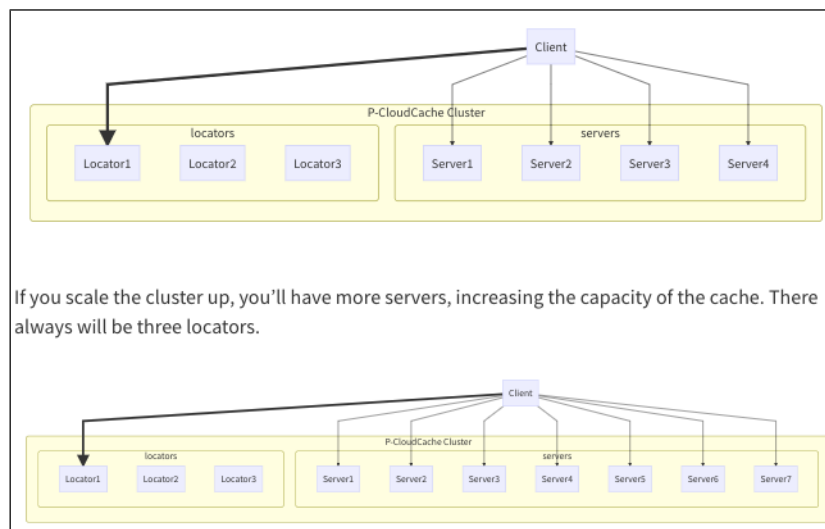
The following table lists and contrasts the different ways that PCF services package on-demand and pre-provisioned service offerings.

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 (based on Percona Server)	v1.x (based on MariaDB and Galera)
PCC	Pivotal GemFire	All versions	NA
GemFire for PCF	Pivotal GemFire	NA	All versions

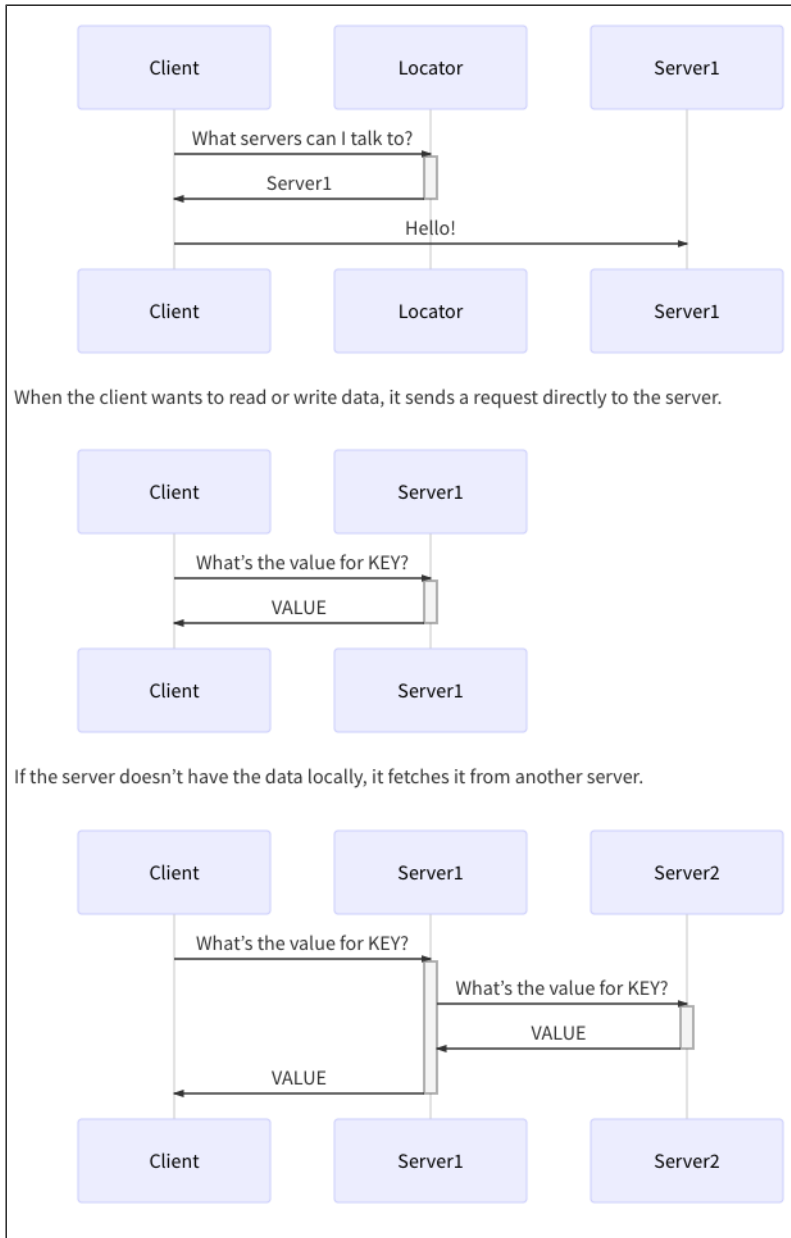
Architecture

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 will have three locators and at least four servers.

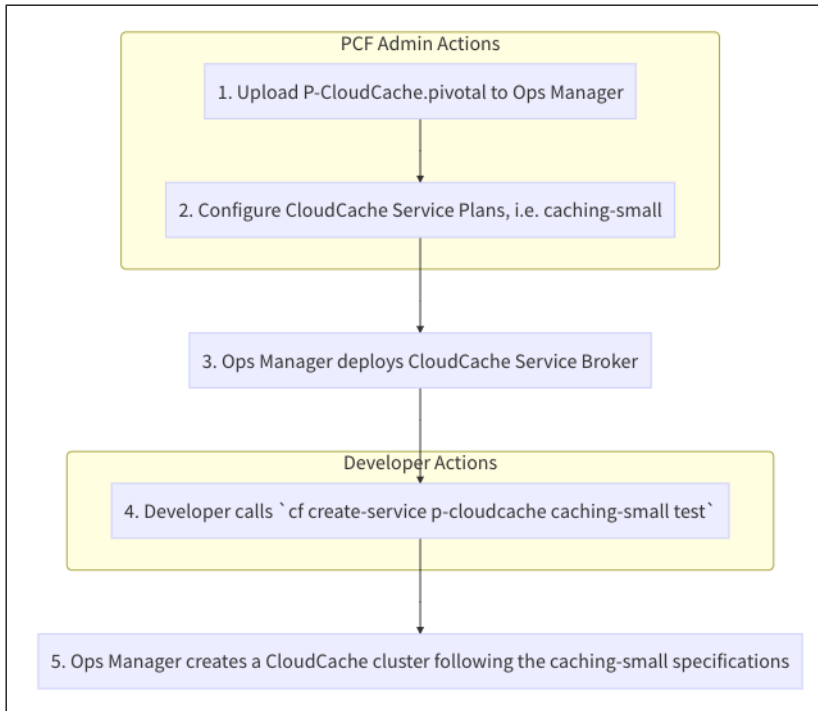


When a client connects to the cluster, it first connects to a locator. The locator replies with the IP address of a server for it to talk to. The client then connects to that server.



Workflow

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



Recommended Usage and Limitations

- PCC can be used as a cache. It supports the [look-aside cache pattern](#).
- PCC can be used to store objects in key/value format, where value can be any object.
- PCC works with gfs. You can only use the gfs version that matches this release's version of GemFire. For the version of GemFire supported in this release, see [Product Snapshot](#), above.
- 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.
- WAN (Cross Data Center) replication is not supported.
- Persistent regions 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.

Authentication

Clusters are created with two default users: `cluster_operator` and `developer`. A cluster can only be accessed using one of these two users. All client applications, gfsH, and JMX clients must authenticate as one of these users accounts to access the cluster.

Authorization

Default user roles `cluster_operator` and `developer` have different permissions:

- `cluster_operator` role has `CLUSTER:WRITE`, `CLUSTER:READ`, `DATA:MANAGE`, `DATA:WRITE`, and `DATA:READ` permissions.
- `developer` role has `CLUSTER:READ`, `DATA:WRITE`, and `DATA:READ` permissions.

You can find more details about these permissions in the Pivotal GemFire [Implementing Authorization](#) [↗](#) topic.

Feedback

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

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.

Minimum Version Requirements

PCC requires PCF v1.10 with PCF Elastic Runtime v1.10.3 or later or PCF v1.11 with PCF Elastic Runtime v1.11.0 or later.

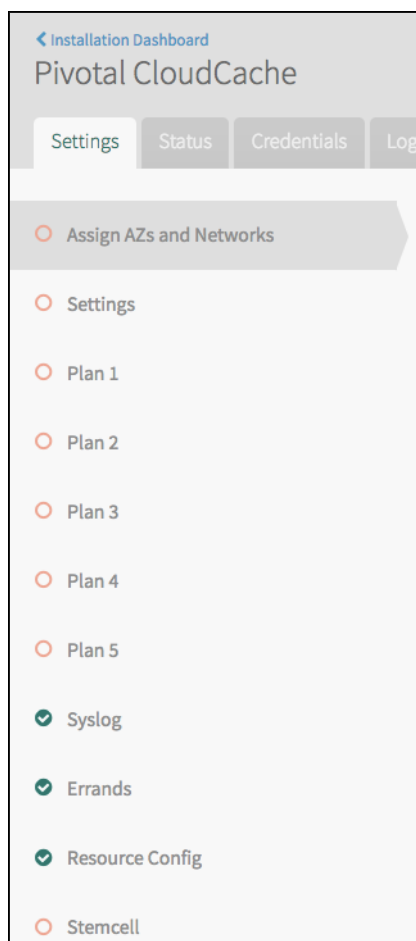
Installing and Configuring Pivotal Cloud Cache

Follow the steps below 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 PCC 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 circle containing a check mark appears next to the section name. Each section name must show this checked green circle before you can complete your installation.

- [Assign AZs and Networks](#)
- [Settings: Smoke Tests and Outbound Internet Access](#)
- [Syslog](#)
- [Plan](#)
- [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 Elastic Runtime network.
Service Network	Select the network to be used for GemFire VMs.

3. Click **Save**.

Settings: Specify Smoke Test Errands and Outbound Internet Acces

1. Click **Settings**. This section provides settings for smoke-test errands and VM options.
2. Under **Plan to use for smoke test**, select a plan to use when the `smoke-tests` errand runs. The smoke-tests errand runs after tile installation and verifies that your installation was successful. By default, the `smoke-test` errand runs on the `system` org and the `p-cloudcache-smoke-test` space.
3. Ensure that 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.

Note: 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.

4. Under **VM options**, allow or disable internet access.
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.
By default, outbound internet access is NOT allowed from service instances.
5. To allow outbound internet access from service instance, click **Allow outbound internet access from service instances (IaaS-dependent)**.

6. After you have configured smoke test errands and internet access options, click **Save**.

Set Up Syslog Forwarding

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

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

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

Field	Instructions
Enable Remote Syslog	Select to enable.

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

3. Click **Save**.

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

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*
☒ default

VM type for the Locator VMs*

small (cpu: 1, ram: 2 GB, disk: 8 GB)

Persistent disk type for the Locator VMs*

10 GB

VM type for the Server VMs*

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

Persistent disk type for the Server VMs*


10 GB

Save

Configure settings for your service plan as described in the table below.

Setting	Instructions
Service Plan Access	Enable Plan is checked by default. If you do not want to add this plan to the CF service catalog, select Disable Plan . You must enable at least one plan.
Plan Name	Allows you to customize the name of the plan. This plan name is displayed to developers when they view the service in the Marketplace..
Plan Description	Allows you to supply a plan description. The description is displayed to developers when they view the service in the Marketplace.
CF Service Access	Enabled by default. This setting determines whether the plan is displayed on the marketplace. Enable Service Access will display the service plan to all developers in the CF marketplace. Disable Service Access will not display the service plan to developers in the CF marketplace and cannot be enabled at a later time. Leave Service Access Unchanged will not display the service plan in the CF

	marketplace by default but can be enabled at a later time.
Maximum service instances	Sets the maximum allowed PCC service instances.
Maximum servers per cluster	Allows operators to set an upper limit on the number of servers developers can request.
Default Number of Servers	The number of servers a new cluster will have by default if developers do not explicitly specify a number of servers in a service instance.
Availability zones for service instances	Determines which AZs are used for a particular cluster. The members of a cluster are distributed evenly across AZs. NOTE: 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.
VM type for the Locator VMs	Specifies the type of VM to be used for locator VMs. Recommended: select a VM that has at least 1 GB of RAM and 4 GB of disk space.
Persistent disk type for the Locator VMs	Specifies the size of the persistent disk to be used for locator VMs. Recommended: select 10 GB or higher.
VM type for the Server VMs	Specifies the type of VM to be used for server VMs. Recommended: select a VM that has at least 4 GB of RAM and 8 GB of disk space.
Persistent disk type for the Server VMs	Specifies the size of the persistent disk to be used for server VMs. Recommended: select 10 GB or higher.

 **Note:** 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.


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

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**.


1. Change the setting for the errands.

 **Note:** If you are using Ops Manager v1.10.0 to v1.10.7, do not set errands to **When Changed**. This is because errands set to When Changed sometimes fail to run when the PCC tile changes.

2. Click **Save**.

Stemcell

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

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

BOSH System Health Metrics

The BOSH layer that underlies PCF generates `healthmonitor` metrics for all VMs in the deployment. However, these metrics are not included in the

Loggregator Firehose by default. To get these metrics, do either of the following:

- To send BOSH HM metrics through the Firehose, install the open-source [HM Forwarder](#).
- To retrieve BOSH health metrics outside of the Firehose, install the [JMX Bridge](#) for PCF tile.

Upgrading Pivotal Cloud Cache

Follow the steps below to upgrade PCC on PCF:

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 PCC 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`, will change the default value of these instance properties, but will 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 will not be 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 one of the following commands:

- **For Ops Manager v1.10 or earlier:** `bosh scp server/0:/var/vcap/store/gemfire-server/statistics.gfs /tmp`
- **For Ops Manager v1.11 or later:** `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 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. You 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 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-cloudcache small my-cloudcache -c '{"num_servers": 5}'
```

Enable Session State Caching with the Java Buildpack

When the service instance name is followed by the `session-replication` tag, the Java buildpack will download all the required resources for session state caching. This is available in Java buildpack v4 starting from v4.3, and in v3 starting from v3.19.

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

- When creating your service instance name, append it with the `session-replication` tag. For example, for the `p-cloudcache` service:

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

- When updating your service instance name (for example, if the updated name is `new-service-instance`), append it with the `session-replication` tag:

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

- End the service instance name with the text `-session-replication`: `my-service-instance-session-replication`.

Enable Session State Caching Using Spring Session

Session state caching for apps using [Spring Session](#) is a beta feature. It uses pre-release versions of Spring Session libraries.

To use `spring-session` with Pivotal Cloud Cache, 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 application config.

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

- Create a region named `ClusteredSpringSessions` in gfs using the `cluster_operator` credentials: `create region --name=ClusteredSpringSessions --type=PARTITION_HEAP_LRU`

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, see [Create Service Keys](#) below.

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 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:

```
{
  "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"
    },
    {
      "password": "cluster_operator-password",
      "username": "cluster_operator"
    }
  ]
}
```

This returned structure contains the following information:

- `cluster_operator` and `developer` credentials
- `gfsh-url`: A URL usable to connect the gfsh client to the service instance
- `pulse-url`: A URL usable to view the Pulse dashboard in a web browser, which allows monitoring of the service-instance status. You can use the developer credentials to authenticate.

The `cluster_operator` and `developer` roles provide access to different functions on the service instance.

You can use the `cluster_operator` credentials to administer the service instance. For example, you can use the `cluster_operator` credentials to define or delete cache regions.

You can use the `developer` credentials to perform create, retrieve, update, and delete (CRUD) data operations. You can also use the `cluster_operator` credentials to perform the same functions the `developer` credentials. As a best practice, you should use the role with the least amount of permissions necessary.

Connect with gfsh over HTTPS

Connecting over HTTPS uses the same certificate you use to secure traffic into ERT; 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.rmyTrustStore
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: ObjectTid: 2.5.29.10 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:0
]

#2: ObjectTid: 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 connect using HTTPS.

1. Download the Pivotal GemFire ZIP archive from [Pivotal Network](#).
2. Unzip and locate the correct binary for your architecture. Use `gfsh` with Unix or `gfsh.bat` with Windows.
3. Add the `gfsh` binary to your path.
4. Set the `JAVA_ARGS` environment variable with the following command:

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"
```

- Using a URL starting with `https`, open the gfsH console and connect:

```
$ gfsfsh
```

9.0.1

Monitor and Manage Pivotal GemFire

```
gfsfh>connect --use-http --url=gfsfh-url --user=cluster_operator --password=cluster_operator-password
```

Using Pivotal Cloud Cache

Create Regions with gfsh

After connecting with `gfsh` as a `cluster_operator`, 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_HEAP_LRU --redundant-copies=1
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"
```

The following region types are recommended for PCC:

- `PARTITION_HEAP_LRU`
- `REPLICATE_HEAP_LRU`

To achieve optimal performance, you should understand the type of region that is relevant to your situation. For more information, see the Pivotal GemFire [Region Types](#) documentation.

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.

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": "http://cloudcache-5574c540-1464-4f9e-b56b-74d8387cb50d.local.pcfdev.io/gemfire/v1",
          "pulse": "http://cloudcache-5574c540-1464-4f9e-b56b-74d8387cb50d.local.pcfdev.io/pulse"
        },
        "users": [
          {
            "password": "some_developer_password",
            "username": "developer"
          },
          {
            "password": "some_password",
            "username": "cluster_operator"
          }
        ]
      },
      "label": "p-cloudcache",
      "name": "test-service",
      "plan": "caching-small",
      "provider": null,
      "syslog_drain_url": null,
      "tags": [],
      "volume_mounts": []
    }
  ]
}
```

A Sample Java Client App for PCC

A sample app written in pure Java has been put together to demonstrate how to connect an app to the service instance.

The code is available at <https://github.com/cf-gemfire-org/cloudcache-sample-app.git>

The code assumes that you have already created a region named `test` in your cluster using `gfsh` as described in the section above.

Copy, update, and save the following sample manifest:

```
---
applications:
- name: sample_app
  path: build/libs/cloudcache-sample-app-1.0-SNAPSHOT-all.jar
  no-route: true
  health-check-type: none
  services:
  - service0 # replace this with the name of your service. Please refer
              # to docs above on how to create a cloud cache service instance.
```

Update the `gradle.properties` with your username and password used for <https://commercial-repo.pivotal.io/>

Run `cf push -f PATH-TO-MANIFEST` to deploy the app. Replace `PATH-TO-MANIFEST` with the path to the sample manifest you created.

After you push the app and it starts, you should see an entry in the `test` region where `key=1` and `value=one`

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` or `developer` credentials to authenticate.

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`](#) 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](#).

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.

Connecting a Spring Boot App to PCC with SSC

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 the Spring Session Data GemFire App

You can connect your Spring app to PCC to do session state caching, but you must use the correct version of the spring-session-data-gemfire library to do so. PCC v1.1.x only supports spring-session-data-gemfire version v2.0.0.M0.

PCC Version	SSDG Version
v1.1.x	v2.0.0.M0

Example `build.gradle` file that works with PCC v1.1.x:

```

version = '0.0.1-SNAPSHOT'

buildscript {
    ext {
        springBootVersion = '2.0.0.M2'
    }
    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.M0")
    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")
    testCompile("org.springframework.boot:spring-boot-starter-test")
}

```

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, perform the following steps:

1. Set `subscriptionEnabled` and `readyForEvents` to `true` on the client cache object through the annotation `@ClientCacheApplication` :

```
@ClientCacheApplication(name = "GemFireSpringApplication", readyForEvents = true, subscriptionEnabled = true)
```

2. To enable durable queries, set `durableClientId` and `keepAlive` through the annotation:

```
@ClientCacheApplication(name = "GemFireSpringApplication", durableClientId = "durable-client-id", keepAlive = true, readyForEvents = true, subscriptionEnabled = true)
```

3. Use the annotation `@ContinuousQuery` to specify the continuous query and bind it to a function to handle the query when it is triggered:

```
@Component
public class ContinuousQuery {

    @ContinuousQuery(name = "PestoQuery", query = "SELECT * FROM /Region", durable = true)
    public void handlechanges(CqEvent event) {
        //PERFORM SOME ACTION
    }
}
```

If you want to specify the continuous query in its own class, you must annotate `@Component` on the class so the Spring component scan detects the `@ContinuousQuery` annotation.

If you want the continuous query to be durable, you must set the `durable` attribute to `true`.

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

Pivotal Cloud Cache Release Notes

v1.1.5

Release Date: March 8, 2018

Features included in this release:

- PCC is now running Pivotal GemFire v9.3.0

v1.1.4

Release Date: January 26, 2018

Features included in this release:

- Updates stemcell to 3468

v1.1.3

Release Date: January 9, 2018

Features included in this release:

- Upgraded Pivotal GemFire to v9.2.0 to address CVEs: CVE-2017-9796, CVE-2017-9795, and CVE-2017-12622

v1.1.2

Release Date: November 2, 2017

Features included in this release:

- Spring Session Data Gemfire 2.0.0.M1 compatible
- When the AZs in the tile configuration are updated and the upgrade-all-service-instances errand is run, existing instances will not change AZs. However new instances that are created will use the newly configured AZs.

Bug fixes:

- Pulse bug: Upgrades failed when a connection to pulse was still open during upgrade.
- Service Instances failed to come up when gfsh start server was slow.
- When smoke-tests failed the would print the service-key exposing credentials.

v1.1.1

Release Date: August 10, 2017

Bug fixes:

- Improved password requirement of the testing utility
- Increased `monit` for smoke tests to be successful
- Improved kill command to force kill in the event of `gfsh stop` failure
- Fixed upgrading a cloud cache service failure when `syslog tls` is disabled and `Send service instance logs to external syslog` is enabled

v1.1.0

Features included in this release:

- HTTP session caching is supported in Pivotal Cloud Cache (PCC).
- PCC is compatible with PCF v1.10 and v1.11.
- PCC signs BOSH deployments with SHA-2 to prevent certificate collisions.
- Syslog setup supports standard RFC 5424 format.