Work With Postgres on Astra

Project Astra

Erika Barcott, Ben Cammett
November 11, 2020

NetApp®

Table of Contents

Work With Postgres on Astra

This guide focuses on Helm as the preferred way to deploy Postgres apps. Plain YAML and Operator-based deployments may be covered in future guides.

For express instructions on launching Postgres on Astra, see Deploy Postgres from a Helm Chart.

Postgres is a validated app for the Astra Beta program. Learn the difference between Validated and Standard apps.

Postgres 11.7 is the only version supported in the Astra beta program.

Requirements

In order to deploy Postgres from a Helm chart on a cluster registered with Astra, you will need the following:

GKE Cluster

An up-to-date Kubernetes cluster (version 1.17+) which is connected to Astra. For help creating your GKE cluster and connecting it to Astra, see the Getting Started Guide.

Kubectl

Kubectl is a standard tool for interacting with Kubernetes. For more information, see the guide Install and Set Up kubectl in the official Kubernetes documentation.

Kubeconfig

The Kubeconfig file contains the credentials which let kubectl communicate with your Kubernetes cluster. to learn how to download your GKE Kubeconfig file, see the Google Cloud guide for configuring cluster access for kubectl.

CVS and Cloud Central

CVS and Cloud Central are the storage layer and connective elements for Astra, respectively. More details on how to configure CVS on GCP may be found in the workflow guide for CVS.

Helm (v3)

Helm is a popular way to organize and install apps on Kubernetes. To install Helm on your local computer, follow their handy install guide.

Postgres Requirements

For a Posgres application, Astra Alpha requires:

- `global.storageClass` value to be set to the storageClass representing either CVS or Trident (or, that storageClass is set as your cluster's default provisioner).
- The namespace set to something other than default, using the `--namespace` argument.
- A single node deployment. Multi-node and HA deployments will be supported in future releases.

The Astra alpha program does not support replicas or failovers. Only single instance versions of the databases are supported. For testing Astra in Alpha, leave replication off, and check that the `global.storageClass` value in `values.yaml` is pointing to the correct storageClass.

Namespace Requirements

You must deploy your app in a namespace other than the default. In the following example, we create and use the namespace `testdb` for the deployment.

A namespace which is empty for more than 60 seconds will be ignored by Astra. Thus, you want to be sure to deploy your app into your namespace within one minute after you create the namespace.

In the following example, we use `&&` to concatenate the commands for creating the namespace and deploying the app. We recommend this approach, as it ensures the commands are run in sequence even if you get interrupted.

We recommend the use of `&&` instead of `;` to concatenate commands. `&&` is conditional, and only runs the second command if the first command completes successfully.

Using psql on Astra

During the Astra alpha program, if you need to perform operations on Postgres pods (such as creating or restoring from backup), be sure to exit out of the psql client if you are using it on the pod.

Astra requires psql access to freeze and thaw the databases. If there is a pre-existing connection the snapshot/backup/clone operation will fail.

Install Postgres

For the Astra alpha program, we recommend the Bitnami Postgres chart. To install this chart, see Deploy Postgres from a Helm Chart.

The values need to be set to consume the volumes provisioned by CVS, be deployed in a namespace other than default, and your stateful app needs to be available to Astra.

By default the Bitnami Postgres chart uses a cluster's default dynamic provisioner. Since Trident (part of Astra) automatically sets CVS as the default storage class, you should be in good shape. Use `kubectl get sc` to see what your cluster's storageClasses are. This produces output like the following:

```
NAME                          PROVISIONER            RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
netapp-cvs-extreme            csi.trident.netapp.io   Delete          Immediate
true                   26h
netapp-cvs-premium (default)  csi.trident.netapp.io   Delete          Immediate
true                   26h
netapp-cvs-standard           csi.trident.netapp.io   Delete          Immediate
true                   26h
standard                      kubernetes.io/gce-pd    Delete          Immediate
true                   27h
```

You have two options for changing settings in your `values.yaml`. The first option is to open the file and edit it directly. The second option is to add an extra argument to your usual Helm CLI command.

To view and export `values.yaml`, use the `helm show` command:

```
helm show values bitnami/postgresql
```

or

```
helm show values bitnami/postgresql > my-values.yaml
```

This creates a `my-values.yaml` file in your local directory. That file is a copy of the official `values.yaml`.

Dry Run

Before deploying, you can do a dry run to make sure everything is set up correctly.

To do this, edit the values in the `my-values.yaml` file you created in the previous step. Test your deployment using the `-f my-values.yaml` and `--dry-run` flags:

```
helm install -f my-values.yaml --namespace testdb --generate-name bitnami/postgresql
--dry-run
```

If the output from our dry run looks correct, we may deploy to your cluster by removing `--dry-run`.

Before we can run the helm charts for real, we need to first create the namespace. We've chosen `testdb` and may use kubectl to create that namespace.

```
kubectl create namespace testdb && helm install -f my-values.yaml --namespace testdb
--generate-name bitnami/postgresql
```

After the Helm chart is deployed, it will be automatically detected by Astra, at which point you can register the app with Astra. Please note that for the Astra alpha program, installed applications can take up to 5 minutes to show up in the Discovered Applications list.

Generate test data

Helm provides instructions for connecting to newly-installed Postgres apps. These instructions should contain a few different methods for connecting to the database.

This process is also discussed here in the Postgres documentation.

```
NOTES:
** Please be patient while the chart is being deployed **
PostgreSQL can be accessed via port 5432 on the following DNS name from within your
cluster:
    postgresql-1591290927.longship.svc.cluster.local - Read/Write connection
To get the password for "postgres" run:
    export POSTGRES_PASSWORD=$(kubectl get secret --namespace longship postgresql-
1591290927 -o jsonpath="{.data.postgresql-password}" | base64 --decode)
To connect to your database run the following command:
    kubectl run postgresql-1591290927-client --rm --tty -i --restart='Never' --namespace
longship --image docker.io/bitnami/postgresql:11.8.0-debian-10-r19
--env="PGPASSWORD=$POSTGRES_PASSWORD" --command -- psql --host postgresql-1591290927 -U
postgres -d postgres -p 5432
To connect to your database from outside the cluster execute the following commands:
    kubectl port-forward --namespace longship svc/postgresql-1591290927 5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres -p 5432
```

From your own instructions, copy the line below `To get the password for "postgres" run:` and run it. Next, copy the lines below `To connect to your database run the following command:` and run them.

This will put you in the psql command line tool. Using psql, you may generate test data for testing Astra snapshot, clone, and restore features.

An example chunk of SQL that generates 10,000 rows is included in this guide.

```
-- create a db
CREATE DATABASE astra_test_db;
-- connect to it
\c astra_test_db;
-- create a table
CREATE TABLE junk(
  id      SERIAL PRIMARY KEY,
  title   VARCHAR(32) NOT NULL UNIQUE
);
-- insert 10,000 rows into the table
INSERT INTO junk (
    title
)
SELECT md5(i::text)
FROM generate_series(1, 10000) g_s(i);
-- check that data looks correct
SELECT * FROM junk LIMIT 20;
```