User Guide - Crunchy Proxy

Crunchy Data Solutions, Inc.

Version 0.0.1, 2016-11-28

image::crunchy_logo.png

Project

crunchy-proxy is an experimental PostgreSQL-aware proxy used to intelligently handle PostgreSQL application requests. In the diagram below, a PostgreSQL client application would connect to the proxy which appears as any other PostgreSQL database connection. The proxy would accept the inbound client requests, and route them to an appropriate PostgreSQL cluster member.

image::proxy-golang.png

To the PostgreSQL server, the proxy is transparent and appears as any other PostgreSQL client would appear. Within the proxy, various features are found including:

- ability to configure a proxy configuration via a JSON configuration file
- ability to perform a PostgreSQL healthcheck on each configured PostgreSQL server instance
- ability to route inbound client messages based on the health of PostgreSQL
- ability to route messages based on the SQL command type, writes are sent to a master and reads are sent to replicas in a round-robin fashion
- ability to provide a REST admin interface
- ability to publish healthcheck events to consumers outside of the proxy

Installation Instructions

Setting up a golang project requires the following directory structure be created:

```
sudo yum -y install golang
mkdir -p gdev/src gdev/pkg gdev/bin
export GOPATH=$HOME/gdev;export GOBIN=$GOPATH/bin
export PATH=$GOPATH:$PATH
cd gdev/src
go get github.com/tools/godep
mkdir github.com/crunchydata
cd github.com/crunchydata
git clone git@github.com:CrunchyData/crunchy-proxy.git
cd crunchy-proxy
cd proxy
godep restore
```

Design

The example shows a message traveling down this path:

pg client→proxy→pg server→proxy→pg client

Lets describe each component and show how to run the test…

Postgres Cluster

Start up a PostgreSQL cluster (master and replica) to test with. This is done by running a pair of PostgreSQL containers that form a replicating cluster.

This requires you install Docker on your dev box! You can do this on centos7 with this:

```
sudo yum -y install docker
sudo group add docker
sudo usermod -a -g docker youruserid
sudo systemctl enable docker.service
sudo systemctl start docker.service
exit
```

After logging back into a new terminal, you should be able to perform the following:

```
cd proxy/pgcontainer
./run-cluster.sh
docker ps
```

If all has worked, you should see 2 Docker containers running, these make up a 2 node PostgreSQL cluster you will test against.

This will start a Postgres 9.5 master container that listens on **localhost:12000** and a replica container that listens on **localhost:12002**

The PostgreSQL user id is **postgres**, the password is **password**, and you would connect to these container database like this using psql:

```
psql -h 127.0.0.1 -p 12000 -U postgres postgres
psql -h 127.0.0.1 -p 12002 -U postgres postgres
```

Stop the containers like this:

```
docker stop master
docker stop replica
```

Start the containers like this:

```
docker start master
docker start replica
```

Proxy Component

There is a proxy component created when you run the **main.go** code. This component accepts PostgreSQL client connections and routes client messages to one of the PostgreSQL containers.

Start it like this:

```
cd proxy
go run main.go -config=config.json
```

This will do the following:

- start an admin service on localhost:10000
- listen on localhost:5432 for client requests
- read config.json and set up a runtime configuration
- route any client messages to the PostgreSQL containers

PG Client Component

There is a test PostgreSQL client program created when you run the **testclient.go** code. This is a client that sends messages to the proxy and will print out the responses from the proxy.

Run the test pg client like this:

```
cd testclient
./run-test-proxy.sh
```

Proxy Administration

There is an administration port created by the proxy that you can interact with to gain status from the proxy.

Events

Events like a healthcheck status are published to any subscribers using a streaming REST API, you can access the admin events as follows:

```
curl -i http://localhost:10000/api/stream
```

As the proxy publishes events, your REST client (e.g. curl) will receive the events.

Configuration

You can get the current configuration of the proxy as follows:

```
curl http://localhost:10000/api/config
```

Statistics

You can get the current statistics of the proxy as follows:

```
curl http://localhost:10000/api/stats
```

Configuration

Configuration of the proxy is determined by a JSON configuration file that is input to the proxy. The configuration file is read at startup of the proxy.

The structures defined in **config/config.go** define the content of the JSON configuration.

Legal Notices