NeonCluster Deployment

on Ubuntu 16.04 LTS

# Overview

This document starts out by describing how the scripts and other assets located within this source folder are to be used to create NeonCluster development and production clusters on Ubuntu 16.04 LTS.

A [Docker Swarm](https://docs.docker.com/swarm/overview/) cluster is a collection of Linux host machines, each running a Docker Engine (in swarm mode), as well as potentially Consul, Vault, and TD-Agent. Docker (swarm mode) creates a virtual Docker Engine across all of the cluster hosts and handles the assignment, deployment, and management of containers across the cluster. Docker refers to this as container *scheduling*. Swarm scheduling is relatively sophisticated and provides for distributing containers across hosts for fault tolerance, relocating containers on failure, accounting for container/hardware as well as container/container affinity. Finally, Swarm can handle the deployment of applications that consist of multiple containers.

# Creating a Development NeonCluster

The first step in creating you cluster is to create the Hyper-V virtual machines that will act as the Docker host nodes. A preconfigured VHDX file is available to make this easy and once you’ve created a cluster you’ll rarely need to create another because Docker makes it easy to restore a cluster to its pristine state by simply removing all deployed containers.

Follow these steps:

1. **Download** and unzip the prepared cluster host image **VHDX** file from:  
     
   <https://s3.amazonaws.com/neon-research/images/ubuntu-16.04-prep.vhdx.zip>
2. Create a **folder** on one or more **SSD** drives to host your VM **VHDX files**.  
     
   Note: Although it is possible to host the virtual drives on spinning hard discs, it is not recommended. Many Neon Research components assume that that are running on modern hardware with SSDs.
3. Decide how many management and worker nodes your cluster will have. You must have at least one management node and at least one worker node. Management nodes host the cluster orchestration, monitoring, as well as the secrets vault. For high availability, you may deploy three or five management nodes. Worker nodes are where application containers will be deployed. Typically, your nodes will be named something like **manager-0**, **manager-1**… and **node-0**, **node-1**…  
     
   **Lill**: I have a 64GB RAM development machine with 2 1-TB SSD drives and I typically create a Hyper-V hosted cluster with **3 manager** and **10 worker** nodes.
4. **Copy** the **VHDX** you downloaded in the new folder for each node and rename it manager-# or **node-#.vhdx** (based on your naming scheme).
5. Using the Hyper-V Manager, create a **VM** for **each node**:  
   1. **Generation 1**
   2. **Manager Node**: **4096MB** RAM  
      **Worker Node**: **4096MB** RAM  
        
      Note: I have dynamic memory enabled for these with the range set to 2048MB-4096MB.
   3. **4 virtual processors**  
        
      Note: The admin credentials[[1]](#footnote-1) for these nodes will be **spot**: **WagTheDog!**
6. Assign a local **static IP** address to each node. There are a couple ways to do this: manually editing the **/etc/network/interfaces** file on each VM as described [here](http://www.cyberciti.biz/faq/setting-up-an-network-interfaces-file/) or configuring **DHCP** on your router.  
     
   **Lill**: I typically configure DHCP by logging into my router, starting and logging into each VM one at a time, using the ifconfig command to discover the MAC address I’ll need to configure the router to reserve an IP for each node.  
     
   Note: For ease of use, assign IPs to the nodes in an easy to remember sequential range.  
     
   **Hint**: To update a Linux machine’s IP address after configuring DHCP, run:  
     
   dhclient -r  
   rm /var/lib/dhcp/\*.leases  
   dhclient
7. **Restart all VMs** so they’ll pick up their new IPs.
8. Optional: If you manage a DNS domain, consider defining **A records** for each local node so you can refer to the hosts via host name, e.g.  
     
   manage-0.lilltek.net. IN A 10.0.1.30   
   manage-1.lilltek.net. IN A 10.0.1.31  
   manage-2.lilltek.net. IN A 10.0.1.32  
   ...  
   node-0.lilltek.net. IN A 10.0.1.40

node-1.lilltek.net. IN A 10.0.1.41

node-2.lilltek.net. IN A 10.0.1.42

...

1. Optional but **Highly Recommended**: Setup an **apt-cache-ng** server to cache Ubuntu/Debian software packages. This will dramatically reduce the amount of traffic on your network when installing and upgrading a cluster.  
     
   Here’s the detailed: [manual](https://www.unix-ag.uni-kl.de/~bloch/acng/html/)  
   1. Follow steps 4-6 above to create an Ubuntu VM for the cache. Configure **RAM=1024MB** and **CPU=4**.
   2. **SSH** into the server and configure its host name like:

sudo bash  
echo apt-cache > /etc/hostname

* 1. The service’s IP address must be statically assigned and/or be resolvable via DNS (e.g. apt-get.lilltek.net).
  2. Install the **apt-get-cacher-ng** server ([manual](https://www.unix-ag.uni-kl.de/~bloch/acng/html/index.html)):  
     1. apt-get update  
        apt-get install -yq apt-cacher-ng
     2. We need to configure the cache to pass-thru SSL requests:  
        Edit **/etc/apt-cacher-ng/acng.conf** by running the following command:  
          
        echo “PassThroughPattern:^.\*:443$” >> /etc/apt-cacher-ng/acng.conf
  3. **Restart** the server to pick up the host name and **apt-cacher-ng** configuration changes.  
       
     Note: You can manage the cache via: http://<apt-cache-server>:3142/acng-report.html
  4. Optional: **neon-cli** will automatically configure cluster nodes to use the cache if **PackageCache** is set in the cluster configuration file. You can also configure any other Ubuntu servers to use the cache by running the commands below (substituting the cache’s IP address or FQDN):  
       
     echo 'Acquire::http::proxy "http://APT-CACHE:3142";' > \  
      /etc/apt/apt.conf.d/02proxy

1. **Cluster Configuration**:  
     
   You’ll be using the neon-cli command line tool to configure your cluster. The latest release binary is located in the Neon source repository in the **~/Build** folder (which will be on the path if you followed the developer setup instructions in **~/Setup-Build.docx** document).  
   1. Create a Neon cluster definition file. This file is formatted as JSON, potentially augmented with comments, variables, and conditionals. Run the command below to obtain a sample cluster definition. The sample includes comments describing the file format and will be a good start for defining your own cluster.  
        
      neon sample
   2. Start all of your node VMs. Wait a couple minutes for the servers to start to avoid a package manager race condition. You’ll have to try again if one of your nodes fails to configure with a log message like:  
        
      E: Could not get lock /var/lib/dpkg/lock - open (11: Resource temporarily unavailable)

E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?

* 1. Create folder to hold setup log files.
  2. Initialize your cluster via:  
       
     neon setup-cluster \  
      --user=spot \  
      --password=WagTheDog! \  
      –-log=<folder> \  
      –-os=Ubuntu-16.04 <config-path>

You’ll be prompted for the user name and password if these aren’t provided as command line options.

* 1. The tool will provision all of the cluster nodes over the course of a few minutes. Once this is complete, you’ll want to verify that the cluster is functional.  
       
     Manually examine the generated log files (one for each node).  
       
     SSH into one of the management nodes and run this command.

1. Note: The default base image credentials are spot/WagTheDog! but the password is changed to a very **strong random password** during cluster setup. [↑](#footnote-ref-1)