neonHIVE Load Balancers

# Introduction

A key requirement for production clusters is can effectively route external (Internet) traffic to cluster 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 four 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 cluster 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/) Implements 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.

# Proxy Services

neonHIVE currently deploys three built-in proxy services:

neon-proxy-vault Handles load balancing and fail-over for the Vault servers running on the cluster 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 neoncluster/haproxy image.

neon-proxy-public Handles routing of external HTTP and TCP traffic (e.g. from the Internet) to cluster services and containers attached to the neon-public network. This is published to ports 5100-5299 on the Docker ingress network. Port **5100** handles **HTTP** traffic, **5101** handles **HTTPS**, with the **remaining ports** are dedicated for **TCP** traffic. External routers or load balancers should be configured to direct cluster traffic here.  
  
This deploys as the neoncluster/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 cluster 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 neoncluster/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 neoncluster/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 neoncluster/neon-proxy image.

neon-proxy-public-cache Implements HTTP caching for public services. This is deployed using the neoncluster/neon-proxy-cache image.

neon-proxy-private-cache Implements HTTP caching for private services. This is deployed using the neoncluster/neon-proxy-cache image.

These proxies should handle most scenarios, but it is possible to deploy additional custom proxy services.

# 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:  
 poll-seconds: 10  
 fallback-poll-seconds: 300  
 cert-warn-days: 30  
  
 status:  
 public: <ProxyStatus json>  
 private: <ProxyStatus json>  
 public-bridge: <ProxyStatus json>  
 private-bridge: <ProxyStatus json>  
   
 proxies:  
 public:  
 proxy-conf: haproxy.zip  
 proxy-hash: <MD5 hash of proxy-conf + certs>  
 public-bridge:  
 proxy-conf: haproxy.zip  
 proxy-hash: <MD5 hash of proxy-conf>  
 private:  
 proxy-conf: haproxy.zip  
 proxy-hash: <MD5 hash of proxy-conf + certs>  
 private-bridge:  
 proxy-conf: haproxy.zip  
 proxy-hash: <MD5 hash of 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:

poll-seconds (**double**) Specifies how often the proxy manager should scan TLS certificates persisted in Vault for expiration checks and updates and also poll the individual proxy definitions for changes.

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

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

proxies/\*/proxy-hash The MD5 hash of the public or private proxy’s conf archive combined with the hash of all of the referenced certificates. This is used by neon-proxy 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 cluster routes for the named proxy.

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 neon-proxy-private-bridge containers on the pet nodes are all based on the neoncluster/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-private-bridge does not have settings or routes because these are derived from the neon-proxy-private configuration.

neon-proxy-manager works by polling **neon/service/neon-proxy-manager/conf/\*** for changes. 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 haproxy.zip configuration will be generated.
   2. An **MD5 hash** will be computed for the haproxy.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.
   4. Each proxy container monitors its proxies/\*/proxy-conf key for changes and will dynamically update itself when the configuration changes.
4. neon-proxy-manager also generates the neon-proxy-public-bridge and neon-proxy-private-bridge HAProxy configurations from the proxy settings and routes as appropriate. Only TCP routes are generated.

The proxy manager also periodically polls the certificates in Vault, proxy settings and route definitions performing the steps outlined above to verify certificates over time when nothing else changes as well as to ensure that proxy configurations don’t inadvertently get out of sync with their definitions.

## 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 cluster services as they are deployed.

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

The public and private cluster proxies 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 5100 – 5299

http: 5100  
 https: 5101  
 custom: 5102 - 5299

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

http: 5300  
 https: 5301  
 custom: 5302 - 5499

The first two ports in each block 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.

**Note** This convention makes it easy to configure edge routers or load balancers. Simply have them direct traffic targeting external ports 80 and 443 to ports 80 and 443[[1]](#footnote-1) on one or more cluster nodes. This one-time configuration will handle many deployment scenarios.

The remaining 198 ports in each 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 cluster 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 (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.

1. Note that all neonHIVE host nodes (including pets) have iptables configured to forward packages received on protected ports 80 and 443 to the public proxy ports 5100 and 5101 to make this work. [↑](#footnote-ref-1)