Hewlett Packard Enterprise

Technical white paper

Check if the document is available in the language of your choice.

DEPLOYING A HIGHLY AVAILABLE CONTAINERIZED DATABASE ON HPE NIMBLE STORAGE WITH HPE CONTAINER PLATFORM 5.0

Using Kubernetes Clusters

CONTENTS

EXECUTIVE SUMMARY

Kubernetes container management setup varies greatly as no two installations are the same. Cluster installation can be done manually, but there are many possibilities for mistakes, and an automated process is much less error prone. Following the official Kubernetes documentation can be daunting—even for Kubernetes professionals. Setting up machines exactly the same way, configuring networking, installing drivers and different software versions can be problematic—even with automated scripts—because configurations can become out of date due to the highly volatile nature of Kubernetes and the container development cycle. Minor version changes can easily cause failures and add enormous complexity to a cluster rollout.

Using HPE Container Platform (HCP) to deploy a Kubernetes cluster ensures that all machines are automatically configured the same way and can be added to a cluster without having master-level knowledge of Linux®, Docker, and Kubernetes, when compared to a manual install. HCP comes with built-in support for HPE Nimble Storage, while the HPE Container Storage Interface (CSI) supplies a foundation for future HPE storage arrays in Kubernetes.

Target audience: This white paper is intended for storage and system administrators looking to set up Kubernetes clusters using HPE Nimble Storage.

SOLUTION OVERVIEW

This solution builds a database environment in a Kubernetes cluster using HPE Nimble Storage for persistent container storage. The environment consists of the following:

- Benefits of HCP:
  - Automated Kubernetes cluster creation from a single control point
  - Integrated dashboard of cluster health
  - Takes care of many prerequisites and manual steps for installing Kubernetes
  - More efficient use of hardware – use of all CPU/Memory between Kubernetes applications
  - Better power and cooling efficiency – less overhead for Kubernetes as resources can be scaled down when less load is required
- Benefits of HPE CSI Driver and HPE Nimble Storage Container Storage Provider (CSP):
  - The HPE CSI driver is an Open Source, multi-platform and multi-vendor container storage interface driver for Kubernetes. With HPE CSI, many data management capabilities become API objects within Kubernetes.
  - Dedupe and compression of application volumes results in significant space saving.
  - When using HPE Nimble replication to another array (or HPE Cloud Volumes), an application administrator can bring up the same instance in another location and run workloads against the same data—offloading the tasks from the primary production systems.
  - An additional benefit to using HPE Nimble Storage is that the volume data is portable outside of Kubernetes. No data is “trapped” in the compute nodes or the Kubernetes cluster. A new cluster on new hardware can be installed, with the ability to import the HPE Nimble Storage volume.
- Demonstrate ease of using HCP to deploy Kubernetes and with native HPE CSI/CSP integration.
- Ease of removing Kubernetes clusters and redeploying without having to manually clean up the old installation—in minutes.

FIGURE 1 shows the solution environment that contains the application in a Kubernetes cluster using HPE Nimble Storage.

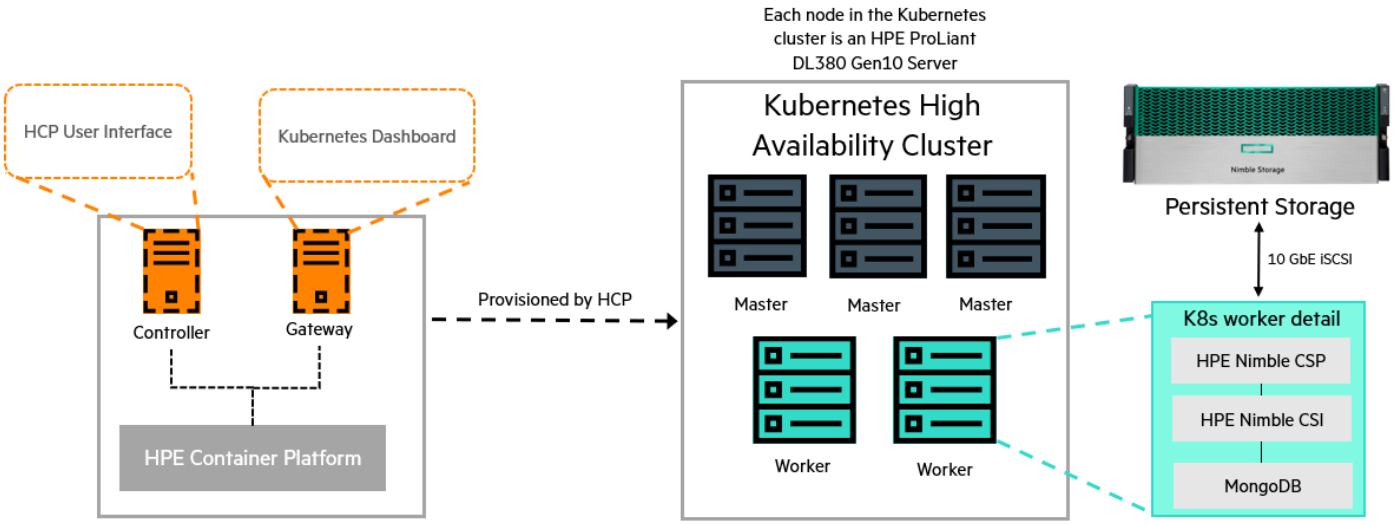

FIGURE 1. HCP and Kubernetes cluster architecture

SOLUTION COMPONENTS

HPE Container Platform and Kubernetes cluster details:

- HPE Container Platform requires a license, but a trial 90-day license can be obtained here.
- CentOS is license free, Open Source, and does not come with support. Red Hat® Enterprise Linux can also be used if licensed support is required.
- The latest HPE DL380 Gen10 servers were used but are not a requirement for HCP or Kubernetes nodes.
- Any model of HPE Nimble Storage could be used in place of the HPE Nimble Storage AF1000 array that was used in this configuration. All models of the HPE Nimble Storage array use the same NimbleOS and should behave the same, with the only difference being performance. Your organization's performance requirements might align with a different array model.

Hardware

- HPE ProLiant DL380 Gen10 server
  - Single Intel® Xeon® Gold 6230 CPU @ 2.10GHz 20/20 cores; 40 threads
  - 384 GB of memory
  - HPE Ethernet 10/25 Gb 2p 640FLR-SFP28 Adapter (for iSCSI data)
- HPE Nimble AF1000 array running NimbleOS 5.1.4.0 with iSCSI over 10 Gb

Software

- CentOS 7.7 (1908) operating system—minimal install—Open-Source and binary compatible stream of Red Hat Enterprise Linux (RHEL)
- HPE Container Platform 5.0 with integrated HPE CSI/CSP driver v1.0. Persistent storage volumes are provisioned, bound, and managed by Kubernetes using the new Container Storage Interface (CSI) standard. Complete HPE Container Platform documentation about setting up HCP is available at: docs.bluedata.com/home.
- Kubernetes v1.17—designed by Google™ and maintained by the Cloud Native Computing Foundation (CNCF)
- MongoDB 4.2.6 (using StatefulSet configuration)
- NimbleOS 5.0.8.x, 5.1.3.x or 5.1.4.x

Kubernetes cluster details

- Three master nodes for High Availability (HA)
- Two worker nodes (where the database application is installed)
- 10 Gb iSCSI network on separate VLAN

BEST PRACTICES AND CONFIGURATION GUIDANCE FOR THE SOLUTION

To configure the Kubernetes environment, perform the instructions in the following subsections. If you already have a Kubernetes environment (1.17 or newer) you can skip to the Prepare persistent storage subsection.

- Image servers and apply OS prerequisites (for prerequisites, see Kubernetes Host Requirements)
- Install and configure the HPE Container Platform (for details, see Installation Overview)
- Deploy Kubernetes cluster using HCP
- Prepare persistent storage

Image servers and apply OS prerequisites

All machines in this environment are imaged and updated with CentOS 7.7. HPE Container Platform also supports Red Hat. The following procedure is a summary of recommendations made in the HPE Container Platform documentation.

NOTE

When deploying the Linux OS, make sure disk partitions meet the HCP requirements and are sized appropriately to the Controller and Gateway machines to avoid partition shuffling post-install.

---

1. Drive and partition layout for Controller, Gateway, and Kubernetes hosts (see the *Storage Recommendations* portion of the HPE Container Platform documentation for more information):
   a. 500 GB disk: The boot disk needs partitioning different from what is partitioned by default during CentOS install (see FIGURE 2). Start with the automatic partitioning, and then add the `/var` partition and adjust sizes:
      I. 1 GB <boot>
      II. 200 GB <root>
      III. 100 GB <home>
      IV. 100 GB <var>

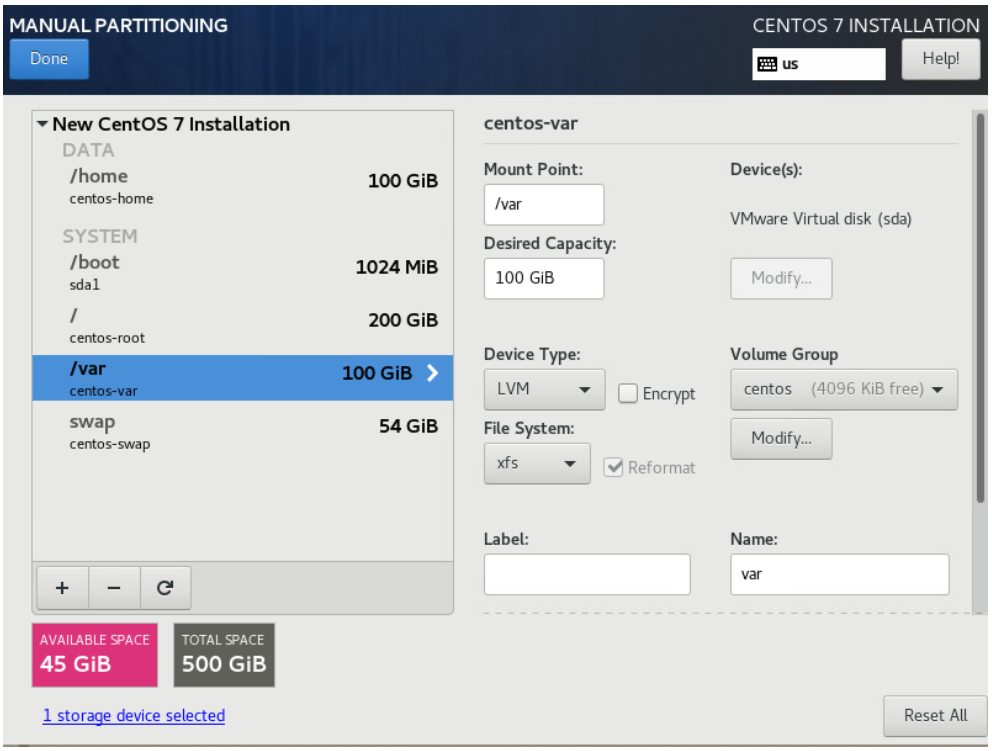

FIGURE 2. Manual partitioning layout

   b. 600 GB – Ephemeral storage (not required for HCP Gateway)
   c. 200 GB – Persistent storage (not required for HCP Gateway)
2. Appropriate memory and CPU (for details, see Host Requirements). In particular:
   a. Controller: 4–16 cores and 32–192 GB RAM
   b. Gateway: 8 cores and 32 GB RAM
   c. Kubernetes hosts: 4–16 cores and 32–192 GB RAM
3. High Availability (HA) Considerations:
   a. If HCP HA is desired (for details, see High Availability), two more hosts are needed (for a total of four hosts for HCP).
   b. For Kubernetes HA, a minimum of five nodes are needed—three masters and two workers.
4. Image each of the machines with CentOS or RHEL 7.7. CentOS was used in this guide.
5. Configure DNS with entries for an HCP Controller machine, HCP Gateway machine, and each Kubernetes host.
6. Ensure the nameserver and search path are in `/etc/resolv.conf`.
7. Define the hostname as an FQDN on all machines.
8. Customize `/etc/profile.d/set_proxy.sh`, see APPENDIX A – Code examples for details.
   The `no_proxy` section is critical. In particular, IP *and* FQDN must be specified for all hosts in the cluster.
   Place the same file on each machine.
   Source this script or log out of the active session to load the variables.
9. After the proxy environment variables are in place, use them to create the Docker proxy configuration. For details, see APPENDIX A – Code examples.
10. If needed in the environment, specify a proxy in `/etc/yum.conf`, for example:
    `proxy=http://<URL>:<PORT>`
11. Install the other supporting packages:
    `yum -y install bind-utils ntp wget`

12. A good general practice is to update the OS:
    `yum -y update`
13. Configure `/etc/ntpd.conf`—add your own NTP servers or ensure the default/public ones are functioning.
14. Setting up password-less SSH from the controller node and all other machines is helpful:
    `ssh-keygen`
    `ssh-copy-id -i ~/.ssh/id_rsa.pub root@<hostname>`
15. Set SELinux appropriately.
    a. The HPE Container Platform supports SELinux, but the mode **may not** be changed after the HPE Container Platform is installed. Reboot for any changes to take effect.
    b. The Kubernetes hosts are currently required to have SELinux set to permissive mode.
    Edit `/etc/selinux/config` and set `SELINUX=permissive`

Install and configure the HPE Container Platform

See Installation Overview for full details. The following is an outline of the requirements. Complete these steps before installing the HPE Container Platform components.

1. Obtain a temporary license from the My HPE Software Center website for the HPE Container Platform, if necessary.
2. Obtain the HPE Container Platform installation bundle (see Bundles for instructions).
3. Complete pre-checks by referencing Using the Pre-Check Script.
    a. This can be performed on the **Controller** node as well as the **Gateway** node.
4. This guide utilized password-less SSH, so the standard install script was executed without arguments.
5. After the HPE Container Platform UI is operational, log in (default credentials are “admin/admin123”) and provision the HPE Container Platform Gateway, as presented in FIGURE 3.

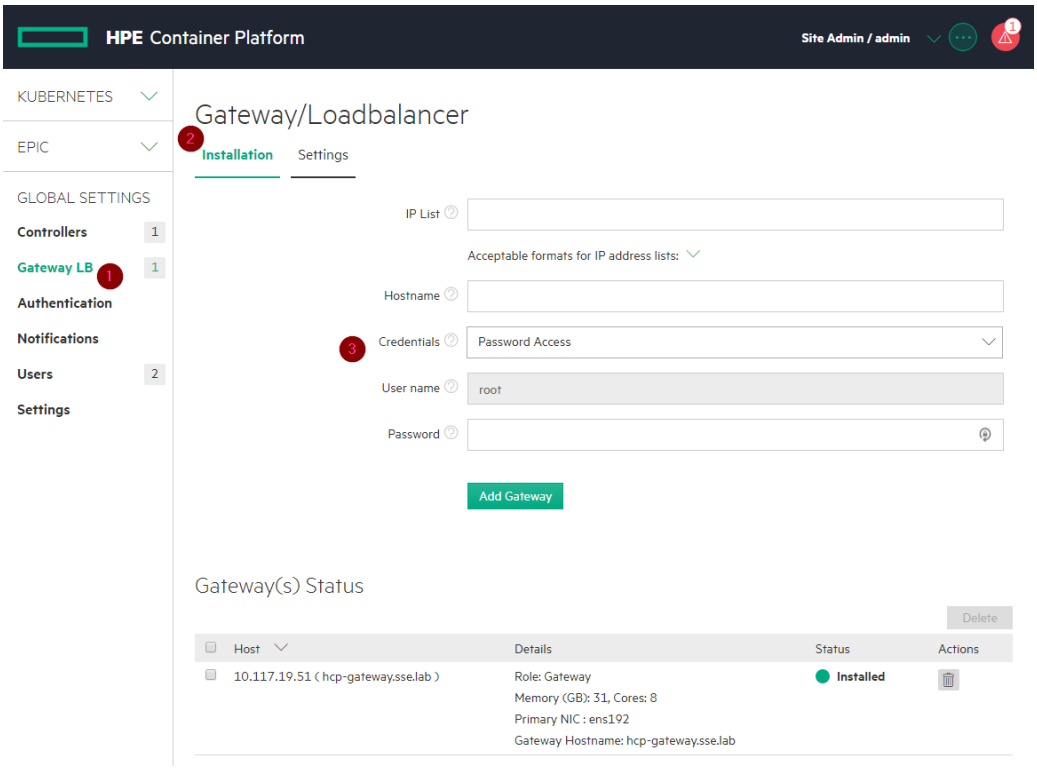

FIGURE 3. HCP Gateway setup

Deploy Kubernetes cluster using HCP

Deploying a highly available Kubernetes cluster is streamlined using the HPE Container Platform.

Add all hosts that will be used for Kubernetes

1. Add hosts to the HPE Container Platform for use as Kubernetes nodes. (See Kubernetes Host Step 2: Select the Host(s) and Kubernetes Host Step 3: Add the Host(s) for details).
   a. Click **Hosts** under the **KUBERNETES** section in the HPE Container Platform UI.
   b. Specify an IP range and credentials for import. This should include all hosts intended for Kubernetes master and worker nodes. Click **Submit**.
2. If not done already, apply the HPE Container Platform license:
   a. Under **Settings** in the left-side navigation, click **License**. Note the **Platform ID**.
   b. Log in to myenterpriselicense.hpe.com/cwp-ui/evaluation/R2E56-70001/5.0/ and find your entitlement for HPE Container Platform. Enter the platform ID in **step 3**.
   c. Download the resulting `.dat` file from the website.
   d. In the HPE Container Platform UI, click **Upload license**. You can now resume the process of adding Kubernetes nodes.
3. Edit storage (ephemeral and persistent) on each host as necessary. Confirm the desired local disks were selected (see Kubernetes Host Step 4: Select Hard Drives for details.)
4. If not already in Lockdown mode, place the HPE Container Platform into Lockdown mode (see Kubernetes Host Step 5: Enter Lockdown Mode for details.)
5. Choose all the hosts, and click the **Install** button (see Kubernetes Host Step 6: Add the Host(s) as Workers for details.) This step takes several minutes to complete.
6. Exit Lockdown mode: In the HCP UI, click  in the top right and select **Exit site lockdown**.
7. As exhibited in FIGURE 4, validate the Kubernetes installation (see Kubernetes Host Step 7: Validate the Worker Installation for details).

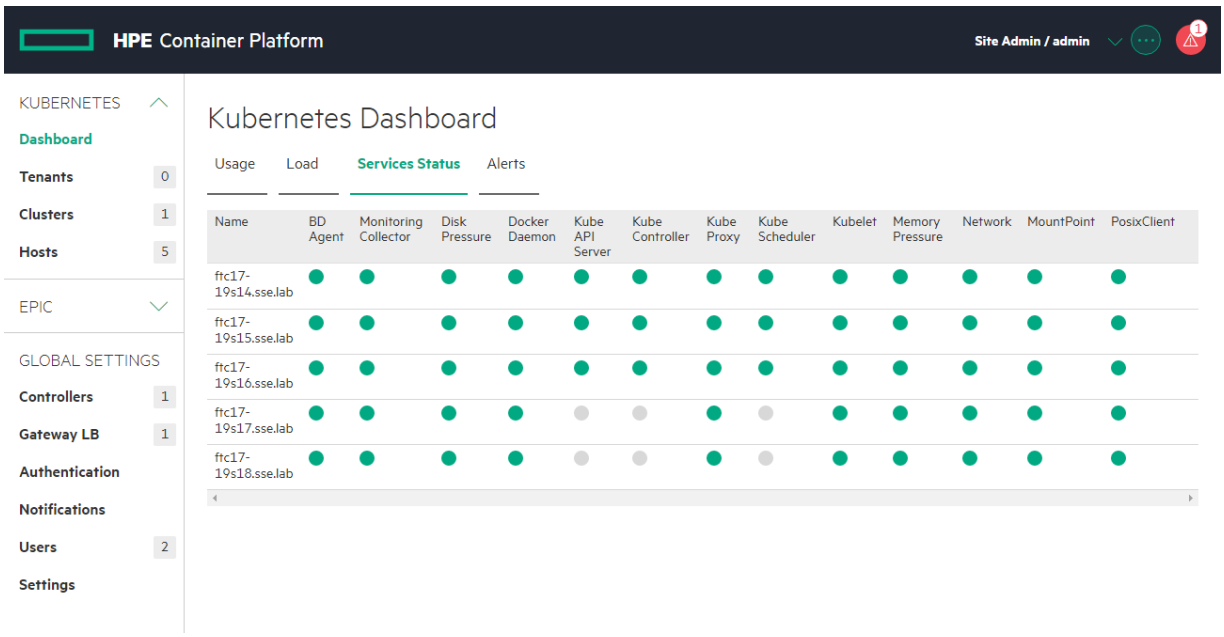

FIGURE 4. HCP Kubernetes Dashboard (not to be confused with the Kubernetes Dashboard running on the HCP Gateway)

Create a Kubernetes cluster

Next, create a Kubernetes cluster from the hosts that were added above. (See Creating a New Kubernetes Cluster for details). From the **KUBERNETES** menu, select **Clusters**, and then click the **Create Kubernetes Cluster** button. This will bring up three steps for cluster creation:

1. Host Configurations: Specify three masters (for HA) and at least two worker nodes.
2. Cluster configurations: Specify Kubernetes version and network. Make sure the checkbox to use HPE Nimble Storage is selected.
3. Summary: Review the roles for each machine are as expected (master vs. worker).

Deploy "kubectl" to manage Kubernetes cluster

After the Kubernetes cluster is deployed, install `kubectl` on a machine that has network connectivity to the Kubernetes cluster. In this environment, the **hcp-controller** node was designated. See Kubernetes Host Requirements for details.

Log in to the HPE Container Platform UI, and navigate to your Kubernetes cluster. The fifth button from the right, as displayed in FIGURE 5, allows you to **Download Admin Kubeconfig** for your cluster.

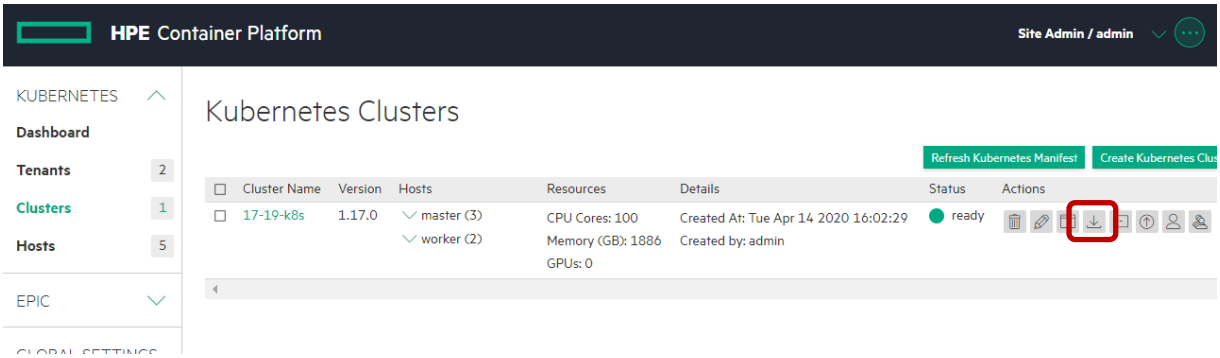

FIGURE 5. HCP Kubernetes Clusters view

After clicking the download button, save the file to `/root/.kube/config`.

1. Define the Kubernetes yum repository at `/etc/yum.repos.d/kubernetes.repo`. See the /etc/profile.d/set_proxy.sh script in the appendix for details.
2. Install the `kubectl` tool:
   `# yum install -y kubectl --disableexcludes=Kubernetes`
3. Test that `kubectl` can run commands against your Kubernetes cluster:

```
# kubectl cluster-info
Kubernetes master is running at https://hcp-gateway.sse.lab:10000…
```

4. Enable `kubectl` autocomplete, if desired. For details, see the kubectl Cheat Sheet.

The Kubernetes cluster is now running and can be configured with `kubectl`.

Visually manage Kubernetes with Kubernetes Dashboard

The Kubernetes cluster operational state is managed using the Kubernetes Dashboard. (See Accessing the Kubernetes Dashboard for details.) This dashboard is provided by Kubernetes and runs locally on each cluster. Storage classes, persistent volumes, services, and secrets are examples of what can be queried using this UI.

Launch the Kubernetes dashboard via the **Clusters** page from the HCP UI, as shown in FIGURE 6.

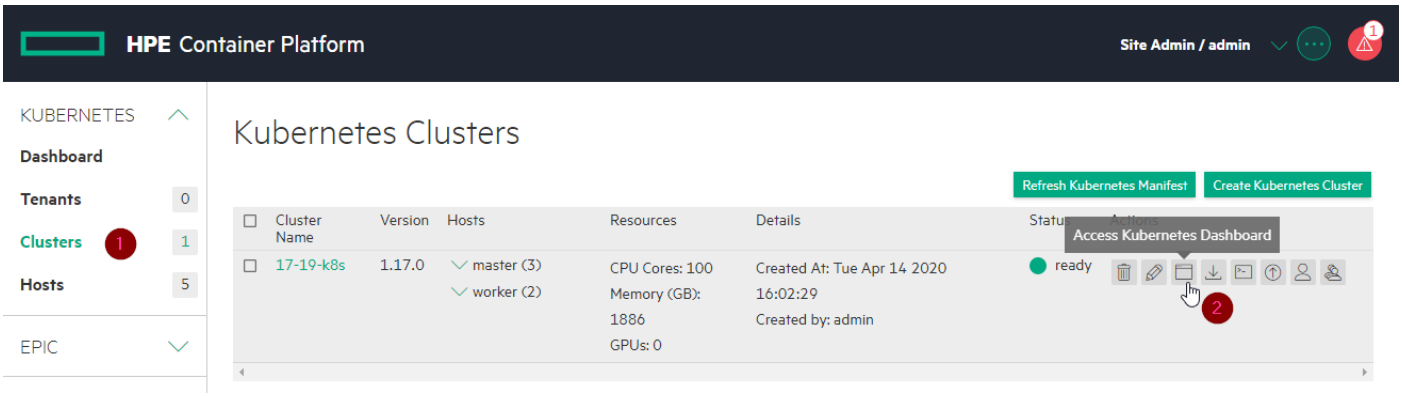

FIGURE 6. Accessing the Kubernetes Dashboard from the HCP Gateway

There is a prompt to transfer a token for authentication. This allows for easy login to the Kubernetes dashboard.

The Kubernetes dashboard has a default timeout of 15 minutes. This can be modified. See Increase (or disable) Kubernetes dashboard timeout for instructions.

Prepare persistent storage

The HPE CSI Driver and the HPE Nimble Storage CSP must be installed. The previous step of deploying a Kubernetes cluster with HCP allowed for this. If your Kubernetes cluster was provisioned in a different manner, install the HPE CSI Driver for Kubernetes. (See Deployment Overview for details.)

All `kubectl` commands are run from the "director" machine or whatever client that has kubeconfig access to the Kubernetes cluster.

Define a secret with storage array details (see Create a secret with backend details for information). Provide the array management IP, username, and base64-encoded password. See hpe-nimble-secret.yaml in *Appendix A* for a code example.

Create the secret with the `kubectl` command:

```
# kubectl create -f hpe-nimble-secret.yaml
```

Set up HCP Tenant for controlling resources to a cluster/namespace (optional but recommended)

Tenants allocate system resources for worker nodes in the cluster. See the official HCP documentation for more information about Tenants.

HCP has separate Tenants sections for both EPIC and Kubernetes. Because a Kubernetes cluster is being used, be sure to select the Tenants for Kubernetes, as shown in FIGURE 7.

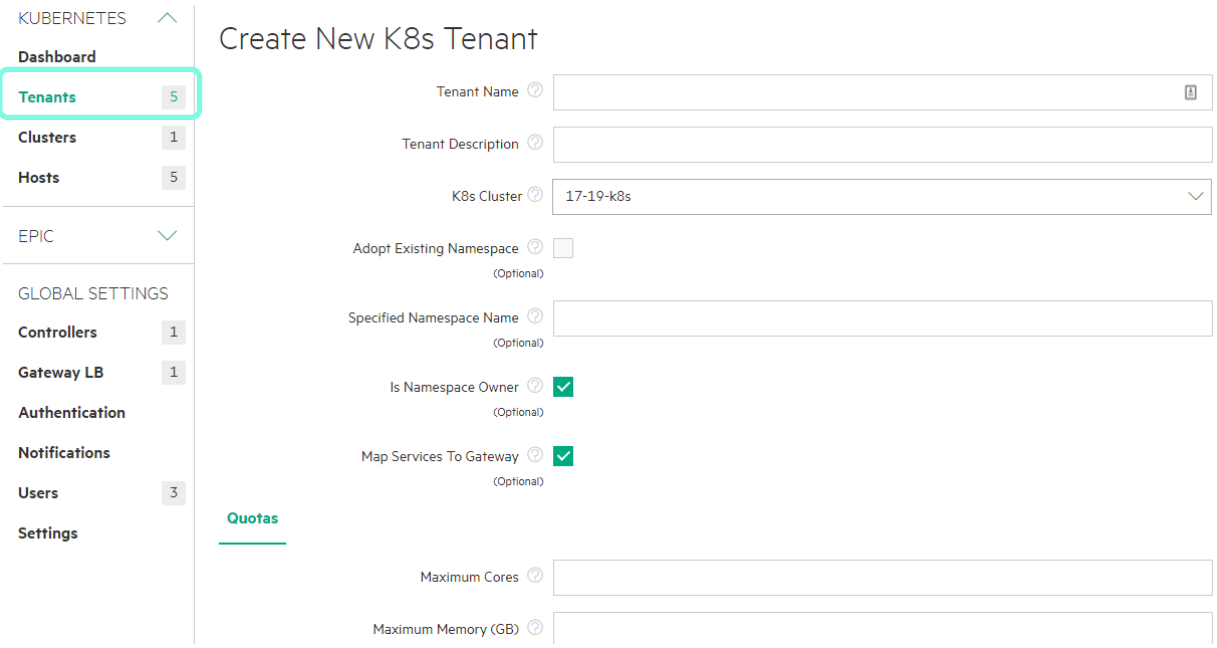

FIGURE 7. HCP Kubernetes Tenant creation

Using the above menu, a Tenant namespace named "mongodb" was created. No quotas or resource limits were set during creation, but this can be changed at any time by clicking the pencil icon (Edit Tenant) on the right, as shown in FIGURE 8.

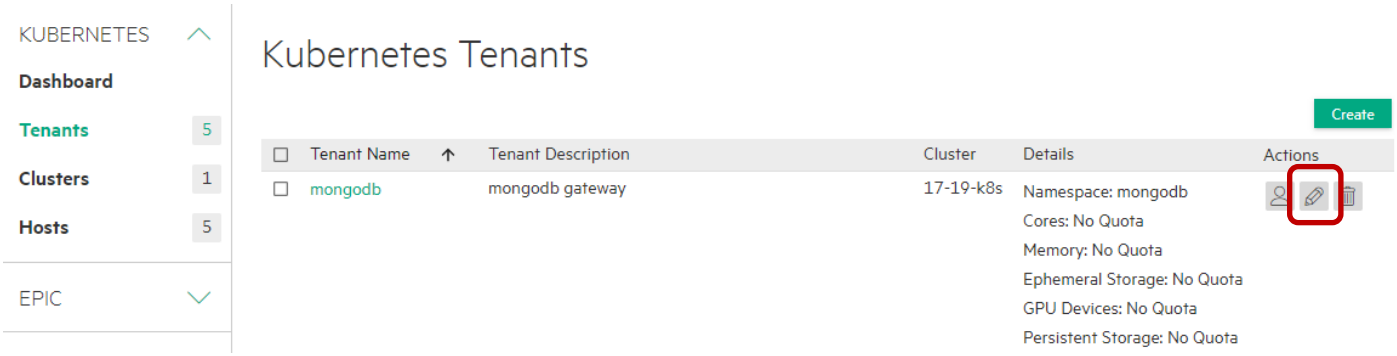

FIGURE 8. HCP "Kubernetes Tenant" view to edit a Tenant

Application installation

The following steps for the installation of the database application (MongoDB) were taken:

- Storage class creation (pointing to the HPE Nimble Storage array for persistent storage)
- Service creation
- StatefulSet creation
- Persistent Volume Claim template creation

The HCP Kubernetes Tenant namespace "mongodb" is used; therefore, the namespace context for `kubectl` should be changed to avoid having to type "`-namespace=mongodb`" at the end of every `kubectl` command.

```
# kubectl config set-context --current --namespace=mongodb
```

Storage Class creation

A storage class file named `sc-mongo.yaml` was created. This allows volume creation on the HPE Nimble Storage array using the HPE CSI Driver. In this example, limits for IOPS and bandwidth were not used.

```
apiVersion: storage.Kubernetes.io/v1
kind: StorageClass
metadata:
  name: sc-mongo
provisioner: csi.hpe.com
parameters:
  description: "Volume provisioned by the HPE CSI Driver for MongoDB"
  accessProtocol: "iscsi"
  csi.storage.Kubernetes.io/fstype: xfs
  csi.storage.Kubernetes.io/provisioner-secret-name: hpe-nimble-secret
  csi.storage.Kubernetes.io/provisioner-secret-namespace: kube-system
  csi.storage.Kubernetes.io/controller-publish-secret-name: hpe-nimble-secret
  csi.storage.Kubernetes.io/controller-publish-secret-namespace: kube-system
  csi.storage.Kubernetes.io/node-stage-secret-name: hpe-nimble-secret
  csi.storage.Kubernetes.io/node-stage-secret-namespace: kube-system
  csi.storage.Kubernetes.io/node-publish-secret-name: hpe-nimble-secret
  csi.storage.Kubernetes.io/node-publish-secret-namespace: kube-system
  csi.storage.Kubernetes.io/controller-expand-secret-name: hpe-nimble-secret
  csi.storage.Kubernetes.io/controller-expand-secret-namespace: kube-system
  # Extras...
  dedupeEnabled: "true"
  performancePolicy: "MongoDB"
# named sub-folder in the "default" pool on the Nimble array
folder: "Kubernetes-Mongo-Vols"
  thick: "false"
  destroyOnDelete: "true"
allowVolumeExpansion: True
```

TIP

When copying and pasting file and code examples from throughout this paper, please make sure that white space is preserved.

Some settings to note:

- `dedupeEnabled: "true"` – uses the HPE Nimble Storage array's deduplication process to reduce used space. As all application Pod volumes in this cluster are in the same dataset, when scaling Pods up (and thus creating new replica volumes), most of the data is deduplicated.
- `performancePolicy: "MongoDB"` – An optional custom Performance Policy for MongoDB was created on the HPE Nimble Storage array, allowing for different performance characteristics (different block size, dedupe and compression enabled or not) of the application volumes.
- `folder` – This is a folder under the default pool in the HPE Nimble Storage array. If you do not want to use a folder for organization, remove this key/value pair.
- `destroyOnDelete: "true"` – allows PVs to be automatically deleted from the HPE Nimble Storage array when removing them from the cluster (see Cleaning up the environment section), instead of having to manually delete them from the HPE Nimble Storage web UI or CLI.
- `allowVolumeExpansion: True` – allows the HPE Nimble Storage array to seamlessly expand the application volumes as database size increases over time, past the initial 28 GB that was deployed.

Declare the StorageClass. This will create the persistent volumes when the MongoDB StatefulSet is deployed.

```
# kubectl create -f sc-mongo.yaml
```

TIP
For a complete list of StorageClass parameters, see the official documentation for the HPE Nimble Storage CSP at: scod.hpedev.io/container_storage_provider/hpe_nimble_storage/index.html.

Headless Service, StatefulSet, and Persistent Volume Claims

This installation uses a StatefulSet, which maintains a sticky identity for each Pod. Pods are created from the same spec, each with a persistent identifier that is maintained across any rescheduling, and are not interchangeable: The persistent Pod identifiers allow it to match existing volumes to new Pods that replace any that have failed.

StatefulSet details:

- Storage for a given Pod must be provisioned by a Persistent Volume Provisioner (see the README at github.com/kubernetes/examples/tree/master/staging/persistent-volume-provisioning/README.md) based on the requested storage class. This is done automatically by the HPE CSI Driver and its configured CSP.
- Deleting or scaling Pods down in a StatefulSet will not delete the volumes associated with the StatefulSet. This is done to ensure data safety, which is generally more valuable than an automatic purge of all related StatefulSet resources.
- A StatefulSet currently requires a Headless Service (see kubernetes Headless Services for details) to be responsible for the network identity of the Pods.
- A StatefulSet does not provide any guarantees on Pod termination when deleted. For ordered and graceful termination of Pods in a StatefulSet, it should be scaled down to 0 prior to deletion.
- When a StatefulSet is being created, the next Pod will not launch until the previous one is ready and running.

The following file named `install-mongo.yaml` creates the MongoDB headless service, StatefulSet, and persistent volume claim template.

```
apiVersion: v1
# Headless service creation
kind: Service
metadata:
  name: mongo
  labels:
    app: mongo
    hpecp.hpe.com/hpecp-internal-gateway: "true"
spec:
  ports:
  - port: 27017
    targetPort: 27017
  clusterIP: None
  selector:
    app: mongo
---
```

(*File listing is continued on next page.*)

(File listing continued from previous page.)

```
# Statefulset creation
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongo
spec:
  selector:
    matchLabels:
      app: mongo
  serviceName: "mongo"
  # Number of initial MongoDB pods to deploy - default is 1 if not set
  replicas: 3
  template:
    metadata:
      labels:
        app: mongo
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mongo
        image: docker.io/mongo
        command:
          - mongod
          - "--replSet"
          - rs0
        ports:
          - containerPort: 27017
            name: mongo
        volumeMounts:
          - name: mongopv
            mountPath: /data/db
      - name: mongo-sidecar
        image: docker.io/cvallance/mongo-k8s-sidecar
        env:
          - name: MONGO_SIDECAR_POD_LABELS
            value: "app=mongo"
# Persistent Volume Claim template for each pod
  volumeClaimTemplates:
  - metadata:
      name: mongopv
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "sc-mongo"
      resources:
        requests:
          storage: 28Gi
```

Deploy MongoDB with the following command:

```
# kubectl create -f install-mongo.yaml
```

Pod creation for the StatefulSet might take some time, but will show the following when they are all running:

```
# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
mongo-0   2/2     Running   0          22h
mongo-1   2/2     Running   0          22h
mongo-2   2/2     Running   0          22h
```

Volumes for the associated Pods are automatically created on the HPE Nimble Storage array by the HPE CSI Driver:

```
# kubectl get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mongopv-mongo-0   Bound    pvc-ca631fd7-67dd-44f4-acf4-2ad3d309f798   28Gi       RWO            sc-mongo       25h
mongopv-mongo-1   Bound    pvc-0d2bf53b-ebdc-426b-b1ce-ce157366e25c   28Gi       RWO            sc-mongo       25h
mongopv-mongo-2   Bound    pvc-79fc26d4-e04e-462d-a17a-02838d62c843   28Gi       RWO            sc-mongo       25h
```

At this point the MongoDB application is fully deployed. If more resources are needed, the number of MongoDB instances can be increased. Scaling the StatefulSet from its initial three Pods to five is very easy using the following command.

```
# kubectl scale --replicas=5 statefulset mongo
```

Again, StatefulSet Pod creation might take a few minutes, and will look like the following when done:

```
# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
mongo-0   2/2     Running   0          25h
mongo-1   2/2     Running   0          25h
mongo-2   2/2     Running   0          25h
mongo-3   2/2     Running   0          23h
mongo-4   2/2     Running   0          23h
```

The same information is available in a nice GUI from the Kubernetes dashboard view through the HCP gateway, as shown in FIGURE 9.

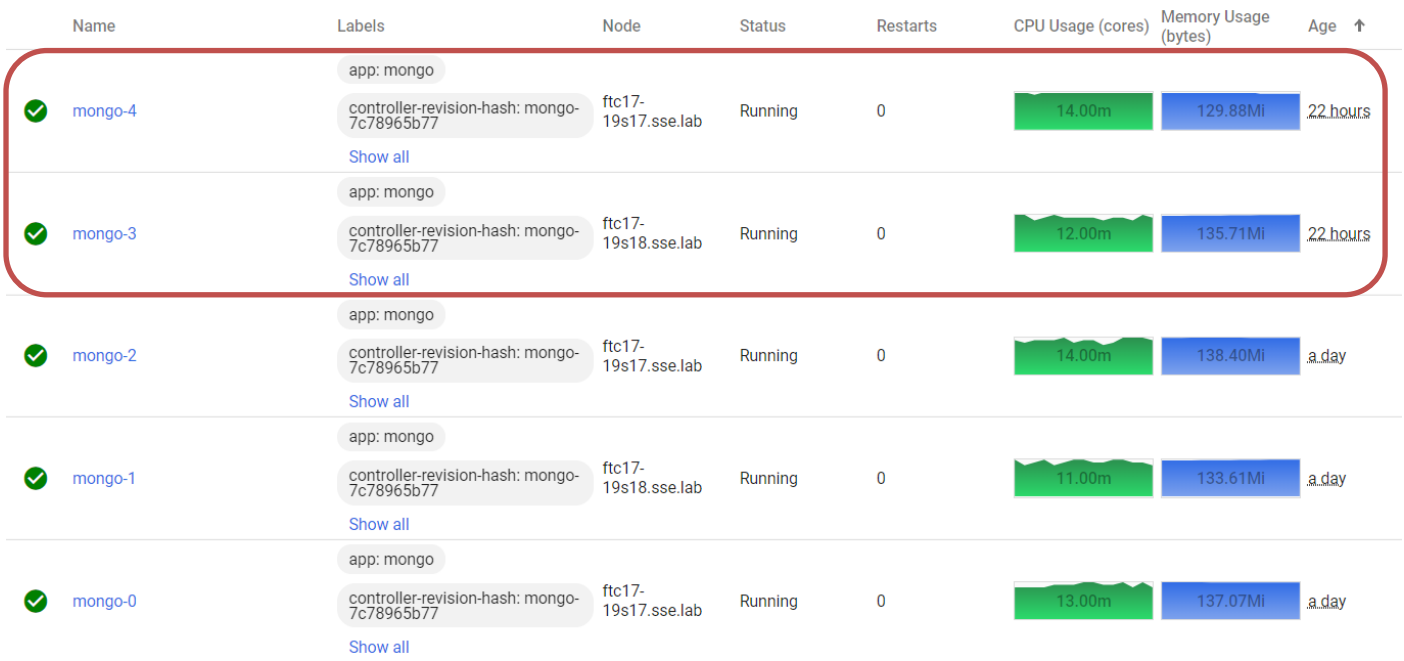

FIGURE 9. Kubernetes Dashboard (from HCP Gateway) showing the MongoDB Pods scaled to five

The newly scaled Pods will now have their associated persistent volumes claims bound to new volumes:

```
# kubectl get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mongopv-mongo-0   Bound    pvc-ca631fd7-67dd-44f4-acf4-2ad3d309f798   28Gi       RWO            sc-mongo       25h
mongopv-mongo-1   Bound    pvc-0d2bf53b-ebdc-426b-b1ce-ce157366e25c   28Gi       RWO            sc-mongo       25h
mongopv-mongo-2   Bound    pvc-79fc26d4-e04e-462d-a17a-02838d62c843   28Gi       RWO            sc-mongo       25h
mongopv-mongo-3   Bound    pvc-83bd646d-2d76-4de3-9bdb-fb0471748aca   28Gi       RWO            sc-mongo       23h
mongopv-mongo-4   Bound    pvc-2d7fb3e8-064e-4d99-8bca-06a28dba2442   28Gi       RWO            sc-mongo       23h
```

The same PVC information can be viewed from the Kubernetes dashboard through the HCP gateway, as shown in FIGURE 10.

Persistent Volume Claims

Name	Labels	Status	Volume	Capacity	Access Modes	Storage Class	Age
mongopv-mongo-4	app: mongo	Bound	pvc-2d7fb3e8-064e-4d99-8bca-06a28dba2442	28Gi	ReadWriteOnce	sc-mongo	22 hours
mongopv-mongo-3	app: mongo	Bound	pvc-83bd646d-2d76-4de3-9bdb-fb0471748aca	28Gi	ReadWriteOnce	sc-mongo	22 hours
mongopv-mongo-2	app: mongo	Bound	pvc-79fc26d4-e04e-462d-a17a-02838d62c843	28Gi	ReadWriteOnce	sc-mongo	a day
mongopv-mongo-1	app: mongo	Bound	pvc-0d2bf53b-ebdc-426b-b1ce-ce157366e25c	28Gi	ReadWriteOnce	sc-mongo	a day
mongopv-mongo-0	app: mongo	Bound	pvc-ca631fd7-67dd-44f4-acf4-2ad3d309f798	28Gi	ReadWriteOnce	sc-mongo	a day

FIGURE 10. Kubernetes Dashboard (from HCP Gateway) showing the MongoDB PVCs associated with the Pods

FIGURE 11 shows the MongoDB volumes as seen on the HPE Nimble Storage array web UI (matching Persistent Volume names).

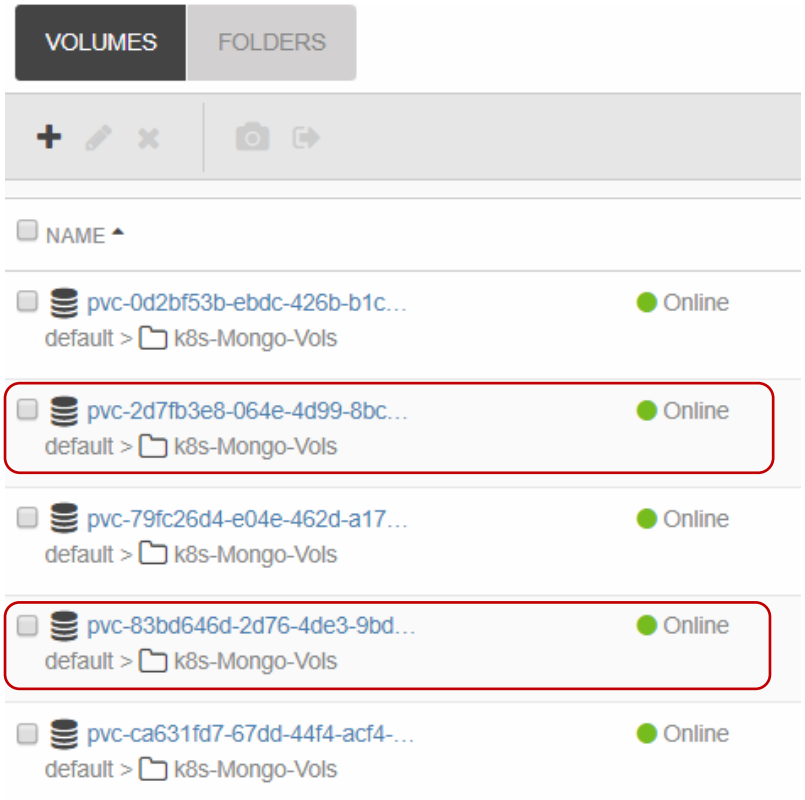

FIGURE 11. MongoDB volumes as seen on the HPE Nimble Storage UI

Because the Pods are part of a StatefulSet, volumes will not be automatically deleted. When scaling Pods down (to the original three in this example), the "mongopv-mongo-3" and "mongopv-mongo-4" volumes that were created on the Pod scaleup will remain. If the Pods are scaled up again (back to five), they will be named the same as the originals, and the existing volumes that were associated with the previous "mongo-3" and "mongo-4" Pods will be re-attached to the new Pods. Scaling the Pods past five will result in new volumes being created for subsequent new Pods. As these Pods are secondary nodes, they will have to resynchronize with the existing nodes since the data on the volumes will be stale.

While not covered in this document, in a sharded cluster configuration, the MongoDB nodes are used to manage throughput from clients to the database shards. Pods can be scaled up for periods of high traffic and scaled down when those resources are no longer required, with no need for resynchronization.

Cleaning up the environment

Because StatefulSet volumes are not automatically deleted, removal of the deployed resources will involve deleting the StatefulSet, Headless Service, and the provisioned Pod volumes. This will delete the persistent volumes from the HPE Nimble Storage array and be unrecoverable.

To delete the StatefulSet and associated Pods:

```
# kubectl delete statefulset mongo
```

To delete the headless service:

```
# kubectl delete svc mongo
```

To delete the persistent volumes:

```
#  kubectl delete pvc -app=mongo
```

SUMMARY

MongoDB is just one of the many containerized solutions that can be easily deployed by the HPE Container Platform with native integration to HPE storage products using the HPE CSI Driver.

In the rapidly changing Kubernetes and container field, easily deployed solutions are highly desirable. The industry wants to get away from tedious and manual text-based setups where human error can occur, and a high level of expertise is required.

This white paper demonstrates the ease of setting up the HPE Container Platform, as well as easily deploying an HA Kubernetes cluster without a lot of the manual steps. Benefits of this solution are:

- Containerization of applications makes for more efficient hardware utilization when compared to virtual machines.
- The HCP orchestration tool aids in the deployment of the HPE CSI Driver and the HPE Nimble Storage CSP.
- The HPE CSI Driver automatically provisions HPE storage to application Pods, avoiding manual setup of those volumes.

This paper describes solution testing performed by Hewlett Packard Enterprise in March and April 2020.

APPENDIX A – CODE EXAMPLES

/etc/profile.d/set_proxy.sh

```
export http_proxy=<url>
export https_proxy=<url>
export ftp_proxy=<url>
export no_proxy="localhost,127.0.0.1,\
 10.96.0.0/12,10.192.0.0/12,\
 <Individual IP of each HCP and Kubernetes host in the environment comma separated>\
 <Individual FQDN of each HCP and Kubernetes host in the environment comma separated>"
```

(provision) /etc/systemd/system/docker.service.d/docker-proxy.conf

```
# Once set_proxy.sh is properly defined, run this to create the docker-proxy
cat > /etc/systemd/system/docker.service.d/docker-proxy.conf <<EOF
[Service]
Environment="HTTP_PROXY=$http_proxy"
Environment="HTTPS_PROXY=$https_proxy"
Environment="NO_PROXY=$no_proxy"
EOF
```

(provision) /etc/yum.repos.d/kubernetes.repo

```
#Import Kubernetes yum repository
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

hpe-nimble-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: hpe-nimble-secret
  namespace: kube-system
stringData:
  serviceName: nimble-csp-svc
  servicePort: "8080"
  backend: 192.168.1.1
  username: admin
data:
  # echo -n "admin" | base64
  password: YWRtaW4=
```

APPENDIX B - TROUBLESHOOTING

Kubernetes Troubleshooting

For Kubernetes clusters deployed by HPE Container Platform, see the following documentation for troubleshooting examples:

- Kubernetes Installation
- Kubernetes Nodes
- Kubernetes Cluster creation
- Kubernetes Web interface
- General Kubernetes Application/Deployment Issues

Increase (or disable) Kubernetes dashboard timeout

Edit the Kubernetes dashboard deployment with the following `kubectl` command:

```
# kubectl edit deployment -n kubernetes-dashboard kubernetes-dashboard
```

Add the `token-ttl=0` attribute, as in the snip below:

```
…
    spec:
      containers:
      - args:
        - --auto-generate-certificates
        - --namespace=kubernetes-dashboard
        - --token-ttl=0
        image: kubernetesui/dashboard:v2.0.0-rc1
        imagePullPolicy: Always
        livenessProbe:
          failureThreshold: 3
…
```

RESOURCES AND ADDITIONAL LINKS

HPE Container Platform
hpe.com/us/en/solutions/container-platform.html

HPE Container Storage Interface
developer.hpe.com/blog/n0J8kpk1DJf4y7xD2D4X/introducing-a-multi-vendor-csi-driver-for-kubernetes

HPE Nimble Storage
hpe.com/us/en/storage/nimble.html

HPE Storage Container Orchestrator Documentation
scod.hpedev.io/container_storage_provider/hpe_nimble_storage/index.html#introduction

Kubernetes
kubernetes.io/

MongoDB
mongodb.com/

LEARN MORE AT

hpe.com/storage

a00100109ENW, July 2020, Rev 1

Hewlett Packard Enterprise