Kubernetes Examples - Crunchy Containers for PostgreSQL

Crunchy Data Solutions, Inc.

Version 1.2.3, 2016-09-27

Kube Environment

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

Installation

Some steps to follow:

Install Kube 1.2.4 by source on a Centos 7 VM.

```
sudo yum -y install etcd-2.2.5
git clone https://github.com/kubernetes/kubernetes.git
cd kubernetes
git checkout v1.2.4
cd hack
./build-go.sh
```

Setup DNS

edit hack/local-up-cluster.sh

```
ENABLE_CLUSTER_DNS=true
DNS_SERVER_IP="10.0.0.10"
DNS_DOMAIN="cluster.local"
```

edit /etc/resolv.conf or configure your network settings to add the DNS server

```
search default.svc.cluster.local crunchy.lab
nameserver 10.0.0.10
```

Starting with Kube 1.3.4, I had to add these into the local-config-up.sh script in order to get DNS to work:

```
export KUBERNETES_PROVIDER=local
export API_HOST=<<docker0 ip address>>
```

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
},
```

Old Notes Not Used!!

- kubectl.sh create ns kube-system
- install the skydns for kube, see kube docs and skydns.yaml example
- to get emptyDir permissions to work for Deployments, I had to remove the SecurityContextDeny admission from the hack/local-cluster-up.sh script then use fsGroup set to 26

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.

Kube Example 1 - single-master

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

Running the example:

```
examples/kube/single-master/run.sh
kubectl get pod single-master
kubectl get service single-master
kubectl logs single-master
```

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

```
psql -h single-master -U postgres postgres
```

Kube Example 2 - master-slave

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

Running the example:

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

It takes about a minute for the slave 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 slave. 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 slave:

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

Kube Example 3 - master-slave-dc

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

Running the example:

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

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

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

```
kubectl get deployment
kubectl scale --current-replicas=2 --replicas=3 deployment/slave-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'
```

Kube Example 4 - master-slave-rc

This example starts a master pod, master service, slave pod, and slave service. The slave is a replica of the master. This example uses emptyDir volumes for persistence. This example runs the slaves in a Replication Controller. A replication controller lets you scale up the slaves 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 completness sake.

Running the example:

```
examples/kube/master-slave-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 slave-rc --replicas=3
kubectl get pod
```

Kube Example 5 - backup-job

This example depends on the single-master example be run prior to this example!

This example performs a database backup on the single-master 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-nfs/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/single-master
```

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.

Kube Example 6 - badger

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
```

Kube Example 6 - metrics

This examples 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-nfs.sh
```

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

Kube Example 7 - collect

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
```

Kube Example 8 - vacuum-job

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

This example runs a Job which performs a SQL VACUUM on a particular table (testtable) in the single-master 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' ')
```

Kube Example 9 - pgpool

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

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

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
```

Kube Example 10 - master-restore

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/single-master/. 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": "single-master/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
```

Kube Example 11 - watch

This example assumes you have run the master-slave 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 slave.

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 slave.

A final test is to see if the old slave 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 slave 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 slave.

Tip

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

```
kubectl describe pod slave | grep Label
```

Kube Example 11 - pgbouncer

This example assumes you have run the master-slave 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 slave. It will also configure a pgbouncer container that sets up a connection pool to the configured master and slave.

Running the example:

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

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

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

The names **master** and **slave** 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 slave pod. After this failover you should be able to execute the command:

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

Kube Example 12 - synchrounous slave

This example deploys a PostgreSQL cluster with a master, a synchrounous slave, and an asynchronous slave. The two slaves share the same Service.

Running the example:

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

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

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

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

Kube Example 13 - kitchensink

This example deploys many of the components all in a single example to demonstrate a more complex overall deployment examples. This examples includes the following objects: * master database service (kitchensink-master) * replica database service (kitchensink-slave) * pgpool database service (kitchensink-pgpool) * master database pod (kitchensink-master) * metrics collection container (kitchensink-master) * pgbadger container (kitchensink-master) * async replica database Deployment (kitchensink-slave-dc-XXXXX) * sync rdatabase pod (kitchensink-sync-slave) * pgpool Replication Controller (kitchensink-pgpool-XXXXX) * watch pod (kitchensink-watch)

Running the example:

```
examples/kube/kitchensink/run.sh
```

The master database pod has the following containers running inside it: * server (postgres container) * pgbadger (pgbadger container) * collect (metrics collection container)

You can scale up the number of async slaves as follows:

```
kubectl get deployment
kubectl scale --current-replicas=1 --replicas=2 deployment/kitchensink-slave-dc
kubectl get deployment
```

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

```
psql -h kitchensink-master -U postgres postgres -c 'table pg_stat_replication'
psql -h kitchensink-master -U testuser userdb -c 'create table mister (id int)'
psql -h kitchensink-master -U testuser userdb -c 'insert into mister values (12)'
psql -h kitchensink-slave -U testuser userdb -c 'table mister'
psql -h kitchensink-pgpool -U testuser userdb -c 'table mister'
psql -h kitchensink-master -U testuser userdb -c 'insert into mister values (112)'
```

Kube Example 14 - pgadmin4

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

Start the container as follows:

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

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

See the pgadmin4 documentation for more details at link: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.

Kube Example 15 - master using gluster fs

This example deploys a master database container that uses a gluster file system as the persistent volume.

Setup gluster according to link:https://wiki.centos.org/SpecialInterestGroup/Storage/gluster-Quickstart

Start the example as follows:

```
cd $BUILDBASE/examples/kube/gluster
./run.sh
```

This will start a container and service for the master database.

You can access the master database as follows:

```
psql -h master-gluster -U postgres postgres
```

This example has a mount point of /mnt/gluster which is mapped to the gluster fs at yourhost:/gv0

Kube Example 15 - petsets (only for kube 1.3 and greater)

This example deploys a master database container and 2 slave containers. The slaves are deployed using a PetSet. The Petset references a pre-provisioned persistent volume claim created using NFS.

Start the example as follows:

```
cd $BUILDBASE/examples/kube/petset
./run.sh
```

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

You can access the master database as follows:

```
psql -h psmaster.default.svc.cluster.local -U postgres postgres
```

You can access the slave databases as follows:

```
psql -h slave-0.psslave.default.svc.cluster.local -U postgres postgres
psql -h slave-1.psslave.default.svc.cluster.local -U postgres postgres
```

You can scale the petset using this command:

```
kubectl patch petset slave -p '{"spec":{"replicas":3}}'
```

Kube Example 16 - 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.

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