



HPE Express Containers with Docker Enterprise Edition: Ops Edition

Deployment Guide and Best Practices

Revision 1.0 – December 04, 2017

Contents

Introduction.....	3
About Ansible	3
About Docker Enterprise Edition.....	3
About HPE SimpliVity	4
Architecture.....	4
Sizing considerations.....	6
Provisioning the operations environment	8
Verify prerequisites.....	8
Enable vSphere High Availability.....	9
Install vSphere Docker Volume Service driver on all ESXi hosts.....	9
Create a VM template	10
Create the Ansible node.....	19
Finalize the template.....	20
Prepare your Ansible configuration.....	20
Editing the inventory.....	21
Editing the group variables.....	22
Editing the vault	26
Running the playbooks.....	26
Post deployment.....	27
Overview of the playbooks.....	27
Create virtual machines	27
Configure network settings.....	27
Distribute public keys	27
Register the VMs with Red Hat	27
Install HAProxy.....	27
Install NTP	27
Install Docker Enterprise Edition.....	27
Install rsyslog.....	27
Configure Docker LVs	27
Docker post-install configuration.....	27
Install NFS server	28
Install NFS clients	28
Install and configure Docker UCP nodes	28

Install and configure DTR nodes.....	28
Install worker nodes	28
Configuring monitoring.....	28
Configure dummy VMs to back up Docker volumes	29
Configure SimpliVity backups.....	30
VM placement and number of HPE SimpliVity servers in the cluster.....	30
Accessing the UCP UI.....	31
Accessing the DTR UI.....	33
Security considerations	35
Prevent tags from being overwritten	35
Isolate swarm nodes to a specific team.....	35
Central logging.....	36
HA considerations.....	36
Host failure.....	36
VM failure	37
Docker volume backup and restore.....	38
Create a Docker volume.....	38
Automated backup	39
Manual backup	39
Restore.....	41
Test the restore.....	45
Disaster Recovery	45
Solution lifecycle management.....	45
Introduction	45
SimpliVity environment.....	47
vSphere Docker Volume Service plug-in	47
Red Hat Enterprise Linux operating system.....	48
Docker EE environment.....	48
Monitoring tools.....	49
High-level dependency map.....	49
Appendix A.....	50
Resources and additional links	51

Introduction

HPE Express Containers with Docker Enterprise Edition (EE) is a complete solution from Hewlett Packard Enterprise that includes all the hardware, software, professional services, and support you need to deploy a containers-as-a-service (CaaS) platform, allowing you to get up and running quickly and efficiently. The solution takes the HPE hyperconverged infrastructure and combines it with Docker's enterprise-grade container platform, popular open source tools, along with deployment and advisory services from HPE Pointnext.

HPE Express Containers with Docker EE is ideal for customers migrating legacy applications to containers, transitioning to a container DevOps development model or needing a hybrid environment to support container and non-containerized applications on a common VM platform. HPE Express Containers with Docker EE provides a solution for both IT development and IT operations, and comes in two versions. The version for IT developers (HPE Express Containers with Docker EE: Dev Edition) addresses the need to provide a cloud-like container development environment with built-in container tooling. The version for IT operations (HPE Express Containers with Docker EE: Ops Edition) addresses the need to have a production ready environment that is very easy to deploy and manage.

This document describes the best practices for deploying and operating the HPE Express Containers with Docker EE: Ops Edition. It describes how to automate the provisioning of the environment using a set of Ansible playbooks. It also outlines a set of manual steps to harden, secure and audit the overall status of the system. A corresponding document focused on setting up HPE Express Containers with Docker EE: Dev Edition will also be available.

Note

The Ansible playbooks described in this document are only intended for Day 0 deployment automation of Docker EE on HPE SimpliVity.

The Ansible playbooks described in this document are not directly supported by HPE and are intended as an example of deploying Docker EE on HPE SimpliVity. We welcome input from the user community via GitHub to help us prioritize all future bug fixes and feature enhancements.

About Ansible

Ansible is an open-source automation engine that automates software provisioning, configuration management and application deployment.

As with most configuration management software, Ansible has two types of servers: the controlling machine and the nodes. A single controlling machine orchestrates the nodes by deploying modules to the nodes over SSH. The modules are temporarily stored on the nodes and communicate with the controlling machine through a JSON protocol over the standard output. When Ansible is not managing nodes, it does not consume resources because no daemons or programs are executing for Ansible in the background. Ansible uses one or more inventory files to manage the configuration of the multiple nodes in the system.

In contrast with other popular configuration management software, such as Chef, Puppet, and CFEngine, Ansible uses an agentless architecture. With an agent-based architecture, nodes must have a locally installed daemon that communicates with a controlling machine. With an agentless architecture, nodes are not required to install and run background daemons to connect with a controlling machine. This type of architecture reduces the overhead on the network by preventing the nodes from polling the controlling machine.

More information about Ansible can be found at: <http://docs.ansible.com>

About Docker Enterprise Edition

Docker Enterprise Edition (EE) is a leading enterprise containers-as-a-service (CaaS) software platform for IT that manages and secures diverse applications across disparate infrastructure, both on-premises and in the cloud. Docker EE provides integrated container management and security from development to production. Enterprise-ready capabilities like multi-architecture orchestration and secure software supply chain give IT teams the ability to manage and secure containers without breaking the developer experience.

Docker EE provides:

- Integrated management of all application resources from a single web admin UI.
- Frictionless deployment of applications and Compose files to production in a few clicks.

- Multi-tenant system with granular role-based access control (RBAC) and LDAP/AD integration.
- Self-healing application deployment with the ability to apply rolling application updates.
- End-to-end security model with secrets management, image signing and image security scanning.

More information about Docker Enterprise Edition can be found at: <https://www.docker.com/enterprise-edition>

About HPE SimpliVity

HPE SimpliVity is an enterprise-grade hyperconverged platform combining best-in-class data services with the world's best-selling server.

Rapid proliferation of applications and the increasing cost of maintaining legacy infrastructure causes significant IT challenges for many organizations. With HPE SimpliVity, you can streamline and enable IT operations at a fraction of the cost of traditional and public cloud solutions by combining your IT infrastructure and advanced data services into a single, integrated solution. HPE SimpliVity is a powerful, simple, and efficient hyperconverged platform that joins best-in-class data services with the world's best-selling server and offers the industry's most complete guarantee.

More information about HPE SimpliVity can be found at: <https://www.hpe.com/us/en/integrated-systems/simplivity.html>

Target Audience: This document is primarily aimed at technical individuals working in the Operations side of the pipeline, such as system administrators and infrastructure engineers, but anybody with an interest in automating the provisioning of virtual servers and containers may find this document useful.

Assumptions: This document assumes a minimum understanding of concepts such as virtualization and containerization and also some knowledge around Linux® and VMware® technologies.

Required Versions: The following software versions were used to implement the playbooks that are described in later sections. Other versions may work but have not been tested.

- Ansible 2.2 and 2.3. Please note that the playbooks will not work with Ansible 2.4 due to an open defect <https://github.com/ansible/ansible/issues/32000>. Do not use Ansible 2.4 until this defect is fixed.
- Docker EE 17.06 (tested with UCP 2.2.3 and 2.2.4 and DTR 2.4.0)
- Red Hat® Enterprise Linux 7.3 and 7.4
- VMware ESXi 6.5.0 and vCenter 6.5.0
- HPE SimpliVity OmniStack 3.7.1.60

Architecture

The Operations environment is comprised of three HPE SimpliVity 380 Gen10 servers. HPE recommends dual socket SimpliVity systems with at least 14 CPU cores per socket (28 total cores per system) for optimal performance and support during HA failover scenario. Please refer to Appendix A for a sample BOM. Since the SimpliVity technology relies on VMware virtualization, the servers are managed using vCenter. The load among the three hosts will be shared as per Figure 1.

Uptime is paramount for any users implementing Docker containers in business critical environments. HPE Express Containers with Docker EE offers various levels of high availability (HA) to support continuous availability. All containers including the Docker system containers are protected by Docker's swarm mode. Swarm mode can protect against individual hardware, network, and container failures based on the user's declarative model.

HPE Express Containers with Docker EE also deploys load balancers in the system to help with container traffic management. There are three load balancer VMs – UCP load balancer, DTR load balancer, and Docker worker node load balancer. Since these load balancers exist in VMs, they have some degree of HA but may incur some downtime during the restoration of these VMs due to a planned or unplanned outage. For optimal HA configuration, the user should consider implementing a HA load balancer architecture using the Virtual Router Redundancy Protocol (VRRP). For more information see <http://www.haproxy.com/solutions/high-availability/>.



Figure 1. Solution Architecture

The Ansible playbooks can be modified to fit your environment and your high availability (HA) needs. By default, the Ansible Playbooks will set up a 3 node environment. HPE and Docker recommend a minimal starter configuration of 3 physical nodes for running Docker in production. This is the minimal configuration that Docker recommends for cluster HA. The distribution of the Docker and non-Docker modules over the 3 physical nodes via virtual machines (VMs) is as follows:

- 3 Docker Universal Control Plane (UCP) VM nodes for HA and cluster management
- 3 Docker Trusted Registry (DTR) VM nodes for HA of the container registry
- 3 Docker Swarm worker VM nodes for container workloads
- 1 Docker UCP load balancer VM to ensure access to UCP in the event of a node failure
- 1 Docker DTR load balancer VM to ensure access to DTR in the event of a node failure
- 1 Docker Swarm Worker node VM load balancer
- 1 Logging server VM for central logging
- 1 NFS server VM for storage Docker DTR images

In addition to the above, the playbooks also set up:

- Docker persistent storage driver from VMware
- Prometheus and Grafana monitoring tools
- SimpliVity backup policy for data volumes and for the NFS storage used by DTR for storing Docker images

These nodes can live in any of the hosts and they are not redundant. The Prometheus and Grafana services are declared in a Docker stack as replicated `services` with one replica each, so if they fail, Docker EE will ensure that they are restarted on one of the UCP VMs. cAdvisor and node-exporter are declared in the same stack as global services, so Docker EE will ensure that there is always one copy of each running on every machine in the cluster. The vSphere Docker volume plug-in stores data in a shared datastore that can be accessed from any machine in the cluster.

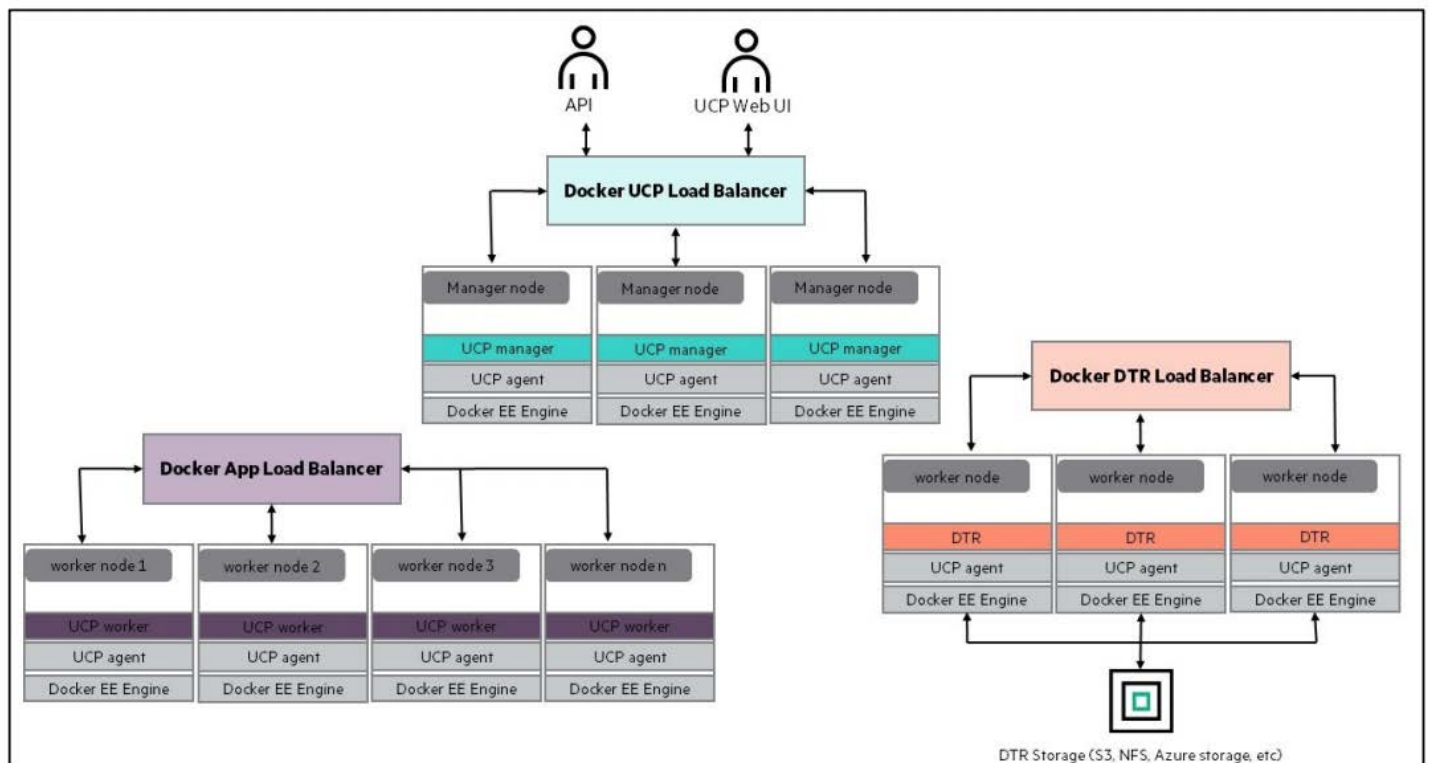


Figure 2. Load balancer architecture

Sizing considerations

A node is a machine in the cluster (virtual or physical) with Docker Engine running on it. When provisioning each node, assign it a role: UCP Controller, DTR, or worker node so that it is protected from running application workloads.

To decide what size the node should be in terms of CPU, RAM, and storage resources, consider the following:

1. All nodes should at least fulfil the minimal requirements, for UCP 2.0, 2GB of RAM and 3GB of storage. More detailed requirements are in the UCP documentation.
2. UCP Controller nodes should be provided with more than the minimal requirements, but won't need much more if nothing else runs on them.
3. Ideally, worker node size will vary based on your workloads so it is impossible to define a universal standard size.

4. Other considerations like target density (average number of containers per node), whether one standard node type or several are preferred, and other operational considerations might also influence sizing.

If possible, node size should be determined by experimentation and testing actual workloads; and they should be refined iteratively. A good starting point is to select a standard or default machine type in your environment and use this size only. If your standard machine type provides more resources than the UCP Controllers need, it makes sense to have a smaller node size for these. Whatever the starting choice, it is important to monitor resource usage and cost to improve the model.

For HPE Express Containers with Docker EE: Ops Edition, the following tables describe sizing configurations. The vCPU allocations are described in Table 1 while the memory allocation is described in Table 2.

Table 1. vCPU

vCPUs	simply01	simply02	simply03
ucp1	4		
ucp2		4	
ucp3			4
dtr1	2		
dtr2		2	
dtr3			2
worker1	4		
worker2		4	
worker3			4
ucb_lb	2		
dtr_lb		2	
worker_lb			2
nfs			2
logger		2	
Total vCPU per node	12	14	14

Note

In the case of one ESX host failure, two nodes are enough to accommodate the amount of vCPU required.

Table 2. Memory allocation

RAM (GB)	simply01	simply02	simply03
ucp1	8		
ucp2		8	
ucp3			8

dtr1	16		
dtr2		16	
dtr3			16
worker1	64		
worker2		64	
worker3			64
ucb_lb	4		
dtr_lb		4	
worker_lb			4
nfs			4
logger		4	
Total RAM required (per node)	92	96	96
Total RAM required		284	
Available RAM	384	384	384

Note

In the case of one ESX host failure, the two surviving hosts can accommodate the amount of RAM required for all VMs.

Provisioning the operations environment

This section describes in detail how to provision the environment described previously in the architecture section. The high level steps this guide will take are shown in Figure 3.

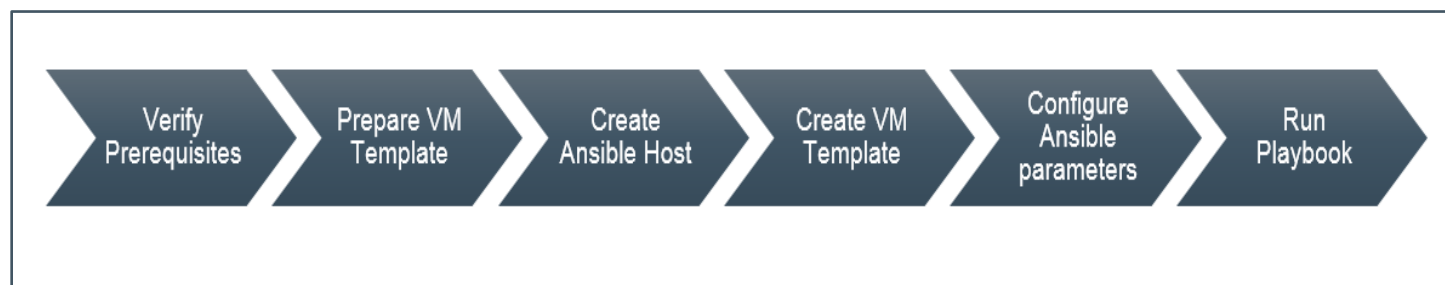


Figure 3. Provisioning steps

Verify prerequisites

You must assemble the information required to assign values for each and every variable used by the playbooks, before you start deployment. The variables are fully documented in the following sections “Editing the group variables” and “Editing the vault”. A summary of the information required is presented in Table 3.

Table 3. Summary of information required

Component	Details
Virtual Infrastructure	The FQDN of your vCenter server and the name of the Datacenter which contains the SimpliVity cluster. You will also need administrator credentials in order to create templates, and spin up virtual machines.
SimpliVity Cluster	The name of the SimpliVity cluster and the names of the member of this cluster as they appear in vCenter. You will also need to know the name of the SimpliVity datastore where you want to land the various virtual machines. You may have to create this datastore if you just installed your SimpliVity cluster. In addition, you will need the IP addresses of the OmniStack virtual machines. Finally you will need credentials with admin capabilities for using the OmniStack API. These credentials are typically the same as your vCenter admin credentials
L3 Network requirements	You will need one IP address for each and every VM configured in the Ansible inventory (see the section “Editing the inventory”). At the time of writing, the example inventory configures 14 virtual machines so you would need to allocate 14 IP addresses to use this example inventory. Note that the Ansible playbooks do not support DHCP so you need static IP addresses. All the IPs should be in the same subnet. You will also have to specify the size of the subnet (for example /22 or /24) and the L3 gateway for this subnet.
DNS	You will need to know the IP addresses of your DNS server. In addition, all the VMs you configure in the inventory should have their names registered in DNS. In addition, you will need the domain name to use for configuring the virtual machines (such as example.com)
NTP Services	You need time services configured in your environment. The solution being deployed (including Docker) uses certificates and certificates are time sensitive. You will need the IP addresses of your time servers (NTP).
RHEL Subscription	A RHEL subscription is required to pull extra packages that are not on the DVD.
Docker Prerequisites	You will need a URL for the official Docker EE software download and a license file. Refer to the Docker documentation to learn more about this URL and the licensing requirements at: https://docs.docker.com/engine/installation/linux/docker-ee/rhel/ in the section entitled “Docker EE repository URL”
Proxy	The playbooks pull the Docker packages from the Internet. If your environment accesses the Internet through a proxy, you will need the details of the proxy including the fully qualified domain name and the port number.

Enable vSphere High Availability

You must enable vSphere High Availability (HA) to support virtual machine failover during an HA event such as a host failure. Sufficient CPU and memory resources must be reserved across the system so that all VMs on the affected host(s) can fail over to remaining available hosts in the system. You configure an Admission Control Policy (ACP) to specify the percentage CPU and memory to reserve on all the hosts in the cluster to support HA functionality.

More information on enabling vSphere HA and configuring Admission Control Policy is available in the HPE SimpliVity documentation. Log in to the HPE Support Center at <https://www.hpe.com/us/en/support.html> and search for “HPE SimpliVity 380”. The administration guide is listed when you select the User document type.

Note

You should not use the default Admission Control Policy. Instead, you should calculate the memory and CPU requirements that are specific to your environment.

Install vSphere Docker Volume Service driver on all ESXi hosts

vSphere Docker Volume Service technology enables stateful containers to access the storage volumes provided by SimpliVity. This is a one-off manual step. In order to be able to use Docker volumes using the vSphere driver, you must first install the latest release of the vSphere Docker Volume Service (vDVS) driver, which is available as a vSphere Installation Bundle (VIB). To perform this operation, log in to each of the ESXi hosts, download and install the latest release of vDVS driver.

```
# esxcli software vib install -v /tmp/vmware-esx-vmkops-<version>.vib --no-sig-check
```

More information on how to download and install the driver can be found on the Docker Store at <https://store.docker.com/plugins/vsphere-docker-volume-service>.

Note

You cannot mount the same persistent volume created through vSphere Docker Volume Service (vDVS) on containers running on two different hosts at the same time.

Create a VM template

The first step of the automated solution is the creation of a VM Template that you will use as the base for all your nodes. In order to create a VM Template you will first create a Virtual Machine with the OS installed and then convert the Virtual Machine to a VM Template. Since the goal of automation is to remove as many repetitive tasks as possible, the VM Template is created as lean as possible, with any additional software installs and/or system configuration performed subsequently using Ansible.

It would be possible to automate the creation of the template. However, as this is a one-off task, it is appropriate to do it manually. The steps to create a VM template manually are described below:

1. Log in to vCenter and create a new Virtual Machine. In the dialog box, shown in Figure 4, select **Typical** and press **Next**:

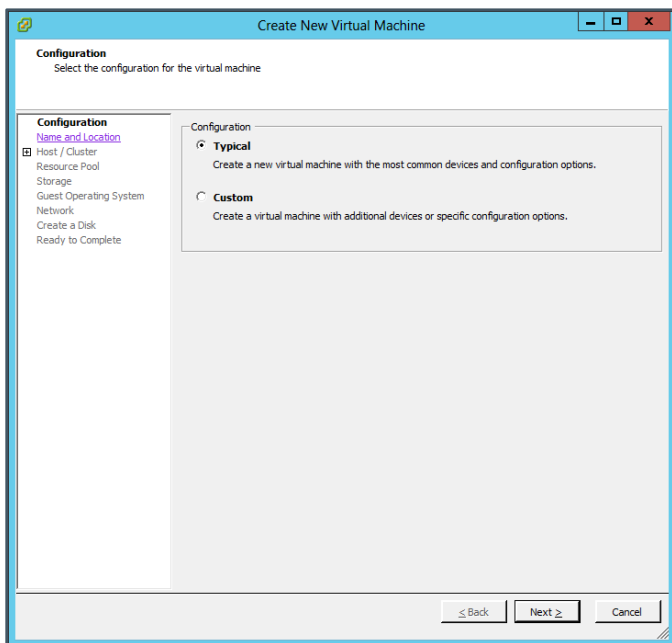


Figure 4. Create New Virtual Machine

- Specify the name and location for your template, as shown in Figure 5:

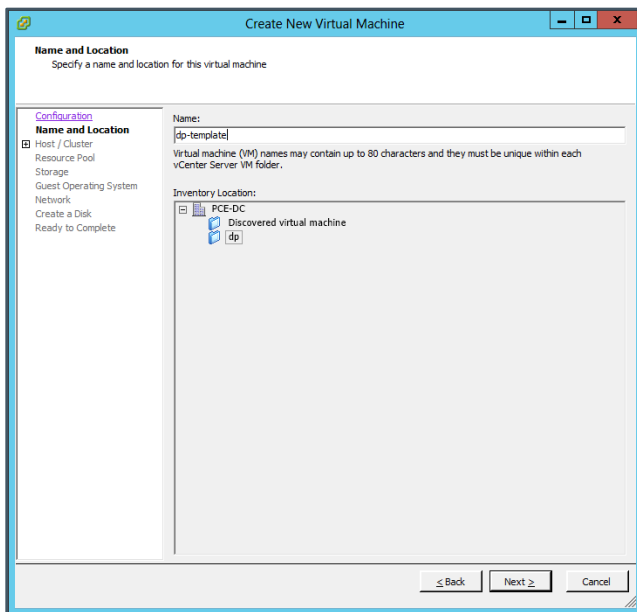


Figure 5. Specify name and location for the virtual machine

- Choose the host / cluster on which you want to run this virtual machine, as shown in Figure 6:

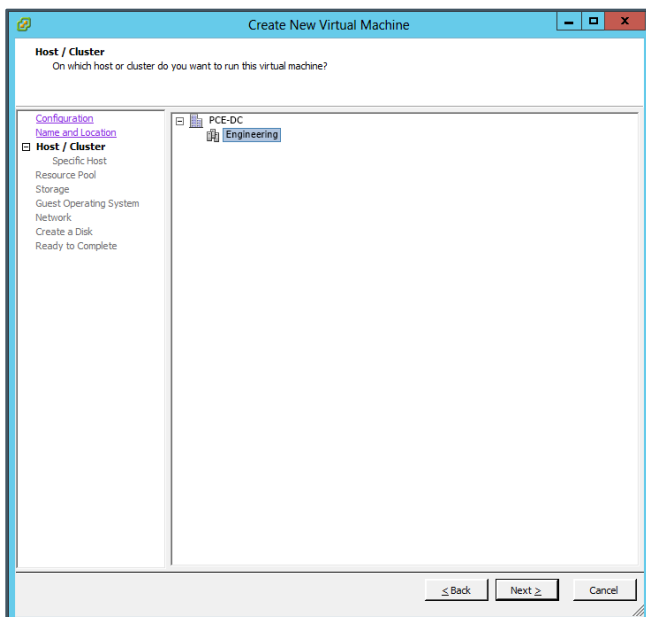


Figure 6. Choose host / cluster

4. Choose a datastore where the template files will be stored, as shown in Figure 7.

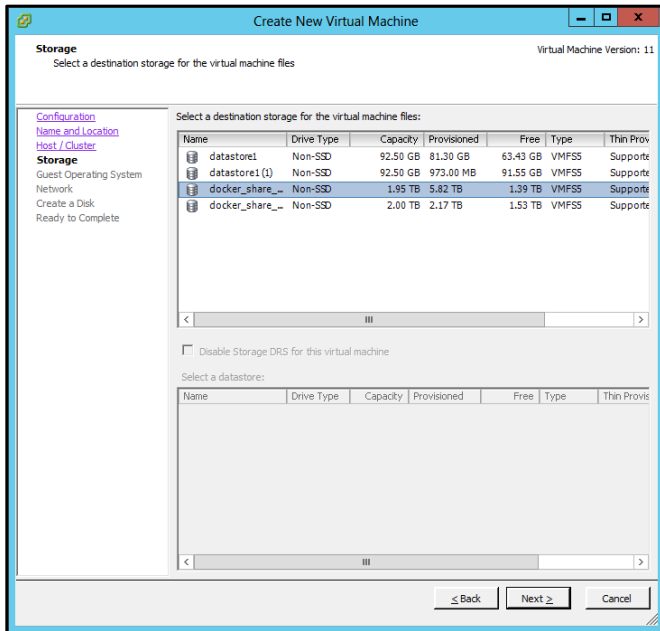


Figure 7. Select storage for template files

5. Choose the OS as shown in Figure 8, in this case Linux, RHEL 7 64bit.

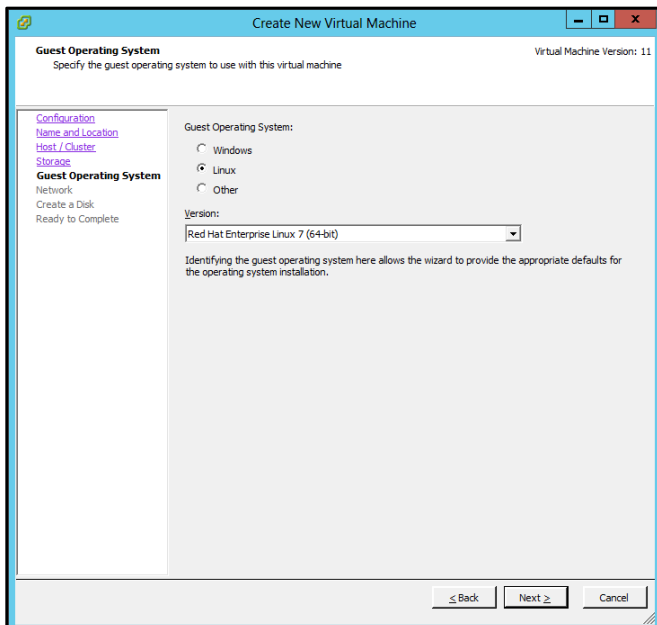


Figure 8. Choose operating system

6. Pick the network to attach to your template as shown in Figure 9. In this example there is only one NIC but depending on how you plan to architect your environment you might want to add more than one.

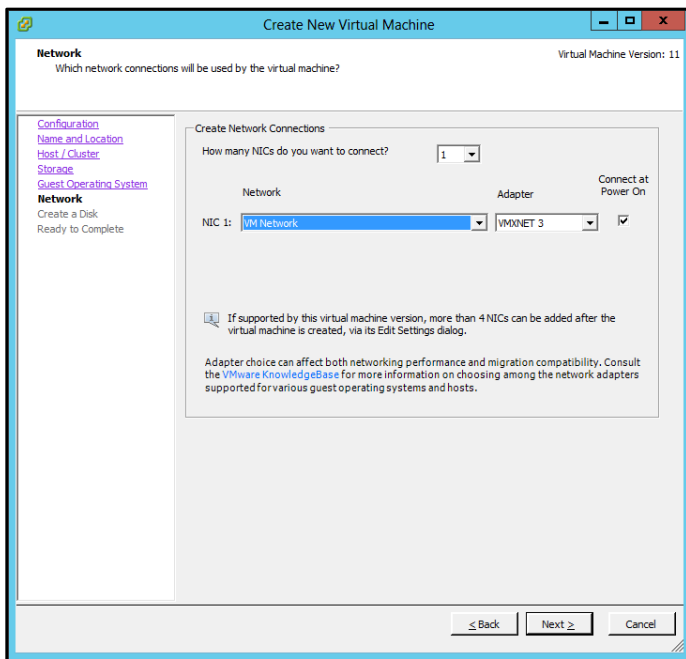


Figure 9. Create network connections

7. Create a primary disk as shown in Figure 10. The chosen size in this case is 50GB but 20GB should typically be enough.

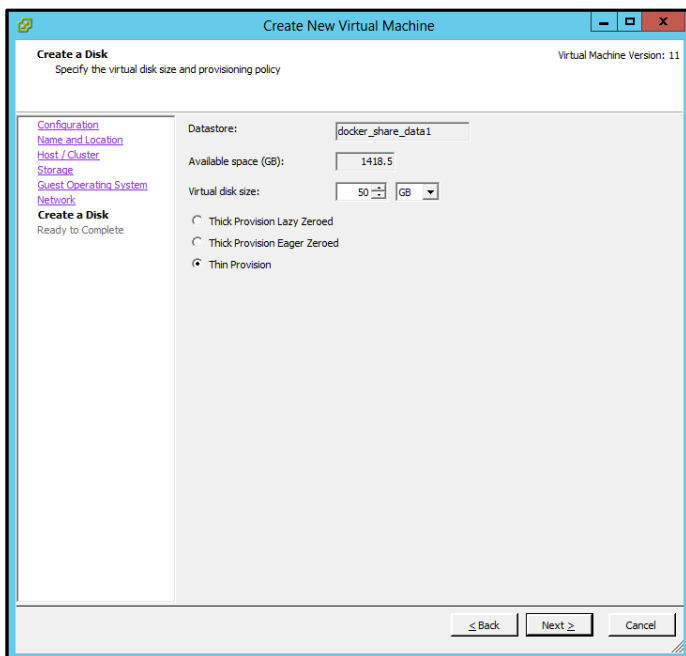


Figure 10. Create primary disk

8. Confirm that the settings are correct and press **Finish** as shown in Figure 11.

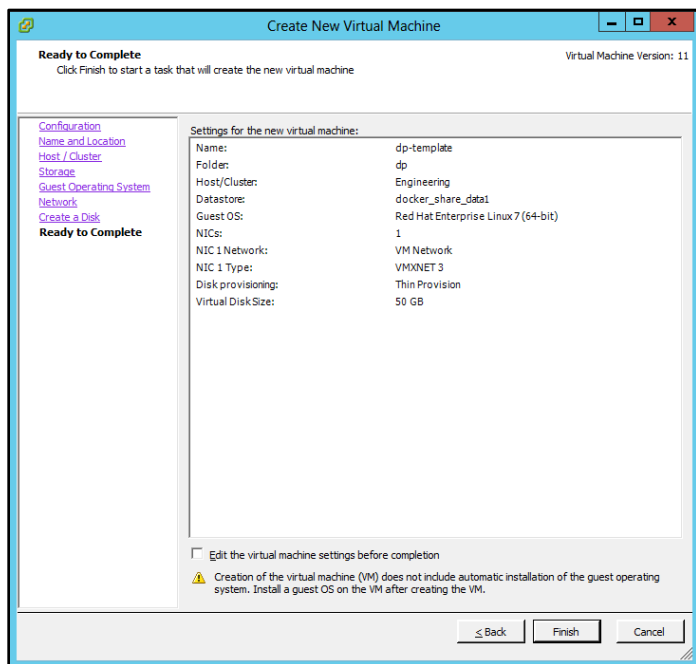


Figure 11. Confirm settings

9. The next step is to virtually insert the RHEL 7 DVD, using the Settings of the newly created VM as shown in Figure 12. Select your ISO file in the Datastore ISO File Device Type and make sure that the **Connect at power on** checkbox is checked.

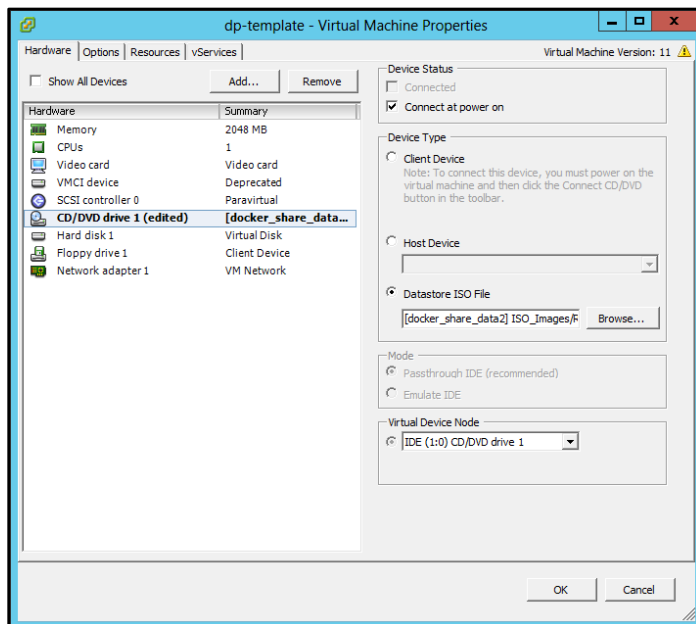


Figure 12. Virtual machine properties

10. Finally, you can optionally remove the Floppy Disk, as shown in Figure 13, as this is not required for the VM.

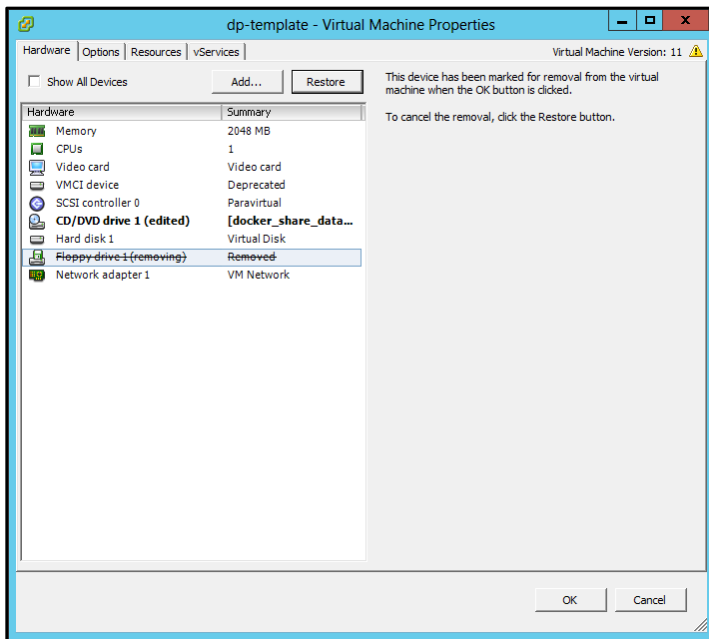


Figure 13. Remove Floppy drive

11. Power on the server and open the console to install the OS. On the welcome screen, as shown in Figure 14, pick your language and press Continue:

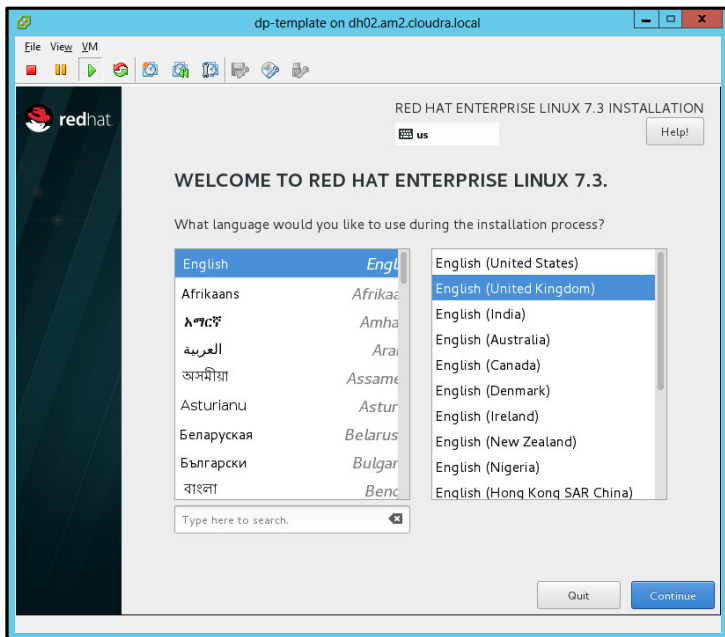


Figure 14. Welcome screen

12. The installation summary screen will appear, as shown in Figure 15.

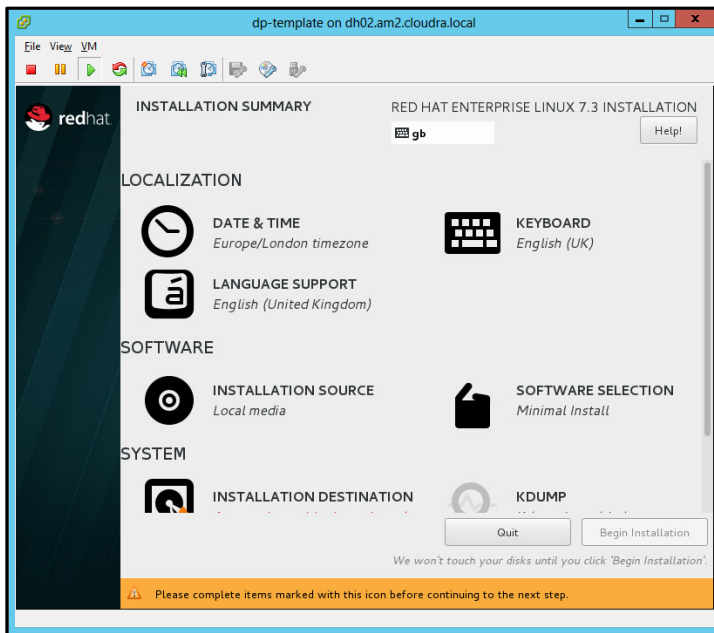


Figure 15. Installation summary

13. Scroll down and click on Installation Destination, as shown in Figure 16.

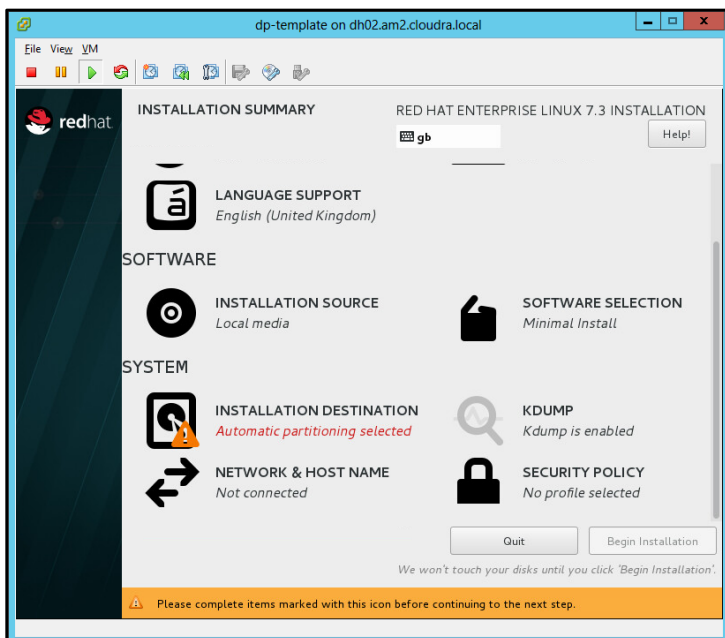


Figure 16. Installation destination

14. Select your installation drive, as shown in Figure 17, and click Done .

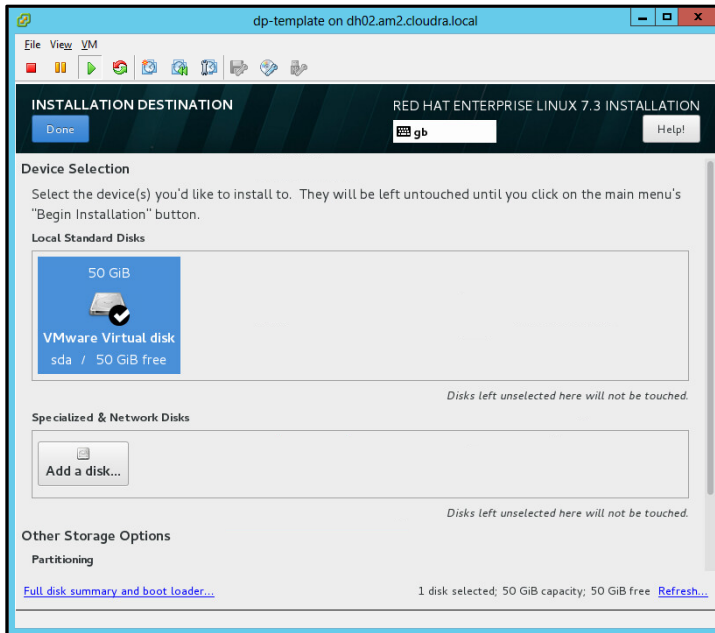


Figure 17. Select installation drive

15. Click Begin Installation, use all the other default settings, and wait for the Configuration – User Settings dialog, shown in Figure 18.

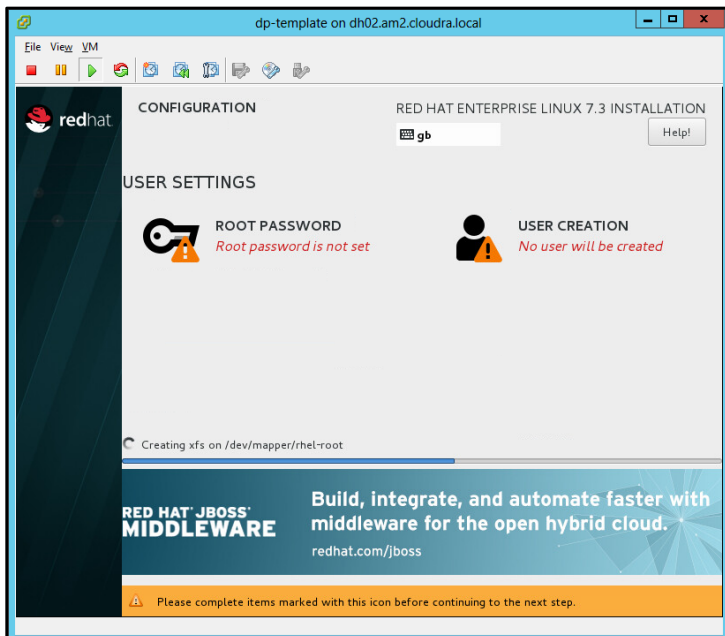


Figure 18. Configure user settings

16. Select a root password, as shown in Figure 19.

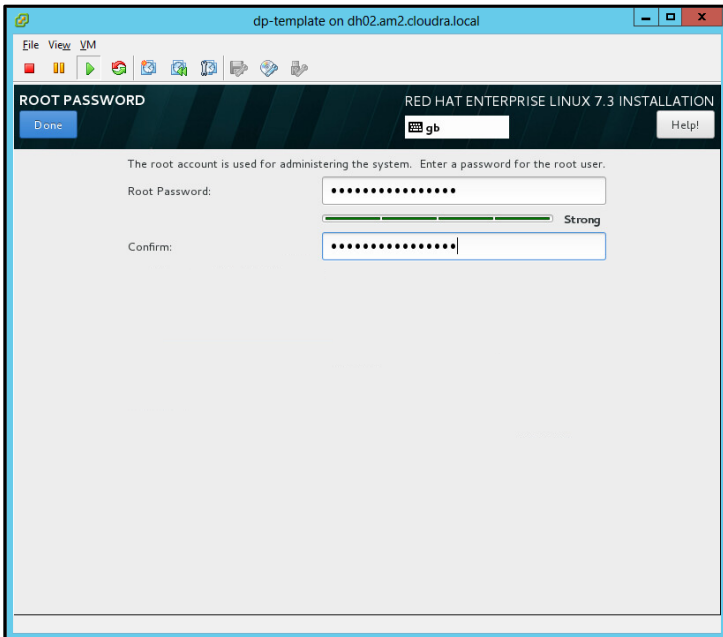


Figure 19. Set root password

17. Click **Done** and wait for the install to finish. Reboot and log into the system using the VM console.

18. The Red Hat packages required during the deployment of the solution come from two repositories: `rhel-7-server-rpms` and `rhel-7-server-extras-rpms`. The first repository can be found on the Red Hat DVD but the second cannot. There are two options, with both requiring a Red Hat Network account.

- a. Use Red Hat subscription manager to register your system. This is the easiest way and will automatically give you access to the official Red Hat repositories. Use the `subscription-manager register` command as follows:

```
# subscription-manager register --auto-attach
```

If you are behind a proxy, you must configure this before running the above command to register:

```
# subscription-manager config --server.proxy_hostname=<proxy IP> --server.proxy_port=<proxy port>
```

If you use this option, the playbooks will automatically enable the `extras` repository on the VMs that need it.

- b. Use an internal repository. Instead of pulling the packages from Red Hat, you can create copies of the required repositories on a dedicated node. You can then configure the package manager to pull the packages from the dedicated node. Your `/etc/yum.repos.d/redhat.repo` could look something like this:

```
[RHEL7-Server]
name=Red Hat Enterprise Linux $releasever - $basearch
baseurl=http://yourserver.example.com/rhel-7-server-rpms/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

```
[RHEL7-Server-extras]
name=Red Hat Enterprise Linux Extra pkg $releasever - $basearch
baseurl=http://yourserver.example.com/rhel-7-server-extras-rpms/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

The following articles explain how you can create a local mirror of the Red Hat repositories and how to share them:

<https://access.redhat.com/solutions/23016>

<https://access.redhat.com/solutions/7227>

Before converting the VM to a template, you will need to set up access for the Ansible host to configure the individual VMs. This is explained in the next section.

Create the Ansible node

In addition to the VM Template, you need another Virtual Machine where Ansible will be installed. This node will act as the driver to automate the provisioning of the environment and it is essential that it is properly installed. The steps are as follows:

1. Create a Virtual Machine and install your preferred OS (in this example, and for the sake of simplicity, RHEL 7 will be used). The rest of the instructions assume that, if you use a different OS, you understand the possible differences in syntax for the provided commands. If you use RHEL 7, select **Infrastructure Server** as the base environment and the **Guests Agents** add-on during the installation.

2. Log in to the root account and create an SSH key pair. Do not protect the key with a passphrase (unless you want to use ssh-agent).

```
# ssh-keygen
```

3. Configure the following yum repositories, `rhel-7-server-rpms` and `rhel-7-server-extras-rpms` as explained in the previous section.

4. Configure the EPEL repository. For more information, see: <http://fedoraproject.org/wiki/EPEL>. Note that `yum-config-manager` comes with the Infrastructure Server base environment, if you did not select this environment you will have to install the `yum-utils` package.

```
# rpm -ivh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

```
# yum-config-manager --enable rhel-7-server-extras-rpms
```

5. Install Ansible. The playbooks were tested with Ansible 2.2 and 2.3. Please note that the playbooks will not work with Ansible 2.4 due to an open defect <https://github.com/ansible/ansible/issues/32000>. Do not use Ansible 2.4 until this defect is fixed. To install the Ansible 2.3 use the following command:

```
# yum install ansible-2.3.2.0-2.el7
```

6. Make a list of all the hostnames and IPs that will be in your system and update your `/etc/hosts` file accordingly. This includes your UCP nodes, DTR nodes, worker nodes, NFS server, logger server and load balancers.

7. Install the following packages which are a mandatory requirement for the playbooks to function as expected. (Update pip if requested).

```
# yum install python-pyvmmomi python-netaddr python2-jmespath python-pip gcc python-devel openssl-devel git
```

```
# pip install --upgrade pip
```

```
# pip install cryptography
```

```
# pip install pysphere
```

8. Copy your SSH public key to the VM Template so that, in the future, your Ansible node can SSH without the need for a password to all the Virtual Machines created from the VM Template.

```
# ssh-copy-id root@<VM_Template>
```

Please note that, in both the Ansible node and the VM Template, you might need to configure the network so that one node can reach the other. Instructions for this step have been omitted since it is a basic step and will vary depending on your environment.

Finalize the template

Now that the VM Template has the public key of the Ansible node, you need to convert this VM to a VM Template. Perform the following steps in the VM Template to finalize its creation:

1. Clean up the template by running the following commands:

```
# rm /etc/ssh/ssh_host_*  
# history -c
```

2. Shut down the VM:

```
# shutdown -h now
```

3. Once the Virtual Machine is ready and turned off, it is time to convert it to a template as shown in Figure 20:

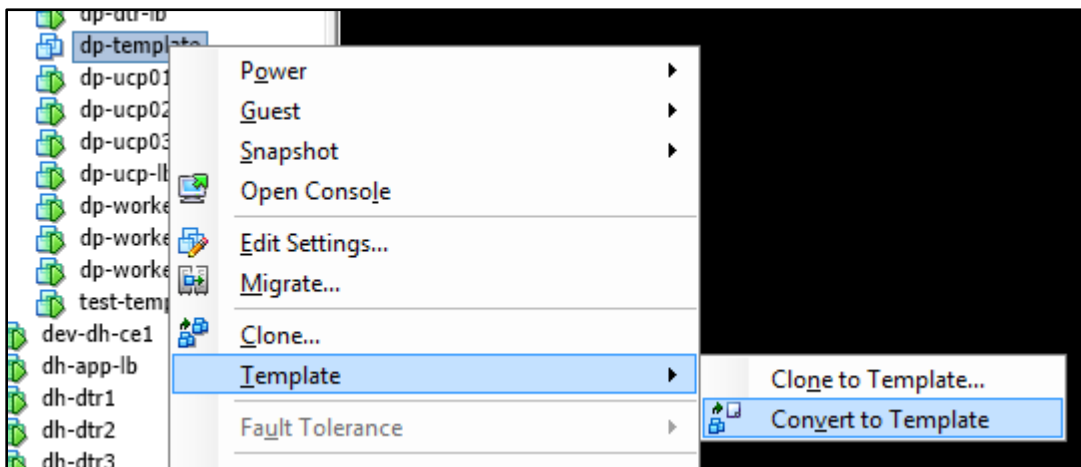


Figure 20. Convert to template

This completes the creation of the VM Template.

Prepare your Ansible configuration

On the Ansible node, retrieve the latest version of the playbooks using git.

```
# git clone https://github.com/HewlettPackard/Docker-SimpliVity
```

Change to the directory which you just cloned:

```
# cd ~/Docker-SimpliVity
```

Change to the ops directory:

```
# cd ops
```

Note

File names are relative to the ops directory. For example `vm_hosts` is located in `~/Docker-SimpliVity/ops` and `group_vars/vars` relates to `~/Docker-SimpliVity/ops/groups_vars/vars`.

You now need to prepare the configuration to match your own environment, prior to deploying Docker EE and the rest of the nodes. To do so, you will need to edit and modify three different files:

- `vm_hosts` (the inventory file)
- `group_vars/vars` (the group variables file)
- `group_vars/vault` (the encrypted group variable file)

You should work from the root account for the configuration steps and later when you run the playbooks.

Editing the inventory

The inventory is the file named `vm_hosts` in the `~Docker-SimpliVity/ops` directory. You need to edit this file to describe the configuration you want to deploy.

The nodes inside the inventory are organized in groups. The groups are defined by brackets and the names are static so they must not be changed. Everything else (including hostnames, specifications, IP addresses) should be edited to match your requirements. The groups are as follows:

- `[ucp_main]`: A group containing one single node which will be the main UCP node and swarm leader. Do not add more than one node under this group.
- `[ucp]`: A group containing all the UCP nodes, including the main UCP node. Typically you should have either 3 or 5 nodes under this group.
- `[dtr_main]`: A group containing one single node which will be the first DTR node to be installed. Do not add more than one node under this group.
- `[dtr]`: A group containing all the DTR nodes, including the main DTR node. Typically you should have either 3 or 5 nodes under this group.
- `[worker]`: A group containing all the worker nodes. Typically you should have either 3 or 5 nodes under this group.
- `[ucp_lb]`: A group containing one single node which will be the load balancer for the UCP nodes. Do not add more than one node under this group.
- `[dtr_lb]`: A group containing one single node which will be the load balancer for the DTR nodes. Do not add more than one node under this group.
- `[worker_lb]`: A group containing one single node which will be the load balancer for the worker nodes. Do not add more than one node under this group.
- `[lbs]`: A group containing all the load balancers. This group will have 3 nodes, also defined in the three groups above.
- `[nfs]`: A group containing one single node which will be the NFS node. Do not add more than one node under this group.
- `[logger]`: A group containing one single node which will be the logger node. Do not add more than one node under this group.
- `[local]`: A group containing the local Ansible host. It contains an entry that should not be modified.

There are also a number of special groups:

- `[docker:children]`: A group of groups including all the nodes where Docker will be installed.
- `[vms:children]`: A group of groups including all the Virtual Machines involved apart from the local host.

Finally, you will find some variables defined for each group:

- `[vms:vars]`: A set of variables defined for all VMs. Currently only the size of the boot disk is defined here.
- `[ucp:vars]`: A set of variables defined for all nodes in the `[ucp]` group.
- `[dtr:vars]`: A set of variables defined for all nodes in the `[dtr]` group.
- `[worker:vars]`: A set of variables defined for all nodes in the `[worker]` group.

- `[lbs:vars]`: A set of variables defined for all nodes in the `[lbs]` group.
- `[nfs:vars]`: A set of variables defined for all nodes in the `[nfs]` group.
- `[logger:vars]`: A set of variables defined for all nodes in the `[logger]` group.

If you wish to configure your nodes with different specifications rather than the ones defined by the group, it is possible to declare the same variables at the node level, overriding the group value. For instance, you could have one of your workers with higher specifications by doing:

```
[worker]
worker01 ip_addr='10.0.0.10/16' esxi_host='esxi1.domain.local'
worker02 ip_addr='10.0.0.11/16' esxi_host='esxi2.domain.local'
worker03 ip_addr='10.0.0.12/16' esxi_host='esxi3.domain.local' cpus='16' ram='32768'

[worker:vars]
cpus='4'
ram='16384'
disk2_size='200'
node_policy='bronze'
```

In the example above, the `worker03` node would have 4 times more CPU and double the RAM compared to the rest of the worker nodes.

The different variables you can use are described in Table 4 below. They are all mandatory unless specified otherwise:

Table 4. Variables

Variable	Scope	Description
<code>ip_addr</code>	Node	IP address in CIDR format to be given to a node.
<code>esxi_host</code>	Node	ESXi host where the node will be deployed. If the cluster is configured with DRS, this option will be overridden.
<code>cpus</code>	Node/Group	Number of CPUs to assign to a VM or a group of VMs.
<code>ram</code>	Node/Group	Amount of RAM in MB to assign to a VM or a group of VMs.
<code>disk2_usage</code>	Node/Group	Size of the second disk in GB to attach to a VM or a group of VMs. This variable is only mandatory on Docker nodes (UCP, DTR, worker) and NFS node. It is not required for the logger node or the load balancers.
<code>node_policy</code>	Node/Group	SimpliVity backup policy to assign to a VM or a group of VMs. The name has to match one of the backup policies defined in the <code>group_vars/vars</code> file described in the next section.

Editing the group variables

Once the inventory is ready, the next step is to modify the group variables to match your environment. To do so, you need to edit the file `group_vars/vars`. The variables can be defined in any order but for the sake of clarity they have been divided into sections.

VMware configuration

All VMware-related variables are mandatory and are described in Table 5.

Table 5. VMware variables

Variable	Description
<code>vcenter_hostname</code>	IP or hostname of the vCenter appliance
<code>vcenter_username</code>	Username to log in to the vCenter appliance. It might include a domain, for example, <code>administrator@vsphere.local</code> . Note: The corresponding password is stored in a separate file (<code>group_vars/vault</code>) with the variable named <code>vcenter_password</code> .
<code>vcenter_validate_certs</code>	'no'
<code>datacenter</code>	Name of the datacenter where the environment will be provisioned
<code>vm_username</code>	Username to log into the VMs. It needs to match the one from the VM Template, so unless you have created a user, you must use <code>root</code> . Note: The corresponding password is stored in a separate file (<code>group_vars/vault</code>) with the variable named <code>vm_password</code> .
<code>vm_template</code>	Name of the VM Template to be used. Note that this is the name from a vCenter perspective, not the hostname.
<code>folder_name</code>	vCenter folder to deploy the VMs. If you do not wish to deploy in a particular folder, the value should be <code>/</code> . Note: If you want to deploy in a specific folder, you need to create this folder in the inventory of the selected datacenter before starting the deployment.
<code>datastores</code>	List of datastores to be used, in list format, i.e. [<code>Datastore1</code> ; <code>Datastore2</code> ...]. Please note that from a SimpliVity perspective, it is a best practice to use only one Datastore. Using more than one will not provide any advantages in terms of reliability and will add additional complexity. This datastore must exist before you run the playbooks.
<code>disk2</code>	UNIX® name of the second disk for the Docker VMs. Typically <code>/dev/sdb</code>
<code>disk2_part</code>	UNIX name of the partition of the second disk for the Docker VMs. Typically <code>/dev/sdb1</code>
<code>vsphere_plugin_version</code>	Version of the vSphere plugin for Docker. The default is <code>0.19</code> which is the latest version at the time of writing this document. The version of the plugin should match the version of the vSphere Installation Bundle (VIB) that you installed on the ESXi servers.

HPE SimpliVity configuration

All HPE SimpliVity variables are mandatory and are described in Table 6.

Table 6. SimpliVity variables

Variable	Description
<code>simplivity_username</code>	Username to log in to the SimpliVity Omnistack appliances. It might include a domain, for example, <code>administrator@vsphere.local</code> . Note: The corresponding password is stored in a separate file (<code>group_vars/vault</code>) with the variable named <code>simplivity_password</code> .
<code>omnistack_ovc</code>	List of Omnistack hosts to be used, in list format, i.e. [<code>omni1.local</code> ; <code>omni2.local</code> ...]. If your OmniStack virtual machines do not have their names registered in DNS, you can use their IP addresses.
<code>backup_policies</code>	List of dictionaries containing the different backup policies to be used along with the scheduling information. Any number of backup policies can be created and they need to match the

`node_policy` variables defined in the inventory. Times are indicated in minutes. All month calculations use a 30-day month. All year calculations use a 365-day year. The format is as follows:

`backup_policies:`

- name: 'daily'
 - days: 'All'
 - start_time: '11:30'
 - frequency: '1440'
 - retention: '43200'
- name: 'hourly'
 - days: 'All'
 - start_time: '00:00'
 - frequency: '60'
 - retention: '2880'

<code>dummy_vm_prefix</code>	In order to be able to back up the Docker volumes, a number of “dummy” VMs need to spin up. This variable will set a recognizable prefix for them.
<code>docker_volumes_policy</code>	Backup policy to use for the Docker Volumes.

Networking configuration

All network-related variables are mandatory and are described in Table 7.

Table 7. Network variables

Variable	Description
<code>nic_name</code>	Name of the device, for RHEL this is typically <code>ens192</code> and it is recommended to leave it as is
<code>gateway</code>	IP address of the gateway to be used
<code>dns</code>	List of DNS servers to be used, in list format, for example <code>['10.10.173.1', '10.10.173.2']</code>
<code>domain_name</code>	Domain name for your Virtual Machines
<code>ntp_servers</code>	List of NTP servers to be used, in list format, i.e. <code>['1.2.3.4', '0.us.pool.net.org'...]</code>

Docker configuration

All Docker-related variables are mandatory and are described in Table 8.

Table 8. Docker variables

Variable	Description
<code>docker_ee_url</code>	Note: This is a private link to your Docker EE subscription. This should be kept secret and defined in <code>group_vars/vault</code> . The value for <code>docker_ee_url</code> is the URL documented at the following URL: https://docs.docker.com/engine/installation/linux/docker-ee/rhel/
<code>rhel_version</code>	Version of your RHEL OS, such as <code>7.3</code> . The playbooks were tested with RHEL 7.3 and RHEL 7.4
<code>dtr_version</code>	Version of the Docker DTR you wish to install. You can use a numeric version or <code>latest</code> for the most recent one. The playbooks were tested with 2.3.3 and 2.4.0.

<code>ucp_version</code>	Version of the Docker UCP you wish to install. You can use a numeric version or <code>latest</code> for the most recent one. The playbooks were tested with UCP 2.2.3 and 2.2.4.
<code>images_folder</code>	Directory in the NFS server that will be mounted in the DTR nodes and that will host your Docker images.
<code>license_file</code>	Full path to your Docker EE license file (it should be stored in your Ansible host). License file is available from the Docker Store
<code>ucp_username</code>	Username of the administrator user for UCP and DTR, typically <code>admin</code> . Note: The corresponding password is stored in a separate file (<code>group_vars/vault</code>) with the variable named <code>ucp_password</code> .

Monitoring configuration

All Monitoring-related variables are described in Table 9. The variables determine the versions of various monitoring software tools that are used and it is recommended that the values given below are used.

Table 9. Monitoring variables

Variable	Description
<code>cadvisor_version</code>	<code>v0.25.0</code>
<code>node_exporter_version</code>	<code>v1.14.0</code>
<code>prometheus_version</code>	<code>v1.7.1</code>
<code>grafana_version</code>	<code>4.4.3</code>
<code>prom_persistent_vol_name</code>	The name of the volume which will be used to store the monitoring data. The volume is created using the vSphere Docker Volume plugin.
<code>prom_persistent_vol_size</code>	The size of the volume which will hold the monitoring data. The exact syntax is dictated by the vSphere Docker Volume plugin. The default value is 10GB.

Logspout configuration

Logspout is a log router for Docker containers that runs inside Docker. It attaches to all containers on a host, then routes their logs to the central logging VM. All Logspout-related variables are described in Table 10.

Table 10. Logspout variables

Variable	Description
<code>logspout_version</code>	<code>'master'</code> More details on Logspout are available later in this document.

Environment configuration

All environment-related variables are described in Table 11.

Table 11. Environment variables

Variable	Description
Env	<p>Dictionary containing all environment variables. It contains three entries described below. Please leave the proxy related settings empty if not required:</p> <ul style="list-style-type: none"> <code>http_proxy</code>: HTTP proxy URL, for example, 'http://15.184.4.2:8080'. This variable defines the HTTP proxy URL if your environment is behind a proxy. <code>https_proxy</code>: HTTPS proxy URL, for example, 'http://15.184.4.2:8080'. This variable defines the HTTPS proxy URL if your environment is behind a proxy <code>no_proxy</code>: List of hostnames or IPs that don't require proxy, for example, 'localhost,127.0.0.1,.cloudra.local,10.10.174.'

Editing the vault

Once your group variables file is ready, the next step is to create a vault file to match your environment. The vault file is similar to the group variables but it will contain all sensitive variables and will be encrypted.

There is a sample vault file named `group_vars/vault.sample` that you can use as a model for your vault file. To create a vault, you create a new file `group_vars/vault` and add entries similar to:

```
---
vcenter_password: 'xxx'
docker_ee_url: 'yoururl'
vm_password: 'xxx'
simplivity_password: 'xxx'
ucp_password: 'xxx'
rhn_orgid: 'Red Hat Organization ID'
rhn_key: 'Red Hat Activation Key'
```

`rhn_orgid` and `rhn_key` are the credentials needed to subscribe the virtual machines with Red Hat Customer Portal. For more info regarding activation keys see the following URL: <https://access.redhat.com/articles/1378093>

To encrypt the vault you need to run the following command:

```
# ansible-vault encrypt group_vars/vault
```

You will be prompted for a password that will decrypt the vault when required. You can update the values in your vault by running:

```
# ansible-vault edit group_vars/vault
```

For Ansible to be able to read the vault, you need to specify a file where the password is stored, for instance in a file called `.vault_pass`. Once the file is created, take the following precautions to restrict access to this file:

1. Change the permissions so only `root` can read it using `chmod 600 .vault_pass`
2. Add the file to your `.gitignore` file if you are pushing the set of playbooks to a git repository

Running the playbooks

At this point, the system is ready to be deployed. Go to the root folder and run the following command:

```
# ansible-playbook -i vm_hosts site.yml --vault-password-file .vault_pass
```

The playbooks should run for 25-35 minutes depending on your server specifications and the size of your environment.

Post deployment

The playbooks are intended to be used to deploy a new environment. You should only use them for Day 0 deployment purposes.

Overview of the playbooks

Create virtual machines

The playbook `playbooks/create_vms.yml` will create all the necessary Virtual Machines for the environment from the VM Template defined in the `vm_template` variable.

Configure network settings

The playbook `playbooks/config_networking.yml` will configure the network settings in all the Virtual Machines.

Distribute public keys

The playbook `playbooks/distribute_keys.yml` distributes public keys between all nodes, to allow each node to password-less log in to every other node. As this is not essential and can be regarded as a security risk (a worker node probably should not be able to log in to a UCP node, for instance), this playbook is commented out in `site.yml` by default and must be explicitly uncommented to enable this functionality.

Register the VMs with Red Hat

The playbook `playbooks/config_subscription.yml` registers and subscribes all virtual machines to the Red Hat Customer Portal. This is only needed if you pull packages from Red Hat. This playbook is commented out by default but you should uncomment it to make sure each VM registers with the Red Hat portal. It is commented out so that you can test the deployment first without having to unregister all the VMs from the Red Hat Customer Portal between each test. If you are using an internal repository, as described in the section "Create a VM template", you can keep this playbook commented out.

Install HAProxy

The playbook `playbooks/install_haproxy.yml` installs and configures the HAProxy package in the load balancer nodes. HAProxy is the chosen tool to implement load balancing between UCP nodes, DTR nodes and worker nodes.

Install NTP

The playbook `playbooks/install_ntp.yml` configures the `chrony` (NTP client) in all Virtual Machines in order to have a synchronized clock across the environment. It will use the server or servers specified in the `ntp_servers` variable in the group variables file.

Install Docker Enterprise Edition

The playbook `playbooks/install_docker.yml` installs Docker along with all its dependencies.

Install rsyslog

The playbook `playbooks/install_rsyslog.yml` installs and configures rsyslog in the logger node and in all Docker nodes. The logger node will be configured to receive all syslogs on port 514 and the Docker nodes will be configured to send all logs (including container logs) to the logger node.

Configure Docker LVs

The playbook `playbooks/config_docker_lvs.yml` performs a set of operations on the Docker nodes in order to create a partition on the second disk and carry out the LVM configuration, required for a sound Docker installation.

Docker post-install configuration

The playbook `playbooks/docker_post_config.yml` performs a variety of tasks to complete the installation of the Docker environment.

Install NFS server

The playbook `playbooks/install_nfs_server.yml` installs and configures an NFS server on the NFS node.

Install NFS clients

The playbook `playbooks/install_nfs_clients.yml` installs the required packages on the DTR nodes to be able to mount an NFS share.

Install and configure Docker UCP nodes

The playbook `playbooks/install_ucp_nodes.yml` installs and configures the Docker UCP nodes defined in the inventory.

Install and configure DTR nodes

The playbook `playbooks/install_dtr_nodes.yml` installs and configures the Docker DTR nodes defined in the inventory. Note that serialization is set to 1 in this playbook as two concurrent installations of DTR may in some cases be assigned the same replica ID.

Install worker nodes

The playbook `playbooks/install_worker_nodes.yml` installs and configures the Docker Worker nodes defined in the inventory.

Configuring monitoring

The playbook `playbooks/config_monitoring.yml` configures a monitoring system for the Docker environment by making use of Grafana, Prometheus, cAdvisor and node-exporter Docker containers.

Note

If you have your own monitoring solution, you can comment out the corresponding line in the main playbook `site.yml`

```
#- include: playbooks/config_monitoring.yml
```

After running the playbook, you can browse (HTTP) to the UCP load balancer IP address or FQDN on port 3000 (using a URL like `http://<ucp_lb>:3000`) and you will see the Grafana UI. The username and password are defaulted to `admin/admin`. When you log in, you can pick up the Dashboard that was imported by the playbooks (Click the `Dashboard` icon and select `Docker Swarm Monitor`) and observe the ongoing monitoring, as shown in Figure 21.

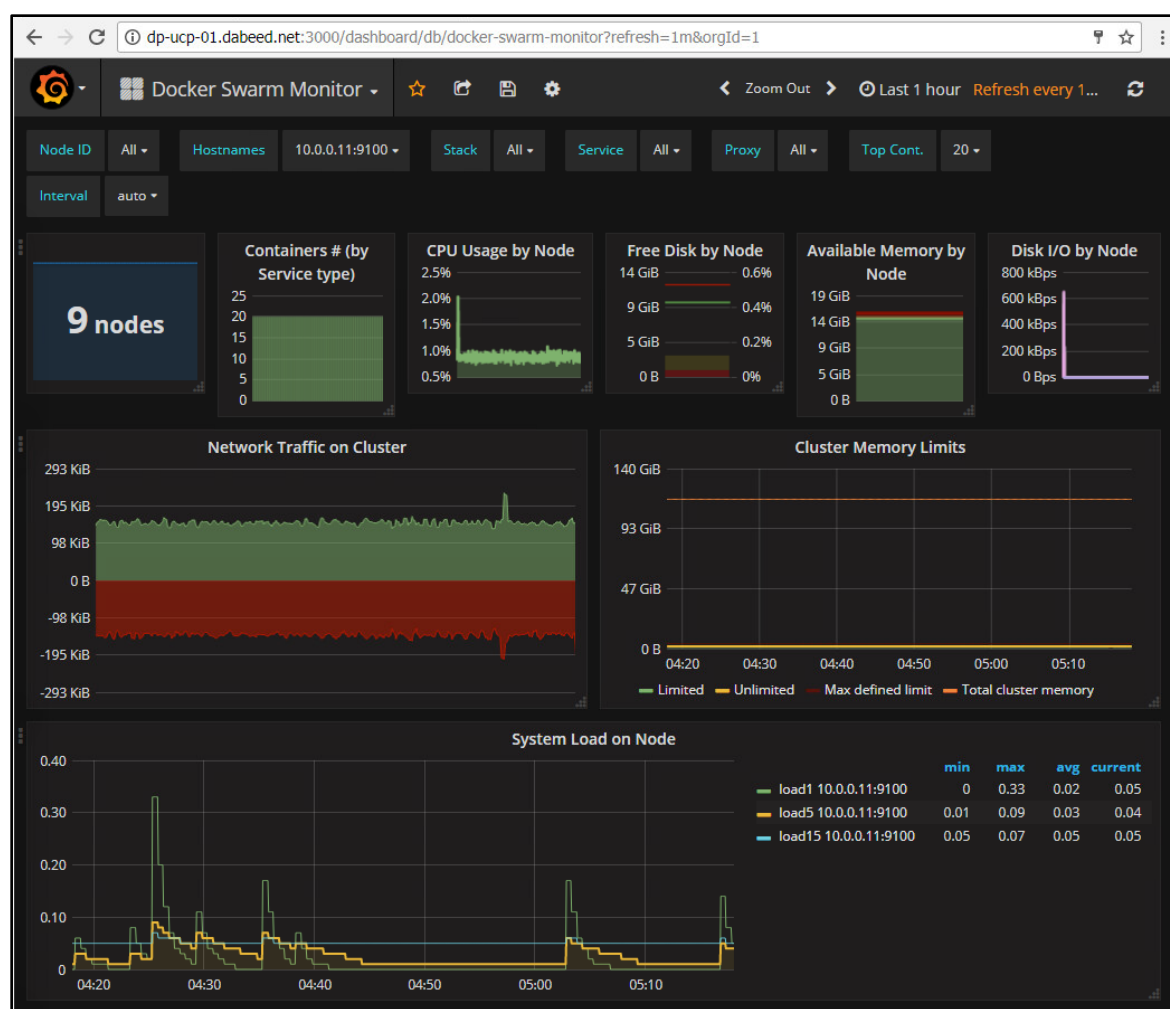


Figure 21. Grafana UI

The deployed Grafana dashboard includes cluster-wide metrics, node-specific metrics, and container-specific metrics. Monitored resources include disk I/O, memory, CPU utilization, network traffic, etc. The dashboard also highlights any containers configured with memory limits and the current memory utilization rate based on those limits. All of these metrics are provided via the node-exporter (responsible for OS and host metrics) and cAdvisor (responsible for container-specific metrics) instances running in the swarm. For more information about these tools and the metrics they expose, see the documentation links in their respective GitHub repositories: https://github.com/prometheus/node_exporter and <https://github.com/google/cadvisor>.

The provided dashboard is editable and extensible for those users who wish to add or remove panels or modify the underlying JSON code to change the look and behavior. Grafana offers many other sample dashboards on their website (<https://grafana.com/dashboards>) that may be used in place of the provided dashboard.

Configure dummy VMs to back up Docker volumes

The playbook `playbooks/config_dummy_vms_for_docker_volumes_backup.yml` ensures that you can back up Docker volumes that have been created using the vSphere plugin (vDVS) in SimpliVity. There is not a straight-forward way to do this, so you need to use a workaround. Since all Docker volumes are going to be stored in the `dockvols` folder in the datastore(s), you need to create a 'dummy' VM per

datastore. The `vmx`, `vmssd` and `vmkd` files from this VM will have to be inside the `dockvols` folder, so when these VMs are backed up, the volumes are backed up as well. Obviously these VMs don't need to take any resources and you can keep them powered off.

Configure SimpliVity backups

The playbook `playbooks/config_simplivity_backups.yml` configures the defined backup policies in the group variables file in SimpliVity and will include all Docker nodes plus the 'dummy' VMs created before, so the existing Docker volumes are also taken into account. The playbook will mainly use the SimpliVity REST API to perform these tasks. A reference to the REST API can be found at: https://api.simplivity.com/rest-api_getting-started_overview/rest-api_getting-started_overview_rest-api-overview.html

VM placement and number of HPE SimpliVity servers in the cluster

The placement of the various VMs deployed by the playbooks depends on whether DRS is enabled or not:

1. If DRS is not enabled, the placement of the VMs is specified in the Ansible inventory file `vm_hosts`
2. If DRS is enabled, the placement of the VMs is outside the control of the playbooks

The playbooks have only been tested with three nodes in the ESX cluster, but the following sections provide guidance on how to use more than three nodes.

Using more than three nodes when DRS is not enabled

The default `vm_hosts` file in the solution GitHub repository corresponds to a deployment on a 3-node HPE SimpliVity cluster. For each Ansible host in the inventory, you use the `esxi_hosts` variable to specify on which ESX hosts the VM should be placed. The following code extract shows 3 UCP VMs distributed across the three members of the cluster. This is the recommended placement as you don't want one node to host two UCP VMs as a failure of that node would result in the cluster losing quorum.

```
[ucp]
clh-ucp01 ip_addr='10.10.174.112/22' esxi_host='simply01.am2.cloudra.local'
clh-ucp02 ip_addr='10.10.174.113/22' esxi_host='simply02.am2.cloudra.local'
clh-ucp03 ip_addr='10.10.174.114/22' esxi_host='simply03.am2.cloudra.local'
```

In the above example, the first UCP VM will be placed on the ESX host named `simply01.am2.cloudra.local`. Note that the value for `esxi_host` is the name of the ESX host in the vCenter inventory.

The default `vm_hosts` inventory configures three Docker worker nodes and distributes them across the three ESX hosts:

```
[worker]
clh-worker01 ip_addr='10.10.174.122/22' esxi_host='simply01.am2.cloudra.local'
clh-worker02 ip_addr='10.10.174.123/22' esxi_host='simply02.am2.cloudra.local'
clh-worker03 ip_addr='10.10.174.124/22' esxi_host='simply03.am2.cloudra.local'
```

If you have more than three ESX hosts in your cluster, you can add an additional worker node as follows:

```
[worker]
clh-worker01 ip_addr='10.10.174.122/22' esxi_host='simply01.am2.cloudra.local'
clh-worker02 ip_addr='10.10.174.123/22' esxi_host='simply02.am2.cloudra.local'
clh-worker03 ip_addr='10.10.174.124/22' esxi_host='simply03.am2.cloudra.local'
clh-worker04 ip_addr='10.10.174.1xx/22' esxi_host='simply04.am2.cloudra.local'
```

You can also distribute the infrastructure VMs across four nodes rather than across the default nodes. For example, the default placement for the NFS server VM is as follows:

```
[nfs]
clh-nfs ip_addr='10.10.174.121/22' esxi_host='simply03.am2.cloudra.local'
```

Instead, you can change the placement NFS server VM, leveraging a fourth ESX node:

```
[nfs]
clh-nfs ip_addr='10.10.174.1xx/22'    esxi_host='simply04.am2.cloudra.local'
```

When you specify the placement of the VM, you should ensure that you follow these placement guidelines:

- Do not place two UCP VMs on the same node. If the node fails, the UCP cluster will lose quorum and the service will go down.
- Do not place two DTR replicas (VMs) on the same node. Once again, the cluster will lose quorum if that node fails.

Note

The OmniStack software maintains two replicas on two different hosts for each VM. As a result, when a VM is scheduled on an ESX server that does not have local access to one of the replicas, the VM will report the warning “SimpliVity VM Data Access Not Optimized”. You can safely ignore this warning.

Using more than three nodes when DRS is enabled

When DRS is enabled, it controls the placement of the VMs and as a result, the placement you have specified in the `vm_hosts` inventory is ignored. Instead, you use DRS rules to make sure that the UCP and DTR VMs are distributed across three nodes for the reasons explained earlier.

Warning

If you do not specify DRS rules to determine the placement, DRS will automatically move the VMs that report the “SimpliVity VM Data Access Not Optimized” warning to a node with a replica of the VM which may break the earlier placement guideline.

Accessing the UCP UI

Once the playbooks have run and completed successfully, the Docker UCP UI should be available by browsing to the UCP load balancer or any of the nodes via HTTPS. The authentication screen will appear as shown in Figure 22:

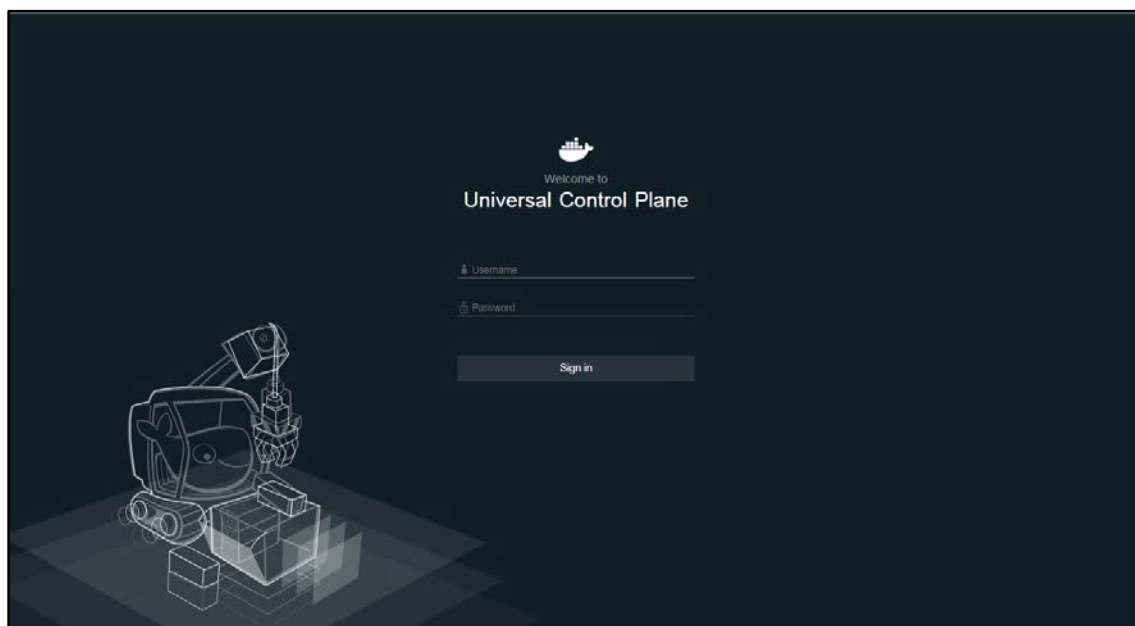


Figure 22. UCP authentication screen

Enter your credentials and the dashboard will be displayed as shown in Figure 23:

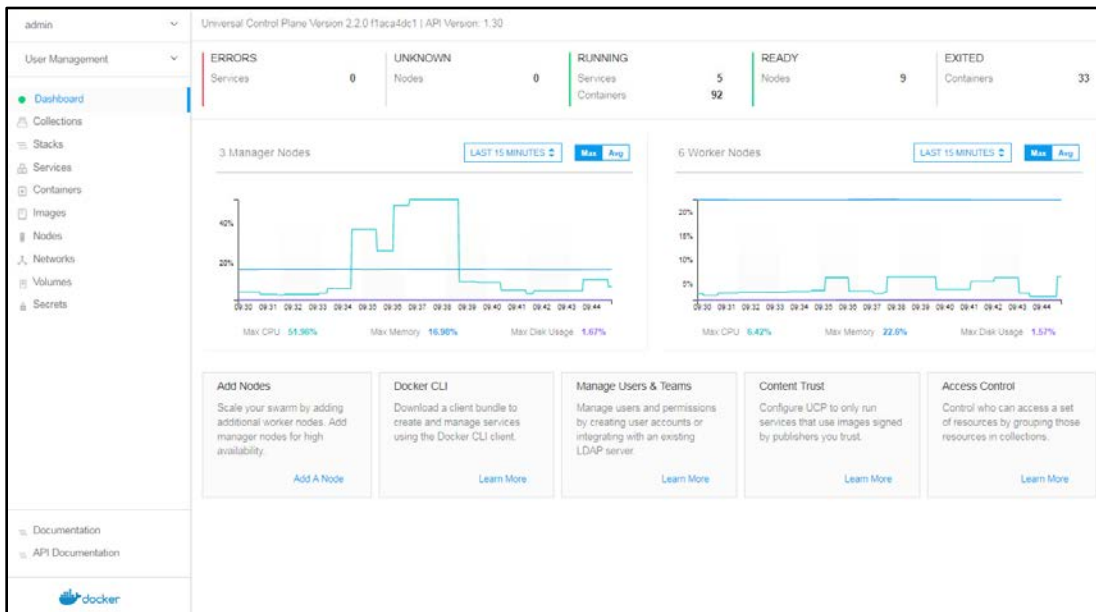


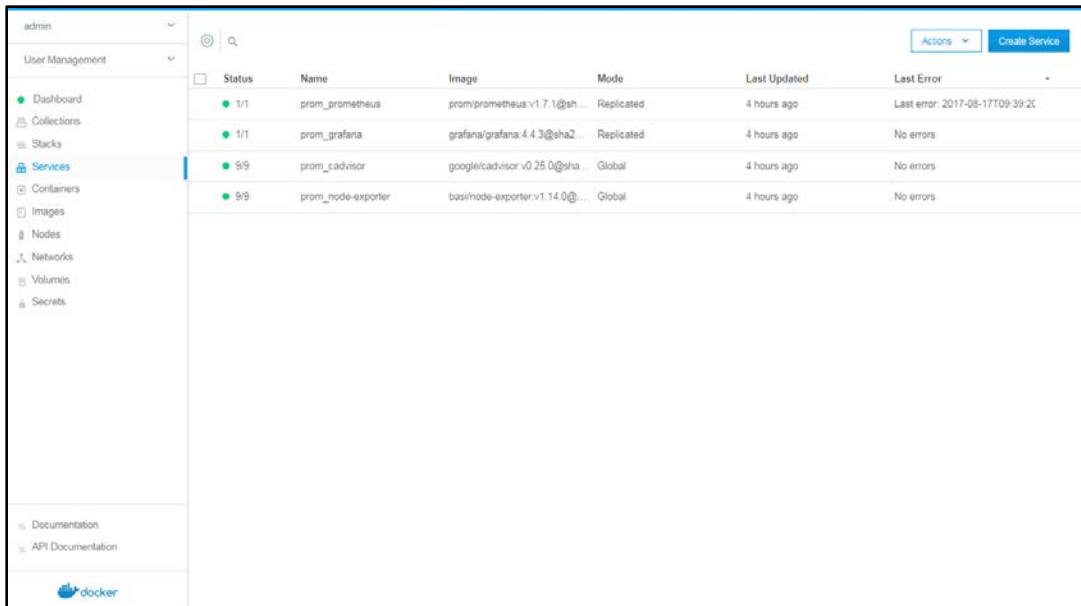
Figure 23. UCP dashboard

You should see all the nodes information in your Docker environment by clicking on Nodes, as shown in Figure 24:



Figure 24. Nodes information

Click on **Services** to see the monitoring services that were installed during the playbooks execution, as shown in Figure 25:



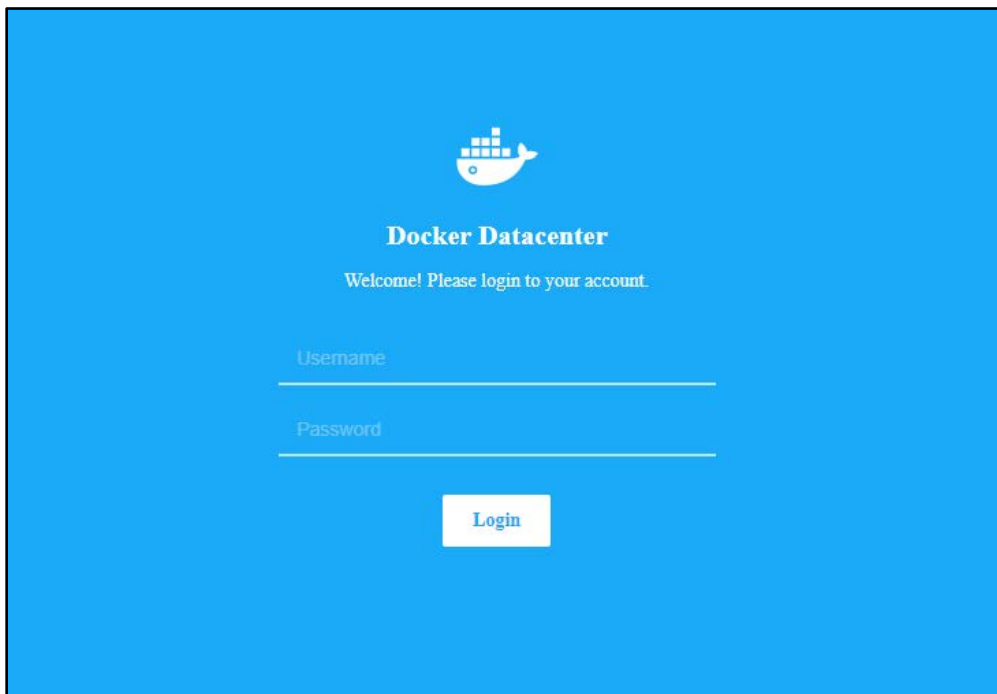
The screenshot shows the Docker Datacenter interface. On the left is a sidebar with navigation links: Dashboard, Collections, Stacks, Services (selected), Containers, Images, Nodes, Networks, Volumes, and Secrets. Below these are links for Documentation and API Documentation, and the Docker logo at the bottom. The main area displays a table of services. At the top right of the table are buttons for 'Actions' and 'Create Service'. The table has columns for Status, Name, Image, Mode, Last Updated, and Last Error. There are four services listed: prom_prometheus, prom_grafana, prom_cadvisor, and prom_node-exporter.

Status	Name	Image	Mode	Last Updated	Last Error
1/1	prom_prometheus	prom/prometheus:v1.7.1@sha2...	Replicated	4 hours ago	Last error: 2017-05-17T09:39:20
1/1	prom_grafana	grafana/grafana:4.4.3@sha2...	Replicated	4 hours ago	No errors
9/9	prom_cadvisor	google/cadvisor:v0.25.0@sha...	Global	4 hours ago	No errors
9/9	prom_node-exporter	basilnode-exporter:v1.14.0@...	Global	4 hours ago	No errors

Figure 25. Services information

Accessing the DTR UI

The Docker DTR UI should be available by browsing to the DTR load balancer or any of the nodes via HTTPS. The authentication screen will appear as shown in Figure 26:



The screenshot shows the Docker Datacenter authentication screen. It has a blue background with the Docker logo (a white whale) at the top center. Below the logo, the text 'Docker Datacenter' is displayed in white. Underneath, it says 'Welcome! Please login to your account.' in a smaller white font. There are two input fields: 'Username' and 'Password', both with white text and white borders. Below these fields is a white 'Login' button with blue text.

Figure 26. DTR authentication screen

Enter your UCP credentials and you should see the empty list of repositories as shown in Figure 27:

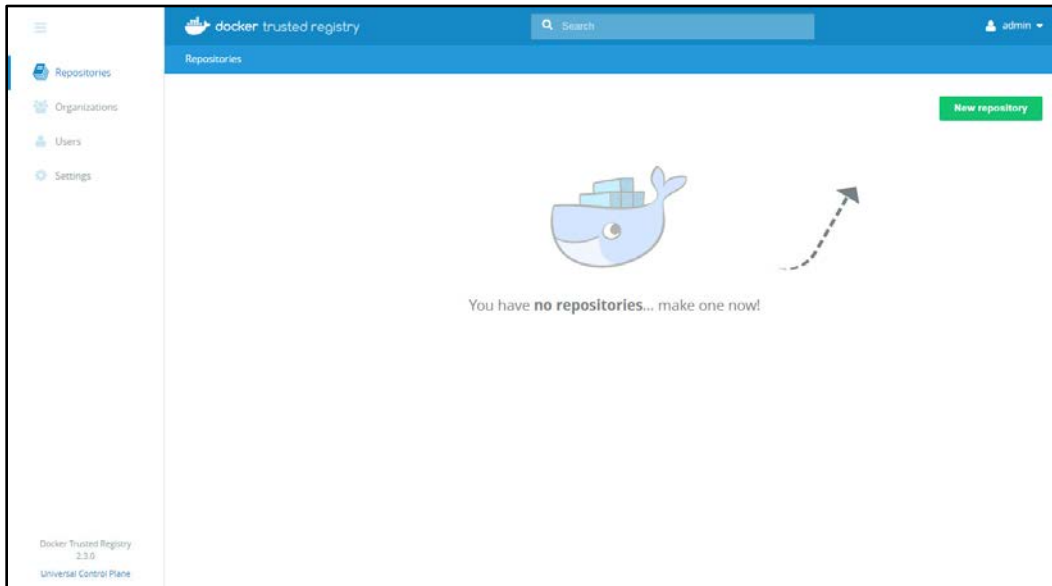


Figure 27. DTR repositories

If you navigate to Settings → Security, you should see the Image Scanning feature already enabled as shown in Figure 28. (Note that you need an Advanced license to have access to this feature.)

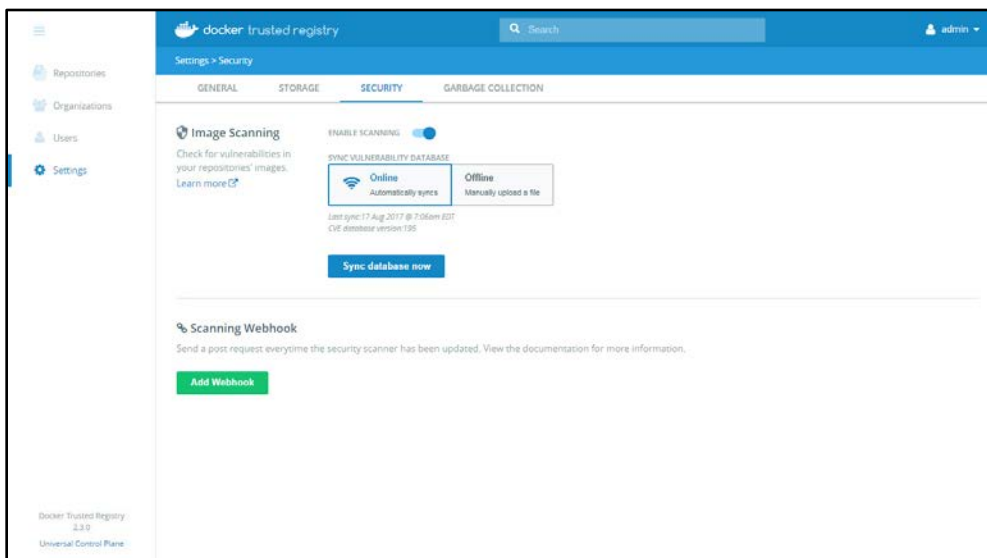


Figure 28. Image scanning in DTR

If you navigate to Settings → Storage, you should see that DTR is configured to use shared NFS storage as shown in Figure 29.

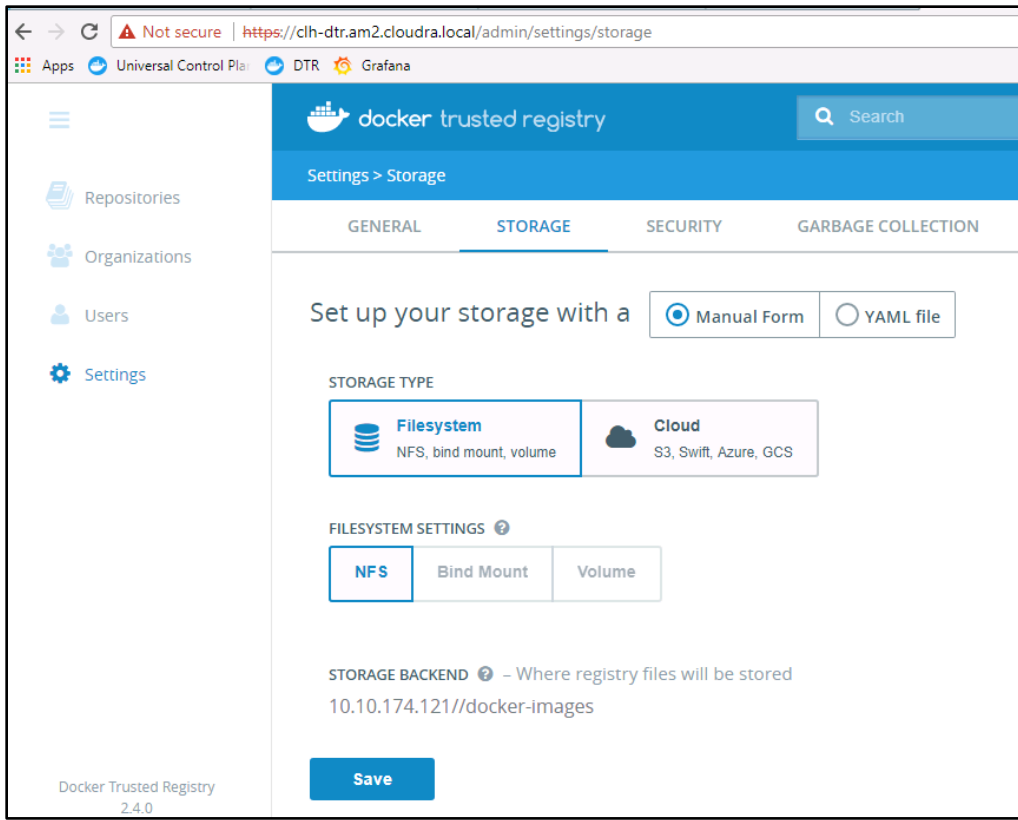


Figure 29. DTR storage settings

Security considerations

In addition to having all logs centralized in a single place and the image scanning feature enabled for the DTR nodes, there are other guidelines that should be followed in order to keep your Docker environment as secure as possible. The HPE Reference Configuration paper for securing Docker on HPE Hardware places a special emphasis on securing Docker in DevOps environments and covers best practices in terms of Docker security. The document can be found at: <http://h20195.www2.hpe.com/V2/GetDocument.aspx?docname=a00020437enw>. Some newer Docker security features that were not covered in the reference configuration are outlined below.

Prevent tags from being overwritten

By default, an image tag can be overwritten by a user with the correct access rights. As an example, an image such as `library/wordpress:latest` can be pushed and tagged by different users, yet have different functionality. This might make it difficult to trace back the image to the build that generated it.

Docker DTR can prevent this from happening with the immutable tags feature that can be configured on a per repository basis. Once an image is pushed with a tag, that particular tag cannot be overwritten.

More information about immutable tags can be found at:

<https://docs.docker.com/datacenter/dtr/2.3/guides/user/manage-images/prevent-tags-from-being-overwritten/>

Isolate swarm nodes to a specific team

With Docker EE Advanced, you can enable physical isolation of resources by organizing nodes into collections and granting Scheduler access for different users. To control access to nodes, move them to dedicated collections where you can grant access to specific users, teams, and organizations.

More information about this subject can be found at: <https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/isolate-volumes-between-teams/>.

Central logging

Central logging is implemented following the best practice documented by Docker which is available at https://success.docker.com/Architecture/Docker_Reference_Architecture%3ADocker_Logging_Design_and_Best_Practices.

Two types of logs are collected and sent to the central logging VM:

- OS system logs, these logs include the logs generated by the Docker engine itself
- `stderr` / `stdout` from containers (including DTR and UCP containers)

The first category of logs is collected by **rsyslog** and sent to the central logging VM. **rsyslog** on the individual VMs is configured using a template named `rsyslog.conf` located in `<directory>/file/` where `<directory>` is the directory into which you cloned the playbook repository.

The second category of logs is collected by the Docker Engine default logging driver (`json_file`) and all the corresponding logs are sent to the central logging VM by Logspout running on all Docker nodes (VMs). Logspout is configured as a global service and hence the Docker swarm will make sure one instance of the container is always running on each and every Docker node in the swarm. More information on Logspout can be found at <https://github.com/gliderlabs/logspout>.

Note

At the time of writing, some functionality required to run Logspout as a global service is only available in the master branch of Logspout so you need to specify the tag 'master' rather than 'latest' (v3.2.3) when pulling the Logspout image.

Operators can SSH to the central logging VM to locate the logs in the `/var/log/messages` directory. Developers can see container logs using the command `docker logs`.

HA considerations

Host failure

For details on the impact of a host failure, see Table 12.

Table 12. Host failure

Component	Technology	Duration	Consequence	Impact
UCP load balancer	Protected by VMware Cluster HA	Minutes	VM is failed-over	No access to service during failover
Workers load balancer	Protected by VMware Cluster HA	Minutes	VM is failed-over	No access to service during failover
DTR load balancer	Protected by VMware Cluster HA	Minutes	VM is failed-over	No access to service during failover
Central logging	Protected by VMware Cluster HA	Minutes	VM is failed-over	Some logs may be lost
NFS	Protected by VMware Cluster HA	Minutes	VM is failed-over	No push/pull of images.
UCP service	Protected by UCP scale out design	Seconds	Potentially new leader elected	Clients are disconnected. Clients should reconnect to the UCP service running on the other nodes.

DTR service	Protected by DTR scale out design	Seconds	DTR continues to be operational	Clients are disconnected. Clients should reconnect to the DTR service running on the other nodes.
Monitoring	Protected by Docker service (replicas)	Seconds	Grafana/Prometheus relocated	Services are restarted on the surviving UCP nodes.
Resource plane (worker nodes)	Protected by Docker scale out design	Seconds	Less capacity	Containers that were deployed as replicated services are restarted on the remaining worker nodes.
Docker volumes	SimpliVity		At least 1 replica will remain	Possible performance degradation if the container is scheduled on a node with no local replica.
User applications	Depends on application (service or standalone container)			If applications have been deployed as stacks or services, Docker swarm will make sure the number of required replicas is maintained.

VM failure

For details on the impact of a single VM failure, for example if it cannot boot or is deleted by accident, see Table 13.

Table 13. VM failure

Component	Technology	Duration	Consequence	Impact
UCP load balancer	Manual recovery	Minutes	Service unavailable	Recovery procedure required (restore from SimpliVity backup)
Workers load balancer	Manual recovery	Minutes	Service unavailable	Recovery procedure required (restore from SimpliVity backup)
DTR load balancer	Manual recovery	Minutes	Service unavailable	Recovery procedure required (restore from SimpliVity backup)
Central logging VM	Manual recovery	Minutes	Service unavailable. Logs may be lost	Recovery procedure required (restore from backup)
NFS VM	Manual recovery	Minutes	DTR push/pull unavailable	Recovery procedure required - see https://docs.docker.com/datacenter/dtr/2.4/guides/admin/backups-and-disaster-recovery/
UCP VM	UCP services are protected by UCP scale out design	Seconds	Transparent except for clients connected to the faulty VM. If monitoring tasks are running on the faulty VM (Grafana or Prometheus) they will be automatically restarted on another UCP VM.	Temporarily remove the faulty VM from the UCP load balancer. If a UCP VM is corrupted or unhealthy, the best way to recover is to deploy a new UCP VM. Do not restore from a SimpliVity backup.
DTR VM	DTR services are protected by DTR scale out design	Seconds	Transparent except for client connections using the DTR services when the VM fails	Temporarily remove the faulty VM from the DTR load balancer. If a DTR VM is corrupted or unhealthy, the best way to

recover is to deploy a new DTR replica. Do not restore from a SimpliVity backup.			
Resource plane (worker node)	Docker swarm scale out design	Reduced capacity Standalone containers may be lost. Required number of replicas for services will be restarted on the other worker nodes.	Temporarily remove the faulty VM from the DTR load balancer. A corrupted worker node should be removed from the swarm and a new worker VM should be deployed.

Docker volume backup and restore

In order to restore a Docker volume, you need to restore a special VM that has been deployed for the sole purpose of backing up Docker volumes. There is one such VM for each datastore defined in the `datastores` array in the `group_vars/vars` file. By default, a single datastore is specified in the playbooks:

```
datastores: [ 'Docker_CLH' ]
```

Note

The use of a single datastore is recommended. If you have configured multiple datastores, you need to understand and keep track of how your Docker volumes are distributed across the datastores.

The name of the special VM follows the pattern `<prefix>-in-dockervols-<Datastore>` where

- `<prefix>` is the value of the variable `dummy_vm_prefix` from the file `group_vars/vars`
- `<Datastore>` is the name of the datastore

For example, based on the default values in the scripts, the VM name would be `clh-VM-in-dockervols-Docker_CLH`.

Create a Docker volume

A single Docker volume will have been created for Prometheus using the vSphere driver as part of the initial deployment. To see this volume, use the `docker volume ls` command and limit the results to those volumes created using the vSphere driver.

```
[root@clh-ucp01 ~]# docker volume ls | grep vsphere
vsphere:latest      prom_clh-db-data@Docker_CLH
[root@clh-ucp01 ~]#
```

To create a Docker volume named `test_01`, you can use the `docker volume create` command specifying the vSphere driver:

```
[root@clh-ucp01 ~]# docker volume create -d vsphere test_01
test_01
```

You can check that the volume exists using the `'docker volume ls'` command:

```
[root@clh-ucp01 ~]# docker volume ls | grep vsphere
vsphere:latest      prom_clh-db-data@Docker_CLH
vsphere:latest      test_01@Docker_CL
```

You can attach a container to the volume and then add data to it by creating a text file with some arbitrary content:

```
[root@clh-ucp01 ~]# docker run -it --rm -v test_01:/tmp alpine sh -c "echo some test data here > /tmp/foo.txt"
[root@clh-ucp01 ~]#
```

If this is the first time you have used the alpine image, you may see additional output relating to download of image layers:

```
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
88286f41530e: Already exists
Digest: sha256:f006ecbb824d87947d0b51ab8488634bf69fe4094959d935c0c103f4820a417d
Status: Downloaded newer image for alpine:latest
```

The container will exit once the shell command has run and any unnamed volumes will be removed. However, the named volume test_01:/tmp will persist. To check that the data is still available after the container exited, spin up a new container and try to retrieve the data:

```
[root@clh-ucp01 ~]# docker run -it --rm -v test_01:/tmp alpine sh -c "cat /tmp/foo.txt"
some test data here
[root@clh-ucp01 ~]#
```

Automated backup

By default, the special VM and any Docker volume in the dockvols folder are backed up every hour. This is controlled by the following settings in the group_vars/vars file.

backup_policies:

```
- name: 'clh-gold'
  days: 'All'
  start_time: '00:00'
  frequency: '60'
  retention: '43200'
dummy_vm_prefix: 'clh-VM'
docker_volumes_policy: 'clh-gold'
```

The backup policy clh-gold is assigned to the special VM that is used to back up the Docker volumes. This policy specifies that a backup is taken every hour (frequency: '60' means sixty minutes) while the backup is retained for one month (retention: '43200' means 43200 minutes or thirty days).

Manual backup

Rather than waiting for an automated backup to take place, you can create a backup immediately. Right-click on the special VM, in this case clh-VM-in-dockervols-Docker_CLH, select All HPE SimpliVity Actions and choose Backup Virtual Machine as shown in Figure 30.

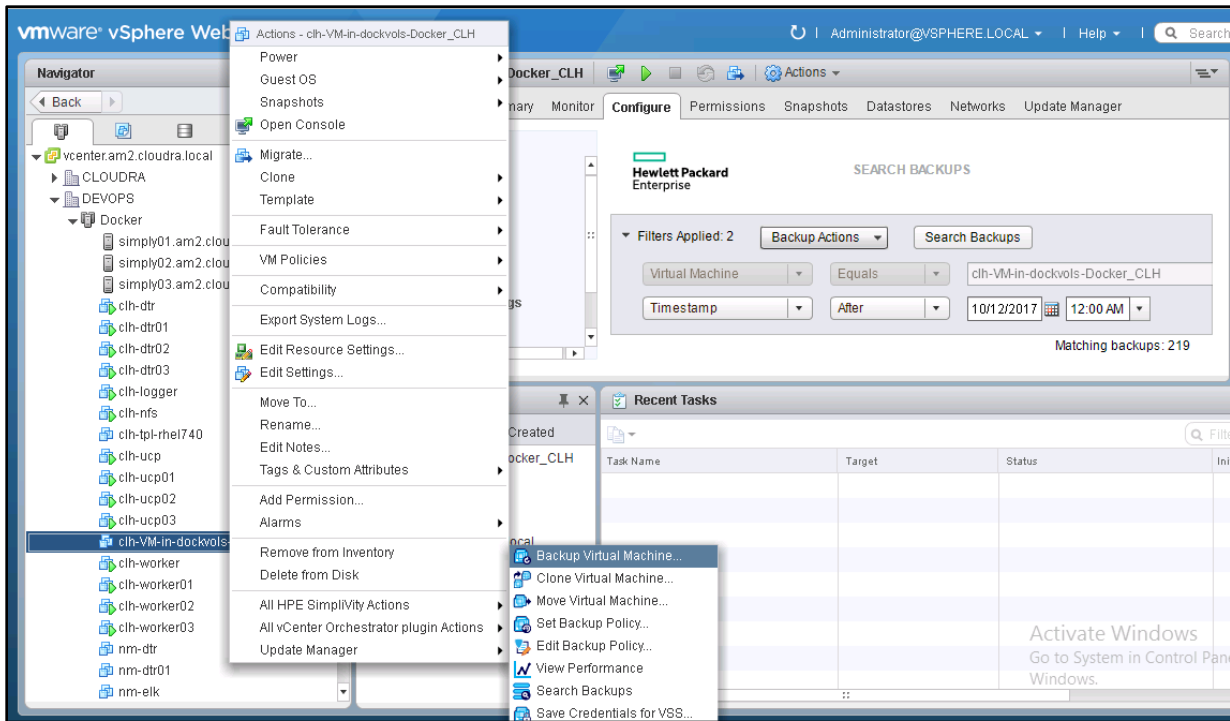


Figure 30. Backup virtual machine

You can specify a backup name, in this case `manual_backup_test_01`, as shown in Figure 31.

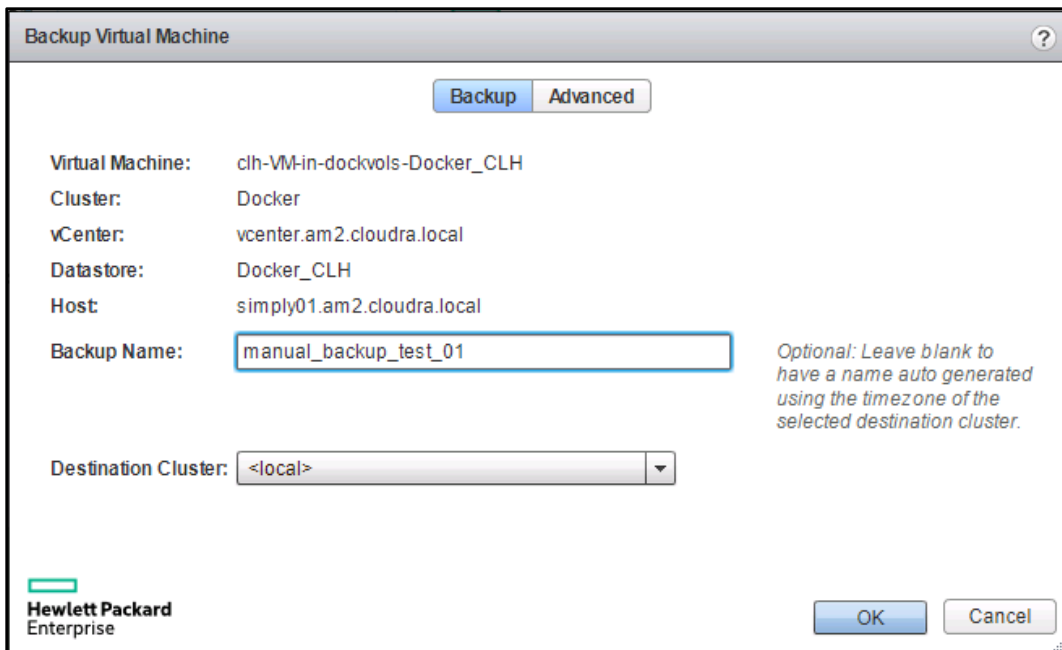


Figure 31. Backup virtual machine details

Restore

Right-click on the special VM, in this case `clh-VM-in-dockwols-Docker_CLH`. On the Configure tab, select HPE SimpliVity Search Backups as shown in Figure 32.

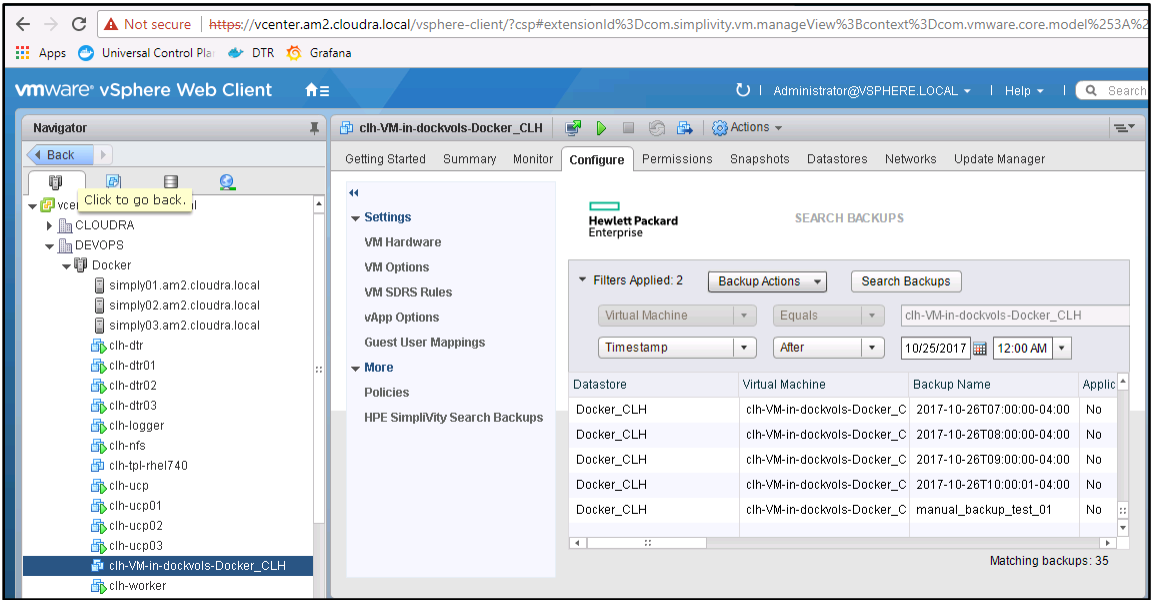


Figure 32. Search backups

You can narrow the search based on the time of the backup. If you are restoring from an automatic backup, the name will be the timestamp of the backup. If you are restoring from a manual backup, the name will be the one you specified earlier when creating the backup, in this case `manual_backup_test_01`.

Right-click on the backup you wish to restore, as shown in Figure 33, and select 'Restore Virtual Machine'.

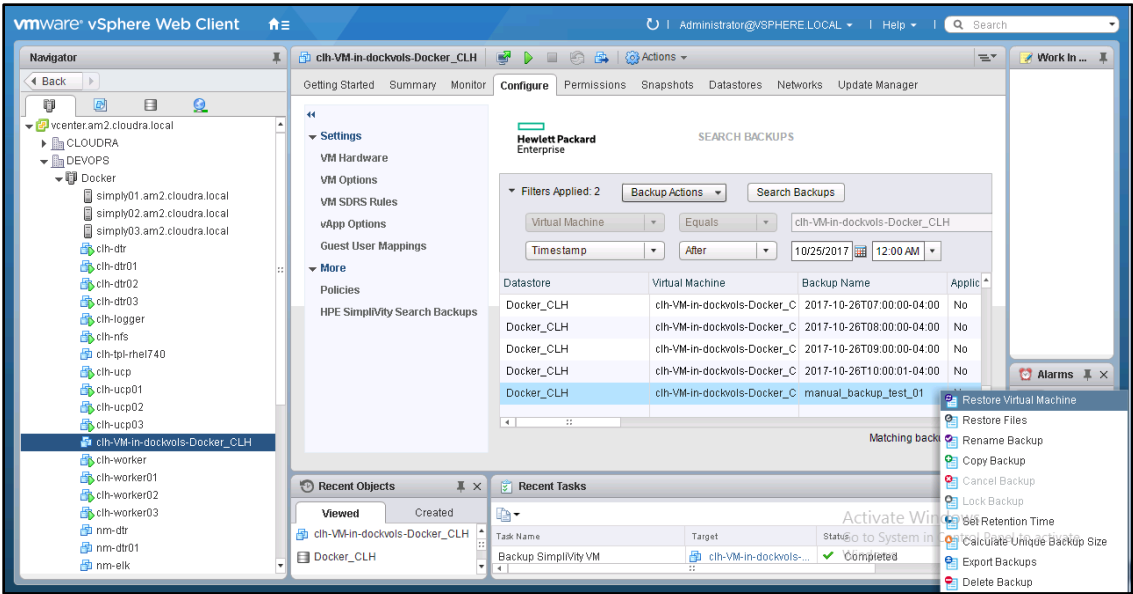


Figure 33. Restore virtual machine

In the details screen, shown in Figure 34, you can choose a name for the new virtual machine and specify the datastore.

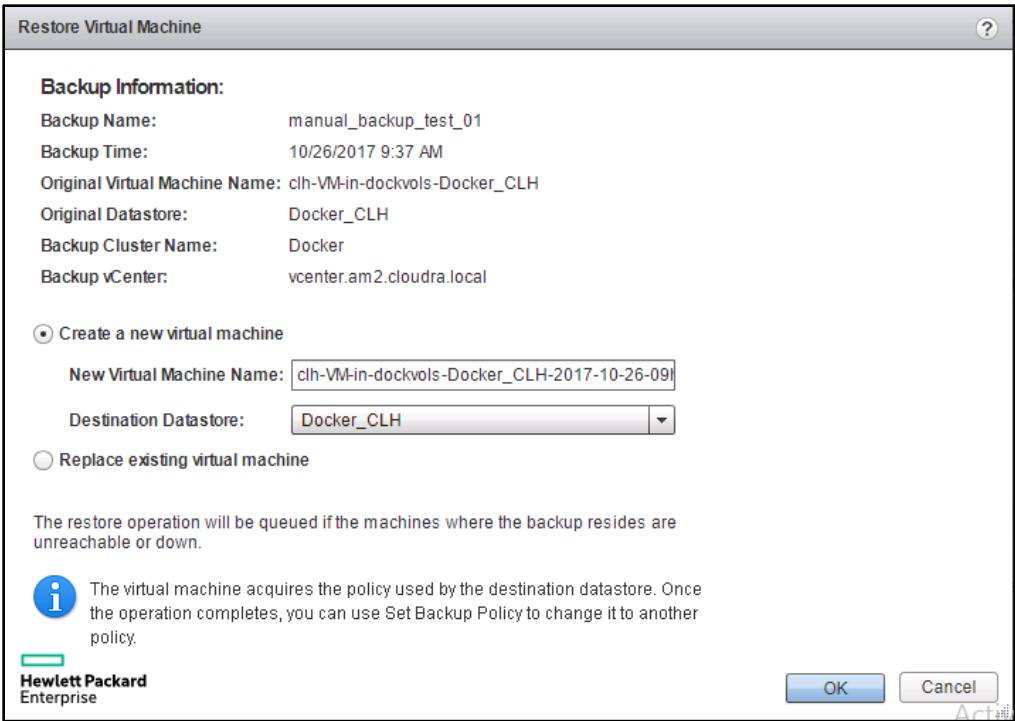


Figure 34. Restore virtual machine details

The name of the new virtual machine will default to a combination of the special VM name and a timestamp, in this instance `clh-vm-in-dockervols-Docker_CLH-2017-10-26-09h47m00s`. The datastore should be the one specified in the `datastores` array from the `group_vars/vars` file. Click OK to restore the virtual machine.

Once the virtual machine has been restored, navigate to the datastore and locate the new VM in the file browser, as shown in Figure 35.

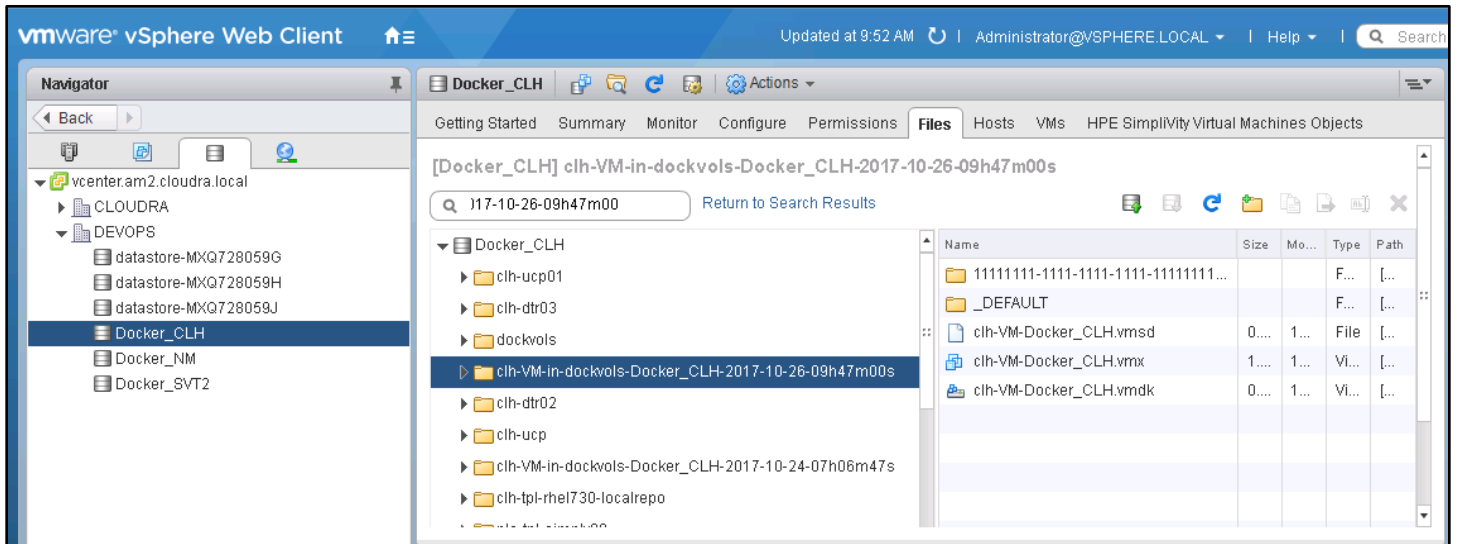


Figure 35. Browse to restored virtual machine

Navigate to the folder named `1111111-1111-1111-1111-...` as shown in Figure 36. You will see files with names based on the Docker volume name that you used at the start, in this instance `test_01.vmdk` and `test_01-478...f1f.vmdfd`

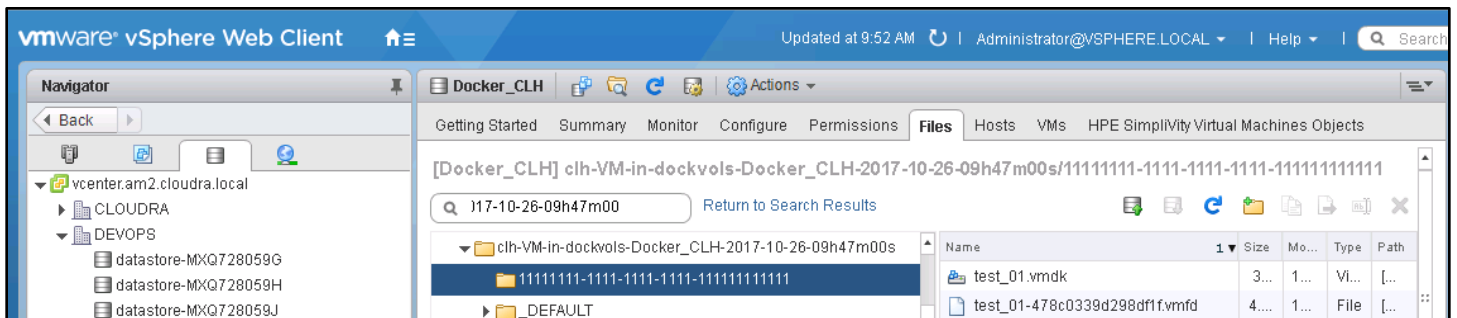


Figure 36. Locate vmdk and vmfd files

You need to move these two files to the `dockvols` sub-directory named `11111111-1111-1111-1111-...` in the same datastore. Right-click on the `.vmdk` file and choose **Move to...** as shown in Figure 37.

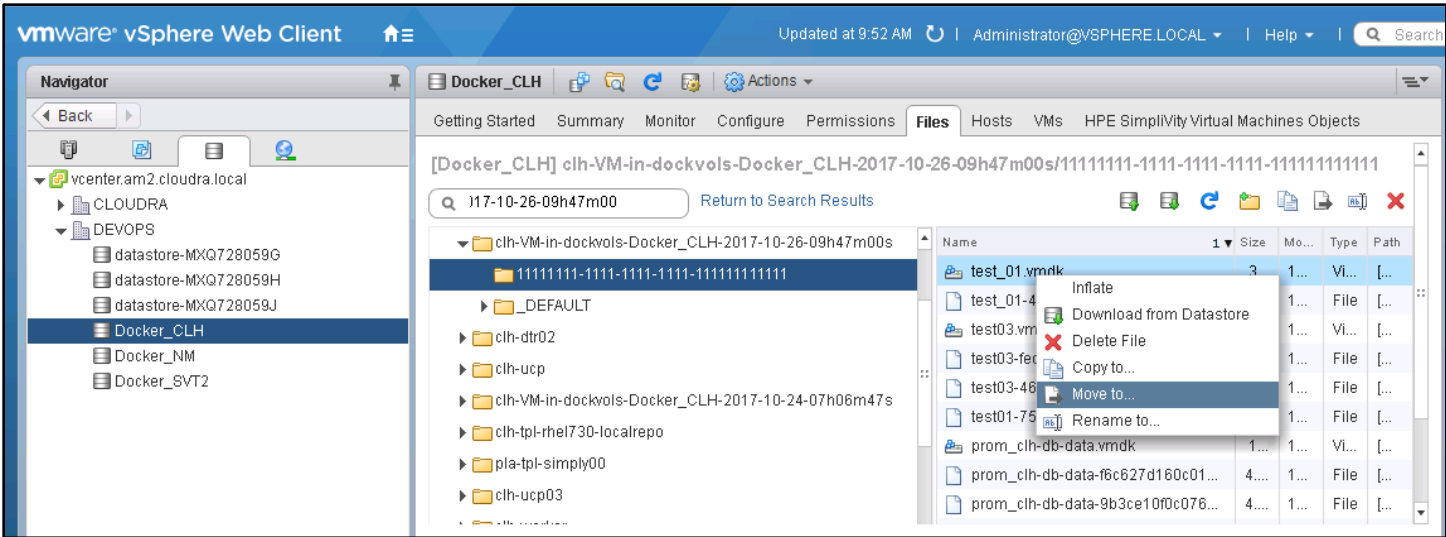


Figure 37. Move files

Set the destination folder to the dockvols sub-directory named 1111111-1111-1111-1111- . . . as shown in Figure 38.

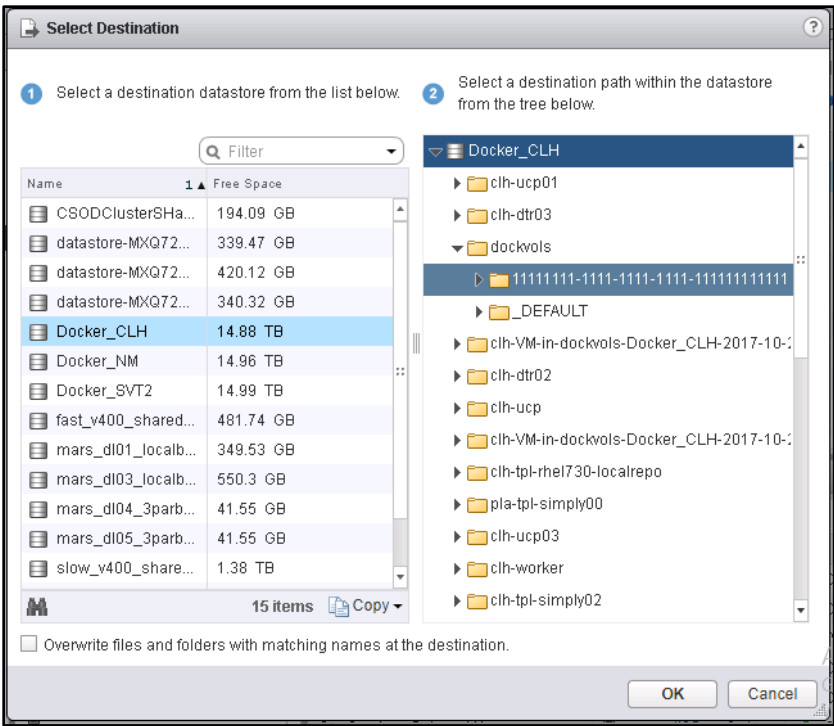


Figure 38. Move to destination

It is only necessary to move the .vmdk file as the .vmfd file will automatically follow. The dockvols sub-directory named 1111111-1111-1111-1111- . . . should now contain both files as shown in Figure 39.

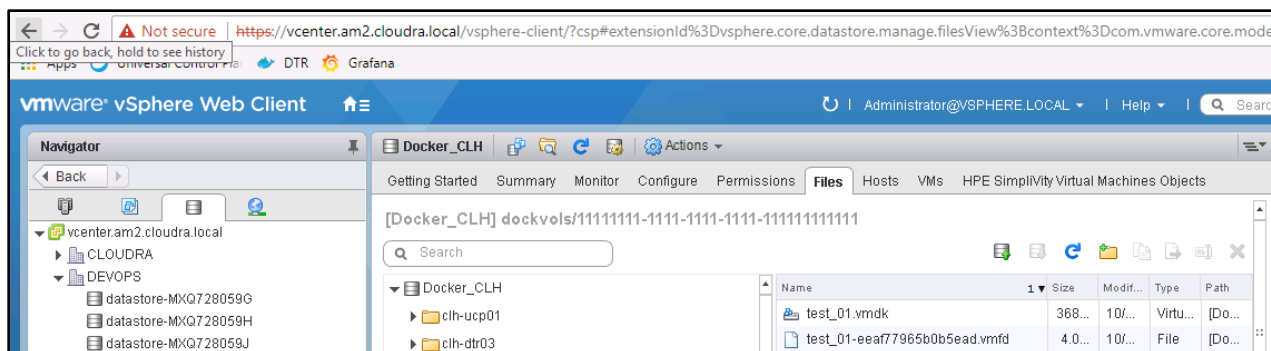


Figure 39. Files moved to destination

Test the restore

You can check that the volume `test_01` has been restored by using the `docker volume ls` command again.

```
[root@clh-ucp01 ~]# docker volume ls | grep vsphere
vsphere:latest      prom_clh-db-data@Docker_CLH
vsphere:latest      test_01@Docker_CLH
```

You can verify that the volume contains the correct data by spinning up a container and running a shell command:

```
[root@clh-ucp01 ~]# docker run -it --rm -v test_01:/tmp alpine sh -c "cat /tmp/foo.txt"
some test data here
```

The data you entered in the text file before performing the backup and deleting the volume is available once again after restoring the volume.

Disaster Recovery

In order to protect your installation from disasters, you need to take regular backups of the swarm, UCP and DTR. In the event where half or more manager nodes are lost and cannot be recovered to a healthy state, the system is considered to have lost quorum and can only be restored through the manual procedure described in the Docker documentation.

For more information, see <https://docs.docker.com/datacenter/ucp/2.2/guides/admin/backups-and-disaster-recovery/>.

Solution lifecycle management

Introduction

Lifecycle management with respect to this solution refers to the maintenance and management of software and hardware of various components that make up the solution stack. Lifecycle management is required to keep the solution up-to-date and ensure that the latest versions of the software are running to provide optimal performance, security and fix any existing defects within the product.

In this section, we will cover lifecycle management of the different components that are used in this solution. The architectural diagram of the solution in Figure 40 shows the software and hardware stacks that make up the solution. Each stack is shown in a different color.

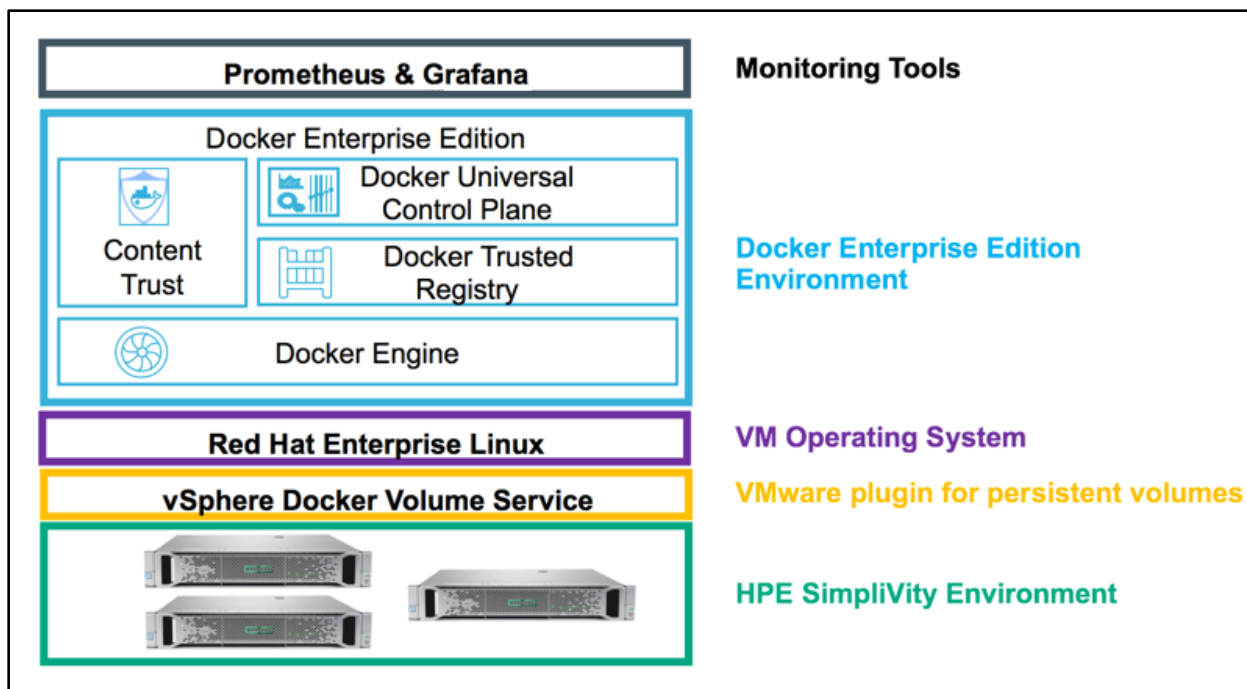


Figure 40. Solution stack

Based on the diagram above, lifecycle of the following stacks need to be maintained and managed.

1. Monitoring (Prometheus and Grafana) Tools
2. Docker Enterprise Edition Environment
3. Virtual Machine Operating systems
4. vSphere Docker Volume Service
5. SimpliVity environment

The general practice and recommendation is to follow a bottom-up approach for updating all components of the environment and making sure the dependencies are met. In our solution, we would start with the SimpliVity stack and end with the Prometheus and Grafana monitoring environment. If all components are not being updated at the same time, the same approach can be followed – updating only the components that require updates while adhering to the interdependencies of each component that is being updated.

HPE SimpliVity environment

The HPE SimpliVity environment is made up of proprietary SimpliVity software, VMware software and HPE firmware. There are interdependencies between the various components that need to be accounted for and are provided in the table below. The components in Table 14 are part of the SimpliVity environment that require lifecycle management.

In general, ensure that the software bits for the Arbiter and vSphere extension corresponding to an OmniStack release are used.

Table 14. SimpliVity components

Order	Component	Dependency (compatibility)	Download/Documentation
1	HPE SimpliVity Arbiter	1. HPE OmniStack	SimpliVity OmniStack for vSphere Upgrade Guide
2	HPE SimpliVity VMware Plug-in	1. HPE SimpliVity Arbiter 2. HPE OmniStack	Download software bits from HPE's support website. http://www.hpe.com/support
3	HPE Omnistack	1. HPE SimpliVity VMware Plug-in 2. HPE SimpliVity Arbiter	

VMware components

The SimpliVity solution used in this deployment guide is built on VMware vSphere. VMware ESXi and vCenter (see Table 15) are the two components from VMware that are leveraged by the SimpliVity software.

The VMware ESXi and vCenter versions must be compatible with each other and with the HPE OmniStack version that is running on the SimpliVity systems.

Table 15. VMware components

Order	Component	Dependency (compatibility)	Download/Documentation
1	VMware vCenter	1. HPE OmniStack 2. VMware ESXi	VMware Upgrade for SimpliVity
2	VMware ESXi	1. HPE OmniStack 2. VMware vCenter	

HPE server software

SimpliVity servers are based on HPE server platforms and require a compatible firmware version to function with HPE OmniStack Software, as shown in Table 16.

Table 16. HPE server components

Order	Component	Dependency (compatibility)	Download/Documentation
1	HPE Firmware	1. HPE OmniStack Software	Firmware Upgrade for SimpliVity

vSphere Docker Volume Service plug-in

vSphere Docker Volume Service plug-in is part of an open source project by VMware that enables running stateful containers by providing persistent Docker volumes leveraging existing storage technology from VMware. There are two parts to the plug-in, namely, client software and

server software (see Table 17). Every version of the plug-in that is released includes both pieces of software and it is imperative that the version number installed on the client side and server side are the same.

When updating the Docker Volume Service plug-in, ensure the ESXi version you are running is supported and that the client software is compatible with the operating system.

Table 17. vSphere Docker Volume Service components

Order	Component	Dependency (compatibility)	Download/Documentation
1	Server Software	1. VMware ESXi	vSphere Docker Volume Service on GitHub
		2. Docker EE	
2	Client Software	1. VM Operating System	
		2. Docker EE	

Red Hat Enterprise Linux operating system

This solution is built using Red Hat Enterprise Linux (see Table 18) as the base operating system. When upgrading the operating system on the VMs, first verify that the OS version is compatible to run Docker EE by looking at the [Docker OS compatibility metric](#).

Table 18. Operating system

Order	Component	Dependency	Download/Documentation
1	Red Hat Enterprise Linux	1. Docker EE	RHEL
		2. vDVS client software plugin	

Docker EE environment

Each release of Docker Enterprise Edition contains three technology components – UCP, DTR and the Docker Daemon or Engine. It is imperative that the components belonging to the same version are deployed or upgraded together – see Table 19.

A banner will be displayed on the UI, as shown in Figure 41, when an update is available for UCP or DTR. You can start the upgrade process by clicking on the banner.

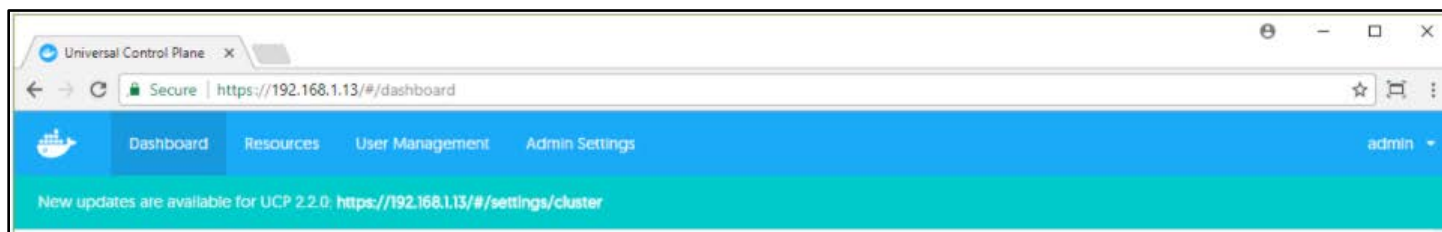


Figure 41. Docker update notification

Table 19. Docker EE components

Order	Component	Dependency	Download/Documentation
1	Docker Daemon/Engine	1. VM Operating System	1. Docker Lifecycle Maintenance
2	Universal Control Plane	2. vDVS plugin	2. Docker Compatibility Matrix

Monitoring tools

Prometheus and Grafana monitoring tools (see Table 20) run as containers within the Docker environment. Newer versions of these tools can be deployed by pulling the Docker images from Docker Hub. Verify that the version of Prometheus that is being used is compatible with the version of Docker EE.

Table 20. Monitoring tools

Order	Component	Dependency	Download/Documentation
1	Prometheus	1. Grafana 2. Docker EE	1. Prometheus Images on Docker Hub 2. Upgrading Grafana
2	Grafana	1. Prometheus 2. Docker EE	

High-level dependency map

Based on the lifecycle management details provided above, Figure 42 is a consolidated diagram that shows the dependencies between the various components in the solution stack. Bi-directional arrows between components indicate that the two components have an interoperability dependence. Before upgrading a component to a newer version, you must ensure that the new version of that component is compatible with the current version of any dependent components.

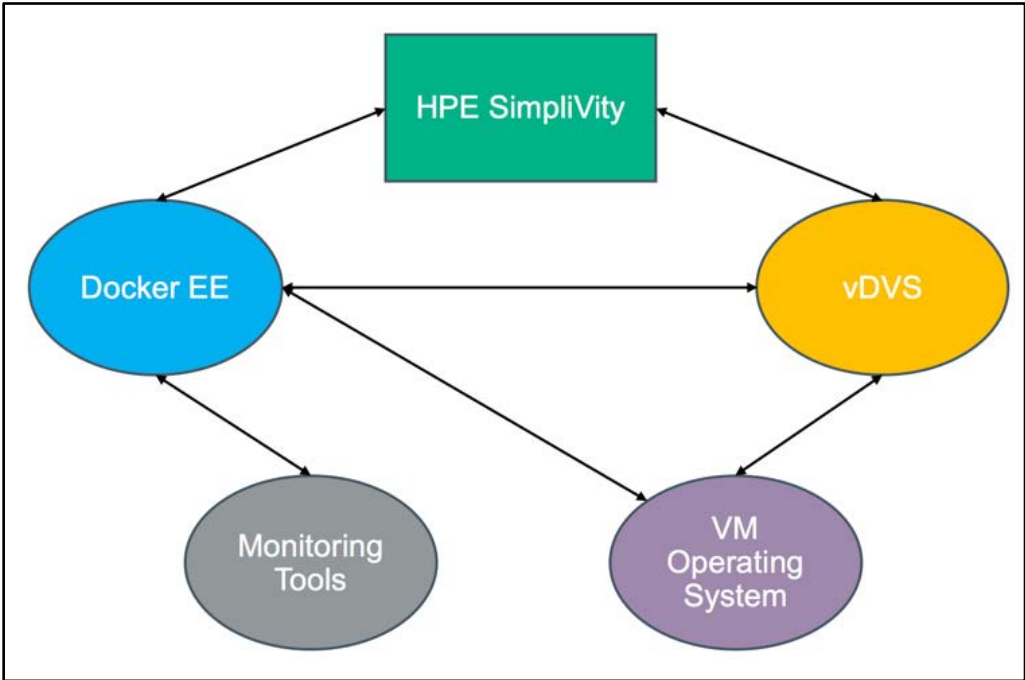


Figure 42. High-level dependency map

Appendix A

This is a recommended BOM for the 3 node HPE Express Containers with Docker EE: Ops Edition. Please verify with your HPE account team to ensure these are the latest BOM SKUs.

Table 21. BOM for each SimpliVity node. Dual Socket – 14 cores per Socket, 5x1.92TB value flash

Product #	Product Description	Qty
Q8D81A	HPE SimpliVity 380 Gen10 Node	1
826856-B21	2.2GHz Xeon Gold 5120 processor (1 chip, 14 cores)	1
826856-L21	2.2GHz Xeon Gold 5120 processor (1 chip, 14 cores)	1
Q8D85A	HPE SimpliVity 240G 12 DIMM FIO Kit	2
Q8D91A	HPE SimpliVity 380 SM Val Kit	1
804331-B21	Smart Array P408i-a SR Gen10 (8 Internal Lanes/2GB Cache) 12G SAS Modular Controller	1
826703-B21	HPE DL380 Gen10 Sys Insght Dsply Kit	1
665243-B21	HPE Ethernet 10Gb 2P 560FLR-SFP+ Adptr (SFP+ connector)	1
864279-B21	HPE TPM 2.0 Gen10 Kit	1
865414-B21	800W Flex Slot Platinum Hot Plug Low Halogen Power Supply Kit	2
733664-B21	HPE 2U Cable Management Arm for Easy Install Rail Kit	1
867809-B21	HPE Gen10 2U Bezel Kit	1
758959-B22	HPE Legacy FIO Mode Setting	1
874543-B21	HPE 1U Gen10 SFF Easy Install Rail Kit	1
BD505A - HPE iLO Adv incl 3yr TSU		
BD505A	1-Svr Lic (HPE)	1
Q8A60A	HPE OmniStack 8-14c 2P Small SW	1

Resources and additional links

HPE Reference Architectures

hpe.com/info/ra

HPE | Docker Alliance page

<http://h22168.www2.hpe.com/us/en/partners/docker/>

HPE Servers

hpe.com/servers

HPE Storage

hpe.com/storage

HPE Networking

hpe.com/networking

HPE Technology Consulting Services

hpe.com/us/en/services/consulting.html



Sign up for updates



**Hewlett Packard
Enterprise**

© Copyright 2017 Hewlett Packard Enterprise Development LP. The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group.