Kubernetes Examples - Crunchy Containers for PostgreSQL

Crunchy Data Solutions, Inc.

Version 1.4.0, 2017-04-27

Kube Environment

Here are instructions for running examples on a pure kube cluster.

Note, some of the examples assume an NFS file system for creating persistent volumes. See the install.asciidoc for details on setting NFS permissions and the use of **supplementalGroups** within pod specs.

Installation

I recommend using kubeadm or minikube to try the examples out.

See the following links for installation instructions:

- https://github.com/kubernetes/minikube
- http://linoxide.com/containers/setup-kubernetes-kubeadm-centos/

Run Kube

Make sure your hostname resolves to a single IP address in your /etc/hosts file! If not, the NFS examples will not work.

You should see a single IP address returned from this command:

```
hostname --ip-address
```

```
sudo PATH=$PATH ALLOW_PRIVILEGED=true ./hack/local-up-cluster.sh
```

Note: specifying ALLOW_PRIVILEGED=true is required if you are running in selinux enforcing mode. This allows you to specify the following in your pod spec to run the container as priviledged:

```
"securityContext": {
    "privileged": true
},
```

Examples for the Kube Environment

The examples/kube directory containers examples for running the Crunchy containers in a kube environment.

The examples are explained below.

The namespace is set for the examples within the **examples/envvars.sh** script and defaults to **default**. Set this variable according to your project configuration.

basic - basic database container example

This example starts a single postgres container and service, the most simple of examples.

Running the example:

```
examples/kube/basic/run.sh
kubectl get pod basic
kubectl get service basic
kubectl logs basic
```

After the database starts up you can connect to it as follows:

```
psql -h basic -U postgres postgres
```

master-replica - master and replica databases

This example starts a master pod, master service, replica pod, and replica service. The replica is a replica of the master. This example uses emptyDir volumes for persistence. This example does not allow you to scale up the replicas.

Running the example:

```
examples/kube/master-replica/run.sh
```

It takes about a minute for the replica to begin replicating with the master. To test out replication, see if replication is underway with this command:

```
psql -h master -U postgres postgres -c 'table pg_stat_replication'
```

If you see a line returned from that query it means the master is replicating to the replica. Try creating some data on the master:

```
psql -h master -U postgres postgres -c 'create table foo (id int)'
psql -h master -U postgres postgres -c 'insert into foo values (1)'
```

Then verify that the data is replicated to the replica:

```
psql -h replica -U postgres postgres -c 'table foo'
```

master-replica-dc - master and scaling replica example

This example starts a master pod, master service, replica pod, and replica service. The replica is a replica of the master. This example uses emptyDir volumes for persistence. This example runs the replicas in a Deployment. A deployment controller lets you scale up the replicas and create an initial replica set.

Running the example:

```
examples/kube/master-replica-dc/run.sh
```

You can insert data in the master and make sure it replicates to the replicas using the commands from Example 2 above. Replace **master** with the **master-dc** name and **replica** with **replica-dc**.

This example creates 2 replicas when it initially starts. To scale up the number of replicas, run this command:

```
kubectl get deployment
kubectl scale --current-replicas=2 --replicas=3 deployment/replica-dc
kubectl get deployment
kubectl get pod
```

You can verify that you now have 3 replicas by running this query on the master:

```
psql -h master-dc -U postgres postgres -c 'table pg_stat_replication'
```

master-replica-rc - master and scaling replica example

This example starts a master pod, master service, replica pod, and replica service. The replica is a replica of the master. This example uses emptyDir volumes for persistence. This example runs the replicas in a Replication Controller. A replication controller lets you scale up the replicas and create an initial replica set. Deployments will likely be the preferred way to create a replica set going forward but I wanted to provide an example for thoroughness.

Running the example:

```
examples/kube/master-replica-rc/run.sh
```

You can also scale up the number of replicas using this replication controller mechanism. The command to scale up is as follows:

```
kubectl get rc
kubectl scale rc replica-rc --replicas=3
kubectl get pod
```

backup-job - backup job example

This example depends on the basic example being run prior to this example!

This example performs a database backup on the basic database. The backup is stored in the /nfsfileshare backup path which is also a dependency. See the installation docs on how to set up the NFS server on this host.

Running the example:

```
examples/kube/backup-job/run.sh
```

Things to point out with this example include its use of persistent volumes and volume claims to store the backup data files to an NFS server.

You can view the persistent volume information as follows:

```
kubectl get pvc
kubectl get pv
```

The Kube Job type executes a pod and then the pod exits. You can view the Job status using this command:

```
kubectl get job
```

While the backup pod is running, you can view the pod as follows:

```
kubectl get pod
```

You should find the backup archive in this location:

```
ls /nfsfileshare/basic
```

Tip

You can view the backup pod log using the **docker logs** command on the exited container. Use **docker ps -a | grep backup** to locate the container.

badger - pgbadger container example

This example runs a pod that includes a database container and a pgbadger container. A service is also created for the pod.

Running the example:

```
examples/kube/badger/run.sh
```

You can access pgbadger at:

```
curl http://badger:10000/api/badgergenerate
```

Tips

You can view the database container logs using this command:

```
kubectl logs -c server badger
```

metrics - postgres metrics backend

This example starts up prometheus, grafana, and prometheus gateway.

It is required to view or capture metrics collected by crunchy-collect.

Running the example:

```
examples/kube/metrics/run.sh
```

This will start up 3 containers and services:

- prometheus (http://crunchy-prometheus:9090)
- prometheus gateway (http://crunchy-promgateway:9091)
- grafana (http://crunchy-grafana:3000)

If you want your metrics and dashboards to persist to NFS, run this script:

```
examples/kube/metrics/run-pvc.sh
```

In the docs folder of the github repo, check out the metrics.asciidoc for details on the exact metrics being collected.

collect - metrics collection container example

This example assumes you have run the metrics example which starts up prometheus, grafana, and prometheus gateway.

This example runs a pod that includes a database container and a metrics collection container. A service is also created for the pod.

Running the example:

```
examples/kube/collect/run.sh
```

You can view the collect container logs using this command:

```
kubectl logs -c collect master-collect
```

You can access the database or drive load against it using this command:

```
psql -h master-collect -U postgres postgres
```

vacuum-job - vacuum job example

This example assumes you have run the basic example prior to this example!

This example runs a Job which performs a SQL VACUUM on a particular table (testtable) in the basic database instance.

Running the example:

```
examples/kube/vacuum-job/run.sh
```

Verify the job completed:

```
kubectl get job
```

Look at the docker log of the vacuum job's pod:

```
docker logs $(docker ps -a | grep crunchy-vacuum | cut -f 1 -d' ')
```

crunchy-proxy - crunchy-proxy pod example

This example assumes you have run the master-replica example prior to this example!

This example runs a crunchy-proxy pod that creates a special purpose proxy to a postgres cluster (master and replica).

crunchy-proxy offers a high performance alternative to pgbouncer and pgpool.

The proxy example copies a configuration file to the PV_PATH and starts up the **crunchy-proxy** within a Deployment.

If you run the example in minikube, you will need to manually copy the crunchy-proxy-config.json file to a file on the minikube named **/data/config.json**.

The proxy reads the configuration file from a **/config** volume mount and begins execution.

Start by running the proxy container:

```
cd $CCPROOT/examples/kube/crunchy-proxy
./run.sh
```

The proxy will listen on port 5432 as specified in the configuration file. The example creates a Service named **crunchy-proxy** that you can use to access the configured PostgreSQL backend containers from the **master-replica** example.

See the following link for more information on the **crunchy-proxy**:

https://github.com/CrunchyData/crunchy-proxy

Test the proxy by running psql commands via the proxy connection:

```
psql -h crunchy-proxy -U postgres postgres
```

SQL "reads" will be sent to the PostgreSQL replica database if your SQL includes the **crunchy-proxy** read annotation. SQL statements that do not include the read annotation will be sent to the master database container within the PostgreSQL cluster.

pgpool - pgpool pod example

This example assumes you have run the master-replica example prior to this example!

This example runs a pgpool pod that creates a special purpose proxy to a postgres cluster (master and replica).

Running the example:

```
examples/kube/pgpool/run.sh
```

The example is configured to allow the **testuser** to connect to the **userdb** database as follows:

```
psql -h pgpool -U testuser userdb
```

master-restore - database restore from backup example

This example assumes you have run the backup-job example prior to this example!

You will need to find a backup you want to use for running this example, you will need the timestamped directory path under /nfsfileshare/basic/. Edit the master-restore.json file and update the BACKUP_PATH setting to specify the NFS backup path you want to restore with, example:

```
"name": "BACKUP_PATH",
"value": "basic/2016-05-27-14-35-33"
```

This example runs a postgres container passing in the backup location. The startup of the container will use rsync to copy the backup data to this new container, and then launch postgres which will use the backup data to startup with.

Running the example:

```
examples/kube/master-restore/run.sh
```

Test the restored database as follows:

```
psql -h restored-master -U postgres postgres
```

watch - automated failover watcher example

This example assumes you have run the master-replica example prior to this example!

This example runs a crunchy-watch container to look for the master within a postgres cluster, if it can not find the master it will proceed to cause a failover to a replica.

Running the example:

```
examples/kube/watch/run.sh
```

Check out the log of the watch container as follows:

```
kubectl log watch
```

Then trigger a failover using this command:

```
kubectl delete pod master
```

Resume watching the watch container's log and verify that it detects the master is not reachable and performs a failover on the replica.

A final test is to see if the old replica is now a fully functioning master by inserting some test data into it as follows:

```
psql -h master -U postgres postgres -c 'create table failtest (id int)'
```

The above command still works because the watch container has changed the labels of the replica to make it a master, so the master service will still work and route now to the new master even though the pod is named replica.

Tip

You can view the labels on a pod with this command:

```
kubectl describe pod replica | grep Label
```

pgbouncer

This example assumes you have run the master-replica example prior to this example!

This example runs a crunchy-pgbouncer container to look for the master within a postgres cluster, if it can not find the master it will proceed to cause a failover to a replica. It will also configure a pgbouncer container that sets up a connection pool to the configured master and replica.

Running the example:

```
examples/kube/pgbouncer/run.sh
```

Connect to the **master** and **replica** databases as follows:

```
psql -h pgbouncer -U postgres master
psql -h pgbouncer -U postgres replica
```

The names **master** and **replica** are pgbouncer configured names and don't necessarily have to match the database name in the actual Postgres instance.

View the pgbouncer log as follows:

```
kubectl log pgbouncer
```

Next, test the failover capability within the crunchy-watch container using the following:

```
kubectl delete pod master
```

Take another look at the pgbouncer log and you will see it trigger the failover to the replica pod. After this failover you should be able to execute the command:

```
psql -h pgbouncer -U postgres master
```

synchronous replica

This example deploys a PostgreSQL cluster with a master, a synchronous replica, and an asynchronous replica. The two replicas share the same Service.

Running the example:

```
examples/kube/sync/run.sh
```

Connect to the **mastersync** and **replicasync** databases as follows:

```
psql -h mastersync -U postgres postgres -c 'create table mister (id int)'
psql -h mastersync -U postgres postgres -c 'insert into mister values (1)'
psql -h mastersync -U postgres postgres -c 'table pg_stat_replication'
psql -h replicasync -U postgres postgres -c 'select inet_server_addr(), * from mister'
psql -h replicasync -U postgres postgres -c 'select inet_server_addr(), * from mister'
psql -h replicasync -U postgres postgres -c 'select inet_server_addr(), * from mister'
```

This set of queries will show you the IP address of the Postgres replica container, notice it changes because of the round-robin Service proxy we are using for both replicas. The example queries also show that both replicas are replicating from the master.

pgadmin4

This example deploys the pgadmin4 (beta4) web user interface for Postgresql.

Start the container as follows:

```
cd $CCPROOT/examples/kube/pgadmin4
./run.sh
```

This will start a container and service for pgadmin4. You can browse the user interface at http://pgadmin4.default.svc.cluster.local:5050

See the pgadmin4 documentation for more details at http://pgadmin.org

The example uses pgadmin4 configuration files which are mounted at an NFS mount point, this NFS data directory is mounted into the container and used by the pgadmin4 application to persist metadata.

statefulsets (only for kube 1.5 and greater)

This example deploys a statefulset named **pgset**. The statefulset is a new feature in Kubernetes as of version 1.5. Statefulsets have replaced PetSets going forward.

This example creates 2 Postgres containers to form the set. At startup, each container will examine its hostname to determine if it is the first container within the set of containers.

The first container is determined by the hostname suffix assigned by Kube to the pod. This is an ordinal value starting with **0**.

If a container sees that it has an ordinal value of **0**, it will update the container labels to add a new label of:

```
name=$PG_MASTER_HOST
```

In this example, PG_MASTER_HOST is specified as **pgset-master**.

By default, the containers specify a value of **name=pgset-replica**

There are 2 services that end user applications will use to access the PostgreSQL cluster, one service (pgset-master) routes to the master container and the other (pgset-replica) to the replica containers.

```
$ kubectl get service
NAME            CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes      10.96.0.1       <none>        443/TCP    22h
pgset           None            <none>        5432/TCP   1h
pgset-master    10.97.168.138   <none>        5432/TCP   1h
pgset-replica   10.97.218.221   <none>        5432/TCP   1h
```

Start the example as follows:

```
cd $CCPROOT/examples/kube/statefulset
./run.sh
```

You can access the master database as follows:

```
psql -h pgset-master -U postgres postgres
```

You can access the replica databases as follows:

```
psql -h pgset-replica -U postgres postgres
```

You can scale the number of containers using this command, this will essentially create an additional replica databse:

```
kubectl scale pgset --replica=3
```

PITR (point in time recovery)

This example is identical to the openshift PITR example, see the Openshift Documentation Example 20 for details on how the PITR example works.

The only differences are the following:

- paths are **examples/kube/pitr**
- JSON and scripts are modifed to work with Kube
- **kubectl** commands are used instead of **oc** commands
- database services resolve to **default.svc.cluster.local** instead of **openshift.svc.cluster.local**

See PITR Documentation for details on PITR concepts and how PITR is implemented within the Suite.

pgbackrest

Starting in release 1.2.5, the pgbackrest utility has been added to the crunchy-postgres container. See the pgbackrest Documentation for details on how this feature works within the container suite.

Start the example as follows:

```
cd $CCPROOT/examples/kube/backrest
./run.sh
```

This will create the following in your Kube environment:

- A configMap named backrestconf which contains the pgbackrest.conf file
- master-backrest pod with pgbackrest archive enabled. An initial stanza db will be created on initialization
- master-backrest service

The crunchy-pvc will be used for /pgdata, and crunchy-pvc2 for the /backrestrepo. Examine the /backrestrepo location to view the archive directory and ensure WAL archiving is working. See backrest.asciidoc for steps to backup and restore using pgbackrest.

backrest restore

This assumes you have run the pgbackrest example above. There are two options to choose from when performing a restore, DELTA and FULL. A FULL is the default; a DELTA will only occur if the environment variable DELTA is specified in the restore-job spec. Consult the pgbackrest user guide to determine which is best suited to run.

Steps for FULL restore

- Delete the master-backrest pod, if still running
- Empty the PGDATA directory (remove all files)
- Navigate to the backrest_restore examples directory. Execute the full-restore.sh script.
- Check the restore logs (db-restore.log) in the /backrestrepo mountpointfor success. You can also view the logs of the completed job pod with kubectl get pod -a
- Re-create the master-backrest pod in the backrest examples directory. The database will recover.

Steps for DELTA restore

- Delete the master-backrest pod, if still running
- rm postmaster.pid from PGDATA.
- Navigate to the backrest_restore examples directory. Execute the delta-restore.sh script.
- Check the restore logs (db-restore.log) in the /backrestrepo mountpointfor success. You can also view the logs of the completed job pod with kubectl get pod -a
- Re-create the master-backrest pod in the backrest examples directory. The database will recover only files that have changed from the last backup.

master-deployment

Starting in release 1.2.8, the postgres container can accept an environment variable named PGDATA_PATH_OVERRIDE. If set, the /pgdata/subdir path will use a path subdir name of your choosing instead of the default which is the hostname of the container.

This example shows how a Deployment of a master postgres is supported. A pod is a deployment uses a hostname generated by Kubernetes, so if you want to restart the master pod, you will get a different hostname as defined by the Deployment. For finding the /pgdata that pertains to the pod,

you will need to specify a /pgdata/subdir name that never changes, and that is the purpose of the PGDATA_PATH_OVERRIDE env var.

Start the example as follows:

```
cd $CCPROOT/examples/kube/master-deployment
./run.sh
```

This will create the following in your Kube environment:

- create a master-dc service, uses a PVC to persist postgres data
- create a replica-dc service, uses emptyDir persistence
- create a master-dc Deployment of replica count 1 for the master postgres database pod
- create a replica-dc Deployment of replica count 2 for the replica(s)

The persisted master postgres data is found under /pgdata/master-dc. If you delete the master pod, the Deployment will create another pod for the master, and will be able to start up immediately since we are using the same /pgdata/master-dc data directory.

upgrade

Starting in release 1.3.1, the upgrade container will let you perform a pg_upgrade on a 9.5 database converting its data to a 9.6 version.

This example assumes you have run **master-pvc** using a 9.5 image such as **centos7-9.5-1.4.0** prior to running this upgrade.

Prior to starting this example, shut down the **master-pvc** database using the **examples/kube/master-pvc/cleanup.sh** script.

Prior to running this example, make sure your CCP_IMAGE_TAG environment variable is using a 9.6 image such as **centos7-9.6-1.4.0**.

Start the upgrade as follows:

```
cd $CCPROOT/examples/kube/upgrade
./run.sh
```

This will create the following in your Kube environment:

- a Kube Job running the **crunchy-upgrade** container
- a new data directory name **master-upgrade** found in the **pgnewdata** PVC

If successful, the Job will end with a Successful status, verify the results of the Job by examining the Job's pod log:

```
kubectl get pod -a -l job-name=upgrade-job
kubectl logs -l job-name=upgrade-job
```

You can verify the upgraded database by running the **examples/kube/master-upgrade** example, this example will mount the newly created and upgraded database files. Database tables and data that were in the **master-pvc** test database should be found in the **master-upgrade** database.

Tip 1

Create a static route from your host to 10.0.0.0/16 if you want to test the user interfaces of the metrics tools.

On my host, 114, and my bridge, br1, this worked for me:

```
ip route add 10.0.0.0/16 via 192.168.0.114 dev br1
```

Legal Notices