neonHIVE Load Balancers

# Introduction

A key requirement for production hives can effectively route external (Internet) traffic to hive services, implementing load balancing and fail-over. The Docker ingress network provides this for Docker swarm mode services, but this works only for containers that are attached to an overlay network. External traffic is not implicitly supported.

Load balancers are also required for other situations, typically to provide load balancing and fail-over for a group of stateful containers that combine to offer a stateful service (e.g. an Elasticsearch or Couchbase database cluster). Basic load balancing is based on the HAProxy open source project: [haproxy.org](http://haproxy.org).

neonHIVE load balancing also supports more advanced features such as HTTP caching and API Gateway using [Varnish Cache](http://varnish-cache.org/) and [Kong API Gateway](https://konghq.com/').

This document describes these scenarios.

# Docker Images

neonHIVE provides five different Docker images for implementing HTTP and TCP reverse proxies.

[haproxy](https://hub.docker.com/r/neoncluster/haproxy/) A simple image based on the [haproxy](https://hub.docker.com/_/haproxy/)/alpine series of official Docker images. This image gets its configuration from an internal or mounted file and it also monitors the file for changes to dynamically reconfigure itself. This image is intended to be used to deploy relatively static proxies.

[neon-proxy](https://hub.docker.com/r/neoncluster/neon-proxy/) A more sophisticated image also based on the [haproxy](https://hub.docker.com/_/haproxy/)/alpine series of official Docker images. This image is deployed as a Docker service and downloads its configuration from a HashiCorp Consul key and then dynamically updates itself whenever the key value changes. The configuration is a ZIP archive including the HAProxy configuration file as well as other artifacts such as TLS certificates. The proxy can also download sensitive assets from HashiCorp Vault. This image is intended to be used for most hive proxies.

[neon-proxy-manager](https://hub.docker.com/r/neoncluster/neon-proxy-manager/) Monitors the proxy routes stored in Consul and the TLS certificates stored in Vault for changes and regenerates the HAProxy configurations served by neon-proxy instances.

[varnish](https://hub.docker.com/r/neoncluster/varnish/) Provides simple HTTP caching.

[neon-proxy-cache](https://hub.docker.com/r/neoncluster/neon-proxy-varnish/) Integrates with neon-proxy-manager to implement neonHIVE service caching. This is based on the [varnish](https://hub.docker.com/r/neoncluster/varnish/) image.

# Proxy Services

neonHIVE currently deploys several built-in proxy services:

neon-proxy-vault Handles load balancing and fail-over for the Vault servers running on the hive managers. This is published to port 5003 on the Docker ingress network. This is a relatively static proxy that will only need to be updated when manager nodes are added or removed. This deploys as the nhive/haproxy image.

neon-proxy-public Handles routing of external HTTP and TCP traffic (e.g. from the Internet) to hive services and containers attached to the neon-public network. This is published to ports 80/443 and 5100-5299 on the Docker ingress network. Port **80** handles **HTTP** traffic, **443** handles **HTTPS**, with the **port range** to be dedicated for **TCP** traffic or specialized HTTP endpoints. External routers or load balancers will typically be configured to direct hive traffic to ports 80 and 443 and to direct other inbound traffic say POP port 101 to one of the ports in the range.  
  
This deploys as the nhive/neon-proxy image that dynamically loads its configuration from Consul and Vault. This proxy will be reconfigured as services are deployed or removed, as TLS certificates are updated and as routing options are changed.

neon-proxy-private Handles the routing of internal hive HTTP and TCP traffic to services and containers on the neon -private network. This is published to ports 5300-5499 on the Docker ingress network. Port **5300** handles **HTTP** traffic, **5301** handles **HTTPS**, with the **remaining ports** are dedicated for **TCP** traffic.  
  
This is intended for situations where standard Docker ingress routing is insufficient. A typical situation is when a stateful service needs to be deployed as individual containers for manageability and clients require a single URL to the containers as a group that will load balance and fail-over properly. This deploys as the nhive/neon-proxy image that dynamically loads its configuration from Consul and Vault. This proxy will be reconfigured as services are deployed or removed, as TLS certificates are updated and as routing options are changed.

neon-proxy-public-bridge Handles the routing of traffic from pet nodes to the neon-proxy-public service running on the Swarm. This is deployed as a container on each pet and works by forwarding TCP traffic from the standard public proxy ports on the pet to the same ports on any of the swarm nodes. The routes will all be configured as TCP pass-thru and any HTTPS decryption will still be terminated by neon-proxy-public.  
  
This is also deployed using the nhive/neon-proxy image.

neon-proxy-private-bridge Handles the routing of traffic from pet nodes to the neon-proxy-private service running on the Swarm. This is deployed as a container on each pet and works by forwarding TCP traffic from the standard private proxy ports on the pet to the same ports on any of the swarm nodes. The routes will all be configured as TCP pass-thru and any HTTPS decryption will still be terminated by neon-proxy-private.  
  
This is also deployed using the nhive/neon-proxy image.

neon-proxy-public-cache Implements HTTP caching for public services. This is deployed automatically when required by neon-proxy-manager using the nhive/neon-proxy-cache image.

neon-proxy-private-cache Implements HTTP caching for private services. This is deployed automatically when required by neon-proxy-manager using the nhive/neon-proxy-cache image.

# Proxy Manager

The neon-proxy-manager service is deployed to manage the neon-proxy-public, neon-proxy-private and **neon-proxy-private-bridge** proxies. The proxy manager is constrained to run on manager nodes and will be configured to run a single instance. Proxy manager settings are persisted to **Consul** as:

neon/service:  
 neon-proxy-manager:  
 cert-warn-days: 30  
 cache-remove-seconds: 300  
  
 status:  
 public: <ProxyStatus json>  
 private: <ProxyStatus json>  
 public-bridge: <ProxyStatus json>  
 private-bridge: <ProxyStatus json>  
   
 proxies:  
 public:  
 proxy-conf: proxy-conf.zip  
 proxy-hash: <MD5 hash of proxy-conf + certs>  
 public-bridge:  
 proxy-conf: proxy-conf.zip  
 proxy-hash: <MD5 hash of proxy-conf>  
 private:  
 proxy-conf: proxy-conf.zip  
 proxy-hash: <MD5 hash of proxy-conf + certs>  
 public-bridge:  
 proxy-conf: proxy-conf.zip  
 proxy-hash: <MD5 hash of proxy-conf>  
 private-bridge:  
 proxy-conf: proxy-conf.zip  
 proxy-hash: <MD5 hash of proxy-conf>  
  
 conf:  
 reload: <uuid>  
  
 public:  
 settings: <ProxySettings json>  
 rules:  
 name1: <ProxyRoute json>  
 name2: <ProxyRoute json>  
 ...  
 private:  
 settings: <ProxySettings json>  
 rules:  
 name1: <ProxyRoute json>  
 name2: <ProxyRoute json>  
 ...

where:

cert-warn-days (double) Specifies the number of days in advance to begin warning of certificate expirations.

cache-remove-seconds (double) Specifies the number of seconds neon-proxy-manager should wait after detecting that the corresponding proxy cache service is no longer necessary before actually removing it. This will help prevent the manager from potentially stopping and restating the cache service unnecessarily as operators add and remove load balancer rules.

proxies/\*/proxy-conf Holds public or private proxy’s generated HAProxy and Vault configurations as a ZIP archive.

proxies/\*/proxy-hash The MD5 hash of the public or private proxy’s proxy-conf archive combined with the hash of all of the referenced certificates. This is used by neon-proxy service instances to detect when the proxy configuration has changed.

status/\* (json) Describes the proxy route status at the time the neon-proxy-manager last processed hive rules for the named load balancer.

conf Root key for load balancer settings and rules.

conf/reload UUID updated whenever any of the configuration properties are changed. neon-proxy-manager polls this frequently and republishes the proxy configurations when a change is detected.

conf/\*/settings Load balancer settings (see the LoadBalancerSettings type).

conf/\*/rules Load balancer rules for the load balancer (see the LoadBalancerRule type).

cert-update Updated with a new UUID by the neon-cli whenever certificates are modified. neon-proxy-manager monitors this and republishes immediately on a change.

The neon-proxy-public and neon-proxy-private services in the Swarm as well as the proxy-public-bridge and neon-proxy-private-bridge containers on any pet nodes are all based on the nhive/neon-proxy image. This image is designed to download a ZIP archive from a Consul key. This ZIP file includes the HAProxy configuration as well as other configuration artifacts. The services then continue to monitor the Consul key for changes to dynamically reconfigure themselves.

Each proxy service settings key holds global definitions (JSON), and the route keys describe how traffic is to be routed (also JSON). The conf key holds the generated HAProxy configuration ZIP archive and artifacts. **proxy-hash** is the MD5 hash of the **proxy-conf** data plus the hashes of any referenced certificates.

neon-proxy-manager works by listening on the neon/proxy-notify HiveMQ channel for ProxyRegenerateMessage messages indicating that it should scan the proxy Consul settings under **neon/service/neon-proxy-manager/conf/\*** and any TLS certificates in Vault for changes. A notification will be broadcast by the neon-cli whenever it updates load balancer rules and neon-hive-manager periodically broadcasts a notification (this defaults to a 5-minute interval). This is a fail-safe that ensures the proxies and rules will eventually converge if neon-cli notifications are somehow lost and also to periodically verify that the proxy configuration is still valid (e.g. that TLS certificates have not expired).

The settings and/or **route/\*** keys will be modified by neon-cli whenever a proxy definition is changed. reload will be touched whenever **neon proxy PROXY reload** is executed. cert-update will be touched whenever TLS certificates are uploaded or modified.

The proxy manager performs the following steps when certificate or proxy definition changes are detected:

1. TLS certificates are downloaded from Vault and are verified. Invalid, expired, or near expired certificates will be logged.
2. MD5 hashes will be generated for each certificate.
3. These steps will be performed for each managed proxy:  
   1. The proxy settings and endpoints will be loaded and a new proxy-conf.zip configurations will be generated, including the HAProxy and Varnish configuration files.
   2. An **MD5 hash** will be computed for the proxy-conf.zip along with the hashes of the certificates referenced by the configuration.
   3. The new hash will be compared against that saved in Consul for the proxy. If they differ, the new configuration and hash will be updated in Consul and a ProxyUpdateMessage will be broadcast to the related HiveMQ proxy-public-update or proxy-private-update channel so that the listening proxy and caching services can quickly pick up the changes.
   4. Each proxy service or container listens for notifications on the related HiveMQ proxy-public-update or proxy-private-update channel and will dynamically update themselves when notified.

neon-proxy-manager also creates and removes the neon-proxy-public-cache and neon-proxy-private-cache services when caching is enabled for one or more load balancing rules or has been disabled for all rules for a period of time. This way, these caches will consume hive resources only as needed.

## Docker Secrets

As of version 1.13.0, Docker supports secrets for swarm mode services. Docker secrets are created by piping the secret (text or data) to the docker secret NAME command. This persists the secret in Docker using the NAME passed. The necessary secrets must be made available to hive services as they are deployed.

Secret names prefixed by neon-\* are reserved for neonHIVE services.

The public and private hive load balancers require read access to the TLS certificates stored in the Vault at neon-secret/cert/\*. Access to this is secured by the **neon-proxy-public** and **neon-proxy-private** Vault AppRoles. The role credentials are persisted as the following Docker secrets and will be made available to the proxies when they are launched.

neon-proxy-manager-credentials Vault credentials for the neon-proxy-manager service.

neon-proxy-public-credentials Vault credentials for the neon-proxy-public service.

neon-proxy-private-credentials Vault credentials for the neon-proxy-private service.

# Proxy Port Ranges

neonHIVEs reserves a block of 200 ports on the ingress network for each of the public and private proxies.

**neon-proxy-public**: ports 80/443 + 5100 – 5299

http: 80  
 https: 443  
 custom: 510- - 5299

Edge routers or load balancers will typically be configured to route external HTTP/HTTPS traffic to port 80/443 to any non-pet hive node. This traffic will be directed to neon-proxy-public instances via the Docker ingress network where the traffic will be directed to the configured backend services.

**neon-proxy-private**: ports 5300 – 5499

http: 5300  
 https: 5301  
 custom: 5302 - 5499

The first two ports are reserved for inbound HTTP and HTTPS traffic. Most, if not all HTTP(S) requests should be directed to these ports and then the proxies should be configured with routes that use the HTTP host header to decide where to deliver traffic.

The remaining 198 ports in this block can be used for routing TCP connections, HTTPS pass-thru, or HTTPS endpoints for older clients that don’t support SNI (server name indication). You’ll need to manually configure your edge router or load balancer to route inbound traffic to the correct port.

# Proxy Bridge Services

As mentioned above, these services are designed to proxy traffic from hive pet nodes to the same endpoints defined for neon-proxy-public and neon-proxy-private services running in the Swarm. This allows traffic to the standard local pet node public proxy ports (80/443, 5100-5299) and private proxy ports (5300-5499) to be forwarded to the same ports on targeted Swarm nodes which will then be forwarded to the ultimate destination.

The neon-proxy-manager automatically generates the bridge HAProxy configurations from the current neon-proxy-public and neon-proxy-private configurations, converting any HTTP/HTTPS proxies into pass-thru TCP proxies. The target swarm nodes are selected randomly by default or may be specified explicitly by IP address via the proxy settings.

# Proxy Cache Services

The neon-proxy-manager service monitors the public and private load balancer rules for rules that require response caching. When this is detected, neon-proxy-manager will spin up neon-proxy-public-cache and/or neon-proxy-private-cache services to handle this and neon-proxy-manager will also include a varnish.vcl file within the Varnish configuration in the proxy-conf.zip archive persisted to Consul. The proxy services will monitor the HiveMQ proxy-notify queue for configuration update messages (just as the proxy services do) to update the local Varnish configuration. So, this is pretty clean.

There’s one bit of trickiness we’ll have to handle. Some observations:

* The public and private proxies each expose HTTP and TCP on up to 300 ports.
* The first two ports on each public/private port range are reserved for HTTP and HTTPS traffic and all load balancer rules on these ports require that requests have unique hostnames so we’ll be able to route requests to the correct backends.
* The other ports may expose either HTTP/HTTPS or TCP endpoints. We don’t cache TCP so we can ignore these.
* It’s possible to expose an HTTP/HTTPS endpoint that doesn’t require hostnames. This effectively routes all requests to the port to the backend without trying to match hostnames.
* Varnish listens on a single port (port 80 for the proxy cache services).

So, here’s the problem: we need to a way to forward cache enabled requests from HAProxy running in the proxy services to Varnish running in the proxy cache services such that Varnish will be able to forward to its backends taking into account the original inbound HAProxy port and the request hostname (if there was one).

We’re going to accomplish this by having neon-proxy-manager generate the HAProxy configuration such that it adds a new X-Neon-Proxy-Target header to each request that passes through. This will identify the inbound proxy PORT and the hostname, if required:

PORT-HOSTNAME The port number and hostname (HOSTNAME may be empty).

The neon-proxy-manager will also generate corresponding code in the varnish.vcl that matches Varnish backends using the X-Neon-Proxy-Target header and also removes this header from the request before forwarding it on to the backend.