Crunchy Containers

Crunchy Data Solutions, Inc.

Version 1.1.1, 2016-04-11

Standalone Examples

We provide some examples of running the Crunchy containers in the examples/standalone directory. Those examples are explained below.

Example 1 - Running a single standalone database

Create the container with this command:

```
./run-pg.sh
```

This script will do the following:

- start up a container named crunchy-pg

- create a data directory for the database in /tmp/crunchy-pg

- initialize the database using the passed in environment variables, building a user, database, schema, and table based on the environment variable values.

- maps the PostgreSQL port of 5432 in the container to your local host port of 12000.

The container creates a default database called 'testdb', a default user called 'testuser' with a default password of 'testpsw', you can use this to connect from your local host as follows:

```
psql -h localhost -p 12000 -U testuser -W testdb
```

To shut down the instance, run the following commands:

```
docker stop crunchy-pg
```

To start the instance, run the following commands:

```
docker start crunchy-pg
```

Example 2 - Creating a master database

Run the container with this command:

```
./run-pg-master.sh
```

This script will do the following:

- start up a container named master

- create a data directory for the database in /tmp/master-data
- initialize the database using the passed in environment variables, building a user, database, schema, and table based on the environment variable values.
- maps the PostgreSQL port of 5432 in the container to your local host port of 12000.

The container creates a default database called 'testdb', a default user called 'testuser' with a default password of 'testpsw', you can use this to connect from your local host as follows:

```
psql -h localhost -p 12000 -U testuser -W testdb
```

To shut down the instance, run the following commands:

```
docker stop master
```

To start the instance, run the following commands:

```
docker start master
```

Example 3 - Performing a backup

In order to run this backup script, you first need to edit run-backup.sh to specify your host IP address you are running on. The script assumes you are going to backup the container created in Example 2.

Run the backup with this command:

```
./run-backup.sh
```

This script will do the following:

- start up a backup container named masterbackup
- run pg_basebackup on the container named master
- store the backup in /tmp/backups/master directory
- exit after the backup

Example 4 - Performing a restore

In order to run this backup script, you first need to edit run-restore.sh to specify your host IP address you are running on. The script assumes you are going to backup the container created in Example 2.

Run the backup with this command:

```
./run-restore.sh
```

This script will do the following:

- start up a container named masterrestore
- copy the backup files from Example 3 into /pgdata
- start up the container using the backup files
- maps the PostgreSQL port of 5432 in the container to your local host port of 12001 as to not conflict with the master running in the previous example.

Example 5 - Running PostgreSQL in Master-Slave Configuration

The container can be passed environment variables that will cause it to assume a PostgreSQL replication configuration with a master instance streaming to a slave replica instance.

The following env variables are specified for this configuration option:

- PG_MASTER_USER - The username used for master-slave replication value=master
- PG_MASTER_PASSWORD - The password for the PG master user
- PG_USER - The username that clients will use to connect to PG server value=user
- PG_PASSWORD - The password for the PG master user
- PG_DATABASE - The name of the database that will be created value=userdb
- PG_ROOT_PASSWORD - The password for the PG admin

This examples assumes you have run Example 1, and that the master container is running.

For running the master-slave configuration , you can run the following scripts:

```
run-pg-replica.sh
```

You can verify that replication is working by connecting to the replica as follows:

```
psql -h 127.0.0.1 -p 12002 -U postgres postgres
```

If you have created tables or data in the master database, they should show up in this replicated copy of that database.

Example 6 - pgpool

A pgpool example is provided that will run a pgpool container that is configured to be used with the master and slave example provided in the run-pg-master.sh and run-pg-replica.sh scripts. After

running those commands to create a master and replica, you can create a pgpool container by running the following example command:

```
sudo ./run-pgpool.sh
```

Enter the following command to connect to the pgpool that is mapped to your local port 12002, in this case the host is named jeffded.crunchy.lab, fill in your hostname instead:

```
psql -h jeffded.crunchy.lab -U testuser -p 12002 testdb
```

You will enter the password of testpsw when prompted. At this point you can execute both INSERT and SELECT statements on the pgpool connection. Pgpool will direct INSERT statements to the master and SELECT statements will be sent round-robin to both master and replica.

Example 7 - pgbadger

A pgbadger example is provided that will run a HTTP server that when invoked, will generate a pgbadger report on a given database.

pgbadger reads the log files from a database to product an HTML report that shows various Postgres statistics and graphs.

To run the example, modify the run-badger.sh script to refer to the Docker container that you want to run pgbadger against, also referring to the container's data directory, then run the example as follows:

```
sudo ./run-badger.sh
```

After execution, the container will run and provide a simple HTTP command you can browse to view the report. As you run queries against the database, you can invoke this URL to generate updated reports:

```
curl http://127.0.0.1:14000/api/badgergenerate
```

Example 8 - dns

Some users will need or want a DNS name to resolve to their container names. The crunchy-dns container provides the following: * listens to a Docker URL or socket for events that would indicate a container is created or destroyed * implements the consul.io DNS server * registers new container information into the DNS server * deregisters container information from the DNS serve when a container is destroyed

Start the crunchy-dns server by running its container as follows:

```
sudo ./examples/standalone/run-dnsbridge.sh
```

This is a privledged container and will bind to your local IP address at port 53. At this point, you can now start a new Postgres container and you should be able to do a DNS lookup as follows:

```
dig @192.168.122.138 containername.service.dc1.crunchy.lab
```

In this example, the local IP address of the DNS container is 192.168.122.138 and it assumes you have started a container named containername

You can alter the DNS domain name within the startup script if desired.

You can also browse the consul web UI at:

```
http://<your ip address>:8500/ui
```

Example 9 - metrics collection

You can collect various Postgres metrics from your database container by running a crunchy-collect container that points to your database container.

Metrics collection requires you run the crunchy 'scope' set of containers that includes: * prometheus * prometheus push gateway * grafana

To start this set of containers, run the following:

```
sudo ./examples/standalone/run-scope.sh
```

These metrics are described in the This document document.

An example has been provided that runs a database container and also the associated metrics collection container, run the example as follows:

```
sudo ./examples/standalone/run-collect.sh
```

Every 3 minutes the collection container will collect postgres metrics and push them to the crunchy prometheus database. You can graph them using the crunchy grafana container.

Legal Notices