Quickstart

PostgreSQL Operator Quickstart

Can't wait to try out the PostgreSQL Operator? Let us show you the quickest possible path to getting up and running.

There are two paths to quickly get you up and running with the PostgreSQL Operator:

- Installation via the PostgreSQL Operator Installer
- Installation via a Marketplace
  - Installation via Operator Lifecycle Manager
  - Installation via Google Cloud Marketplace

Marketplaces can help you get more quickly started in your environment as they provide a mostly automated process, but there are a few steps you will need to take to ensure you can fully utilize your PostgreSQL Operator environment. You can find out more information about how to get started with one of those installers in the Installation section.

PostgreSQL Operator Installer

Below will guide you through the steps for installing and using the PostgreSQL Operator using an installer that works with Ansible.

Installation

Install the PostgreSQL Operator

On environments that have a default storage class set up (which is most modern Kubernetes environments), the below command should work:

```
kubectl create namespace pgo
kubectl apply -f
https://raw.githubusercontent.com/CrunchyData/po
operator/v4.5.1/installers/kubectl/postgres-
operator.yml
```

This will launch the `pgo-deployer` container that will run the various setup and installation jobs. This can take a few minutes to complete depending on your Kubernetes cluster.

If your install is unsuccessful, you may need to modify your configuration. Please read the "Troubleshooting" section. You can still get up and running fairly quickly with just a little bit of configuration.

Install the `pgo` Client

During or after the installation of the PostgreSQL Operator, download the `pgo` client set up script. This will help set up your local environment for using the PostgreSQL Operator:

```
curl
https://raw.githubusercontent.com/CrunchyData/po
operator/v4.5.1/installers/kubectl/client-
setup.sh > client-setup.sh
chmod +x client-setup.sh
```

When the PostgreSQL Operator is done installing, run the client setup script:

```
./client-setup.sh
```

This will download the `pgo` client and provide instructions for how to easily use it in your environment. It will prompt you to add some environmental variables for you to set up in your session, which you can do with the following commands:

```
export PGOUSER="${HOME?}/.pgo/pgo/pgouser"
export
PGO_CA_CERT="${HOME?}/.pgo/pgo/client.crt"
export
PGO_CLIENT_CERT="${HOME?}/.pgo/pgo/client.crt"
export
PGO_CLIENT_KEY="${HOME?}/.pgo/pgo/client.key"
export
PGO_APISERVER_URL='https://127.0.0.1:8443'
export PGO_NAMESPACE=pgo
```

If you wish to permanently add these variables to your environment, you can run the following:

```
cat <<EOF >> ~/.bashrc
export PGOUSER="${HOME?}/.pgo/pgo/pgouser"
export
PGO_CA_CERT="${HOME?}/.pgo/pgo/client.crt"
export
PGO_CLIENT_CERT="${HOME?}/.pgo/pgo/client.crt"
export
PGO_CLIENT_KEY="${HOME?}/.pgo/pgo/client.key"
export
PGO_APISERVER_URL='https://127.0.0.1:8443'
export PGO_NAMESPACE=pgo
EOF

source ~/.bashrc
```

NOTE: For macOS users, you must use `~/.bash_profile` instead of `~/.bashrc`

Post-Installation Setup

Below are a few steps to check if the PostgreSQL Operator is up and running.

By default, the PostgreSQL Operator installs into a namespace called `pgo`. First, see that the Kubernetes Deployment of the Operator exists and is healthy:

```
kubectl -n pgo get deployments
```

If successful, you should see output similar to this:

```
NAME                READY   UP-TO-DATE
AVAILABLE   AGE
postgres-operator   1/1     1            1
16h
```

Next, see if the Pods that run the PostgreSQL Operator are up and running:

```
kubectl -n pgo get pods
```

If successful, you should see output similar to this:

```
NAME                                  READY
STATUS    RESTARTS   AGE
postgres-operator-56d6ccb97-tmz7m    4/4
Running   0          2m
```

Finally, let's see if we can connect to the PostgreSQL Operator from the `pgo` command-line client. The Ansible installer installs the `pgo` command line client into your environment, along with the username/password file that allows you to access the

PostgreSQL Operator. In order to communicate with the PostgreSQL Operator API server, you will first need to set up a port forward to your local environment.

In a new console window, run the following command to set up a port forward:

```
kubectl -n pgo port-forward svc/postgres-
operator 8443:8443
```

Back to your original console window, you can verify that you can connect to the PostgreSQL Operator using the following command:

```
pgo version
```

If successful, you should see output similar to this:

```
pgo client version 4.5.1
pgo-apiserver version 4.5.1
```

Create a PostgreSQL Cluster

The quickstart installation method creates a namespace called `pgo` where the PostgreSQL Operator manages PostgreSQL clusters. Try creating a PostgreSQL cluster called `hippo`:

```
pgo create cluster -n pgo hippo
```

Alternatively, because we set the `PGO_NAMESPACE` environmental variable in our `.bashrc` file, we could omit the `-n` flag from the `pgo create cluster` command and just run this:

```
pgo create cluster hippo
```

Even with `PGO_NAMESPACE` set, you can always overwrite which namespace to use by setting the `-n` flag for the specific command. For explicitness, we will continue to use the `-n` flag in the remaining examples of this quickstart.

If your cluster creation command executed successfully, you should see output similar to this:

```
created Pgcluster hippo
workflow id 1cd0d225-7cd4-4044-b269-
aa7bedae219b
```

This will create a PostgreSQL cluster named `hippo`. It may take a few moments for the cluster to be provisioned. You can see the status of this cluster using the `pgo test` command:

```
pgo test -n pgo hippo
```

When everything is up and running, you should see output similar to this:

```
cluster : hippo
        Services
                primary (10.97.140.113:5432):
UP
        Instances
                primary (hippo-7b64747476-
6dr4h): UP
```

The `pgo test` command provides you the basic information you need to connect to your PostgreSQL cluster from within your Kubernetes environment. For more detailed information, you can use `pgo show cluster -n pgo hippo`.

Connect to a PostgreSQL Cluster

By default, the PostgreSQL Operator creates a database inside the cluster with the same name of the cluster, in this case, `hippo`. Below demonstrates how we can connect to `hippo`.

How Users Work

You can get information about the users in your cluster with the `pgo show user` command:

```
pgo show user -n pgo hippo
```

This will give you all the unprivileged, non-system PostgreSQL users for the `hippo` PostgreSQL cluster, for example:

```
CLUSTER USERNAME PASSWORD
EXPIRES STATUS ERROR
------- -------- ------------------------ ----
--- ------ -----
hippo   testuser datalake
never   ok
```

To get the information about all PostgreSQL users that the PostgreSQL Operator is managing, you will need to use the `--show-system-accounts` flag:

```
pgo show user -n pgo hippo --show-system-
accounts
```

which returns something similar to:

```
CLUSTER USERNAME       PASSWORD
EXPIRES STATUS ERROR
------- -------------- -----------------------
- ------- ------ -----
hippo   postgres       <REDACTED>
never   ok
hippo   primaryuser    <REDACTED>
never   ok
hippo   testuser       datalake
never   ok
```

The `postgres` user represents the database superuser and has every privilege granted to it. The PostgreSQL Operator securely interfaces through the `postgres` account to perform certain actions, such as managing users.

The `primaryuser` is the used for replication and high availability. You should never need to interface with this user account.

Connecting via `psql`

Let's see how we can connect to `hippo` using `psql`, the command-line tool for accessing PostgreSQL. Ensure you have installed the `psql` client.

The PostgreSQL Operator creates a service with the same name as the cluster. See for yourself! Get a list of all of the Services available in the `pgo` namespace:

```
kubectl -n pgo get svc

NAME                         TYPE
CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
hippo                        ClusterIP
10.96.218.63    <none>
2022/TCP,5432/TCP            59m
hippo-backrest-shared-repo   ClusterIP
10.96.75.175    <none>        2022/TCP
59m
postgres-operator            ClusterIP
10.96.121.246   <none>
8443/TCP,4171/TCP,4150/TCP   71m
```

Let's connect the `hippo` cluster. First, in a different console window, set up a port forward to the `hippo` service:

```
kubectl -n pgo port-forward svc/hippo
5432:5432
```

You can connect to the database with the following command, substituting `datalake` for your actual password:

```
PGPASSWORD=datalake psql -h localhost -p 5432
-U testuser hippo
```

You should then be greeted with the PostgreSQL prompt:

```
psql (12.5)
Type "help" for help.

hippo=>
```

Connecting via pgAdmin 4

pgAdmin 4 is a graphical tool that can be used to manage and query a PostgreSQL database from a web browser. The PostgreSQL Operator provides a convenient integration with pgAdmin 4 for managing how users can log into the database.

To add pgAdmin 4 to `hippo`, you can execute the following command:

```
pgo create pgadmin -n pgo hippo
```

It will take a few moments to create the pgAdmin 4 instance. The PostgreSQL Operator also creates a pgAdmin 4 service. See for yourself! Get a list of all of the Services available in the `pgo` namespace:

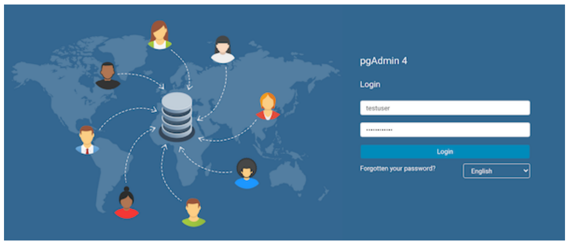
```
kubectl -n pgo get svc

NAME                          TYPE
CLUSTER-IP      EXTERNAL-IP    PORT(S)
AGE
hippo                         ClusterIP
10.96.218.63    <none>
2022/TCP,5432/TCP             59m
hippo-backrest-shared-repo    ClusterIP
10.96.75.175    <none>         2022/TCP
59m
hippo-pgadmin                 ClusterIP
10.96.165.27    <none>         5050/TCP
5m1s
postgres-operator             ClusterIP
10.96.121.246   <none>
8443/TCP,4171/TCP,4150/TCP    71m
```

Let's connect to our `hippo` cluster via pgAdmin 4! In a different terminal, set up a port forward to pgAdmin 4:

```
kubectl -n pgo port-forward svc/hippo-pgadmin
5050:5050
```

Navigate your browser to http://localhost:5050 and use your database username (`testuser`) and password (e.g. `datalake`) to log in. Though the prompt says "email address", using your PostgreSQL username will work:



testuser

(There are occasions where the initial credentials do not properly get set in pgAdmin 4. If you have trouble logging in, try running the command `pgo update user -n pgo hippo --username=testuser --password=datalake`).

Once logged into pgAdmin 4, you will be automatically connected to your database. Explore pgAdmin 4 and run some queries!

For more information, please see the section on pgAdmin 4.

Troubleshooting

Installation Failures

Some Kubernetes environments may require you to customize the configuration for the PostgreSQL Operator installer. The below provides a guide on the common parameters that require modification, though this may vary based on your installation. For a full reference, please visit the Installation section.

If you already attempted to install the PostgreSQL Operator and that failed, the easiest way to clean up that installation is to delete the Namespace that you attempted to install the PostgreSQL Operator into. Note: This deletes all of the other objects in the Namespace, so please be sure this is OK!

To delete the namespace, you can run the following command:

```
kubectl delete namespace pgo
```

Get the PostgreSQL Operator Installer Manifest

You will need to download the PostgreSQL Operator Installer manifest to your environment, which you can do with the following command:

```
curl
https://raw.githubusercontent.com/CrunchyData/po
operator/v4.5.1/installers/kubectl/postgres-
operator.yml > postgres-operator.yml
```

Configure the PostgreSQL Operator Installer

There are many configuration parameters to help you fine tune your installation, but there are a few that you may want to change to get the PostgreSQL Operator to run in your environment. Open up the `postgres-operator.yml` file and edit a few variables.

Find the `pgo_admin_password` variable. This is the password you will use with the pgo client to manage your PostgreSQL clusters. The default is `password`, but you can change it to something like `hippo-elephant`.

You may also need to set the storage default storage classes that you would like the PostgreSQL Operator to use. These variables are called `primary_storage`, `replica_storage`, `backup_storage`, and `backrest_storage`. There are several storage configurations listed out in the configuration file under the heading `storage[1-9]_name`. Find the one that you want to use, and set it to that value.

For example, if your Kubernetes environment is using NFS storage, you would set these variables to the following:

```
backrest_storage: "nfsstorage"
backup_storage: "nfsstorage"
primary_storage: "nfsstorage"
replica_storage: "nfsstorage"
```

If you are using either Openshift or CodeReady Containers and you have a `restricted` Security Context Constraint, you will need to set `disable_fsgroup` to `true` in order to deploy the PostgreSQL Operator.

For a full list of available storage types that can be used with this installation method, please review the configuration parameters.

When you are done editing the file, you can install the PostgreSQL Operator by running the following commands:

```
kubectl create namespace pgo
kubectl apply -f postgres-operator.yml
```