**Guide for Integration of HPE 3PAR Volume Plug-in with OpenShift/Kubernetes**

This document will describe the high-level information to install Kubernetes Cluster & OpenShift platform, setup etcd instance/cluster, install HPE 3PAR Volume Plug-in for Docker, build and usage of dory (FlexVolume Driver) and doryd (Dynamic Provisioner) binaries.

**Sequence**

1. Install Kubernetes Cluster OR OpenShift platform.
2. Setup Etcd Cluster or Instance.
3. Install HPE 3PAR Volume Plug-in for Docker.
4. Setup dory (FlexVolume Driver) and doryd (Dynamic Provisioner) binaries.
5. Explore different operations of HPE 3PAR storage provisioning using Kubernetes resources like PersistentVolume, PersistentVolumeClaim, StorageClass, Pods, etc.

**Support Matrix**

Following details are collated as per the validation done by test team:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Platforms | Legacy Docker Volume plugin (Version 1) | Managed Docker Volume plugin (Version 2) | Docker Engine  Version | HPE 3PAR Docker Volume Plug-in for Docker | HPE 3PAR OS version |
| Kubernetes 1.6.13 | Yes | No | 1.12.6 | 2.1 | 3.3.1 MU2 |
| Kubernetes 1.7.6 | Yes | No | 1.12.6 | 2.1 | 3.3.1 MU2 |
| Kubernetes 1.8.9 | Yes | No | 17.06 | 2.1 | 3.3.1 MU2 |
| OpenShift Origin 3.7 (Kubernetes 1.7.6) | Yes | No | 1.12.6 | 2.1 | 3.3.1 MU2 |

**Installation of Kubernetes Cluster**

Install kubernetes 1.8 and above with Docker engine 17.03/17.06 CE/EE,

OR Install kubernetes 1.7 and below with Docker engine 1.12.6

Follow the steps from this website to create K8s cluster: <https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/>

If you want to install any specific version of K8s (e.g. 1.6.13), run the command “yum install -y kubectl-1.6.13 kubernetes-cni-0.5.1 kubelet-1.6.13 kubeadm-1.6.13”

For Container network interface, use “Romana” or “Weave Net”. Flannel is widely used but it didn’t work for us with our flat interface lab network.

**Installation of OpenShift platform**

You can install OpenShift platforms via two installing methods: RPM based installer and Containerized Installer. An RPM installer installs all services through package management and configures services to run within the same user space, while a containerized installer installs services using container images and runs separate services in individual containers.

You can select one based on your requirements and operating systems.

You will need RHEL Server 7.4 to run RPM based installer, and RHEL Atomic host to run Containerized installer. Red Hat Enterprise Linux Atomic Host is a lightweight operating system optimized to run Linux containers.

To install OpenShift 3.7 multinode setup using ansible playbooks, follow below steps which has details of Redhat OpenShift’s official link:

1. Plan your OpenShift multinode setup with different Environment scenarios mentioned in this link:

<https://docs.openshift.org/3.7/install_config/install/planning.html>

1. Follow all the required Pre-requisites from this link:

<https://docs.openshift.org/3.7/install_config/install/prerequisites.html#install-config-install-prerequisites>

1. Prepare all the hosts (master and worker nodes) by following instructions from this link:

<https://docs.openshift.org/latest/install_config/install/host_preparation.html>

1. Prepare your ansible inventory file by configuring all required parameters:

<https://docs.openshift.org/3.7/install_config/install/advanced_install.html>

1. Run the RPM-based installer or Containerized installer:

<https://docs.openshift.org/3.7/install_config/install/advanced_install.html#running-the-advanced-installation-rpm>

Run this command for OpenShift 3.7 and earlier versions:

# ansible-playbook [-i /path/to/inventory] \

    ~/openshift-ansible/playbooks/byo/config.yml

Run this command for OpenShift 3.8 and later version:

# ansible-playbook [-i /path/to/inventory] \

    ~/openshift-ansible/playbooks/deploy\_cluster.yml

1. Verify your installation: <https://docs.openshift.org/3.7/install_config/install/advanced_install.html#Verifying%20the%20Installatio>

**IMPORTANT NOTE:** *HPE 3PAR Volume Plug-in for Docker is only tested against RPM-based OpenShift Origin 3.7. Containerized version of OpenShift Origin/Enterprise 3.7 and RPM based OpenShift Enterprise 3.7 might work, but have not yet been tested and verified by the volume plugin engineering team.*

**Installation of Etcd instance/cluster**

If you are planning to setup a 3 node secured etcd cluster, follow this guide: <https://github.com/hpe-storage/python-hpedockerplugin/blob/master/docs/etcd_cluster_setup.md>

$ export HostIP=<host\_ip>

$ docker run -d --name "etcd" \

-p 40010:40010 -p 23800:23800 -p 23790:23790 \

"quay.io/coreos/etcd:v3.2.10" \

"/usr/local/bin/etcd" \

-name "etcd0" \

-advertise-client-urls "http://${HostIP}:23790,http://${HostIP}:40010" \

-listen-client-urls "http://0.0.0.0:23790,http://0.0.0.0:40010" \

-initial-advertise-peer-urls "http://${HostIP}:23800" \

-listen-peer-urls "http://0.0.0.0:23800" \

-initial-cluster-token "etcd-cluster-1" \

-initial-cluster "etcd0=http://${HostIP}:23800" \

-initial-cluster-state "new"

NOTE: Ports used here must be opened and different than default ports (2379 and 4001). This is because default ports are used by internal e etcd instance of OpenShift cluster and HPE 3PAR Docker volume plugin won’t be able to communicate to this internal etcd instance without client certificates and keys which are stored in /etc/origin/master path (for OpenShift Origin).

### On the other hand, if you are setting up Kubernetes cluster, you can use the same pre-created etcd instance using kubeadm init with a configuration file.

This file is passed in the --config option.

apiVersion: kubeadm.k8s.io/v1alpha1

kind: MasterConfiguration

api:

advertiseAddress: <address|string>

controlPlaneEndpoint: <string>

bindPort: <int>

etcd:

endpoints:

- <endpoint1|string>

- <endpoint2|string>

caFile: <path|string>

certFile: <path|string>

keyFile: <path|string>

dataDir: <path|string>

extraArgs:

<argument>: <value|string>

<argument>: <value|string>

image: <string>

.

.

.

.

**Installation of HPE 3PAR Volume Plug-in for Docker**

Install Docker volume managed/legacy plugin on all the worker nodes.

For k8s 1.7 and below, install legacy plugin (version 1) using this guide:  <https://github.com/hpe-storage/python-hpedockerplugin/tree/plugin_v2/quick-start>

For K8S 1.8 and above, install managed plugin (version2) using this guide:  <https://github.com/hpe-storage/python-hpedockerplugin/blob/master/quick-start/README.md>

OpenShift 3.7 has K8s 1.7.6 and Docker 1.12.6, so you can install legacy docker volume plugin (version 1) here. Docker 1.12.6 does not support managed plugin system.

**Build, Install and Configure Dory/Doryd binaries**

There are two ways to build dory/doryd binaries: one way is an automated way via dory\_installer which is self-extracted binary and other way is to perform all the required steps manually.

**Automated tool for building dory/doryd binaries**

We have created an automated tool which is called dory\_installer which will get dory and doryd binaries in the required path. Run below steps on **all the schedulable nodes** (where Pods would get deployed) to use automated tool:

wget <https://github.com/hpe-storage/python-hpedockerplugin/raw/master/dory_installer>

chmod u+x ./dory\_installer

sudo ./dory\_installer

# Confirm if the binaries are installed properly in this location

$ ls -l /usr/libexec/kubernetes/kubelet-plugins/volume/exec/dev.hpe.com~hpe/

total 52360

-rwxr-xr-x. 1 docker docker 47046107 Apr 20 06:11 doryd

-rwxr-xr-x. 1 docker docker  6561963 Apr 20 06:11 hpe

-rw-r--r--. 1 docker docker      237 Apr 20 06:11 hpe.json

Once the installation is done, use this command to start the doryd process - dynamic provisioner.

sudo /usr/libexec/kubernetes/kubelet-plugins/volume/exec/dev.hpe.com~hpe/doryd  /etc/kubernetes/admin.conf dev.hpe.com

**Build, Install and Configure Dory binary (Manual Steps)**

Build and configure dory binary on all the schedulable master/worker nodes. Dory is written in Go and requires golang on your machine. The following example installs the necessary tools and builds Dory on a RHEL 7.4 system:

sudo subscription-manager repos --enable=rhel-7-server-optional-rpms

sudo yum install -y golang make

git clone https://github.com/hpe-storage/dory.git

make gettools

make dory

You should end up with a dory executable in the ./bin directory and be ready for installation.

cd /usr/libexec/kubernetes/kubelet-plugins/volume/exec/

mkdir dev.hpe.com~hpe

cd dev.hpe.com~hpe

cp <path>/dory hpe

# create a file called hpe.json in this folder

Dory looks for a configuration file with the same name as the executable with a .json extension. Following the example above, the configuration file would be /usr/libexec/kubernetes/kubelet-plugins/volume/exec/dev.hpe.com~hpe/hpe.json

**Contents of hpe.json**

{

"dockerVolumePluginSocketPath": "/run/docker/plugins/hpe.sock",

"logFilePath": "/var/log/dory.log"

"logDebug": true,

"supportsCapabilities": true,

"stripK8sFromOptions": true,

"createVolumes": true,

"listOfStorageResourceOptions": ["size"]

}

* dockerVolumePluginSocketPath: For legacy docker volume plugin (version 1), this is used to tell Dory where the socket file is located for the Docker Volume Plugin. For managed docker volume plugin (version 2), the name of the plugin used in place of the path to the socket file e.g. “hpe:latest” where plugin has been aliased to “hpe”.

To provision and mount volumes, you need to restart the kubelet node service if you're using K8s < 1.8.

If running Kubernetes:

$ sudo systemctl restart kubelet‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍

If running OpenShift on atomic host:

$ sudo systemctl restart atomic-openshift-node‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍

If running OpenShift origin:

$ sudo systemctl restart origin-node‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍

If everything works fine, you should be able to inspect your log file for successful initialization (/var/log/dory.log):

Info : 2017/09/18 16:37:40 dory.go:52: [127775] entry  : Driver=hpe Version=1.0.0-ae48ca4c Socket=/run/docker/plugins/hpe.sock Overridden=true  
Info : 2017/09/18 16:37:40 dory.go:55: [127775] request: init []  
Info : 2017/09/18 16:37:40 dory.go:58: [127775] reply  : init []: {"status":"Success"}‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍

For more information, study below Dory guide:

<https://github.com/hpe-storage/dory/blob/master/docs/dory/README.md>

**Build, Install and Configure Doryd binary (Manual Steps)**

Build and configure doryd binary on all master nodes. There are two ways provided to build Doryd. A host machine build and a fully containerized build. The containerized build require Docker 17.05 or newer on both client and daemon.

**Host build**

Doryd is written in Go and requires golang on your machine. The following example installs the necessary tools and builds Doryd on a RHEL 7.4 system:

sudo subscription-manager repos --enable=rhel-7-server-optional-rpms

sudo yum install -y golang make

git clone https://github.com/hpe-storage/dory.git

make gettools

make vendor

make doryd

You should end up with a doryd executable in ./bin directory.

The doryd binary needs access to the cluster via a kubeconfig file which is available on path: /etc/kubernetes/admin.conf

Doryd command line in OpenShift platform which is known to work and start the doryd process is:

sudo ./bin/doryd /root/.kube/config dev.hpe.com

OR

sudo ./bin/doryd /etc/kubernetes/admin.conf dev.hpe.com

**Containerized build**

Building Doryd in a container uses a multi-stage build and only require Docker 17.05 or newer.

docker build -t doryd:latest https://raw.githubusercontent.com/hpe-storage/dory/master/build/docker/doryd/Dockerfile.staged

Doryd is available on Docker Hub and an [example DaemonSet specification is](https://github.com/hpe-storage/dory/blob/master/examples/ds-doryd.yaml) available. /etc/kubernetes/admin.conf file needs to exist on all nodes prior to deploying the DaemonSet.

**NOTE**: *This is not validated with HPE 3PAR Volume Plug-in for Docker yet. We will update information about Doryd daemonset once it is properly tested and validated.*

For more information, study below Doryd guide:

<https://github.com/hpe-storage/dory/blob/master/docs/doryd/README.md>

**Usage of Dory (FlexVolume Driver, Just-In-Time Provisioner)**

Dory is a Kubernetes [FlexVolume Plugin driver](https://kubernetes.io/docs/concepts/storage/volumes/" \l "out-of-tree-volume-plugins) that leverages the [Docker Volume API](https://docs.docker.com/engine/extend/plugins_volume/) and allows Kubernetes to leverage Docker Volume plugins, both [legacy](https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins) and [managed](https://store.docker.com/search?category=volume&q=&type=plugin).

In OpenShift environment, to relax the security in your cluster so that pods are allowed to use the hostPath volume plug-in without granting everyone access to the privileged SCC:

1. Edit the restricted SCC:

$ oc edit scc restricted

1. Add allowHostDirVolumePlugin: true.
2. Save the changes
3. Restart node service.

“Kubectl” is a command line interface for running commands against Kubernetes clusters. Similarly, “oc” is a command line interface for running commands against OpenShift platform.

Below is an example yaml specification to create PersistantVolume.

$ sudo kubectl create -f - << EOF

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv1

spec:

capacity:

storage: 20Gi

accessModes:

- ReadWriteOnce

flexVolume:

driver: dev.hpe.com/hpe

options:

name: hpe\_volume

size: "20"

EOF

Under **flexVolume** spec, we need to mention the docker volume provisioner details in “**driver**” attribute. In “**options**” section, we can mention all the required driver options. Valid values are : 'name’, ‘compression', 'size', 'provisioning', 'flash-cache', 'cloneOf', 'virtualCopyOf', 'expirationHours', 'retentionHours', 'qos-name', 'mountConflictDelay'

The valid values of access modes are:

1. ReadWriteOnce – the volume can be mounted as read-write by a single node
2. ReadOnlyMany – the volume can be mounted read-only by many nodes
3. ReadWriteMany – the volume can be mounted as read-write by many nodes

**Here, value of “accessModes” should always be set to ReadWriteOnce as HPE 3PAR Docker Volume can be mounted as read-write by a single node at given point of time.**

Now, let’s create a claim (PVC) against the above volume:

$ sudo kubectl create -f - << EOF

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: pvc1

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 20Gi

EOF

You should notice that no actual volume is created after both the above steps. The FlexVolume driver is very basic and Docker Volume will get created during mount phase.

Now, we will create a pod with “nginx” container that requires some persistent storage:

$ sudo kubectl create -f - << EOF

---

apiVersion: v1

kind: Pod

metadata:

name: pod1

spec:

containers:

- name: nginx

image: nginx

volumeMounts:

- name: export

mountPath: /export

restartPolicy: Always

volumes:

- name: export

persistentVolumeClaim:

claimName: pvc1

EOF

When the pod gets created and a mount request comes in, you should see the actual volume created:

$ docker volume ls  
DRIVER              VOLUME NAME  
hpe               hpe\_volume‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍‍

On the Kubernetes/OpenShift side, it should now look something like this:

$ kubectl get pv,pvc,pod -o wide  
NAME       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM            STORAGECLASS   REASON   AGE  
pv/pv1    20Gi       RWO           Retain          Bound     default/pvc1                            11m  
  
NAME         STATUS    VOLUME    CAPACITY   ACCESSMODES   STORAGECLASS   AGE  
pvc/pvc1    Bound     pv100     20Gi       RWO                          11m  
  
NAME                          READY     STATUS    RESTARTS   AGE       IP             NODE  
po/pod1                       1/1       Running   0          11m       10.128.1.53    cld6b16

Now, you should delete the pod to unmount the docker volume. For deleting the docker volume, manual clean-up is required. You will see docker volume present even after deleting PV and PVC. You need to delete docker volume with the help of “docker volume rm <volume-name>” command from the node where docker volume plugin is enabled.

**Usage of Doryd (Dynamic Provisioner)**

Dynamic volume provisioning allows storage volumes to be created on-demand. To enable dynamic provisioning, a cluster administrator needs to pre-create one or more StorageClass objects for users. StorageClass objects define which provisioner should be used and what parameters should be passed to that provisioner when dynamic provisioning is invoked.

This provisioner is a very simple daemon that listens for PVCs and satisfies those claims based on the defined SCs.

The following manifest creates a storage class “sc1” which provisions compressed volume with the help of HPE 3PAR Volume Plug-in for Docker.

$ sudo oc create -f - << EOF

---

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: sc1

provisioner: dev.hpe.com/hpe

parameters:

size: "16"

compression: "true"

EOF

We need to mention the docker volume provisioner details in “**provisioner**” attribute. In “**parameters**” section, we can mention all the required driver options. Valid values are : 'name’, ‘compression', 'size', 'provisioning', 'flash-cache', 'cloneOf', 'virtualCopyOf', 'expirationHours', 'retentionHours', 'qos-name', 'mountConflictDelay'

Now, let’s create a claim (PVC) against the above volume:

$ sudo kubectl create -f - << EOF

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: pvc1

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 16Gi

storageClassName: sc1

EOF

You should observe that the PersistentVolume and actual docker volume is created after both the above steps.

Now, we will create a pod with nginx container that requires some persistent storage:

$ sudo kubectl create -f - << EOF

---

apiVersion: v1

kind: Pod

metadata:

name: pod1

spec:

containers:

- name: nginx

image: nginx

volumeMounts:

- name: export

mountPath: /export

restartPolicy: Always

volumes:

- name: export

persistentVolumeClaim:

claimName: pvc1

EOF

You can confirm the docker volume created using docker volume ls command:

$ docker volume ls  
DRIVER              VOLUME NAME  
hpe               sc1-fc0054ea-2920-11e8-9d58-ecb1d7a4b070

On the Kubernetes/OpenShift side, it should now look something like this:

$ sudo oc get pod,pvc,pv –o wide

NAME CAPACITY ACCESSMODES RECLAIMPOLICY STATUS CLAIM STORAGECLASS REASON AGE

8s

pv/sc1-fc0054ea-2920-11e8-9d58-ecb1d7a4b070 10Gi RWO Delete Bound default/pvc1 sc1 41s

NAME STATUS VOLUME CAPACITY ACCESSMODES STORAGECLASS AGE

pvc/pvc1 Bound sc1-fc0054ea-2920-11e8-9d58-ecb1d7a4b070 10Gi RWO sc1 42s

NAME READY STATUS RESTARTS AGE IP NODE

po/docker-registry-1-tdjl8 1/1 Running 1 15d 10.129.0.5 openshift-master-node1

po/router-1-z5jhl 1/1 Running 1 15d 10.50.9.17 openshift-master-node1

po/pod1                      1/1       Running   0          17m       10.128.1.98  openshift-worker-node2

Now, you should delete the pod to unmount the docker volume. For deleting the docker volume, manual clean-up is not required because doryd provides automatic clean-up. You can just delete the PersistentVolumeClaim, and observe PersistentVolume and docker volume is automatically deleted. This is how dynamic storage provisioning can be attain using doryd.

**Scaling Application Up and Down**

Controller Manager component of Kubernetes ensures that the cluster’s desired state matches the current state by scaling workloads up and down. This can be achieved by using different types of controllers like ReplicationController, ReplicaSets, Deployments, etc

Controllers create pods on behalf of a user in OpenShift platform. For pods created on behalf of a user, **in most cases** by the system itself, **access should be given to a service account** under which related controller is operated upon.

In this case, an administrator might want to allow users or groups outside the administrator group access to create more privileged pods. To do so, you can:

1. Determine the user or group you would like to have access to the SCC.
2. Run:

$ oc adm policy add-scc-to-user <scc\_name> <user\_name>

$ oc adm policy add-scc-to-group <scc\_name> <group\_name>

Below is the example for a default namespace:

$ sudo oc adm policy add-scc-to-user privileged system:serviceaccount:default:default

$ sudo oc adm policy add-scc-to-group anyuid system:authenticated

If you are planning to scale up the pods to multiple instances which have a same 3PAR volume mounted, with HPE 3PAR Volume Plug-in for Docker, make sure all the pods are running within the same node by disabling the scheduling for all the nodes but one, OR with the help of node selector and node affinity. Node Selector and node affinity are the forms of constraints which constrain a [pod](https://kubernetes.io/docs/concepts/workloads/pods/pod/) to only be able to run on particular [nodes](https://kubernetes.io/docs/concepts/architecture/nodes/) or to prefer to run on particular nodes.

You would see the node status as below if you keep one node with scheduling enabled:

$ sudo oc get nodes

NAME                     STATUS                     AGE       VERSION

openshift-master-node1   Ready,SchedulingDisabled   31d       v1.7.6+a08f5eeb62

openshift-worker-node2   Ready                      31d       v1.7.6+a08f5eeb62

openshift-worker-node3   Ready,SchedulingDisabled   31d       v1.7.6+a08f5eeb62

After that, a ReplicationController can be created and you can scale the pods with replicas as ‘3’ where PVC will get mounted to three different Pods (scheduled on the same node).

$ sudo oc scale --replicas=3 rc/nginx

replicationcontroller "nginx" scaled

$ sudo oc get pods –o wide

NAME                         READY     STATUS    RESTARTS   AGE       IP            NODE

po/docker-registry-1-tdjl8   1/1       Running   1          31d       10.129.0.5    openshift-master-node1

po/nginx-fsh9c               1/1       Running   0          2m        10.130.0.43   openshift-worker-node3

po/nginx-jwskt               1/1       Running   0          2m        10.128.0.33   openshift-worker-node3

po/nginx-zdf5r               1/1       Running   0          2m        10.128.0.34   openshift-worker-node3

po/router-1-z5jhl            1/1       Running   1          31d       10.50.9.17    openshift-master-node1

If scheduling is enabled on all the nodes, you can see the status of nodes as below:

$ sudo oc get nodes

NAME                     STATUS                     AGE       VERSION

openshift-master-node1   Ready   31d       v1.7.6+a08f5eeb62

openshift-worker-node2   Ready                      31d       v1.7.6+a08f5eeb62

openshift-worker-node3   Ready    31d       v1.7.6+a08f5eeb62

Now, if ReplicationController is scaled with replicas as ‘3’ without nodeSelector constraint where PVC will get mounted to three different Pods (scheduled on the three different nodes).

$ sudo oc scale --replicas=3 rc/nginx

replicationcontroller "nginx" scaled

$ sudo oc get pods –o wide

NAME                         READY     STATUS    RESTARTS   AGE       IP            NODE

po/docker-registry-1-tdjl8   1/1       Running   1          31d       10.129.0.5    openshift-master-node1

po/nginx-fsh9c               1/1       Running   0          9m        10.130.0.45   openshift-worker-node2

po/nginx-jwskt               1/1       Running   0          7m        10.128.0.37   openshift-worker-node3

po/nginx-zdf5r               1/1       Running   0          6m        10.128.0.38   openshift-worker-node1

po/router-1-z5jhl            1/1       Running   1          31d       10.50.9.17    openshift-master-node1

**You will not see error on OpenShift but the 3PAR Docker volume is actually mounted to only ONE POD in the most recent node**. This is because 3PAR Docker volume cannot be mounted on two or more nodes concurrently as mentioned earlier in access modes details of “Usage of Dory” section and mountConflictDelay feature will come into play in this case.

**Known limitations of HPE 3PAR Volume Plug-in for Docker**

**Issue #23:**

The /run/docker/plugins/hpe/hpe.sock and /run/docker/plugins/hpe/hpe.sock.lock files are not automatically removed when you stop the container. Therefore, these files will need to be removed between each run of the plugin.

Documented as IMPORTANT NOTE in this link: <https://github.com/hpe-storage/python-hpedockerplugin/tree/plugin_v2/quick-start>

**Issue #100:**

Inappropriate QoS rules can cause delay in creating file system and hamper the mount operation subsequently.

Following QoS rules is a valid example which can be configured in pre-existed VVSet:

Min IOPS: 10000 IOs/sec

Min IOPS: 2000000 IOs/sec

Min BWS: 30 MB/sec

Max BWS: 40 MB/sec

Latency: 500 seconds

Priority: High, Normal or Low

**Issue #117:**

Cloned volume created using -cloneOf=<src\_vol> does'nt have Comment field populated on 3PAR Volume. This issue will not be observed when size of a cloned volume is greater than source volume.

**Issue #99 and Issue #151**

Summary

After performing enable and disable of a Docker plugin repeatedly, system reports “CPU Stuck Error”

Root cause

When Docker plugin is enabled, a bind-mount entry with source as “/etc/hosts” gets created as can be seen below.



After performing repeated enable / disable Docker commands, duplicate bind-mount entries get added and over a period of time, when these duplicate entries reach over hundred thousand, the system throws “CPU stuck error”.

As you can see below, duplicate bind-mount entries are growing:



Observation

It has been observed that after plugin is enabled when few operations like create, mount, unmount, etc. are performed, attempting to disable the plugin **without** force flag results in “resource in use” kind of message. And upon using force flag, disable is unable to clear bind-mount entry. Once the setup is in this condition, further enable and (forced) disable make the bind-mount entries swell (almost doubles up the previous count).

Resolution

Removing the bind-mounts periodically resolves the issue.

GitHub Issue

This was originally reported on GitHub which can be viewed at <https://github.com/hpe-storage/python-hpedockerplugin/issues/151>

Docker Ticket

Additional information can be found at <https://support.docker.com/5000f00001HyPpu>

Use the following credentials to view the ticket:

**User Id:** sandanar **Password:** Welcome2Hp!!

Docker team has also filed an internal issue #563 to track the issue.

RHEL Ticket

RHEL defect has also been raised by HPE which can be viewed at: <https://access.redhat.com/support/cases/#/case/02071131>

List of OS under which this is observed:

RHEL, Ubuntu, and CentOS

**Issue #157**

# When mountConflictDelay driver option is set to => 60 seconds and mount operation is requested for the volume from the different node/host, Docker engine can throw timeout error: “context deadline exceeded”

**Issue #172**

When docker volume driver options are configured in PersistentVolume (FlexVolume Driver) and StorageClass (Dynamic Provisioner) yaml specifications, make sure that they have valid values. If invalid values are set, Pod can get stuck with ContainerCreating state infinitely.

For example, in PersistentVolume yaml specification, if qos-name option is specified with VVSet which is not associated with any QoS rule in 3PAR array and Pod is created out of PersistentVolumeClaim (which is bound to mentioned PersistentVolume), this Pod can get stuck with ContainerCreating state infinitely.