Honeycomb refinery - redis error investigation

Summary

This is an **investigation** of RUNTIME-1236, a recurring error by the **Honeycomb refinery**.

Symptom

We see **two** type of **errors** in the logs:

```
msg="registration failed" err=EOF name="http://10.12.106.14:8081" timeoutSec=10
```

and

```
msg="redis scan encountered an error" err=EOF keys_returned=0 timeoutSec=5s
```

The assumption by Chris was that these are **both** most likely **related** to the Redis connection from the refinery, to be more precise it's assumed that the redis **connection** is **terminated** somewhere.

Invetigation

First, we try to get **more context** of these error messages. After reading the code, we can **confirm** that **both** messages are related to the **Redis code** in the refinery.

registration failed, in redimem.go:

```
_, err = conn.Do("SET", key, "present", "EX", timeoutSec)
if err != nil {
  logrus.WithField("name", memberName).
    WithField("timeoutSec", timeoutSec).
    WithField("err", err).
    Error("registration failed")
  return err
}
return nil
```

redis scan encountered an error, in redimem.go:

```
  }
  for err := range errChan {
    logrus.WithField("keys_returned", len(memberList)).
      WithField("timeoutSec", redisScanTimeout).
      WithField("err", err).
      Error("redis scan encountered an error")
  }
  return memberList, nil
}
```

Let's find the code where the refinery is **initializing** the redis connection pool, which is in redis.go:

```
  pool := &redis.Pool{
    MaxIdle:     3,
    MaxActive:   30,
    IdleTimeout: 5 * time.Minute,
    Wait:        true,
    Dial: func() (redis.Conn, error) {
      // if redis is started at the same time as refinery, connecting to redis can
      // fail and cause refinery to error out.
      // Instead, we will try to connect to redis for up to 10 seconds with
      // a 1 second delay between attempts to allow the redis process to init
      var (
```

We can also see that the refinery is using the **redigo** module:

```
  "time"

  "github.com/gomodule/redigo/redis"
  "github.com/sirupsen/logrus"
```

Interesting to note that the **redigo** is not the offical library for redis, and its last release was a year ago. The official go library is go-redis and it's actively maintained. This is not a major point here, redigo still should work, but will make a slight relevant point about it later.

Next, we check **redigo's** documentation for the **Pool** function, where we see something **interesting**:

```
    // Close connections after remaining idle for this duration. If the value
    // is zero, then idle connections are not closed. Applications should set
    // the timeout to a value less than the server's timeout.
    IdleTimeout time.Duration
```

According to the documentation the **IdleTimeout** value should be **lower** than the **server** (idle) **timeout**. This makes sense, especially if the **server** is closing connection **without notifying** the clients first, which is what this note **suggest**. If you remember just before I pasted where the refinery **initialize** the **Pool** with a **hard-coded 5 minutes** (300s) idle timeout.

Let's **check** the **server** side.

First, let's get the **IP** of the **redis** service:

```
aspitzer@4978 graphite-envoy % kubectl get service -n refinery
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE
redis            ClusterIP   10.253.15.65    <none>        6379/TCP            238d
redis-dev        ClusterIP   10.253.15.206   <none>        6379/TCP            39d
redis-infra      ClusterIP   10.253.12.85    <none>        6379/TCP            40d
refinery         ClusterIP   None            <none>        8080/TCP,4317/TCP   204d
refinery-dev     ClusterIP   None            <none>        8080/TCP,4317/TCP   39d
refinery-infra   ClusterIP   None            <none>        8080/TCP,4317/TCP   40d
```

And **fetch** the **timeout** parameter from the Redis server:

```
root@sendai-testpod:/# redis-cli -h 10.253.15.65
10.253.15.65:6379> config get timeout
1) "timeout"
2) "60"
```

It's **set** to **60** (all 3 redis servers are set to 60). This may not be the only reason why we see these errors, but it's definitely a possibility, so I recommend to **correct** this. Since the refinery idle timeout is hard coded, we have to **increase** the **timeout** on the **server side**, to more than 300. We can do this in the **configmap** for the Redis server:

```
aspitzer@4978 terraform-config % kubectl get configmap redis-config -n refinery -o yaml
apiVersion: v1
data:
  redis.conf: |
    loglevel notice
    logfile ""
    save ""
    appendonly no
    maxmemory 23622320128
    maxmemory-policy allkeys-lru
    timeout 60
```

Also to **documentation** for the **timeout** parameter server side:

By default recent versions of Redis don't close the connection with the client if the client is idle for many seconds: the connection will remain open forever.

However if you don't like this behavior, you can configure a timeout, so that if the client is idle for more than the specified number of seconds, the client connection will be closed.

You can configure this limit via redis.conf or simply using CONFIG SET timeout .

This is pretty much **confirming** what we **see**. And now, back to redis-go, the official Redis Go library. They have a retry option for their connections, unfortunately redigo doesn't have that.

Conclusion

The first finding is that the **idle timeout** is significantly **lower** (60s) on the server side than on the **client side** (300s), which can cause the **redis** server to close idle client connections, which can cause error messages we experience. I recommend to **increase** the server side **timeout** to **330**. If we keep seeing these error messages after the increase, I'm happy to continue the investigation.

sendai