neon-cluster-manager

# Introduction

The neon-cluster-manager container performs a cluster maintenance functions:

* Periodically queries the Docker swarm to obtain information on the available nodes as well as the node labels so that the cluster definition saved to Consul is always close to being up to date. The compressed cluster definition and hash will be persisted to Consul at:  
    
  neon:  
   cluster:  
   definition.deflate – (json/compressed) the current cluster definition  
   definition.hash - MD5 hash of the definition (base64)
* Monitoring the Vault seal status and optionally unsealing Vault automatically.

# Implementation Note I really wanted to deploy this as a Docker service with one replica running on manager nodes. This won't work at this time for three reasons:

# We can't expose the Docker socket outside of the host for security reasons.

# .NET Core is not currently capable of performing HTTP queries against Unix sockets, so we can't mount **/var/run/docker.sock** into our container.

# The alternative to #1 and #2 is to expose a Docker socket on the **127.0.0.1** loopback address. The problem here is that Docker services are unable to mount the host network.

# The solution is to run **neon-cluster-manager** as a container on each of the manager nodes, mount the host network, and access Docker via the loopback address.

# Ideally, we'd also use a Consul lock to ensure that only one instance is active at any time but I'm going to defer until later. Doing this for the time being this means that each manager node will be polling for node status every 30 seconds. This works out to roughly a 1.5K download per cluster node during each poll; so maybe 150K total for a 100-node cluster. This should be manageable for most clusters and this could be mitigated by increasing the polling interval.

# Docker Image

The service is provisioned using the [neoncluster/neon-cluster-manager](https://hub.docker.com/r/neoncluster/neon-cluster-manager/) image.

# Consul Configuration

The container requires only a handful of settings from Consul to operate. A default value will be used if a specific setting isn’t present in Consul. You’ll need to force the service to update (restarting it) to pick up a settings change.

neon/services  
 neon-cluster-manager:  
 node\_poll\_seconds: 30  
 vault\_poll\_seconds: 30

Here are the descriptions:

**node\_poll\_seconds** (double) The number of seconds the service will wait between retrieving the current cluster node information from the manager hosting the service and then updating the definition in Consul if it has changed. This defaults to 30 seconds.

**vault\_poll\_seconds** (double) The number of seconds the service will wait between Vault seal status checks.

The service will attempt to unseal any sealed Vault instances if the **neon-vault-unseal-keys** Docker secret has been set and has been made available to the service. This secret must be formatted as:

{  
 "UnsealKeys": [  
 "nNobx5u5q3JeQM6d/kGkJflUwbg7QSQyEnMGf9wgzdE="  
 ],  
 "KeyThreshold": 1  
}

# Proxy Manager

The neon-proxy-manager service is deployed to manage the neon-proxy-public and neon-proxy-private 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:  
 leader  
 leader\_ttl\_seconds: 60  
 poll-seconds: 300  
 cert-warn-days: 30  
  
 status:  
 public: <ProxyStatus json>  
 private: <ProxyStatus json>  
   
 proxies:  
 public:  
 conf: haproxy.zip  
 hash: <MD5 hash of conf+certs>  
 private:  
 conf: haproxy.zip  
 hash: <MD5 hash of conf+certs>  
  
 conf:  
 reload  
 cert-update  
  
 public:  
 settings: <ProxySettings json>  
 routes:  
 name1: <ProxyRoute json>  
 name2: <ProxyRoute json>  
 ...  
 private:  
 settings: <ProxySettings json>  
 routes:  
 name1: <ProxyRoute json>  
 name2: <ProxyRoute json>  
 ...

where:

leader is used to ensure that only one instance of neon-proxy-manager is actually active.

leader\_ttl\_seconds (double) specifies the number of seconds a leader will hold onto the leader lock without renewing the session. This should be somewhat longer than the time it takes for the neon-proxy-manager to generate the proxy configurations. You may wish to increase this time for clusters with very extensive proxy routing rules.  
  
Note that under certain circumstances, it may up to this much time for a new leader to take over when the previous leader was terminated so you don’t want to set this too high.

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

proxies/\*/hash is 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 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 proxy settings that need to be monitored for changes.

reload is touched when the neon proxy NAME reload command is executed.

cert-update is touched by the neon tool whenever certificates are modified.

settings global per proxy settings for a proxy formatted as JSON (see the ProxySettings type).

routes named per proxy routes formatted as JSON (see the ProxyRoute type).

The neon-proxy-public and neon-proxy-private services are both 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. **hash** is the MD5 hash of the **conf** data plus the hashes of any referenced certificates.

Vault works by monitoring **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/\*/conf key for changes and will dynamically update itself when the configuration changes.

The proxy manager also periodically polls the certificates in Vault, proxy settings and route definitions performing the steps outlined above as a way 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 neonCLUSTER 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

neonCLUSTERs reserves a block of 100 ports on the Docker 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 5300 and 5301 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.