Load Balancing and Proxy Configuration

Contents

1. HAProxy
2. Nginx

The recommended best practice mode of production Riak operation suggests placing Riak behind a load balancing or proxy solution, either hardware or software based, and never exposing Riak directly to public network interfaces.

Riak users have reported success in using Riak with a variety of load balancing and proxy solutions. Common solutions include proprietary hardware based load balancers, cloud based load balancing options, such as Amazon's Elastic Load Balancer, and open source software based projects like HAProxy and Nginx.

This guide briefly explores the commonly used open source software based solutions HAProxy and Nginx, and provides some configuration and operational tips gathered from community users and operations oriented engineers at Basho.

While it is by no means an exhaustive overview of the topic, this guide should provide a starting point for choosing and implementing your own solution.

HAProxy

HAProxy is a fast and reliable open source solution for load balancing and proxying of HTTP and TCP based application traffic.

Users have reported success in using HAProxy in combination with Riak in a number of configurations and scenarios. Much of the information and example configuration for this section is drawn from experiences of users in the Riak community in addition to suggestions from Basho engineering.

Example Configuration

The following is an example starting point configuration for HAProxy to act as a load balancer to a 4 node Riak cluster for access by clients using the Protocol Buffers and HTTP interfaces.

> The operating system's open files limits need to be greater than 256000 for the example configuration that follows. Consult the Open Files Limitdocumentation for details on configuring the value for different operating systems.

Config

```
global
        log 127.0.0.1     local0
        log 127.0.0.1     local1 notice
        maxconn           256000
        chroot            /var/lib/haproxy
        user              haproxy
        group             haproxy
        spread-checks     5
        daemon
        quiet

defaults
        log               global
        option            dontlognull
        option            redispatch
        option            allbackups
        maxconn           256000
        timeout connect   5000

backend riak_rest_backend
       mode               http
       balance            roundrobin
```

```
        option              httpchk GET /ping
        option              httplog
        server riak1 riak1.<FQDN>:8098 weight 1 maxconn 102
        server riak2 riak2.<FQDN>:8098 weight 1 maxconn 102
        server riak3 riak3.<FQDN>:8098 weight 1 maxconn 102
        server riak4 riak4.<FQDN>:8098 weight 1 maxconn 102

frontend riak_rest
        bind                127.0.0.1:8098
        mode                http
        option              contstats
        default_backend     riak_rest_backend


backend riak_protocol_buffer_backend
        balance             leastconn
        mode                tcp
        option              tcpka
        option              srvtcpka
        server riak1 riak1.<FQDN>:8087 weight 1 maxconn 102
        server riak2 riak2.<FQDN>:8087 weight 1 maxconn 102
        server riak3 riak3.<FQDN>:8087 weight 1 maxconn 102
        server riak4 riak4.<FQDN>:8087 weight 1 maxconn 102


frontend riak_protocol_buffer
        bind                127.0.0.1:8087
        mode                tcp
        option              tcplog
        option              contstats
        mode                tcp
        option              tcpka
        option              srvtcpka
        default_backend     riak_protocol_buffer_backend
```

Note that the above example is considered a starting point and is a work in progress based upon this example. You should carefully examine the configuration and change it according to your specific environment.

Maintaining Nodes Behind HAProxy

When using HAProxy with Riak, you can instruct HAProxy to ping each node in the cluster and automatically remove nodes which do not respond.

You can also specify a round robin configuration in HAProxy and have your application handle connection failures by retrying after a timeout, thereby reaching a functioning node upon retrying the connection attempt.

HAPproxy also has a standby system you can use to remove a node from rotation while allowing existing requests to finish. You can remove nodes from HAProxy directly from the command line by interacting with the HAProxy stats socket with a utility such as socat:

Shell

```
echo "disable server <backend>/<riak_node>" | socat stdio /e
```

At this point, you can perform maintenance on the node, down the node, and so on. When you've finished working with the node and it is again available for requests, you can re-enable the node:

Shell

```
echo "enable server <backend>/<riak_node>" | socat stdio /et
```

Consult the following HAProxy documentation resources for more information on configuring HAProxy in your environment:

- HAProxy Documentation
- HAProxy Architecture

Nginx

Some users have reported success in using the Nginx HTTP server to proxy requests for Riak clusters. An example that provides access to a Riak cluster *through*

GET requests only is provided here for reference.

Example Configuration

The following is an example starting point configuration for Nginx to act as a front end proxy to a 5 node Riak cluster.

This example forwards all GET requests to Riak nodes while rejecting all other HTTP operations.

Nginx version notes

This example configuration was verified on **Nginx version 1.2.3**. Please be aware that early versions of Nginx did not support any HTTP 1.1 semantics for upstream communication to backends. You should carefully examine this configuration and make changes appropriate to your specific environment before attempting to use it.

```
upstream riak_hosts {
  # server  10.0.1.10:8098;
  # server  10.0.1.11:8098;
  # server  10.0.1.12:8098;
  # server  10.0.1.13:8098;
  # server  10.0.1.14:8098;
}

server {
  listen   80;
  server_name  _;
  access_log  /var/log/nginx/riak.access.log;

  # your standard Nginx config for your site here...
  location / {
    root /var/www/nginx-default;
  }
```

```
    # Expose the /riak endpoint and allow queries for keys only
    location /riak/ {
        proxy_set_header Host $host;
        proxy_redirect off;

        client_max_body_size       10m;
        client_body_buffer_size    128k;

        proxy_connect_timeout      90;
        proxy_send_timeout         90;
        proxy_read_timeout         90;

        proxy_buffer_size          64k;  # If set to a smaller val
                                         # nginx can complain with
                                         # "too large headers" err
        proxy_buffers              4 64k;
        proxy_busy_buffers_size    64k;
        proxy_temp_file_write_size 64k;

      if ($request_method != GET) {
        return 405;
      }

      # Disallow any link with the MapReduce query format "bucket,
      if ($uri ~ "/riak/[^/]*/[^/]*/[^,]+,[^,]+," ) {
        return 405;
      }

      if ($request_method = GET) {
        proxy_pass http://riak_hosts;
      }
    }
  }
```

Note

Even when filtering and limiting requests to GETs only as done in the example, you should strongly consider additional access controls beyond what Nginx can provide directly, such as specific firewall rules to limit inbound connections to trusted sources.

Querying Secondary Indexes Over HTTP

When accessing Riak over HTTP and issuing Secondary Index queries, you can encounter an issue due to the default Nginx handling of HTTP header names containing underscore (`_`) characters.

By default, Nginx will issue errors for such queries, but you can instruct Nginx to handle such header names when doing Secondary Index queries over HTTP by adding the following directive to the appropriate `server` section of `nginx.conf`:

```
underscores_in_headers on;
```

These May Also Interest You

- Downloads
- Log Messages FAQs
- Operating Riak FAQs
- Configuration Files
- Configuring Secondary Indexes
- Basic Cluster Setup