User Guide - Crunchy Proxy

Crunchy Data Solutions, Inc.

Version 0.0.1, 2017-01-22

Releases

Users can download a compiled version of **crunchy-proxy** from the github repo site in the Releases tab.

A Docker image is also found in the DockerHub at https://hub.docker.com/r/crunchydata/crunchy-proxy/

Command Usage

The command syntax of the proxy is as follows:

```
crunchyproxy -config=config.json
```

The only command option is the **-config** flag. This option specifies the proxy configuration to use. A sample configuration file is located at https://github.com/crunchydata/crunchy-proxy/config.json

The proxy uses glog for logging, to see the log output, run the proxy with this command:

```
crunchyproxy -config=config.json -logtostderr=true
```

Increase the logging level this way:

```
crunchyproxy -config=config.json -logtostderr=true -v 2
```

Normal execution does not produce any logging:

```
crunchyproxy -config=config.json
```

Configuration

The proxy configuration is controlled by a single configuration file which is written in JSON format.

The JSON file is read at startup and is currently not reloaded after execution starts.

The JSON parameters include:

Parameter	Purpose	Example
Name	a name you want to give the configuration	"myconfig"
HostPort	the proxy host:port to listen to	"localhost:5432"

Parameter	Purpose	Example
AdminHostPort	the proxy admin host:port	"localhost:10000"
ReadAnnotation	the string to use for a read annotation	"read" is the default if not specified
StartAnnotation	the string to use for a start annotation	"start" is the default if not specified
FinishAnnotation	the string to use for a finish annotation	"finish" is the default if not specified
heathcheck.delay	seconds to delay between checks	60
heathcheck.query	SQL to use for check	"select now()"
pool.enabled	enable connection pooling	true
pool.capacity	size of pools	2
credentials.username	postgres username for pool connections	"postgres"
credentials.password	postgres password for pool connections	"password"
credentials.database	postgres database for pool connections	"postgres"
adapters	not implemented yet	"logging"
Master.HostPort	the master backend hostname:port	"127.0.0.1:12000"
Replicas.HostPort	the replica backend hostname:port	"127.0.0.1:12002"

Testing

A test script is provided that will run a PostgreSQL cluster, with a single master and replica. Run the database script as follows:

```
export CCP_IMAGE_TAG=centos7-9.5-1.2.6
bin/run-cluster.sh
```

This will start two docker containers that execute the PostgreSQL cluster.

The Postgres 9.5 master container listens on **localhost:12000** and a replica container 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
```

Test Execution

Start the **crunchy-proxy** like this:

```
go run crunchyproxy.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

Benchmark

For some simple benchmark results, run some tests using the **crunchy-proxy**:

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

You can also run the **psql** command against the proxy as a test client.

Overhead

Overhead of the proxy was measured and shows the following for the typical case of handling a SQL statement:

Test	Proxy	No-Proxy	Overhead
Single SQL Statement	2.240026ms	2.085424ms	+0.154602ms

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.

Current 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
```

Compiling the Source

If you are a developer and want to build the proxy from source code, follow these steps...

Assuming an installation directory of **$HOME/gdev**, follow the following steps to build **crunchy-proxy** from source:

```
mkdir -p $HOME/gdev/src $HOME/gdev/pkg $HOME/gdev/bin
export GOPATH=$HOME/gdev;export GOBIN=$GOPATH/bin;export PATH=$PATH:$GOBIN
export BUILDBASE=$GOPATH/github.com/crunchydata/crunchy-proxy
```

First, install a golang compiler. As an example, on centos7:

```
sudo yum -y install golang
```

Next, pull the source code as follows:

```
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
```

Next, build the binary as follows:

```
cd crunchy-proxy
godep restore
make
```

Design

The example shows a message traveling down this path:

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

Packages

The proxy code is implemented in the following golang packages:

Table 1. proxy golang packages

Package Name	Purpose
adapter	adapters can be applied to in-bound and out-bound message flows to do add capabilities like logging or auditing
admin	the administration interface, a REST API
config	the configuration file format
proxy	the main proxy processing
testclient	a test client that uses libpq, useful for testing
tests	standalone unit tests

Makefile Targets

The Makefile has the following targets defined:

Table 2. Makefile targets

Makefile Target	Purpose
gendeps	calls godep to generate dependencies for golang compilation
docsbuild	calls asciidoctor to generate PDF and HTML versions of the documentation
clean	removes the proxy binaries
dockerimage	builds the docker image for the proxy
push	pushes the docker image to dockerhub
run	runs the proxy in foreground using the default configuration file
test	executes the standalone unit tests

Wire Protocol

crunchy-proxy operates at the PostgreSQL wire protocol (network) layer to understand PostgreSQL client authentication requests and SQL statements passed by a client to a PostgreSQL backend.

The proxy does very little processing of the messages sent between a client and an actual backend, mostly examining the SQL statements for a proxy-specific annotation. The annotation is used to route the message to the backend.

Its important to note that the proxy does not implement all features of libpq or provide an application interface similar to a JDBC driver or other language driver.

Connection Pooling

crunchy proxy provisions a connection pool for each backend (master and replica(s)) that is defined in the proxy configuration file. The connection pool is a fixed size currently and established before the proxy begins to accept connections from clients.

The connections in the pool are determined by the pool settings found within the configuration parameters **credentials** and **pool**.

Currently **crunchy proxy** only supports basic PostgreSQL password authentication using username and password.

As client requests come into the proxy, the proxy will choose to which backend to route the SQL statement and then pick a free connection from the backend's connection pool.

For each connection pool there is a golang channel defined to manage which connections are available for use when processing a SQL statement. After the SQL statement is processed, the connection is returned to the pool. You can think of the pool's channel as a queue of available connections.

Client Authentication

Each client must authenticate against the master backend before the proxy will process future client requests. **crunchy proxy** does not include an authentication store itself, but instead relies on the master backend to perform authentication.

Once a client does authenticate, the proxy will terminate the client's connection to the master and subsequently begin using the connections from the connection pools.

Annotations

SQL statements that start with a SQL comment of a particular format will be used to determine the routing of a SQL statement either to a master or a replica.

To simplify the proxy parsing, we require the annotation begin at the first byte of the SQL statement as follows:

```
/* read */ select from foo.....
```

If no annocation is found in a SQL statement, **it is assumed the statement is a write**.

In certain circumstances, it may be desriable to route all the SQL statements within a transaction to the same backend.

In order to support this case, it is possible include a **start** annotation in the first SQL statement and a **finish** annotation in the last SQL statement as follows:

```
/* start */ begin;
select .....;
/* finish */commit;

/* start,read */ begin;
select .....;
/* finish */commit;
```

Health Checking

The **crunchy-proxy** status health check is currently a simple implementation - essentially determining only whether the backend can process a SQL statement.

The health check is performed every few second on each backend by a separate goroutine that runs until the proxy exits.

The backend status is checked by the active connection processing in order to determine which backends are available to process a SQL statement.

As the status of a backend changes, the global configuration is updated.

Health status is captured and placed into an event channel. The event channel is used to publish events to any number of subscribers to the REST API.

Legal Notices