

Pivotal HD Service Broker For Pivotal-CF

Version 1.0.0.0

Release Notes

Rev: 01

Table of Contents

1	Pivotal HD Service Broker for Cloud Foundry	3
1.1	Availability	3
1.2	Installation	3
1.3	Start/Stop/Status Commands	3
1.4	Utility Commands	3
1.5	Configuring Communication with the CF Cloud Controller	4
1.5.1	Network Configuration to allow External Broker to talk to Cloud Foundry	4
1.6	Configuring Provisioners	6
1.6.1	HDFS Provisioner Settings	6
1.6.2	HAWQ Provisioner Settings	7
1.6.3	Kerberos Configuration Settings	8
1.6.4	Defaults and Examples	9

1 Pivotal HD Service Broker for Cloud Foundry

The Pivotal HD Service Broker component is provided for download and install with the PHD 1.1.x distributions. This broker, when installed and configured, allows Cloud Foundry applications to bind to and use non-CF PHD clusters. When you have completed the installation and configuration of your broker refer to [PHD Service Provisioning via Pivotal CF](#).

1.1 Availability

This component is available as a download with PHD releases starting with PHD 1.1.1.

1.2 Installation

1. Download and untar the `phd-cf-broker-1.0.0.0-1.tar` package.
2. Change to the installation directory:

```
cd phd-cf-broker-1.0.0.0-1/
```
3. Make a copy of the sample `phd.yml` file:

```
cp phd.example.yml to phd.yml
```
4. Edit the `phd.yml` to reflect your cluster configuration and the Cloud Controller info (see sections below for instructions).
5. Install the broker by running:

```
./install.sh
```
6. Start the broker by running:

```
service pcf-phd-broker start
```

After installation you can the configuration file in `/etc/pcf/phd-broker/conf/phd.yml`. Any changes will not take effect until you restart the broker (`service pcf-phd-broker restart`).

1.3 Start/Stop/Status Commands

After installation the following standard service commands will be available.

- `service pcf-phd-broker start`
- `service pcf-phd-broker stop`
- `service pcf-phd-broker restart`
- `service pcf-phd-broker try-restart`
- `service pcf-phd-broker status`

1.4 Utility Commands

After installation the following utility command is provided:

- `phd-broker-tokenize`

Since the service broker requires several system passwords to be able to provision application users, and it is a security risk to have these in cleartext, this command provides a means to encrypt passwords. The command has two options: `--tokenize-secret` and `--tokenize-string` as indicated below. It generates an encrypted token that can be used in the `phd.yml` configuration file instead of the cleartext password. The encryption is currently fixed at 128 bits and is protected by a broker created keytab file readable only by the broker user (and `root` on the host).

Command options:

`--tokenize-secret` Prompts user to enter the password twice for verification. Password is not echoed to terminal. Outputs encrypted token.

`--tokenize-string <string>` Encrypts string and outputs encrypted token.

You must run this tool as `root` or the `phdbroker` user (`sudo -u phdbroker phd-broker-tokenize <options>`).

The full command path is `/usr/bin/phd-broker-tokenize`.

The token should be copied and entered into the appropriate place in your `phd.yml` file. For example if you wanted to create an encrypted token for the host password you would run `phd-broker-tokenize --tokenize-secret`, enter the password twice for verification, then copy the resulting field into the value part for the `host:` key in your provisioner configuration (see examples below).

1.5 Configuring Communication with the CF Cloud Controller

By default, the service broker does not talk to the Cloud Controller. If it finds a key `advertise-interval`, it will start advertising to the Cloud Controller repeatedly at the interval indicated by the value for that key (in milli-seconds).

1.5.1 Network Configuration to allow External Broker to talk to Cloud Foundry

If your Cloud Foundry deployment is on a different private network than your PHD installation you need to place the HA proxy component of the CF runtime on a network that has access to your PHD installation.



If your Cloud Foundry deployment is on the same network as your PHD installation skip this section.

Cloud Foundry requires a DNS (either external, such as Route 53, or internal) setup that resolves all CF services. If the CF domain name ends with, for example, `.my-org.com`, the DNS must resolve all `*.my-org.com` to the HA Proxy component within the CF runtime. If Pivotal CF is installed in a private subnet, an external Service Broker will not be able to communicate with the CF runtime. Therefore, in order for your external Service Broker to talk to the CF runtime components, the HA proxy is required to have an external IP. The standard installation of Pivotal CF only provisions one interface for all their components including the HA Proxy, so you need to make the changes after CF runtime deployment to add the external interface.

To configure an external interface, the admin has to perform the following steps :

1. Install the CF runtime component
2. Modify the generated BOSH deployment manifest to add a second network:
 - The generated BOSH manifest can be found in the `/var/tempest/workspaces/default/deployments/` directory of the Pivotal CF VM
 - An example of the BOSH manifest specifying a second (external) network interface is shown below.

```
...
...
networks:
- name: default
  subnets:
  - range: 192.168.0.0/24
    gateway: 192.168.0.1
    dns:
    - 192.168.0.11
    - 10.5.215.126
    static:
    - 192.168.0.15
    ....
    - 192.168.0.60
    reserved:
    ....
    - 192.168.0.81-192.168.0.254
    cloud_properties:
      name: VLAN_AS
- name: external
  subnets:
  - range: 10.5.215.0/22
    gateway: 10.5.212.1
    static:
    - 10.5.215.26
    dns:
    - 192.168.0.11
    - 10.5.215.126
    cloud_properties:
      name: VM Network
...
...
```

3. Associate second network to the HA Proxy job:

For example:

```
...
...
jobs:
...
- name: ha_proxy
  template: haproxy
  instances: 1
  resource_pool: ha_proxy
  persistent_disk: 0
  networks:
  - name: default
    static_ips:
    - 192.168.0.15
  - name: external
    default: [dns, gateway]
    static_ips:
    - 10.5.215.26
...
...
```

4. Re-deploy using the BOSH CLI command: `bosh deploy` (See [Re-Deploying using the BOSH CLI](#) if you originally used the Installer UI to deploy)

1.6 Configuring Provisioners

An example template (`phd.yml.example`) is provided to start. In this file places where cluster specific information is needed (such as FQDN of namenode) are indicated. Additional keys for service provisioning are indicated below. Copy the example template to `phd.yml` in the installation directory before installing or editing. If you want to edit this file after installation you must restart the service broker after editing the file.



Support for Kerberized clusters is an alpha feature and full certification of CF applications working with Kerberized clusters is not yet available.


1.6.1 HDFS Provisioner Settings

```
host: <host to use for provisioning e.g., namenode>
host_user: <username to login as; must be able to run useradd/usermode/usedel commands>
host_password: <password for host_user>
auth_type: <simple || kerberos ; default is simple>
hdfs_admin_user: <hdfs admin user>
hdfs_user_dir: <optional; directory (leading / included, no trailing /) used for creating user
directories; default is /instances.>
hdfs_uri: <URI for HDFS>
```


The next two settings are applicable only to HDFS provisioners and are useful on non-CF deployed clusters to allow the use of distributed shell tools. The `client_hosts_file` is only used when creating the provision instance (admin) user. The files must not contain duplicate hosts. This is required for Kerberized clusters.

```
server_hosts_path: <required only for kerberized clusters; path to file containing server hosts,
default is null>
client_hosts_path: <required only for kerberized clusters; path to file containing client hosts,
default is null>
```


These next five settings allow a parameterized description of the user creation command to be provided. Keywords: `$HOST_PATH` is replaced by the applicable `[server | client]_hosts_path` value; `$USER` is replaced by the `host_user`; `$GROUP` by the group.

 You may omit `$GROUP` and specify the group in your `user/group add/delete` commands but always use `$USER` in user add commands; DO NOT specify the username (this is assigned by the SB).

```
addgroup_command: <optional; group add command>
client_adduser_command: <optional; user add command for Hadoop client hosts>
server_adduser_command: <optional; user add command for Hadoop server hosts>
delgroup_command: <optional; group delete command>
deluser_command: <optional; user delete command>
```


 If you want to create server or client user accounts on multiple hosts for non-CF clusters you can do so by:

1. Set the `server_hosts_path` and/or `client_hosts_path` parameters to point to your hosts files;
2. Set all the add/delete command values so that they use the command format required by the distributed shell tool you are using.


 The HDFS PROVISION and BIND requests both create a user and return a user ID. For PROVISION (cli create service) only if `client_hosts_path` is non-empty the broker will create a user with shell on client systems. This user is for admin purposes and not user application usage. If `auth_type` is "kerberos" a password, representing the user's Kerberos principal password, will also be returned in both BIND and PROVISION.


1.6.2 HAWQ Provisioner Settings

```
host: <HAWQ master host FQDN; default is null>
port: <HAWQ master port for client connections; default is 5432>
database: postgres <database to use for admin actions; default is postgres>
username: <HAWQ superuser; default is gpadmin>
password: <HAWQ superuser password; default is empty string>
auth_type: <simple || kerberos ; default is simple>
```


 only indicate "kerberos" if HAWQ is configured to use kerberos for incoming client connections. If HAWQ is running on top of kerberized HDFS but is not using kerberos for incoming client authentication use "simple". The HAWQ provisioner will return a "password" key: value for logging into the HAWQ database in the BIND response for both auth types. Note that unlike the BIND response the HAWQ PROVISION (cli create service command) response does not create a user and will not include a password.

1.6.3 Kerberos Configuration Settings

 Support for Kerberized clusters is currently an alpha feature as use of Cloud Foundry applications provisioned on Kerberized clusters has not yet been certified.

 If you are provisioning on a Kerberized cluster the host the SB is on must be able to communicate with other hosts using any distributed shell tools you are using. For PHD this would typically require that the broker be on the PCC/ICM admin host.

```
auth_type: kerberos
kadmin_host: <optional; specify the host to login to to run "kadmin" if not localhost; default null>
kadmin_host_user: <optional; user to login to kadmin_host with; default host_user>
kadmin_host_password: <optional; password to login to kadmin_host with; default host_password>
kadmin_realm: <Kerberos realm to use, if not default; optional>
kadmin_user: <optional; Kerberos kadmin username; expect admin to be of form user/admin>
kadmin_password: <optional; Kerberos kadmin password>
hdfs_kuser_password: <required for HDFS provisioners only; kerberos password for HDFS admin user>
```

 For HDFS and HAWQ provisioners if the `auth_type` is set to `kerberos` and the provision succeeds a key: value of `password: <password>` will be returned in credentials and represent the Kerberos password.

1.6.4 Defaults and Examples

HDFS non-kerberized group and user add/delete command defaults:

```
addgroup_command: sudo groupadd $GROUP
client_adduser_command: sudo useradd -s /bin/bash $USER -g $GROUP
server_adduser_command: sudo useradd -s /sbin/nologin -M $USER -g $GROUP
delgroup_command: sudo groupdel $GROUP
deluser_command: sudo userdel -f $USER
```

HDFS kerberized group and user add/delete command defaults:

```
addgroup_command: sudo massh $HOST_PATH bombed "sudo groupadd $GROUP"
client_adduser_command: sudo massh $HOST_PATH bombed "sudo useradd -s /bin/bash -g $GROUP $USER"
server_adduser_command: sudo massh $HOST_PATH bombed "sudo useradd -M -s /sbin/nologin -g $GROUP $USER"
delgroup_command: sudo massh $HOST_PATH bombed "groupdel $GROUP"
deluser_command: sudo massh $HOST_PATH bombed "userdel -f $USER"
```



If kadmin login info is not provided the administrator must add the principal manually before the app will be able to access services.

Example HDFS provisioner config:



The host here should be the namenode host.

```
provisioner-conf:
  host: centos61-2.localdomain
  host_user: root
  host_password: $SBENCTOKEN$:65QjaricLfoVSYoG+pFA4g==
  hdfs_admin_user: hdfs
  hdfs_user_dir: /user
  hdfs_uri: hdfs://centos61-2.localdomain:8020
```

Example HAWQ provisioner config:



The host here should be the HAWQ master host.

```
provisioner-conf:
  host: centos61-2.localdomain
  port: 5432
  database: postgres
  username: gpadmin
  password: $SBENCTOKEN$:LTFofttf0nusNPuYSMdPfA==
```

Example HDFS provisioner config (kerberized):

```
provisioner-conf:
  host: centos61-2.localdomain
  host_user: root
  host_password: $SBENCTOKEN$:FYy6dJCKBF4q29w5pujnLQ==
  hdfs_admin_user: hdfs
  hdfs_user_dir: /user
  hdfs_uri: hdfs://centos61-2.localdomain:8020
  auth_type: kerberos
  kadmin_host: localhost
  kadmin_realm: PHD.BIGDATA.COM
  kadmin_user: user
  kadmin_password: $SBENCTOKEN$:MCI8fHu8mtmGeJPrtwj/gA==
  hdfs_kuser_password: $SBENCTOKEN$:PiElHTjkSRImZ+cW0psO+g==
  server_hosts_path: /etc/pcf/phd-broker/conf/shosts.txt
  client_hosts_path: /etc/pcf/phd-broker/conf/chosts.txt
  addgroup_command: massh $HOST_PATH bombed "sudo groupadd $GROUP"
  client_adduser_command: massh $HOST_PATH bombed "/usr/sbin/useradd -s /bin/bash -g $GROUP $USER"
  server_adduser_command: massh $HOST_PATH bombed "/usr/sbin/useradd -s /sbin/nologin -M -g $GROUP $USER"
```

Example HAWQ provisioner config (kerberized):

```
provisioner-conf:
  host: centos61-2.localdomain
  port: 5432
  database: postgres
  username: gpadmin
  password: $SBENCTOKEN$:TKi8odkeogeBwtmrEt/4w==
  auth_type: kerberos
  kadmin_realm: PHD.BIGDATA.COM
  kadmin_host: localhost
  kadmin_host_user: kdcadmin
  kadmin_host_password: $SBENCTOKEN$:aJ4UCMQMm3Z7RF6uokQndg==
  kadmin_user: user
  kadmin_password: $SBENCTOKEN$:zs/SwCKgue8iTLVw4d2lXA==
```