Solutions

Project Astra

NetApp
November 17, 2020

This PDF was generated from https://docs.netapp.com/us-en/project-astra/solutions/solutions/mariadb-deploy-from-helm-chart.html on November 17, 2020. Always check docs.netapp.com for the latest.

Table of Contents

Solutions

MySQL/MariaDB

Deploy MariaDB From a Helm Chart

Learn how to exercise the Astra Beta program workflow by deploying MariaDB from a Helm chart. After you deploy MariaDB on your cluster, you can manage the application with Astra.

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

The Astra Beta program only supports MySQL 0.3.22 and MariaDB 14.14.

System Requirements

In order to deploy MariaDB from a Helm chart for the Astra Beta program, you need the following:

- A GKE cluster which has been added to Astra.
- Updated versions of Helm (version 3.2+) and Kubectl installed.
- Kubeconfig configured using the gcloud tool with a command like `gcloud container clusters get-credentials my-cluster-name`

Install MariaDB

To exercise the Astra Beta program workflow, we recommend you use the Helm chart.

Deploy MariaDB with the command:

```
helm install mariadb bitnami/mariadb --namespace testdb --create-namespace --set
db.database=test_db,db.user=test_db_user,db.password=NKhjs2wQPt8 > /dev/null 2>&1
```

This does the following:

- Creates the `testdb` namespace.
- Deploys MariaDB on the `testdb` namespace.
- Creates a database named `test_db`
- Creates a user `test_db_user` with password `NKhjs2wQPt8`

This method of setting the password at deployment is insecure. Only use this command when setting up MariaDB for a sandbox deployment to use Astra Beta program. We do not recommend this for a production environment.

After the Helm chart is deployed, it will be automatically discovered by Astra, at which point you can manage the app with Astra.

Deploy MySQL From a Helm Chart

Learn how to exercise the Astra Beta workflow by deploying MySQL from a Helm chart. After you deploy MySQL on your Kubernetes cluster, you can manage the application with Astra.

MariaDB and MySQL are validated apps for the Astra Beta program. Learn the difference between Validated and Standard apps.

The Astra beta program only supports MySQL 0.3.22 and MariaDB 14.14.

System Requirements

In order to deploy MySQL from a Helm chart for the Astra Beta program, you need the following:

- A GKE cluster which has been added to Astra.
- Updated versions of Helm (version 3.2+) and Kubectl installed.
- Kubeconfig configured using the gcloud tool with a command like `gcloud container clusters get-credentials my-cluster-name`

You must deploy your app after the cluster is added to Astra, not before.

Install MySQL

To exercise the Astra Beta workflow, we recommend the standard stable chart.

You must deploy the Helm chart in a namespace other than the default.

Deploy MySQL with the command:

```
helm install mysql stable/mysql --namespace testdb--set
db.database=test_db,db.user=test_db_user,db.password=NKhjs2wQPt8
if you need to deploy mysql under a new namespace; please use the following command
helm install mysql stable/mysql --namespace testdb --create-namespace --set
db.database=test_db,db.user=test_db_user,db.password=NKhjs2wQPt8
```

This does the following:

- Creates the `testdb` namespace.
- Deploys MySQL on the `testdb` namespace.
- Creates a database named `test_db`

- Creates a user `test_db_user` with password `NKhjs2wQPt8`

This method of setting the password at deployment is insecure. You can use own secrets and config maps and passing along with helm command.

After the Helm chart is deployed, it will be automatically discovered by Astra, at which point you can manage the app with Astra.

Work With MySQL/MariaDB 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 MySQL/MariaDB on Astra, see Deploy MySQL/MariaDB from a Helm Chart.

MariaDB and MySQL are validated apps for the Astra Beta program. Learn the difference between Validated and Standard apps

MySQL 0.3.22 and MariaDB 14.14 are the only versions supported in the Astra Beta program.

Requirements

In order to deploy MySQL/MariaDB from a Helm chart on a cluster registered with the Astra Beta program, 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.

Cloud Volume Serivice in Google Cloud Platform (CVS-GCP)

CVS is 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.

MySQL/MariaDB Requirements

For a MySQL/MariaDB application, Astra 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).

Install MariaDB/MySQL

For the Astra beta, we recommend the custom Helm chart we have created for this purpose. For instructions on how to deploy from this custom chart, see Deploy MySQL/MariaDB 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 chart uses a cluster's default storage class. Kubernetes clusters registered with Astra beta use Trident CSI fron NetApp. Trident automatically sets CVS as the default storage class. 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:

```
# mariaDB
helm show values bitnami/mariadb
# mySQL
helm show values bitnami/mysql
```

or

```
# mariaDB
helm show values bitnami/mariadb > my-values.yaml
# mySQL
helm show values bitnami/mysql > 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:

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

# MySQL
helm install -f my-values.yaml --namespace testdb --generate-name bitnami/mysql --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, you can choose to use an existing namespace or specify to create a new namespace with helm command like below;

```
# MariaDB
helm install -f my-values.yaml --namespace testdb --generate-name bitnami/mariadb
--create-namespace

# MySQL
 helm install -f my-values.yaml --namespace testdb --generate-name bitnami/mysql --create
-namespace
```

After deploying the application using Helm chart Astra will be automatically discover the application. After a successful discovery you can manage the app with Astra.

Postgres

Deploy Postgres From a Helm Chart

Learn how to exercise the Astra beta program workflow by deploying Postgres from a Helm chart. After you deploy Postgres on your cluster, you can register the application with Astra.

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

> The Astra Beta Program only supports Postgres 11.7.

System Requirements

In order to deploy Postgres from a Helm chart for the Astra alpha program, you need the following:

- A fresh GKE cluster which has been added to Astra.
- Updated versions of Helm (version 3.2+) and Kubectl installed.
- Kubeconfig configured using the gcloud tool with a command like `gcloud container clusters get-credentials my-cluster-name`

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.

> You must deploy your app after the cluster is added to Astra, not before.

Install Postgres

To exercise the Astra alpha workflow, we recommend the standard stable chart.

> You must deploy the Helm chart in a namespace other than the default.

Deploy Postgres with the command:

```
kubectl create namespace testdb && helm install stable/postgresql --namespace testdb
--set postgresqlPassword=U9dH9HT4pWS,postgresqlDatabase=test_db --generate-name
```

This does the following:

- Creates the `testdb` namespace.

- Deploys Postgres on the `testdb` namespace.
- Creates a database named `test_db`
- Creates a user `test_db_user` with password `U9dH9HT4pWS`

This method of setting the password at deployment is insecure. Only use this command when setting up Postgres for a sandbox deployment to use Astra alpha program. We do not recommend this for a production environment.

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, it can take up to 5 minutes for applications to show up in the Discovered Applications list after being installed.

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;
```

Jenkins

Deploy Jenkins From a Helm Chart

Learn how to exercise the Astra beta program workflow by deploying Jenkins from a Helm chart. After you deploy Jenkins on your cluster, you can register the application with Astra.

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

Requirements

The following requirements are necessary for installing and running Jenkins on a Kubernetes cluster for the Astra beta program.

Compatibility Requirements

Only the current version of Jenkins (5.0.26) has been officially validated for use with Astra. Other versions may work, but may only run as a standard application.

Astra does not support the Kubernetes plugin for Jenkins at this time. This functionality will be added soon. You can run Jenkins in a Kubernetes cluster without the plugin. The plugin provides scalability to your Jenkins cluster.

System Requirements

- A new Kubernetes cluster which has been added to Astra.

- Helm (version 3.2+) and kubectl installed on your local computer.
- Kubeconfig configured using the gcloud tool with a command like `gcloud container clusters get-credentials my-cluster-name`

Namespace Requirements

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

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.

You must deploy your app after the cluster is added to Astra, not before.

Install Jenkins

To exercise the Astra beta workflow, we recommend you use the Bitnami Helm chart. Add the Bitnami chart repo:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

You must deploy the Helm chart in a namespace other than the default.

Create the `jenkins` namespace and deploy Jenkins into it with the command:

```
kubectl create namespace jenkins && helm install jenkins --namespace jenkins --set
persistence.storageClass=netapp-cvs-perf-premium,persistence.size=100Gi bitnami/jenkins
```

This does the following:

- Creates the `jenkins` namespace.
- Sets the correct storage class.
- Sets the persistent volume storage size to 100Gi.

After the Helm chart is deployed, it will be automatically detected by Astra. You can then register the app with Astra.

Copyright Information

Copyright © 2020 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.