HPE Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure
Contents

Executive Summary ........................................................................................................... 4
Solution overview .............................................................................................................. 4
  New in this release .......................................................................................................... 4
Solution configuration ...................................................................................................... 5
High availability ................................................................................................................. 6
Sizing considerations ........................................................................................................ 7
Disaster recovery ............................................................................................................. 9
Security ............................................................................................................................. 10
Solution components ...................................................................................................... 10
  Hardware ....................................................................................................................... 10
  Software ....................................................................................................................... 11
  Application software ................................................................................................. 12
Preparing the environment.............................................................................................. 17
  Verify prerequisites ...................................................................................................... 17
  Enable vSphere High Availability (HA) ...................................................................... 17
  Install vSphere Docker Volume Service driver on all ESXi hosts ................................ 18
  Create the Ansible node ............................................................................................ 18
  Create the Red Hat Linux template ............................................................................ 19
  Create the Windows Template (optional) .................................................................... 20
Configuring the solution components .............................................................................. 21
  Ansible configuration ................................................................................................. 21
  Editing the inventory .................................................................................................... 22
  VMware configuration ................................................................................................. 24
  Networking configuration ............................................................................................. 24
  Environment configuration ......................................................................................... 25
  Docker configuration ................................................................................................... 25
  Orchestrator configuration ......................................................................................... 25
  Kubernetes configuration ......................................................................................... 25
  Windows configuration ............................................................................................... 26
  Splunk configuration .................................................................................................. 27
  Sysdig configuration .................................................................................................... 28
  Prometheus and Grafana configuration ...................................................................... 30
  Protecting sensitive information ................................................................................ 31
  Inventory group variables ......................................................................................... 32
Overview of the playbooks .............................................................................................. 33
  site.yml and related playbooks .................................................................................. 33
  Windows playbooks ................................................................................................. 34
## Reference Architecture

Backup and restore playbooks ................................................................. 35
Convenience playbooks ..................................................................... 35
Convenience scripts ......................................................................... 35
Running the playbooks ...................................................................... 35
  Linux-only deployment .................................................................... 35
  Hybrid Windows and Linux deployment ........................................... 35
Post deployment ................................................................................ 35
Deploying the NFS provisioner for Kubernetes ................................. 43
Deploying Sysdig monitoring ............................................................. 53
Solution lifecycle management .......................................................... 56
  Synergy environment ..................................................................... 57
  VMware components ..................................................................... 57
  vSphere Docker Volume Service plug-in ........................................... 57
  Red Hat Enterprise Linux operating system .................................... 57
  Docker EE Environment ................................................................. 57
  Monitoring tools ........................................................................... 58
  Windows operating system and Docker EE ..................................... 58
Backup and restore ........................................................................... 59
  Backup and restore UCP and DTR .................................................... 59
  Backup and restore Docker persistent volumes .............................. 66
  Integrate UCP and DTR backup with HPE RMC and HPE StoreOnce ......................................................... 69
Summary .......................................................................................... 71
Appendix A: Bill of materials .............................................................. 71
  Software Licenses .......................................................................... 73
Appendix B: Using customer supplied certificates for UCP and DTR ............................................................. 73
  Generating and testing certificates ................................................... 74
  Verify your certificates ................................................................... 76
Appendix C: Enabling SSL between the universal forwarders and the Splunk indexers using your certificates ................................................................. 76
  Limitations ..................................................................................... 77
  Prerequisites .................................................................................. 77
  Before you deploy ......................................................................... 78
  Hybrid environment Linux / Windows ............................................. 78
Appendix D: How to check that certs were deployed correctly ............ 78
Resources and additional links ........................................................... 81
Executive Summary

HPE Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure 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 Synergy 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 Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure 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. This Reference Configuration provides a solution for IT operations, addressing the need for a production-ready environment that is easy to deploy and manage.

This release supports Kubernetes via Docker Enterprise Edition (EE) 2.0, which is the only platform that manages and secures applications on Kubernetes in multi-Linux, multi-OS and multi-cloud customer environments. This document describes the best practices for deploying and operating Containers as a Service with Docker Enterprise Edition (EE). 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.

Target Audience: This document is primarily aimed at technical individuals working in the operations side of the software pipeline, such as infrastructure architects, 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: The present document assumes a minimum understanding in concepts such as virtualization and containerization and also some knowledge around Linux®, Microsoft Windows® and VMware® technologies.

Solution overview

The HPE Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure consists of a set of Ansible playbooks that run on top of a VMware virtualization platform on HPE Synergy and HPE 3PAR storage hardware. The solution allows you to configure a flexible OS environment (with both RHEL and Windows workers) providing built-in high availability (HA), container monitoring and security, and backup and restore functionality. This solution assumes that you have already set up your HPE Synergy hardware, that you have installed your VMware virtualization platform and have configured HPE 3PAR for storage.

New in this release

Version 2.0 of the solution provides support for Kubernetes via Docker EE 2.0, while monitoring of Kubernetes clusters is provided by updated versions of Splunk and Sysdig. For more details on what is new in this release, see the release notes at https://github.com/HewlettPackard/Docker-Synergy/blob/master/ops/ReleaseNotes20.md.
Solution configuration
The Ansible playbooks are available to download at https://github.com/HewlettPackard/Docker-Synergy. By default, the playbooks are configured as shown in Figure 2 to set up a 3 node environment. This is the minimal starter configuration recommended by HPE and Docker for production.

Figure 2. Three node HPE Synergy configuration

The playbooks can also be used for larger container environments, with a 3 frame, 6 node HPE Synergy system, with 2 nodes in each frame as shown in Figure 3.

Figure 3. Six node HPE Synergy configuration
Two separate configurations are available out of the box, with one restricted to a Linux-only deployment while the other supports a hybrid deployment including Windows workers as well as Linux ones. All the software is distributed over the physical nodes via VMware virtual machines (VMs), depending on the size of your environment, as follows:

**Linux-only VM configuration**
- 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
  The Docker UCP and DTR nodes are spread across 3 physical nodes, with one on each physical node. An odd number of manager nodes is recommended to avoid split-brain issues. It is possible to restrict the deployment to 1 UCP and 1 DTR, or to expand to more than 3, but the recommended minimum for an enterprise production deployment is 3 UCPs and 3 DTRs
- 3 Docker Linux worker VM nodes for container workloads - Kubernetes or Docker swarm or a mix
  The Docker worker nodes will be co-located with the UCP and DTR nodes in a 3 physical node deployment, whereas in a 6 physical node setup, the worker nodes will typically be separated onto the extra nodes. It is possible to specify that more than one worker node will be deployed per physical node but this decision will depend on the requirements of your applications
- 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
  Three load balancers are provided to increase availability of the UCP, DTR and worker nodes and these are typically distributed evenly across 3 physical nodes
- 1 Logging server VM for central logging
- 1 NFS server VM for storage of Docker DTR images

With the addition of the NFS and logging VMs, a total of 14 VMs are created for the default Linux-only deployment. In addition to these VMs, the playbooks also set up the Docker persistent storage plug-in from VMware. The vSphere Docker volume plug-in facilitates the storage of data in a shared datastore that can be accessed from any machine in the cluster.

**Hybrid VM configuration (Windows and Linux)**
The hybrid deployment will typically add 3 Windows worker nodes to the above configuration, co-located with the Linux workers.

- 3 Docker swarm Windows worker VM nodes for container workloads (optional)

**Note**
Some of the application software supported by this configuration does not currently run on Windows, for example, the Sysdig Software Agent (see the section Monitoring with Sysdig).

**High availability**
Uptime is paramount for businesses implementing Docker containers in business critical environments. The HPE Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure offers various levels of high availability (HA) to support continuous availability. The Docker EE system components run on multiple manager nodes in the cluster. The management plane continues to operate even in the event of a manager node failure. Application containers can be protected through the use of Services running on top of swarm. The swarm orchestrator works to maintain the number of containers declared as part of the service. The Ansible playbooks can be modified to fit your environment and your high availability (HA) needs.

**Load Balancers**
This Reference Configuration 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 as shown in Figure 4. 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 system administrator should consider implementing an HA load balancer architecture using the Virtual Router Redundancy Protocol (VRRP). For more information, see http://www.haproxy.com/solutions/high-availability/.

**Figure 4. Load balancer architecture**

**Sizing considerations**

A node is a machine in the cluster (virtual or physical) with Docker Engine running on it. There are two types of nodes: managers and workers. UCP will run on the manager nodes. Although DTR runs on a worker node, Docker does not recommend running other application containers on them. 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 fulfill the minimal requirements, for UCP 2.0, 8GB of RAM and 3GB of storage. For production systems, 16GB of RAM is recommended for manager nodes. More detailed requirements are in the Docker EE UCP documentation at https://docs.docker.com/ee/ucp/admin/install/system-requirements/.

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 for all nodes in the environment. If your standard machine type provides more resources than the UCP Controller nodes 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 this solution, the following tables describe sizing configurations, assuming 3 Linux workers and 3 Windows workers. The vCPU allocations are described in Table 1 while the memory allocation is described in the Memory allocation for 6 node solution section.

### Table 1. vCPU

<table>
<thead>
<tr>
<th>vCPUs</th>
<th>node01</th>
<th>node02</th>
<th>node03</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucp1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucp2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucp3</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>dtr1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr3</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>worker1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worker2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>worker3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>win-worker1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>win-worker2</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>win-worker3</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ucb_lb</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr_lb</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>worker_lb</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>nfs</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>logger</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td><strong>Total vCPU per node</strong></td>
<td><strong>16</strong></td>
<td><strong>18</strong></td>
<td><strong>18</strong></td>
</tr>
</tbody>
</table>

**Note**

In the case of one ESX host failure, 2 nodes are enough to accommodate the amount of vCPU required.

The memory allocation for this solution (3 Linux workers and 3 Windows workers), is described in the Memory allocation for 6 node solution section below.

### Table 2. Memory allocation

<table>
<thead>
<tr>
<th>RAM (GB)</th>
<th>node01</th>
<th>node02</th>
<th>node03</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucp1</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucp2</td>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>ucp3</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>dtr1</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr2</td>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>dtr3</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>worker1</td>
<td></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>worker2</td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>worker3</td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>win-worker1</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>win-worker2</td>
<td></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>win-worker3</td>
<td></td>
<td></td>
<td>64</td>
</tr>
</tbody>
</table>
Memory allocation for 6 node solution
For a 6 node solution, Table 3 outlines the memory requirements where the control plane is on 3 nodes and the worker nodes are on the other 3 nodes. In this example, it is assumed that there are 2 Linux worker nodes and 1 Windows worker node, but the actual number of worker nodes is not limited to 3 and depends entirely on the workload requirements.

Table 3. Memory allocation for 6 nodes

<table>
<thead>
<tr>
<th>RAM (GB)</th>
<th>node01</th>
<th>node02</th>
<th>node03</th>
<th>node04</th>
<th>node05</th>
<th>node06</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucp1</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucp2</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucp3</td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr1</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr2</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr3</td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worker1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>worker2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>win-worker1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>ucb_lb</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtr_lb</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>worker_lb</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nfs</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>logger</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total RAM required (per node)</td>
<td>36</td>
<td>44</td>
<td>44</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Available RAM</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

Disaster recovery
Recovery Time Objective (RTO) refers to the time that it takes to recover your data and applications while Recovery Point Objective (RPO) refers to the point in time you can recover to in the event of a disaster. In essence, RPO tells you how often you will need to make new backups.

In order to protect your installation from disasters, you need to take regular backups and transfer the backups to a safe location. This solution provides a range of convenience scripts and Ansible playbooks to help automate the backup of UCP, DTR, your swarm and your Docker volumes. See the section **Backup and restore** for best practices, procedures and utilities for implementing disaster recovery.
Security


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.

In addition, the Sysdig product also provides a strong level of container security and monitoring (see the section Monitoring with Sysdig).

Solution components

The section describes the various components that were utilized in this Reference Configuration.

Hardware

Table 4 lists the hardware components that are utilized in this Reference Configuration.

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPE Synergy 12000 Frame</td>
<td>Rack enclosure for compute, storage, and network hardware</td>
</tr>
<tr>
<td>HPE Synergy 480 Gen10 Compute Modules</td>
<td>Hosts for running ESX servers that support UCP, DTR, worker and other nodes in the solution</td>
</tr>
<tr>
<td>HPE 3PAR StoreServ 8200</td>
<td>Provides the storage for the virtual machines and the Docker backups</td>
</tr>
<tr>
<td>HPE StoreOnce</td>
<td>High performance backup system</td>
</tr>
</tbody>
</table>

HPE Synergy

HPE Synergy, the first platform built from the ground up for composable infrastructure, empowers IT to create and deliver new value instantly and continuously. This single infrastructure reduces operational complexity for traditional workloads and increases operational velocity for the new breed of applications and services. Through a single interface, HPE Synergy composes compute, storage and fabric pools into any configuration for any application. It also enables a broad range of applications from bare metal to virtual machines to containers, and operational models like hybrid cloud and DevOps. HPE Synergy enables IT to rapidly react to new business demands.

HPE Synergy Frames contain a management appliance called the HPE Synergy Composer which hosts HPE OneView. HPE Synergy Composer manages the composable infrastructure and delivers:

- Fluid pools of resources, where a single infrastructure of compute, storage and fabric boots up ready for workloads and demonstrates self-assimilating capacity.
- Software-defined intelligence, with a single interface that precisely composes logical infrastructures at near-instant speeds, and demonstrates template-driven, frictionless operations.
- Unified API access, which enables simple line-of-code programming of every infrastructure element; easily automates IT operational processes; and effortlessly automates applications through infrastructure deployment.

Server requirements

The minimum platform requirement for this configuration, shown in Figure 2, is a three node HPE Synergy 480 Gen10 deployment with 1 node in each Synergy frame and:

- 384 GB DDR4–2133 RAM
- 2 Intel® Xeon® CPU Gold 6130 2.10GHz x 16 core
- Single ESXi cluster with control plane and Docker workers spread out on all 3 nodes
The solution has also been tested on a 6 node HPE Synergy environment, with 2 nodes in each frame. In this setup, the extra 3 nodes are dedicated to Docker worker nodes. The 6 node deployment is depicted graphically in Figure 3 with the following suggested requirements for each node:

- 128 GB DDR4-2133 RAM
- 2 Intel® Xeon® CPU Gold 6130 2.10GHz x 16 core
- Single ESXi cluster with the control plane on 3 nodes and the Docker workers spread on the other 3 nodes.

**Storage requirements**

An HPE 3PAR array is required for the ESXi datastore. This solution makes use of an HPE 3PAR StoreServ 8200 populated with:

- 8x 480GB SSD for the vSphere cluster datastore
- 8x 1.8TB HDD for the backup datastore

You should create a large virtual volume on the HPE 3PAR StoreServ to host the virtual machines and another large virtual volume for Docker backups. Create datastores on your vSphere cluster using these virtual volumes. If desired, you can create separate HPE 3PAR StoreServ virtual volumes and attach them to all vSphere cluster hosts for backing up Docker persistent volumes. It is recommended that you configure the volumes that are used for virtual machine deployments on the SSD. Storage for backups can be configured on the HDDs.

Table 5 provides an overview of how the storage requirements for various components are addressed in this solution.

<table>
<thead>
<tr>
<th>Storage requirement</th>
<th>Provide by</th>
<th>Storage resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMs</td>
<td>vSphere</td>
<td>HPE 3PAR</td>
</tr>
<tr>
<td>Persistent storage for stateful containers running under Docker swarm orchestrator</td>
<td>vSphere Docker Volume plugin</td>
<td>HPE 3PAR</td>
</tr>
<tr>
<td>Persistent storage for K8S - production</td>
<td>K8s NFS provisioner configured for 3PAR</td>
<td>HPE 3PAR Virtual File Server</td>
</tr>
<tr>
<td>Storage for K8S – non-production</td>
<td>K8s NFS provisioner configured for NFS VM</td>
<td>NFS VM</td>
</tr>
<tr>
<td>Persistent storage backup</td>
<td>Clone using vSphere Docker Volume plugin</td>
<td>HPE 3PAR</td>
</tr>
<tr>
<td>UCP, DTR backup</td>
<td>backup.sh utility script or individual backup playbooks</td>
<td>Ansible VM</td>
</tr>
<tr>
<td>Backup to HPE StoreOnce</td>
<td>HPE RMC</td>
<td>HPE StoreOnce</td>
</tr>
<tr>
<td>Restore from HPE StoreOnce</td>
<td>HPE RMC</td>
<td>HPE 3PAR</td>
</tr>
</tbody>
</table>

**Software**

The software components used in this Reference Configuration are listed in Table 6 and Table 7.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>2.4.2</td>
</tr>
<tr>
<td>Docker EE</td>
<td>2.0 with Docker EE Engine 17.06 (tested with UCP 3.0.4 and DTR 2.5.3)</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>7.4.7.5</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>Server 2016</td>
</tr>
<tr>
<td>VMware</td>
<td>ESXi 6.5.0 and vCenter 6.5.0</td>
</tr>
</tbody>
</table>

Table 7. HPE Software

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPE Recovery Manager Central</td>
<td>5.0.1</td>
</tr>
</tbody>
</table>
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 Linux 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.

When deploying Windows nodes in a hybrid deployment, the Ansible playbooks make use of the Python pywinrem module which carries out actions via the Windows remote manager.

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

About Docker Enterprise Edition
Docker Enterprise Edition (EE) is the leading enterprise-ready container 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.


Application software
A number of different logging and monitoring solutions are supported by this solution:

- Splunk
- Sysdig
- Prometheus and Grafana

The application software components used in this Reference Configuration are listed in Table 8.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splunk</td>
<td>7.12</td>
</tr>
<tr>
<td>Sysdig</td>
<td>latest</td>
</tr>
<tr>
<td>Prometheus</td>
<td>V2.32</td>
</tr>
<tr>
<td>Grafana</td>
<td>5.23</td>
</tr>
</tbody>
</table>
Monitoring with Splunk and Sysdig

The solution can be configured to use either Splunk or Sysdig or to enable both simultaneously. While there is some overlap in the functionality provided by these tools, they are ultimately complimentary in what they offer. Splunk aggregates logging and tracing for a wide variety of sources and provides a clean, high-level dashboard for all your enterprise systems. Sysdig, on the other hand, has been engineered from the ground up to focus on containerized environments and includes both monitoring and security features, with built-in understanding of the different workloads running on your cloud. The load among the three hosts in a hybrid deployment will be shared as per Figure 5.

**Figure 5.** Solution architecture: Hybrid Linux and Windows workers with Splunk and Sysdig

Monitoring with Splunk

Splunk Enterprise allows you to collect and index any data from any source, and to monitor systems and infrastructure in real time to preempt issues before they happen. It allows you to analyze your data to understand trends, patterns of activity and behavior, giving you valuable intelligence across your entire organization.
This solution allows you to integrate your CaaS deployment with an existing Splunk Enterprise installation or to deploy a stand-alone Splunk Enterprise demo environment as a Docker stack in your cloud. In both instances, Universal Forwarders are used to collect data from your applications running on your Linux and Windows worker nodes in your cloud, as well as log data from the Docker platform itself and from the infrastructure VMs and servers. Figure 6 shows the Splunk architecture.

![Splunk architecture](image)

Figure 6. Splunk architecture

All the Universal Forwarders run natively on the operating system to allow greater flexibility in terms of configuration options. Each forwarder sends the data it collects to one or more indexers in the central Splunk.

**Linux worker nodes:** The Universal Forwarders on the Linux worker nodes collect log and metrics data. The log data includes:

- `/var/log/messages` from the Docker host (including the daemon engine logs)
- `/var/log/secure` from the Docker hosts
- container logs via a Splunk technical add-on

The metrics data is collected via a technical add-on and includes:

- `docker stats`
- `docker top`
- `docker events`
- `docker service stats`

**Windows worker nodes:** The Universal Forwarders running on the Windows worker nodes collect the following data:

- Windows logs
- CPU stats
- Memory stats
- Network Interface stats
- and more

For more information on configuring standalone Splunk for Linux and Windows worker nodes, see the section on Splunk prerequisites.

**UCP and ESXi:** UCP operational logs and ESXi logs are forwarded to the logger VM via TCP ports 1514 and 514 respectively. Port 1514 is assigned a special sourcetype of ucp which is then used by the Splunk Docker APP to interpret UCP logs. The Universal Forwarder runs the rsyslog daemon which will record the log messages coming from the ESX machines into the /var/log/messages file on the VM.

**Non-Docker VMs:** Other VMs, for example, NFS, use a Splunk monitor to collect and forward data from the following files:
- /var/log/messages
- /var/log/secure (Red Hat)

---

**Note**
You can configure the list of files monitored by the Universal Forwarder.

Other syslog senders can be configured to send their data to the logger VM or directly to central Splunk.

**Monitoring with Sysdig**
Sysdig's approach to Docker monitoring uses transparent instrumentation to see inside containers from the outside, with no need for agents in each container. Metrics from Docker containers, and from your applications running inside them, are aggregated in real-time across each service to provide meaningful monitoring dashboards and alerts for your application. Figure 7 provides an overview of the Sysdig architecture.

![Sysdig architecture](image)

**Figure 7.** Sysdig architecture

**Sysdig Monitor** allows you to analyze response times, application performance metrics, container and server utilization metrics, and network metrics. You can build dashboards across applications, micro-services, container and networks, and explore metadata from Docker, Kubernetes, Mesos and AWS. For more information, see the Sysdig Container Monitoring video overview and the Sysdig Monitor 101 training course.

**Sysdig Secure** provides security at the orchestrator as well as the container level. You create service-aware policies that allow you to take actions (like killing a container) or send alerts (to Slack, Splunk, etc) whenever a policy violation occurs. All commands are audited to help you identify
anomalous actions, along with taking snapshots of all activities pre-and-post a policy violation. For more information, see the Sysdig Secure video overview and the Sysdig Secure 101 training course.

The implementation in this solution uses the Software as a Service (SaaS) version of Sysdig. The playbooks deploy Sysdig Agent software on each UCP, DTR and Linux worker node, as well as the NFS, logger and load balancer VMs and captured data is relayed back to your Sysdig SaaS Cloud portal. The deployment provides access to a 90 day try-and-buy, fully featured version of the Sysdig software.

Note
The Sysdig functionality is not turned on by default in this solution - see the section on Sysdig configuration for more information on how to enable Sysdig. For more information on how to access the 90 day try-and-buy version, see the GitHub repository at https://github.com/HewlettPackard/Docker-Synergy.

Monitoring with Prometheus and Grafana
The solution can be configured to enable the use of Prometheus and Grafana for monitoring. In this setup, there is no need for native installs and all the required monitoring software runs in containers, deployed as either services or stacks. The load among the three hosts will be shared as per Figure 8.

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.
Note
Prometheus and Grafana functionality is not turned on by default in this solution - see the section on Prometheus and Grafana configuration for more information on how to enable these tools. Additionally, this functionality will not work for the Windows worker nodes in your environment at present.

Preparation of the environment
This section describes in detail how to prepare the environment that was outlined in the architecture section. The following high level steps are required:

- Verify prerequisites
- Enable vSphere High Availability (HA)
- Install vSphere Docker Volume Service driver on all ESXi hosts
- Create the Ansible node
- Create the Red Hat Linux Template and configure the repository
- Create the Windows Template (optional)
- Finalize the template

Verify prerequisites
Before you start deployment, you must assemble the information required to assign values for each and every variable used by the playbooks. The variables are fully documented in the section Configuring the solution components. A brief overview of the information required is presented in Table 9.

<table>
<thead>
<tr>
<th>Component</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Infrastructure</td>
<td>The FQDN of your vCenter server and the name of the Datacenter. You will also need administrator credentials in order to create templates and spin up virtual machines.</td>
</tr>
<tr>
<td>L3 Network requirements</td>
<td>You will need one IP address for each and every VM configured in the Ansible inventory (see the section Configuring the solution components). The recommended minimal deployment (Linux-only) configures 14 virtual machines so you would need to allocate 14 IP addresses to use this example inventory. If you have a hybrid environment with Windows workers, you will need to increase the allocation. 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.</td>
</tr>
<tr>
<td>DNS</td>
<td>You will need to know the IP addresses of your DNS server. In addition, all the VMs you configure in the inventory must have their names registered in DNS prior to deployment. In addition, you will need to know the domain name to use for configuring the virtual machines (such as example.com)</td>
</tr>
<tr>
<td>NTP Services</td>
<td>You need time services configured in your environment. The deployed solution uses certificates that are time-sensitive. You will need to specify the IP addresses of your time servers (NTP).</td>
</tr>
<tr>
<td>RHEL Subscription</td>
<td>A RHEL subscription is required to pull extra packages that are not on the DVD.</td>
</tr>
<tr>
<td>Docker Prerequisites</td>
<td>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 <a href="https://docs.docker.com/engine/installation/linux/docker-ee/rhel/">https://docs.docker.com/engine/installation/linux/docker-ee/rhel/</a> in the section entitled “Docker EE repository URL.”</td>
</tr>
<tr>
<td>Proxy</td>
<td>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.</td>
</tr>
</tbody>
</table>

Enable vSphere High Availability (HA)
You must enable vSphere High Availability (HA) to support virtual machine failover during a 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.
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. Setting this up 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 and then download and install the latest release of vDVS driver.

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

More information on how to download and install the driver can be found at [http://vmware.github.io/vsphere-storage-for-docker/documentation/install.html](http://vmware.github.io/vsphere-storage-for-docker/documentation/install.html). The version of the driver tested in this configuration is 0.21.2.

**Create the Ansible node**

The Ansible node will act as the driver to automate the provisioning of the environment and it is essential that it be properly installed.

1. Create a Virtual Machine and install your preferred OS (in this example, and for the sake of simplicity, RHEL7 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 section Configure the yum repositories. The `extras` repo can be enabled as follows:

```
# subscription-manager repos --enable=rhel-7-server-extras-rpms
```

4. Configure the EPEL repository. For more information, see [http://fedoraproject.org/wiki/EPEL](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.

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

5. Install Ansible 2.4.2 or higher. The playbooks have been tested with 2.4.2 and 2.5.0.

```
# yum install ansible
```

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

```
# yum install python-pyvmomi python-netaddr python2-jmespath python-pip gcc python-devel openssl-devel git
# pip install --upgrade pip
# pip install cryptography
# pip install pysphere
# pip install --ignore-installed "pywinrm>=0.2.2"
```

**Configure the yum repositories**

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 is on the Red Hat DVD but the second is not. There are two options, with both options requiring a Red Hat Network account. Logon in your VM template using SSH with the credentials you configured for the root account and implement one of the two options below.

**Option 1:** 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.
1. Use the `subscription-manager register` command as follows.
   
   ```bash
   # subscription-manager register --auto-attach
   ```

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

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

3. Verify that you don't have the issue described here: [https://access.redhat.com/solutions/3317671](https://access.redhat.com/solutions/3317671) by entering the following command.

   ```bash
   # yum repolist
   ```

4. If you have the issue, fix it with the following command

   ```bash
   # subscription-manager repos --disable=rhel-7-server-rt-beta-rpms
   ```

   The playbooks will later automatically enable the `extras` repository on the VMs that need it.

**Option 2:** 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 as follows.

```bash
[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
```

To see how you can create a local mirror of the Red Hat repositories and how to share them, check the Red Hat documentation at [https://access.redhat.com/solutions/23016](https://access.redhat.com/solutions/23016), [https://access.redhat.com/solutions/265523](https://access.redhat.com/solutions/265523) and at [https://access.redhat.com/solutions/7227](https://access.redhat.com/solutions/7227).

**Create the Red Hat Linux template**

To create the Red Hat Linux VM template that you will use as the base for all your nodes, you first create a Virtual Machine with the OS installed and then convert the Virtual Machine to a VM Template. The VM Template is created as lean as possible, with any additional software installs and/or system configuration performed subsequently using Ansible.

As the creation of the template is a one-off task, this procedure has not been automated. The steps required to manually create a VM template are outlined below.

Log in to vCenter and create a new Virtual Machine with the following characteristics:

- Guest OS Family: Linux, Guest OS Version: Red Hat Enterprise Linux (64-bit)
- Hard Disk size: 50GB, (Thin provisioning)
- A single network controller connected to the network or VLAN of your choice. All VMs will connect to this same network.
- Optionally you can remove the floppy drive

Install Red Hat Enterprise 7:

1. Select a language which is selected by Docker
2. For the software selection, choose **Infrastructure Server** as the base environment and add the **Guest Agents** from the lists of add-ons available for this environment. The Infrastructure Server environment is selected here versus the Minimal Install because Customization of Linux guest operating systems requires that Perl is installed in the Linux guest operating system.

3. Configure the network settings so that you can later access the VM using SSH. Specify an IP address for the network interface, a default gateway, DNS settings and possibly any HTTP/HTTPS proxies that apply in your environment.

4. Specify a password for the root account and optionally created an admin user.

5. Wait for the installation to finish and for the VM to reboot.

**Finalize the template**

Log in to the root account on the Ansible box and copy the SSH public key to the VM Template. This will allow your Ansible node to SSH to all the Virtual Machines created from the VM Template without the need for a password.

```bash
ssh-copy-id root@<IP of your VM_Template>
```

Perform the following steps on the VM Template to finalize its creation:

1. Clean up the template by running the following commands from the Virtual Machine Console:

   ```bash
   # rm /etc/ssh/ssh_host_*
   # nmcli con del ens192
   # logrotate -f /etc/logrotate.conf
   # rm /var/log/*-201*
   # history -c
   
   # shutdown -h now
   # shutdown -h now
   # Turn the VM into a template by right-clicking on your VM and selecting Template -> Convert to Template. This will create a new template visible under VM Templates in Folders, ready for future use.

**Note**

In both the Ansible node and the VM Template you might need to configure the network so one node can reach the other. Instructions for this step have been omitted since it is a basic step and could vary depending on the user’s environment.

---

**Create the Windows Template (optional)**

To create the Windows VM Template that you will use as the base for all your Windows worker nodes, you will first create a Virtual Machine with the OS installed and then convert the Virtual Machine to a VM Template. The VM Template is created as lean as possible, with any additional software installs and/or system configuration performed subsequently using Ansible.

As the creation of the template is a one-off task, this procedure has not been automated. The steps to create a VM template manually are outlined below.

Log in to vCenter and create a new Virtual Machine with the following characteristics:

- **Guest OS Family:** Windows, **Guest OS Version:** Microsoft Windows Server 2016 (64-bit)
- **Hard Disk size:** 100GB (Thin provisioning), 1 vCPU and 4 GB of RAM. Both vCPU and memory can be altered later after you deploy from this template.
- A single network controller connected to the network or VLAN of your choice. All VMs will connect to this same network.
- Change the network type to VMXNET3, and attach the Windows Server 2016 ISO image from a datastore ensuring you connect the CD/DVD drive on boot.
• Click on the VM Options tab, and in the Boot Options section, select Force BIOS setup(*) to ensure that the machine enters the BIOS setup screen on next boot of this VM. This will allow you to adjust the boot order, placing the virtual CDROM in front of your hard drive.

• Optionally you can remove the floppy drive.

Install Windows Server 2016:

1. Power on the selected VM and then Open Console. Once connected to the console, you will be placed in the BIOS setup screen.

2. Select the Boot tab, click on CD-ROM Drive and move up the CDROM drive above the hard drive. This allows your Windows Server 2016 ISO image to be loaded first on boot. F10 Save and exit is next step.

3. Enter your choices for Language, Time/Currency Format, Keyboard and then Install Now.

4. Select the OS you want to install, and then select Custom: Install Windows Only.

5. Select drive 0, the 100 GB drive you specified earlier, as the location for installing windows.

6. Add a password for the Administrator user.

7. Install VMware Tools and reboot.

8. Once the VM has re-booted, add a temporary network IP address.

9. Use sconfig utility from (MS-DOS) command line to install windows updates and enable remote desktop.

10. Perform any other customizations you require at this point.

11. Prior to converting the VM to Template, run Sysprep: C:\Windows\System32\Sysprep\Sysprep.exe

12. Ensure ‘System Out-of-Box Experience (OOBE)’ is selected.

13. Select the ‘Generalize’ option.

14. Select ‘Shutdown’ from the Shutdown Options.

15. Shutdown VM, and untick Connect CD/DVD so that the Windows Server 2016 ISO is no longer mounted. Boot the Windows VM one final time and enter regional settings applicable to your location and keyboard mapping, enter a password and then Shutdown VM.

---

**Note**
The vmware_guest module used by the playbooks will generate a new SID.

---

Turn the VM into a template by right-clicking on your VM and selecting Template -> Convert to Template. This will create a new template visible under VM Templates in Folders, ready for future use.

### Configuring the solution components

Once you have prepared your environment, you need to download the solution software and edit the configuration variables to match your setup.

### Ansible configuration

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

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

2. Change to the directory that you just cloned:

   ```shell
   # cd ~/Docker-Synergy
   ```

3. Change to the ops directory:

   ```shell
   # cd ops
   ```
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 modify a number of files including:

- `site.yml`, the main entry point for the playbooks.
- `vm_hosts`, the inventory file.

You also need to create and populate a number of files:

- `group_vars/vars`, the group variables file.
- `group_vars/vault`, containing sensitive information that needs to be protected.
- `group_vars/backups`, containing backup-related variables.

For the latter group, a set of sample files has been provided to help you get started:

- `group_vars/vars.sample`, a sample group variables file.
- `group_vars/vault.sample`, a sample vault file.
- `group_vars/backups.sample`, a sample backup configuration file.

The file `group_vars/win_worker.yml` supports advanced configuration of Windows remote management and in general should not require modification.

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

**Editing the inventory**

The inventory is the file named `vm_hosts` in the `~/Docker-Synergy/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 group names are static so they must not be changed. Other fields (hostnames, specifications, IP addresses…) are edited to match your setup. 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 Linux worker nodes.
- `[win_worker]`: A group containing all the Windows worker nodes.
- `[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 individually 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 few 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, with the exception of 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.
• [win_worker:vars]: A set of variables defined for all nodes in the [win_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 to 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 Linux workers with higher specifications by setting:

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

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

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 10 below. They are all mandatory unless otherwise specified.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_addr</td>
<td>Node</td>
<td>IP address in CIDR format to be given to a node</td>
</tr>
<tr>
<td>esxi_host</td>
<td>Node</td>
<td>ESXi host where the node will be deployed. If the cluster is configured with DRS, this option will be overridden</td>
</tr>
<tr>
<td>cpus</td>
<td>Node/Group</td>
<td>Number of CPUs to assign to a VM or a group of VMs</td>
</tr>
<tr>
<td>ram</td>
<td>Node/Group</td>
<td>Amount of RAM in MB to assign to a VM or a group of VMs</td>
</tr>
<tr>
<td>disk2_usage</td>
<td>Node/Group</td>
<td>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.</td>
</tr>
</tbody>
</table>
## VMware configuration

All VMware-related variables are mandatory and are described in Table 11.

### Table 11. VMware variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vcenter_hostname</td>
<td>group_vars/vars</td>
<td>IP or hostname of the vCenter appliance</td>
</tr>
<tr>
<td>vcenter_username</td>
<td>group_vars/vars</td>
<td>Username to log in to the vCenter appliance. It might include a domain, for example, <code>administrator@vsphere.local</code>.</td>
</tr>
<tr>
<td>vcenter_password</td>
<td>group_vault</td>
<td>The password corresponding to the vcenter_username user above.</td>
</tr>
<tr>
<td>vcenter_validate_certs</td>
<td>group_vars/vars</td>
<td>'no'</td>
</tr>
<tr>
<td>datacenter</td>
<td>group_vars/vars</td>
<td>Name of the datacenter where the environment will be provisioned</td>
</tr>
<tr>
<td>vm_username</td>
<td>group_vars/vars</td>
<td>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 root.</td>
</tr>
<tr>
<td>vm_password</td>
<td>group_vault</td>
<td>The password for the vm_username user above.</td>
</tr>
<tr>
<td>vm_template</td>
<td>group_vars/vars</td>
<td>Name of the RHEL VM Template to be use. Note that this is the name from a vCenter perspective, not the hostname.</td>
</tr>
<tr>
<td>folder_name</td>
<td>group_vars/vars</td>
<td>vCenter folder to deploy the VMs. If you do not wish to deploy in a particular folder, the value should be / . 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.</td>
</tr>
<tr>
<td>datastores</td>
<td>group_vars/vars</td>
<td>List of datastores to be used, in list format, i.e. ['Datastore1','Datastore2']. The datastores must exist before you run the playbooks. Note that each datastore should be mounted on each of the ESXi hosts.</td>
</tr>
<tr>
<td>disk2</td>
<td>group_vars/vars</td>
<td>UNIX name of the second disk for the Docker VMs. Typically /dev/sdb</td>
</tr>
<tr>
<td>disk2_part</td>
<td>group_vars/vars</td>
<td>UNIX name of the partition of the second disk for the Docker VMs. Typically /dev/sdb1</td>
</tr>
<tr>
<td>vsphere_plugin_version</td>
<td>group_vars/vars</td>
<td>Version of the vSphere plugin for Docker. The default is 0.21.2 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.</td>
</tr>
<tr>
<td>vm_portgroup</td>
<td>group_vars/vars</td>
<td>Used by the playbook create_vms.yml, this variable is used to specify the portgroup connected to the network that connects all the VMs. There is currently only one network. It is recommended that the template which is used as the base for all deployed VMs specifies a network adapter but it is not required. If a network adapter is specified, you should not attach this adapter to a standard switch if the portgroup designated by vm_portgroup is connected to a distributed vSwitch. In addition, you should make sure that the adapter specifies Connect At Power On.</td>
</tr>
</tbody>
</table>

## Networking configuration

All network-related variables are mandatory and are described in Table 12.

### Table 12. Network variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nic_name</td>
<td>group_vars/vars</td>
<td>Name of the device, for RHEL this is typically ens192 and it is recommended to leave it as is.</td>
</tr>
<tr>
<td>gateway</td>
<td>group_vars/vars</td>
<td>IP address of the gateway to be used</td>
</tr>
<tr>
<td>dns</td>
<td>group_vars/vars</td>
<td>List of DNS servers to be used, in list format, i.e. ['10.10.173.1','10.10.173.2']</td>
</tr>
<tr>
<td>domain_name</td>
<td>group_vars/vars</td>
<td>Domain name for your Virtual Machines</td>
</tr>
<tr>
<td>ntp_server</td>
<td>group_vars/vars</td>
<td>List of NTP servers to be used, in list format, i.e. ['1.2.3.4','0.us.pool.net.org']</td>
</tr>
</tbody>
</table>
**Environment configuration**

All Environment-related variables are described in Table 13 below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
</table>
| env | group_vars/vars | Dictionary containing all environment variables. It contains three entries described below. Please leave the proxy related settings empty if not required:  
  - **http_proxy**: HTTP proxy URL, such as "http://15.184.4.2:8080". This variable defines the HTTP proxy URL if your environment is behind a proxy.  
  - **https_proxy**: HTTPS proxy URL, such as "http://15.184.4.2:8080". This variable defines the HTTPS proxy URL if your environment is behind a proxy.  
  - **no_proxy**: List of hostnames or IPs that don't require proxy, such as 'localhost,127.0.0.1,.cloudra.local,10.10.174.' |

**Docker configuration**

All Docker-related variables are mandatory and are described in Table 14.

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docker_ee_url</td>
<td>group_vars/vault</td>
<td>Note: This is a private link to your Docker EE subscription. The value for docker_ee_url is the URL documented at the following address: <a href="https://docs.docker.com/engine/installation/linux/docker-ee/rhel/">https://docs.docker.com/engine/installation/linux/docker-ee/rhel/</a></td>
</tr>
<tr>
<td>docker_ee_version</td>
<td>group_vars/vars</td>
<td>If this variable is omitted, install_docker.yml will install the latest stable version of docker-ee available in the repo specified with docker_ee_url. If you want to install a specific version of Docker EE, enter the full specification of the packages, for example: docker_ee_version: 'docker-ee-17.06.2.3.el7.rhel.x86_64'</td>
</tr>
<tr>
<td>rhel_version</td>
<td>group_vars/vars</td>
<td>For the Docker installation, this sets the version of your RHEL OS, such as 7.5. The playbooks were tested with RHEL 7.4 and 7.5.</td>
</tr>
<tr>
<td>dtr_version</td>
<td>group_vars/vars</td>
<td>Version of the Docker DTR you wish to install. You can use a numeric version or latest for the most recent one. The playbooks were tested with UCP 3.0.4.</td>
</tr>
<tr>
<td>ucp_version</td>
<td>group_vars/vars</td>
<td>Version of the Docker UCP you wish to install. You can use a numeric version or latest for the most recent one. The playbooks were tested with UCP 3.0.4.</td>
</tr>
<tr>
<td>images_folder</td>
<td>group_vars/vars</td>
<td>Directory in the NFS server that will be mounted in the DTR nodes and that will host your Docker images.</td>
</tr>
<tr>
<td>license_file</td>
<td>group_vars/vars</td>
<td>Full path to your Docker EE license file on your Ansible host. The license file is available from the Docker Store.</td>
</tr>
<tr>
<td>ucp_username</td>
<td>group_vars/vars</td>
<td>Username of the administrator user for UCP and DTR, typically admin.</td>
</tr>
<tr>
<td>ucp_password</td>
<td>group_vars/vault</td>
<td>The password for the ucp_username account.</td>
</tr>
</tbody>
</table>

To see how to use customer-supplied certificates with UCP and DTR, see Appendix B.

**Orchestrator configuration**

The variable orchestrator in the [worker] group is used to specify if a worker node should be assigned to the Kubernetes orchestrator (orchestrator: 'kubernetes') or to the swarm orchestrator (orchestrator: 'swarm'). In general, you should only change the orchestrator for worker nodes.

**Note**

Docker supports a third type, mixed, that enables workloads to be scheduled by both Kubernetes and Docker swarm on the same node. Mixing orchestrator types on the same node is not recommended for production deployments because of the likelihood of resource contention. As a result, these playbooks do not support the mixed type.
The following example shows how to set Kubernetes as the default orchestrator for worker nodes, and how to override the default to use Docker swarm on one specific node instead.

```yaml
# WORKER
[wzrk]
hep-wzrk01 ip_addr='10.60.59.21/16' esxi_host='esxi-hep-1.cloudra.local'
hep-wzrk02 ip_addr='10.60.59.22/16' esxi_host='esxi-hep-2.cloudra.local'
hep-wzrk03 ip_addr='10.60.59.23/16' esxi_host='esxi-hep-3.cloudra.local' orchestrator=swarm

[wzrk:vars]
cpus='4'
ram='65536'
disk2_size='500'
orchestrator=kubernetes
```

**Note**
The playbooks do not change Docker's default orchestrator type which is `swarm`. Instead, the inventory is used to configure worker nodes for Kubernetes workloads or swarm workloads as explained above. If you want to change the default orchestrator type, use the method explained in the Docker documentation at [https://docs.docker.com/ee/ucp/admin/configure/set-orchestrator-type/#set-the-default-orchestrator-type-for-new-nodes](https://docs.docker.com/ee/ucp/admin/configure/set-orchestrator-type/#set-the-default-orchestrator-type-for-new-nodes).

It is possible to manually change the orchestrator type for a node. When you do this, existing workloads are evicted and they are not migrated automatically to the new orchestrator. If you want the workloads to be scheduled by the new orchestrator, you must migrate them manually. More information is available in the Docker documentation at [https://docs.docker.com/ee/ucp/admin/configure/set-orchestrator-type/#what-happens-when-you-change-a-nodes-orchestrator](https://docs.docker.com/ee/ucp/admin/configure/set-orchestrator-type/#what-happens-when-you-change-a-nodes-orchestrator).

**Kubernetes configuration**
The current playbooks support the deployment of UCP 3.0.* which deploys Kubernetes version 1.8.*. This version of the playbooks will not work with a version of UCP that is lower than 3. If you wish to deploy using UCP 2.*, you will need to download the previous release of the playbooks, which is available on the GitHub site.

The preceding section *Orchestrator configuration* explains how to assign a worker node to the Kubernetes orchestrator. This section covers specific Kubernetes configuration, including how to set the pod CIDR and how to configure Kubernetes Persistent Volumes.

**Pod CIDR**
The variable `k8s_pod_cidr` is specified in `group_vars/vars` and configures a custom range of IP addresses to be used by pods. The specific range that you use should be dedicated to the cluster.

The default value is `192.168.0.0/16`. To set an alternative value, use the variable as shown in the example:

```yaml
k8s_pod_cidr: 192.168.128.0/17
```

**Kubernetes Persistent Volume configuration**
Variables related to the configuration of Kubernetes Persistent Volumes are shown in Table 15.

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs_provisioner_role</td>
<td>group_vars/vars</td>
<td>Name of the role to create, for example, <code>nfs-provisioner-runner</code></td>
</tr>
<tr>
<td>nfs_provisioner_name</td>
<td>group_vars/vars</td>
<td>Name of the provisioner, for example, <code>hpe.com/nfs</code></td>
</tr>
<tr>
<td>nfs_provisioner_storage_class_name</td>
<td>group_vars/vars</td>
<td>Name of the storage class to create, for example, <code>nfs</code></td>
</tr>
<tr>
<td>nfs_provisioner_server_ip</td>
<td>group_vars/vars</td>
<td>IP address (or FQDN) of your external NFS server, for example, <code>hpe-nfs.cloudra.local</code></td>
</tr>
<tr>
<td>nfs_provisioner_server_share</td>
<td>group_vars/vars</td>
<td>Name of the NFS share where all the persistent volume data will be stored, for example, <code>/k8s</code></td>
</tr>
</tbody>
</table>
Related playbooks
The playbook playbooks/k8s-nfs-provisioner.yml is used to enable a dynamic NFS provisioner which can be used to automatically create and allocate Kubernetes persistent volumes. The backend storage is provided by an NFS backend. This playbook is run from the Ansible box after downloading a UCP client bundle for the admin account and sourcing the downloaded env.sh file. For more information on using this playbook, see the section Deploying the NFS provisioner for Kubernetes.

Windows configuration
Window-related variables are shown in Table 16.

### Table 16. Windows variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_windows</td>
<td>group_vars/vars</td>
<td>If true, the creation of Windows 2016 worker nodes will be actioned. The default value is false.</td>
</tr>
<tr>
<td>win_vm_template</td>
<td>group_vars/vars</td>
<td>Name of the Windows 2016 VM Template to use. Note that this is the name from a vCenter perspective, not the hostname.</td>
</tr>
<tr>
<td>win_username</td>
<td>group_vars/vars</td>
<td>Windows user name. The default is Administrator</td>
</tr>
<tr>
<td>win_password</td>
<td>group_vars/vault</td>
<td>The password for the Windows account</td>
</tr>
<tr>
<td>windows_vdvs_ps</td>
<td>group_vars/vars</td>
<td>Variable used to download the PowerShell script that is used to install vDVS for Windows. For example, <a href="https://raw.githubusercontent.com/vmware/vsphere-storage-for-docker/master/install-vdvs.ps1">https://raw.githubusercontent.com/vmware/vsphere-storage-for-docker/master/install-vdvs.ps1</a></td>
</tr>
<tr>
<td>windows_vdvs_path</td>
<td>group_vars/vars</td>
<td>Variable used to download vSphere Docker Volume Service software. This variable is combined with windows_vdvs_version (below) to generate a URL of the form &lt;windows_vdvs_path&gt;/&lt;windows_vdvs_version&gt;.zip to download the software. For example, to download version 0.21, set windows_vdvs_path equal to <a href="https://vmware.bintray.com/vDVS/vsphere-storage-for-docker.windows">https://vmware.bintray.com/vDVS/vsphere-storage-for-docker.windows</a> and windows_vdvs_version equal to 0.21</td>
</tr>
<tr>
<td>windows_vdvs_version</td>
<td>group_vars/vars</td>
<td>Combined with windows_vdvs_path, this variable is used to generate the URL for downloading the software</td>
</tr>
<tr>
<td>windows_vdvs_directory</td>
<td>group_vars/vars</td>
<td>Variable used to determine where vDVS software will be unzipped and installed from. The default is C:\Users\Administrator\Downloads</td>
</tr>
</tbody>
</table>

### group_vars/win_worker.yml
There is a separate file in the group_vars directory named win_worker.yml for advanced, Windows-specific configuration. These variables are used in the following playbooks:

- playbooks/create_windows_vms.yml
- playbooks/install_docker_window.yml
- playbooks/scale_workers_windows.yml

In general, it should not be necessary to modify this file, but the variables are documented in Table 17 for the sake of completeness.

### Table 17. Advanced windows variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansible_user</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to the Windows user account win_username as specified in group_vars/vars</td>
</tr>
<tr>
<td>ansible_password</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to the Windows user password win_password as specified in group_vars/vault</td>
</tr>
<tr>
<td>ansible_port</td>
<td>group_vars/win_worker.yml</td>
<td>5986</td>
</tr>
<tr>
<td>ansible_connection</td>
<td>group_vars/win_worker.yml</td>
<td>winrm</td>
</tr>
<tr>
<td>ansible_winrm_server_cert_validation</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to ignore</td>
</tr>
<tr>
<td>ansible_winrm_operation_timeout_sec</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to 250</td>
</tr>
<tr>
<td>ansible_winrm_read_timeout_sec</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to 300</td>
</tr>
<tr>
<td>windows_timezone</td>
<td>group_vars/win_worker.yml</td>
<td>Defaults to 15</td>
</tr>
</tbody>
</table>
Splunk configuration

This solution supports two types of Splunk deployment. Firstly, there is a built-in deployment useful for demos and for getting up to speed with Splunk. Alternatively, the solution can be configured to interact with a standalone, production Splunk deployment that you set up independently. In this case, you must explicitly configure the universal forwarders with external “forward servers” (Splunk indexers), whereas this happens automatically with the built-in option.

In the standalone deployment, you can enable SSL authentication between the universal forwarders and the indexers, by setting the `splunk_ssl` variable to `yes` in the file `group_vars/vars`. The built-in deployment does not support SSL and so, in this instance, the value of the `splunk_ssl` variable is ignored. For more information on enabling SSL, see Appendix C.

After the installation is complete, the Splunk UI can be reached at `http://<fqdn>:8000`, where `<fqdn>` is the FQDN of one of your Linux Docker nodes. Mesh routing does not currently work on Windows so you must use a Linux node to access the UI.

Splunk prerequisites

You should select the Splunk deployment type that you require by setting the variable `monitoring_stack` in the `group_vars/vars` file to either `splunk` to use a standalone Splunk deployment, or `splunk_demo` for the built-in version. If you omit this variable, or if it has an invalid value, no Splunk deployment will be configured.

For both types of deployment, you need to download the Splunk universal forwarder images/packages from https://www.splunk.com/en_us/download/universal-forwarder.html. Packages are available for 64-bit Linux and 64-bit Windows 8.1/Windows 10. Download the RPM package for Linux 64-bit (2.6+ kernel Linux distributions) to `/files/splunk/linux`. If you are deploying Windows nodes, download the MSI package for Windows 64 bit to `/files/splunk/windows`. For a dual Linux/Windows deployment, the images and packages must have same name and version, along with the appropriate extensions, for example:

- files/splunk/windows/splunkforwarder-7.1.2.msi
- files/splunk/linux/splunkforwarder-7.1.2.rpm

You need to set the variable `splunk_architecture_universal_forwarder_package` to the name you selected for the package(s), not including the file extension. Depending on the Splunk deployment you have chosen, edit the file `templates/monitoring/splunk/vars.yml` or the file `templates/monitoring/splunk_demo/vars.yml` and set the variable, for example:

```
splunk_architecture_universal_forwarder_package: 'splunkforwarder-7.1.2'
```

As of Splunk version 7.1, the Splunk universal forwarder must be deployed with a password. This password is specified using the variable `splunk.uf.password` which is configured in `group_vars/vault`.

If you are using a standalone Splunk deployment, you must specify the list of indexers using the variable `splunk_architecture_forward_servers` in `group_vars/vars`, for example:

```
splunk_architecture_forward_servers:
  - splunk-indexer1.cloudra.local:9997
  - splunk-indexer2.cloudra.local:9997
```

By default, the indexers are configured in a single load balancing group. This can be changed by editing the file `outputs.conf.j2` in the folder `template/monitoring/splunk/`. For more information on forwarding using Universal Forwarder, see the Splunk documentation at http://docs.splunk.com/Documentation/Forwarder/7.0.2/Forwarder/Configureforwardingwithoutoutputs.conf.

On your standalone Splunk installation, you need to install the following add-ons and apps.

To monitor **Linux worker nodes**, the **Docker app** should be installed on central Splunk. More info on this Docker app can be found at https://github.com/splunk/docker-itmonitoring and at https://hub.docker.com/r/splunk/universalforwarder/.

To monitor the **Windows worker nodes**, install the **Splunk App for Windows Infrastructure** on central Splunk and its dependencies:

- Splunk App for Windows Infrastructure. The Splunk App for Windows Infrastructure is not compatible with The Splunk Add-on for Windows 5.0 at this time. See https://splunkbase.splunk.com/app/1680/
• Splunk Add-on for Microsoft Windows version 4.8.4 - see https://splunkbase.splunk.com/app/742/

• Splunk Add-On for Microsoft Active Directory version 1.0.0 - see https://splunkbase.splunk.com/app/3207/

• Splunk Add-on for Microsoft Windows DNS version 1.0.1 (if this is not installed on central Splunk, you will see yellow icons on some dashboards with the message eventtype wineventlog-dns does not exist or is disabled) - see https://splunkbase.splunk.com/app/3208/

• Splunk Supporting Add-on for Active Directory version 2.1.7 (if this is not installed on central Splunk, you will see yellow icons on some dashboards with the message eventtype wineventlog-ds does not exist or is disabled) - see https://splunkbase.splunk.com/app/1151/

If you want to use your own certificates in your standalone Splunk deployment to secure the communications between the indexers and the universal forwarders, see Appendix D.

You can specify advanced Splunk configuration in the following files:

• files/splunk/linux/SPLUNK_HOME
• files/splunk/linux/DOCKER_TAS
• files/splunk/windows/SPLUNK_HOME

These files will be copied as-is to the systems running the universal forwarder.

**Configuring syslog in UCP**

In order to see some data in the UCP operational dashboard, you need to have UCP send its logs to the VM configured in the [logger] group. For example, for the following `vm_host` file:

```
[logger]
    hpe-logger ip_addr='10.60.59.24/16' esxi_host='esxi-hpe-2.cloudra.local'
```

This will configure UCP to send its logs to `hpe-logger.cloudra.local:1514`. You need to select the TCP protocol as shown in Figure 9.

![Figure 9. Configure Remote Syslog Server in UCP](image-url)
Configuring syslog in ESX

This configuration must be done manually for each ESX server. The syslog server should be the server configured in the [logger] group in your vm_hosts inventory. The protocol should be tcp and the port 514 as shown in Figure 10.

Figure 10. Configure Syslog on ESXi Hosts

For more information, see the VMware documentation at https://docs.vmware.com/en/VMware-vSphere/6.5/com.vmware.vsphere.security.doc/GUID-9F67DB52-F469-451F-B6C8-DAE8D95976E7.html

Related playbooks
- playbooks/monitoring.yml installs and configures the Splunk Universal Forwarders
- playbooks/splunk_demo.yml installs a demo of Splunk Enterprise in the cluster (if the splunk_demo deployment option is selected)

Limitations
- The Dockerized Splunk App has a number of open issues
  - https://github.com/splunk/docker-itmonitoring/issues/19
  - https://github.com/splunk/docker-itmonitoring/issues/20
- The Docker events tab is not working

Sysdig configuration

Separate playbooks are used to install Sysdig for Docker swarm and Sysdig for Kubernetes.

Sysdig configuration for Docker swarm

The playbook playbooks/install_sysdig.yml is used to automate the configuration of the SaaS setup for Docker swarm. By default, this playbook is commented out in site.yml and must be explicitly enabled. The variables used to configure Sysdig for Docker swarm are detailed in Table 18.

Table 18. Sysdig variables for Docker swarm

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysdig_access_key</td>
<td>group_vars/vault</td>
<td>After the activation of your account on the Sysdig portal, you will be provided with your access key. This is used by the playbooks to install the agent on each UCP, DTR and Linux worker node, as well as the NFS, logger and load balancer VMs.</td>
</tr>
<tr>
<td>sysdig_tags</td>
<td>group_vars/vars</td>
<td>Tagging your hosts is highly recommended. Tags allow you to sort the nodes of your infrastructure into custom groups in Sysdig Monitor. Specify location, role, and owner in the format: 'location:City,role:Enterprise CaaS on Synergy,owner:Customer Name'</td>
</tr>
</tbody>
</table>
Sysdig configuration for Kubernetes
The playbook playbooks/k8s-install-sysdig.yml is used to automate the configuration of the SaaS setup for Kubernetes. By default, this playbook is commented out in site.yml and must be explicitly enabled. The variables used to configure Sysdig for Kubernetes are detailed in Table 19.

Table 19. Sysdig variables for Kubernetes

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysdig_access_key</td>
<td>group_vars/vault</td>
<td>After the activation of your account on the Sysdig portal, you will be provided with your access key. This is used by the playbooks to install the agent on each UCP, DTR and Linux Kubernetes worker nodes.</td>
</tr>
<tr>
<td>sysdig_restricted_control_role</td>
<td>group_vars/vars</td>
<td>The Sysdig service account requires the ‘Restricted Control’ grant. So this value will typically be set as follows: sysdig_restricted_control_role: 'Restricted Control'</td>
</tr>
<tr>
<td>k8s_cluster</td>
<td>group_vars/vars</td>
<td>This should match the cluster name displayed when you source the environment setup script, for example, # source env.sh Cluster &quot;ucp_gab-ucp.cloudra.local:6443_admin&quot; set. Use &quot;ucp_gab-ucp.cloudra.local:6443_admin&quot; set. For more information, see the section on installing the UCP client bundle in the section Deploying Sysdig monitoring on Kubernetes.</td>
</tr>
</tbody>
</table>

Prometheus and Grafana configuration
All monitoring-related variables for Prometheus and Grafana are described in Table 20. The variables determine the versions of various software tools that are used and it is recommended that the values given below are used.

Table 20. Monitoring variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cadvisor_version</td>
<td>v0.28.3</td>
</tr>
<tr>
<td>node_exporter_version</td>
<td>v1.15.0</td>
</tr>
<tr>
<td>prometheus_version</td>
<td>V2.3.2</td>
</tr>
<tr>
<td>grafana_version</td>
<td>5.2.3</td>
</tr>
<tr>
<td>prom_persistent_vol_name</td>
<td>The name of the volume which will be used to store the monitoring data. The volume is created using the vSphere Docker Volume plugin.</td>
</tr>
<tr>
<td>prom_persistent_vol_size</td>
<td>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.</td>
</tr>
</tbody>
</table>

Protecting sensitive information
A vault file is used to protect any sensitive variables that should not appear in clear text in your {group_vars/vars} file. The vault file will be encrypted and will require a password to be entered before it can be read or updated.

A sample vault file is provided 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 called {group_vars/vault} and add entries similar to:

```bash
---
docker_ee_url: 'your_url_here'
vcenter_password: 'xxxxx'
vm_password: 'xxxx'
ucp_password: 'zzzz'
win_password: 'yourpass'
sysdig_access_key: 'enter_sysdig_access_key'
rhn_orgid: "YourOrgId"
rhn_key: "YourActivationKey"
splunk_uf_password: 'YourPa$$word12'
```
rhn_orgid and rhn_key are the credentials needed to subscribe the virtual machines with Red Hat Customer Portal. For more information 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
```

In order 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 avoid illegitimate 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 using a Git repository to manage your playbooks.

**Inventory group variables**

Additional configuration files for each group in the inventory are available, including `group_vars/vms.yml`, `group_vars/ucp.yml`, `group_vars/dtr.yml`, `group_vars/worker.yml` and `group_vars/nfs.yml`.

These group files facilitate more sophisticated settings, such as additional drives and additional network interfaces. For example, here is the `group_vars/nfs.yml` file.

```yaml
networks:
  - name: '{{ vm_portgroup }}'
    ip: "{{ ip_addr | ipaddr('address') }}"
    netmask: "{{ ip_addr | ipaddr('netmask') }}"
    gateway: "{{ gateway }}"

disks_specs:
  - size_gb: '{{ disk1_size }}'
    type: thin
    datastore: "{{ datastores | random }}"
  - size_gb: '{{ disk2_size }}'
    type: thin
    datastore: "{{ datastores | random }}"
  - size_gb: 10
    type: thin
    datastore: "{{ datastores | random }}"
```

In this example, the size of the first two drives is specified using the values of the variables disk1_size and disk2_size that are declared in the `group_vars/vars` file. This maintains compatibility with `vm_hosts` inventories from the previous release of the playbooks. However, it is possible to provide explicit values, depending on your requirements, for the individual UCP, DTR, worker or NFS VMs. For example, you may want to increase the size of the second drive for the NFS VM as this is used to store the DTR images, so the default value of 500GB may not be sufficient to meet your needs.

In this release, support has been added for configuring a third drive that can be used to hold Kubernetes persistent volume data. The default size (10GB) is set low as the use of the NFS VM for storing persistent volume data is only considered suitable for demo purposes and should not be used in a production environment.

In the following example, the `group_vars/nfs.yml` has been modified to configure the NFS VM with a 50GB boot disk, a 500GB drive for DTR images and a 800GB drive for Kubernetes persistent volumes data.

```yaml
networks:
  - name: '{{ vm_portgroup }}'
    ip: "{{ ip_addr | ipaddr('address') }}"
```
netmask: "{{ ip_addr | ipaddr('netmask') }}"
gateway: "{{ gateway }}"

disks_specs:
- size_gb:  50
  type: thin
  datastore: "{{ datastores | random }}"
- size_gb:  500
  type: thin
  datastore: "{{ datastores | random }}"
- size_gb:  800
  type: thin
  datastore: "{{ datastores | random }}"

Note
The number of drives and the purpose of each drive is determined by the role of the VM and the specific playbooks that use the information. The first disk is always used as the boot disk, irrespective of VM role, while the purpose of the second or third disk is specific to the role.

Overview of the playbooks
The Ansible playbooks are available to download at https://github.com/HewlettPackard/Docker-Synergy. Once you have cloned the repository, change directory to /root/Docker-Synergy/ops.

There are two main entry points for the playbooks:
- site.yml deploys a Linux-only environment
- hybrid.yml deploys a mixed environment with Linux and Windows worker nodes

site.yml and related playbooks
The playbook ./site.yml is the day 0 playbook you use to deploy the solution and it invokes the following playbooks:
- playbooks/create_vms.yml will create all the necessary virtual machines for the environment from the VM Template defined in the vm_template variable. All Linux VMs are now created in one go, regardless of the number of drives they have. This playbook also has the potential to configure additional network adapters.
- playbooks/config_networking.yml will configure the network settings in all the virtual machines.
- playbooks/resize_syspart.yml resizes the logical volume that holds the / partition of the Linux VMs to use all the space available on the drive. In this release, there is no equivalent for the Windows machines.
- 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 the Red Hat Linux template, you can keep this playbook commented out.
- playbooks/install_haproxy.yml installs and configures the HAProxy package in the load balancer nodes. HAProxy is the tool chosen to implement load balancing for UCP nodes, DTR nodes and worker nodes.
- playbooks/config_ntp.yml configures the chrony client package in all virtual machines in order to have a synchronized clock across the environment. It will use the list of servers specified in the ntp_servers variable in the file group_vars/vars.
- playbooks/install_docker.yml installs Docker along with all of its dependencies.
- 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.
• `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.

• `playbooks/docker_post_config.yml` performs a variety of tasks to complete the installation of the Docker environment, including configuration of the HTTP/HTTPS proxies, if any, and installation of the VMware vSphere Storage for Docker volume plugin.

• `playbooks/install_nfs_server.yml` installs and configures an NFS server on the NFS node. This playbook has been updated to configure a third drive which is used to hold the data of the persistent volumes created with the NFS provisioner. This default size for this drive is purposefully kept small because using the NFS VM to store persistent volumes is not recommended for production use. However, this can be useful for demo purposes.

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

• `playbooks/create_main_ucc.yml` installs and configures the first Docker UCP instance on the target node defined by the group `ucp_main` in the `vm_hosts` inventory.

• `playbooks/scale_ucc.yml` installs and configures additional instances of UCP on the target node defined by the group `ucp` in the `vm_hosts` inventory, except for the node defined in the group `ucp_main`.

• `playbooks/create_main_dtr.yml` installs and configures the first Docker DTR instance on the target nodes defined by the group `dtr_main` in the `vm_hosts` inventory.

• `playbooks/scale_workers.yml` installs and configures additional Linux workers on the target nodes defined by the group `worker` in the `vm_hosts` inventory.

• `playbooks/install_logspout.yml` installs and configures `Logspout` on all Docker nodes. Logspout is responsible for sending logs produced by containers running on the Docker nodes to the central logger VM. By default, this playbook is commented out in `site.yml`.

• `playbooks/config_monitoring.yml` configures a monitoring system for the Docker environment based on Grafana, Prometheus, cAdvisor and node-exporter Docker containers. By default, this playbook is commented out in `site.yml` so if you want to use the solution to automatically deploy a Prometheus/Grafana monitoring system, you must explicitly uncomment both this and the `playbooks/install_logspout.yml` playbook.

• `playbooks/splunk_demo.yml` installs a demo of Splunk Enterprise in the cluster (if the `splunk_demo` deployment option is selected)

• `playbooks/splunk_uf.yml` installs and configures the Splunk Universal Forwarder on each Linux machine in the inventory.

• `playbooks/config_scheduler.yml` configures the scheduler to prevent regular users (i.e. non-admin users) scheduling containers on the Docker nodes running instances of UCP and DTR.

• `playbooks/scale_dtr.yml` installs and configures additional instances (or replicas) of DTR on the target nodes defined by the group `dtr` in the `vm_hosts` inventory, with the exception of the node defined in the group `dtr_main`.

• `playbooks/reconfigure_dtr.yml` is used to reconfigure DTR with the FQDN of the UCP Load Balancer and also enables image scanning.

• `playbooks/install_sysdig.yml` is used to configure Sysdig for Docker swarm. It opens the required port in the firewall, and installs the latest version of the Sysdig agent image on the nodes. By default, this playbook is commented out in `site.yml` so if you want to use the solution to automatically configure Sysdig for Docker swarm, you must uncomment this line.

Windows playbooks

`hybrid.yml` invokes the following Windows-specific playbooks, together with the preceding ones for Linux.

• `playbooks/create_windows_vms.yml` will create all the necessary Windows 2016 VMs for the environment based on the Windows VM Template defined in the `win_vm_template` variable.

• `playbooks/install_docker_windows.yml` installs Docker along with all its dependencies on your Windows VMs.

• `playbooks/scale_workers_win.yml` installs and configures additional Windows workers on the target nodes defined by the group `win_worker` in the `vm_hosts` inventory.

• `playbooks/splunk_uf_win.yml` installs and configures the Splunk Universal Forwarder on each Windows machine in the inventory.
Backup and restore playbooks
Best practices and procedures are described in the section Backup and restore. The following playbooks are used to perform backups:

- `playbooks/backup_swarm.yml` is used to back up the swarm data
- `playbooks/backup_ucp.yml` is used to back up UCP
- `playbooks/backup_dtr_meta.yml` is used to back up DTR metadata
- `playbooks/backup_dtr_images.yml` is used to back up DTR images

The following playbooks are used to restore the system:

- `playbooks/restore_dtr_images.yml` is used to restore DTR images
- `playbooks/restore_dtr_metadata.yml` is used to restore DTR metadata
- `playbooks/restore_ucp.yml` is used to restore UCP

Convenience playbooks
- `playbooks/clean_all.yml` powers off and deletes all VMs in your inventory.
- `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.

Convenience scripts
- `backup.sh` can be used to take a backup of the swarm, UCP, DTR metadata and the DTR images in one go.
- `restore_dtr.sh` can be used to restore DTR metadata and DTR images.
- `scale_worker.sh` can be used to scale the worker nodes.

Running the playbooks
At this point, the system is ready to be deployed. Make sure you are logged on as `root` in your ansible box and that your current directory is `/root/Docker-Synergy/ops`

Note
As well as configuring your `vars` and `vault` files, you must also provide a `backups` configuration file in the `group_vars` folder when running `site.yml` or `hybrid.yml`. An example file is provided in the repository named `backups.sample`. Rename it to `backups` before running the playbooks. Details on how to configure this file are available in the section Backup and restore.

Linux-only deployment
To start a Linux-only deployment, use the following command:

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

The playbooks should run for 35-40 minutes depending on your server specifications and the size of your environment.

Hybrid Windows and Linux deployment
To start a hybrid, Windows and Linux deployment, use the following command:

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

The playbooks should run for 70-80 minutes depending on your server specifications and the size of your environment. The increase in running time is primarily due to the need to update Windows after creating the VMs.
Post deployment

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

The Ansible log is stored in the ops folder of the repo (/root/Docker-Synergy/ops). If the deployment fails, you may find useful hints in this log. To see how to check if your certs have been deployed correctly, see Appendix D. If you have installed Grafana, you will be asked to reset the default admin password the first time you access the user interface.

Installing kubectl

You can find the version number for the current stable version of kubectl at https://kubernetes.io/docs/tasks/tools/install-kubectl/. At the time of writing, the stable version is 1.11.2.

However, this version of kubectl is not compatible with UCP 3.0.*. Instead, you need to install an earlier version, for example, v1.10.4. To download this specific version, follow the instructions below.

# version=v1.10.4
# wget -O kubectl https://storage.googleapis.com/kubernetes-release/release/${version}/bin/linux/amd64/kubectl
# chmod +x ./kubectl
# sudo mv ./kubectl /usr/local/bin/kubectl

# kubectl version

Client Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.4", GitCommit:"5ca598b4ba5abb9bb773071ce452e33fb66339d", GitTreeState:"clean", BuildDate:"2018-06-08T08:13:03Z", GoVersion:"go1.9.3", Compiler:"gc", Platform:"linux/amd64"}


More details on installing kubectl are available at https://kubernetes.io/docs/tasks/tools/install-kubectl/.

Kubernetes guestbook example with Redis

Many sample Kubernetes applications are available at https://kubernetes.io/docs/tutorials/. This section details how to deploy the stateless guestbook application with Redis as documented at https://kubernetes.io/docs/tutorials/stateless-application/guestbook/. When deploying applications, you must be aware that Kubernetes version 1.8 shipped with Docker EE 2.0 and, as a result, you may have to make changes in some places to the configuration files, particularly in relation to the apiVersion field.

The manifest file, included below, specifies a deployment controller that runs a single replica Redis master pod.

# cat redis-master-deployment.yaml

apiVersion: apps/v1beta2  # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
cole: master
tier: backend
replicas: 1
template:
  metadata:
    labels:
app: redis
tier: backend

spec:
  containers:
    - name: master
      image: k8s.gcr.io/redis:e2e  # or just image: redis
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
        - containerPort: 6379

Apply the Redis master deployment from the `redis-master-deployment.yaml` file:

```bash
# kubectl apply -f redis-master-deployment.yaml
```

Query the list of Pods to verify that the Redis master pod is running.

```bash
# kubectl get pods
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>redis-master-5f8d48b75f-8zt4</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>33s</td>
</tr>
</tbody>
</table>

Use the `kubectl logs` command to view the logs from the Redis master pod:

```bash
# kubectl logs -f redis-master-5f8d48b75f-8zt4
```

[1] 24 Aug 15:24:06.575 # Server started, Redis version 2.8.19
[1] 24 Aug 15:24:06.576 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
[1] 24 Aug 15:24:06.576 * The server is now ready to accept connections on port 6379
The guestbook application needs to communicate with the Redis master to write its data. You need to apply a service to proxy the traffic to the Redis master pod. A service defines a policy to access the pods.

```yaml
# cat redis-master-service.yaml
apiVersion: v1
class: Service
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: master
    tier: backend
```

Apply the Redis master service from the `redis-master-service.yaml` file:

```
# kubectl apply -f redis-master-service.yaml
service "redis-master" created
```

Query the list of services to verify that the Redis master service is running.

```
# kubectl get service
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP    2d
redis-master   ClusterIP   10.96.174.129   <none>        6379/TCP   1m
```
Although the Redis master is a single pod, you can make it highly available to meet traffic demands by adding replica Redis slaves.

```yaml
# cat redis-slave-deployment.yaml
apiVersion: apps/v1beta2 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-slave
labels:
  app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: slave
      tier: backend
  replicas: 2
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
        - name: slave
          image: gcr.io/google_samples/gb-redisslave:v1
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
# Using 'GET_HOSTS_FROM=dns' requires your cluster to
# provide a dns service. As of Kubernetes 1.3, DNS is a built-in
# service launched automatically. However, if the cluster you are using
# does not have a built-in DNS service, you can instead
# access an environment variable to find the master
# service's host. To do so, comment out the 'value: dns' line above, and
# uncomment the line below:
# value: env
          ports:
            - containerPort: 6379
Create the Redis slaves from the redis-slave-deployment.yaml file.

# kubectl apply -f redis-slave-deployment.yaml
deployment.apps "redis-slave" created

Query the list of Pods to verify that the Redis slave pods are running.

# kubectl get pods
NAME                             READY     STATUS    RESTARTS   AGE
redis-master-5f8d48b75f-8ztt4    1/1        Running   0          39m
The guestbook application needs to communicate to Redis slaves to read data. To make the Redis slaves discoverable, you need to set up a service that provides transparent load balancing to the set of pods.

```
# cat redis-slave-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
labels:
  app: redis
  role: slave
  tier: backend
spec:
  ports:
  - port: 6379
  selector:
    app: redis
    role: slave
    tier: backend
```

Deploy the Redis slave service from the `redis-slave-service.yaml` file

```
# kubectl apply -f redis-slave-service.yaml
service "redis-slave" created
```

Query the list of services to verify that the Redis slave service is running.

```
# kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP    2d
redis-master   ClusterIP   10.96.174.129   <none>        6379/TCP   30m
redis-slave    ClusterIP   10.96.82.175    <none>        6379/TCP   3m
```

The guestbook application has a web frontend written in PHP serving the HTTP requests. It is configured to connect to the `redis-master` service for write requests and the `redis-slave` service for read requests.

```
# cat frontend-deployment.yaml
apiVersion: apps/v1beta2 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: frontend
labels:
  app: guestbook
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
```

```yaml
tier: frontend
spec:
  containers:
    - name: php-redis
      image: gcr.io/google-samples/gb-frontend:v4
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      env:
        - name: GET_HOSTS_FROM
          value: dns
          # Using 'GET_HOSTS_FROM=dns' requires your cluster to
          # provide a dns service. As of Kubernetes 1.3, DNS is a built-in
          # service launched automatically. However, if the cluster you are using
          # does not have a built-in DNS service, you can instead
          # access an environment variable to find the master
          # service's host. To do so, comment out the 'value: dns' line above, and
          # uncomment the line below:
          # value: env
      ports:
        - containerPort: 80

Create the frontend deployment using the `frontend-deployment.yaml` file.

# kubectl apply -f frontend-deployment.yaml
deployment.apps "frontend" created

Query the list of pods to verify that the three frontend replicas are running.

# kubectl get pods -l app=guestbook -l tier=frontend
NAME                        READY     STATUS    RESTARTS   AGE
frontend-574b66f649-2qh6h   1/1       Running   0          7m
frontend-574b66f649-782mr   1/1       Running   0          7m
frontend-574b66f649-mnbc9   1/1       Running   0          7m

If you want guests to be able to access your guestbook, you must configure the frontend service to be externally visible, so a client can request the service from outside the container cluster.

# cat frontend-service.yaml
apiVersion: v1
category: Service
metadata:
  name: frontend
  labels:
    app: guestbook
tier: frontend
spec:
  # comment or delete the following line if you want to use a LoadBalancer
type: NodePort
  # if your cluster supports it, uncomment the following to automatically create
  # an external load-balanced IP for the frontend service.
  # type: LoadBalancer
  ports:
    - port: 80
  selector:
```

app: guestbook
tier: frontend

Deploy the frontend service using the `frontend-service.yaml` file

```
# kubectl apply -f frontend-service.yaml
service "frontend" created
```

Query the list of services to verify that the frontend service is running.

```
# kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
frontend       NodePort    10.96.14.76     <none>        80:35440/TCP   15s
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        2d
redis-master   ClusterIP   10.96.174.129   <none>        6379/TCP       54m
redis-slave    ClusterIP   10.96.82.175    <none>        6379/TCP       27m
```

Access the UI, using the identified port, at `http://gab-ucp03.cloudra.local:35440` as shown in Figure 11.

![Guestbook UI](gab-ucp03.cloudra.local:35440)

**Figure 11.** Guestbook UI

### Redeploying Splunk demo

The Splunk demo deployment, whilst fully featured, is severely restricted in the amount of data it can process. Once this limit has been reached, often after running for just one or two days, it is necessary to redeploy the application if you want to continue experimenting with the demo.

Before you redeploy, it is necessary to remove the corresponding Docker stack and delete the associated volumes.

```
# ssh gab-ucp02
[root@gab-ucp02 ~]# docker stack rm splunk_demo
Removing service splunk_demo_splunkenterprise
Removing network splunk_demo_default
[root@gab-ucp02 ~]# docker volume ls | grep splunk
vsphere:latest      splunk_demo_vsplunk-opt-splunk-etc@Docker_GAB
vsphere:latest      splunk_demo_vsplunk-opt-splunk-var@Docker_GAB
[root@gab-ucp02 ~]# docker volume rm splunk_demo_vsplunk-opt-splunk-etc@Docker_GAB
splunk_demo_vsplunk-opt-splunk-etc@Docker_GAB
[root@gab-ucp02 ~]# docker volume rm splunk_demo_vsplunk-opt-splunk-var@Docker_GAB
splunk_demo_vsplunk-opt-splunk-var@Docker_GAB
```
Then re-run the playbook on your ansible node.

ansible-playbook -i vm_hosts playbooks/splunk_demo.yml --vault-password-file .vault_pass

**Deploying the NFS provisioner for Kubernetes**

NFS can be provisioned using either the NFS VM for proof of concept or demo systems, and HPE 3PAR for production environments.

**Prerequisites**

- Configure the variables described in the section Kubernetes Persistent Volume configuration
- Install the kubectl binary on your Ansible box as documented at https://docs.docker.com/ee/ucp/user-access/kubectl/
- Install the UCP Client bundle for the admin user as described at https://docs.docker.com/ee/ucp/user-access/cli/#download-client-certificates-by-using-the-rest-api
- Confirm that you can connect to the cluster by running a test command, for example, kubectl get nodes

**Running the playbook**

Once the prerequisites are satisfied, run the appropriate playbook on your Ansible node.

# cd Docker-Synergy/ops
# ansible-playbook -i vm_hosts playbooks/k8s-nfs-provisioner.yml --vault-password-file .vault_pass

**Using NFS VM for post-deployment verification**

In this example, it is assumed that the relevant variables are configured as shown in Table 21.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs_provisioner_role</td>
<td>nfs-provisioner-runner</td>
</tr>
<tr>
<td>nfs_provisioner_name</td>
<td>hpe.com/nfs</td>
</tr>
<tr>
<td>nfs_provisioner_storage_class_name</td>
<td>nfs</td>
</tr>
<tr>
<td>nfs_provisioner_server_ip</td>
<td>hpe-nfs.cloudra.local</td>
</tr>
<tr>
<td>nfs_provisioner_server_share</td>
<td>/k8s</td>
</tr>
</tbody>
</table>

In this instance, the server IP is set to the NFS VM that has been deployed.

Ensure that your environment satisfies the prerequisites as described in Deploying the NFS provisioner for Kubernetes, and then run the playbook:

# cd Docker-Synergy/ops
# ansible-playbook -i vm_hosts playbooks/k8s-nfs-provisioner.yml --vault-password-file .vault_pass

Running the command kubectl get sc will show the storage class named nfs:

# kubectl get sc
NAME   PROVISIONER    AGE
nfs    hpe.com/nfs   5m

Create a temporary file /tmp/pvc.yml for a persistent volume claim (PVC) named dynnfs-testpvc with a storage class of nfs.

# cat /tmp/pvc.yml <<EOF
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: dynnfs-testpvc
annotations:
  volume.beta.kubernetes.io/storage-class: "nfs"
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
    storage: 100Mi
EOF

Create the PVC resource by running `kubectl apply` on this file.

```
# kubectl apply -f /tmp/pvc.yml
persistentvolumeclaim "dynnfs-testpvc" created
```

Verify that the corresponding persistent volume (PV) was created at the same time.

```
# kubectl get pv
NAME                          CAPACITY  STATUS    CLAIM                    STORAGECLASS    AGE
pvc-e685a9d2-8a6f-11e8-...    100Mi     Bound     default/dynnfs-testpvc   nfs             4s
```

Define a pod that will mount the persistent volume by using the persistent volume claim. The persistent volume is mounted under `/tmp/foo`.

```
# cat /tmp/pod.yml <<EOF
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: dynnfs-testpod
spec:
  selector:
    matchLabels:
      app: dynnfs-testpod
  replicas: 1
  template:
    metadata:
      labels:
        app: dynnfs-testpod
    spec:
      volumes:
      - name: pod-data
        persistentVolumeClaim:
          claimName: dynnfs-testpvc
      containers:
      - name: dynnfs-testpod
        command:
        - sh
        - -c
        - while true; do sleep 1; done
        image: radial/busyboxplus:curl
        volumeMounts:
        - mountPath: /tmp/foo
          name: pod-data
EOF
```
Create the pod resource by running `kubectl apply` on the file.

```bash
# kubectl apply -f /tmp/pod.yml
deployment.apps "dynnfs-testpod" created
```

Retrieve the pod ID and then execute a command in the pod to create a test file on the persistent volume. The file is named `/tmp/foo/bar.txt` and contains the string `hello`.

```bash
# pod=$(kubectl get pod | awk '/dynnfs-testpod-/ {print $1}')
# kubectl exec -it $pod -- sh -c "echo hello > /tmp/foo/bar.txt"
```

In this example, where the NFS VM is being used as the storage back-end, you can examine the content of the folder containing the persistent volumes. Given the values specified above, where the NFS VM is named `hpe-nfs` and the `nfs_provisioner_server_share` is `k8s`, you can connect to the VM and explore the folder as follows.

```bash
# ssh hpe-nfs ls -R /k8s
/k8s:
default-dynnfs-testpvc-pvc-e685a9d2-8a6f-11e8-9025-0242ac110010
/k8s/default-dynnfs-testpvc-pvc-e685a9d2-8a6f-11e8-9025-0242ac110010:
  bar.txt
```

Examine the contents of the file to ensure that the string `hello` has been persisted in the file `bar.txt`.

```bash
# ssh hpe-nfs cat /k8s/default-dynnfs-testpvc-pvc-e685a9d2-8a6f-11e8-9025-0242ac110010/bar.txt
hello
```

**Using HPE 3PAR for post-deployment verification**

For a production environment, using the NFS VM is not appropriate so this solution also shows how to set up HPE 3PAR as persistent storage for Kubernetes.

**Setting up HPE 3PAR**

The following section outlines the steps you need to follow in order to configure a Virtual File Server and a share for use by the Kubernetes NFS provisioner.

**Step 1:** Log in to the HPE 3PAR StoreServ Management console and create a virtual file server (VFS):

1. In the General section, as shown in Figure 12, specify a name, in this instance `gab_vfs3par`.

![Create Virtual File Server - General](image)
2. In the Storage Allocation Settings section shown in Figure 13, set the Provisioning to Thin Provisioned, select an appropriate CPG, in this instance FC_r1, and set the size, for example, 1 terabyte.

![Create Virtual File Server - Storage Allocation Settings](image)

3. Add a virtual IP address as shown in Figure 14.

![Add Virtual IP Address](image)
These steps result in a Virtual File Server shown in Figure 15.

**Figure 15. Virtual File Server**

**Step 2: Create a File Store**

1. In the General section, as shown in Figure 16, specify a name, in this instance `gab_filestore3par`, and select the Virtual File Server that you just created.

**Figure 16. Create File Store - General**
2. Use the default Security settings, as shown in Figure 17.

![Create File Store - Security](image)

Figure 17. Create File Store – Security

These steps result in the File Store shown in Figure 18.

![File Store](image)

Figure 18. File Store
Step 3: Create a File Share

1. In the General section of the Create File Share dialog, shown in Figure 19, set the share type to NFS Share and set a share name, for example, `gab_fileshare3par`.

2. In the Share Path section, select the virtual file server and file store that you created earlier, as shown in Figure 20.
3. In the Additional Settings section, set the Permission to Read/Write allowed and the Privilege to root squashing is off (no root squash) as shown in Figure 21.

**Figure 21.** Create File Share – Additional Settings

The overview for File Share is shown below in Figure 22 and contains the information you need to specify the configuration variables.

**Figure 22.** File Share
Configuration
In the following example, it is assumed that the relevant variables are configured as shown in Table 22:

**Table 22. Configuration values**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs_provisioner_role</td>
<td>nfs-provisioner-runner-3par</td>
</tr>
<tr>
<td>nfs_provisioner_name</td>
<td>hpe.com/nfs-3par</td>
</tr>
<tr>
<td>nfs_provisioner_storage_class_name</td>
<td>nfs-3par</td>
</tr>
<tr>
<td>nfs_provisioner_server_ip</td>
<td>10.60.59.102</td>
</tr>
<tr>
<td>nfs_provisioner_server_share</td>
<td>/gab_vfs3par/gab_vfs3par/gab_filestore3par/gab_fileshare3par</td>
</tr>
</tbody>
</table>

Verification
Ensure that your environment satisfies the prerequisites, as described in Deploying the NFS provisioner for Kubernetes, and then run the playbook:

```
# cd Docker-Synergy/ops
# ansible-playbook -i vm_hosts playbooks/k8s-nfs-provisioner.yml --vault-password-file .vault_pass
```

Running the command `kubectl get sc` will show the storage class named `nfs`:

```
# kubectl get sc
NAME      PROVISIONER        AGE
nfs-3par  hpe.com/nfs-3par   5m
```

Create a temporary file `/tmp/pvc-3par.yml` for a persistent volume claim (PVC) named `nfs-3par-test` with a storage class of `nfs-3par`:

```
# cat /tmp/pvc.yml <<EOF
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-3par-test
  annotations:
    volume.beta.kubernetes.io/storage-class: "nfs-3par"
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 100Mi
EOF
```

Create the PVC resource by running `kubectl apply` on this file:

```
# kubectl apply -f /tmp/pvc-3par.yml
persistentvolumeclaim "nfs-3par-test" created
```

Verify that the corresponding persistent volume (PV) was created at the same time:

```
# kubectl get pv
NAME                         CAPACITY   STATUS    CLAIM                   STORAGECLASS   AGE
pvc-5dd91d9b-a520-11e8-...   100Mi      Bound     default/nfs-3par-test   nfs-3par       58s
```
Define a pod that will mount the persistent volume by using the persistent volume claim. The persistent volume is mounted under `/tmp/foo`.

```
# cat /tmp/pod.yml <<EOF
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nfs-3par-pod
spec:
  selector:
    matchLabels:
      app: nfs-3par-pod
  replicas: 1
  template:
    metadata:
      labels:
        app: nfs-3par-pod
    spec:
      volumes:
      - name: pod-data
        persistentVolumeClaim:
          claimName: nfs-3par-test
        containers:
        - name: nfs-3par-pod
          command:
          - sh
          - -c
          - while true; do sleep 1; done
          image: radial/busyboxplus:curl
          volumeMounts:
          - mountPath: /tmp/foo
            name: pod-data
EOF
```

Create the pod resource by running `kubectl apply` on the file.

```
# kubectl apply -f /tmp/pod-3par.yml
deployment.apps "nfs-3par-pod" created
```

Retrieve the pod ID and then execute a command in the pod to create a test file on the persistent volume. The file is named `/tmp/foo/bar.txt` and contains the string `hello`.

```
# pod=$(kubectl get pod | awk '/nfs-3par-pod-/ {print $1}')
# kubectl exec -it $pod -- sh -c "echo hello >/tmp/foo/bar.txt"
```

Check that the file was written correctly:

```
# kubectl exec -it $pod -- sh -c "cat /tmp/foo/bar.txt"
hello
```

Now delete the deployment associated with the pod.

```
# kubectl get deploy nfs-3par-pod
NAME    DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
nfs-3par-pod  1       1         1          1        28m
```
# kubectl delete deploy nfs-3par-pod
deployment.extensions "nfs-3par-pod" deleted

Wait until the status is no longer Terminating and the pod is deleted.

# kubectl get pod

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfs-3par-pod-5c4896699d-zfq9n</td>
<td>1/1</td>
<td>Terminating</td>
<td>0</td>
<td>28m</td>
</tr>
</tbody>
</table>

Re-create the pod as before, by running kubectl apply on the file /tmp/pod-3par.yml.

# kubectl apply -f /tmp/pod-3par.yml
deployment.apps "nfs-3par-pod" created

Retrieve the pod id:

# pod=$(kubectl get pod | awk '/nfs-3par-pod-/ {print $1} ')

# echo $pod
nfs-3par-pod-5c4896699d-kl6bw

# kubectl exec -it $pod -- sh -c "cat /tmp/foo/bar.txt"
hello

**Deploying Sysdig monitoring**

By default, the playbooks for deploying Sysdig are commented out in site.yml and must be explicitly enabled in that file if you want it included in the initial deployment. Alternatively, you can run the specific playbooks detailed in this section in a stand-alone manner, subsequent to the initial deployment.

**Note**

By default, you must have outgoing port 6666 open in your firewall, to allow data to flow to collector.sysdigcloud.com. You can configure the agent to use a different port by setting the collector.port parameter in the configuration file /opt/draios/etc/dragent.yaml. For more information, see the relevant Sysdig documentation at [https://support.sysdig.com/hc/en-us/articles/204205969](https://support.sysdig.com/hc/en-us/articles/204205969).

If you are using a proxy, it must be configured to be "fully-transparent". Non-transparent proxies will not allow the agent to connect.

**Registering for Sysdig trial**

Hewlett Packard Enterprise has teamed up with Sysdig to offer a fully featured 90-day trial version of Sysdig Monitor and Secure as part of the HPE Reference Configuration for Docker Containers as a Service on HPE Synergy Composable Infrastructure. For more details on how to sign up, see the GitHub repository at [https://github.com/HewlettPackard/Docker-Synergy](https://github.com/HewlettPackard/Docker-Synergy).
After registering for the trial, you will be presented with options for setting up your environment, as shown in Figure 23.

**Figure 23.** Sysdig Monitor set up environment

**Sysdig Monitoring for Kubernetes**

If you are deploying Sysdig monitoring on Kubernetes, select the **Kubernetes | GKE | OpenShift** option. You will be presented with an access code, as shown in Figure 24.

**Figure 24.** Sysdig Monitor access code for Kubernetes
Use the `sysdig_access_key` field in your `group_vars/vault`, as described in the section Sysdig configuration for Kubernetes. Once you deploy your environment and your Kubernetes nodes connect to the Sysdig SaaS platform, Sysdig will automatically display information regarding your setup, as shown in Figure 25.

**Figure 25** Sysdig Monitor Spotlight for Kubernetes

Select **View Dashboard** for an entry point to accessing all your monitoring data. Alternatively, you can browse to [https://app.sysdigcloud.com](https://app.sysdigcloud.com) at any time to access your dashboards.

**Sysdig Monitor for Docker swarm**

If you are deploying Sysdig monitoring on Docker swarm, select the **Non-Orchestrated: Native Linux** option. You will be presented with a screen containing details for the URL to download the Sysdig agent, along with your access code embedded in the command, as shown in Figure 26.

**Figure 26** Sysdig Monitor download location and access code for Docker
The download URL is used in the `sysdig_agent` field in `group_vars/vars`, while the access code is stored in the `sysdig_access_key` field in your `group_vars/vault`, as described in the section Sysdig configuration for Docker swarm.

Once you deploy your environment and your Docker swarm nodes connect to the Sysdig SaaS platform, Sysdig will automatically display information regarding your setup. Alternatively, you can browse to https://app.sysdigcloud.com at any time to access your dashboards.

### Deploying Sysdig monitoring on Kubernetes

The latest version of Sysdig supports monitoring of Kubernetes logs and metrics.

#### Prerequisites

- Install the kubectl binary on your Ansible box - see https://docs.docker.com/ee/ucp/user-access/kubectl/
- Install the UCP Client bundle for the admin user - see https://docs.docker.com/ee/ucp/user-access/cli/
- Confirm that you can connect to the cluster by running a test command, for example, `kubectl get nodes`
- Ensure that you have configured the required variables, as described in the section Sysdig configuration for Kubernetes

For example, you add the the relevant variables in the `group_vars/vars` file.

```yaml
sysdig_restricted_control_role: "Restricted Control"
k8s_cluster: ucp.gab-ucp.cloudra.local
```

You should add the access key to the encrypted `group_vars/vault` using the command `ansible-vault edit group_vars/vault`.

```yaml
sysdig_access_key: '10****97-9160-****-9061-84bfd0f****0'
```

#### Running the playbook

The playbook `playbooks/k8s-install-sysdig.yml` is used to automate the configuration of the SaaS setup for Docker swarm.

```
# cd Docker-Synergy/ops
# ansible-playbook -i vm_hosts playbooks/k8s-install-sysdig.yml --vault-password-file .vault_pass
```

Using the Sysdig software as a solution (SaaS) website https://app.sysdigcloud.com, you are able to view, analyze and inspect various different dashboards. Initially, you will just see the monitoring information for the infrastructure itself. Deploy a sample application, as detailed in the section Kubernetes guestbook example with Redis, and use the Sysdig solution to analyze the different facets of the deployed application.

### Deploying Sysdig monitoring on Docker Swarm

The playbook `playbooks/install_sysdig.yml` is used to automate the configuration of the SaaS setup for Docker swarm. By default, this playbook is commented out in `site.yml` and must be explicitly enabled. An access key variable must be set in the `group_vars/vault` file as detailed in Table 18.

```
# cd Docker-Synergy/ops
# ansible-playbook -i vm_hosts playbooks/install_sysdig.yml --vault-password-file .vault_pass
```

Using the Sysdig software as a solution (SaaS) website https://app.sysdigcloud.com, you are able to view, analyze and inspect various different dashboards.

### Solution lifecycle management

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 to fix any existing defects within the product.

In this section, we will cover life cycle management of the different components that are used in this solution. The lifecycle of the following stacks need to be maintained and managed:

- Monitoring Tools (Splunk or Prometheus and Grafana)
- Docker Enterprise Edition 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 Synergy and end with the 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.

**Synergy environment**

HPE Synergy Composer powered by HPE OneView provides fast, reliable, and simplified firmware and driver management across many HPE Synergy components. HPE OneView manages firmware to reduce manual interactions and errors, in addition to minimizing downtime. Firmware updates of management appliances and shared infrastructure are non-disruptive to the production workload.


**VMware components**

The solution in this deployment guide is built on VMware vSphere and leverages VMware ESXi and vCenter. For more information on upgrading vSphere, see the VMware documentation, Introduction to vSphere Upgrade, at https://docs.vmware.com/en/VMware-vSphere/6.5/com.vmware.vsphere.upgrade.doc/GUID-EB29D42E-7174-467C-AB40-DB37236FEAF5.html.

**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 23). 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>
<thead>
<tr>
<th>Order</th>
<th>Component</th>
<th>Dependency (compatibility)</th>
<th>Download/Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Server Software</td>
<td>1. VMware ESXi</td>
<td>vSphere Docker Volume Service on GitHub</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Docker EE</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Client Software</td>
<td>1. VM Operating System</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Docker EE</td>
<td></td>
</tr>
</tbody>
</table>

**Red Hat Enterprise Linux operating system**

This solution is built using Red Hat Enterprise Linux (see Table 24) as the base operating system. When upgrading the operating system on the VMs, first verify that the OS version is compatible with Docker EE by looking at the Docker OS compatibility matrix.

<table>
<thead>
<tr>
<th>Order</th>
<th>Component</th>
<th>Dependency (compatibility)</th>
<th>Download/Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Red Hat Enterprise Linux</td>
<td>1. Docker EE</td>
<td>RHEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. vDVS client software plugin</td>
<td></td>
</tr>
</tbody>
</table>

**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 25.
A banner will be displayed on the UI, as shown in Figure 27, when an update is available for UCP or DTR. You can start the upgrade process by clicking on the banner.

![Universal Control Plane](image)

**Figure 27.** Docker update notification

<p>| Table 25. Docker EE components |</p>
<table>
<thead>
<tr>
<th>Order</th>
<th>Component</th>
<th>Dependency (compatibility)</th>
<th>Download/Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Docker Daemon/Engine</td>
<td>1. VM Operating System</td>
<td>Docker Lifecycle Maintenance</td>
</tr>
<tr>
<td>2</td>
<td>Universal Control Plane</td>
<td>2. vDVS plugin</td>
<td>Docker Compatibility Matrix</td>
</tr>
<tr>
<td>3</td>
<td>Docker Trusted Registry</td>
<td>3. Prometheus and Grafana</td>
<td></td>
</tr>
</tbody>
</table>

**Monitoring tools**

To learn more about upgrading Splunk, see the relevant documentation at [How to upgrade Splunk Enterprise](#).

The Sysdig agent runs as a container and the latest version is pulled from the Docker hub on first installation. Re-run the `install_sysdig.yml` playbook to update to the newest version if required.

Prometheus and Grafana monitoring tools (see Table 26) 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.

<p>| Table 26. Monitoring tools: Prometheus and Grafana |</p>
<table>
<thead>
<tr>
<th>Order</th>
<th>Component</th>
<th>Dependency (compatibility)</th>
<th>Download/Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prometheus</td>
<td>1. Grafana</td>
<td>1. Prometheus Images on Docker Hub</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Docker EE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Grafana</td>
<td>1. Prometheus</td>
<td>2. Upgrading Grafana</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Docker EE</td>
<td></td>
</tr>
</tbody>
</table>

**Windows operating system and Docker EE**

Docker Enterprise Edition for Windows Server (Docker EE) enables native Docker containers on Windows Server. Windows Server 2016 and later versions are supported. This solution has been tested with Windows worker nodes running Windows Server 2016 and with Docker EE 17.06.

**Note**

Docker Universal Control Plane is not currently supported on Windows Server 1709 due to image incompatibility issues. For more information, see the Docker documentation [Install Docker Enterprise Edition for Windows Server](#).

This solution recommends that you only run Windows Server 2016 on your Windows worker nodes and that you install any required updates to your Windows nodes in a timely manner.

For information on how to update Docker EE on Windows Server 2016, see the Docker documentation [Update Docker EE](#).
Backup and restore

This Reference Configuration provides playbooks and scripts to help you back up and restore:

- Docker UCP and DTR
- Docker volumes

Backup and restore UCP and DTR

The playbooks provided in this solution implement the backup and restore procedures as they are described in the Docker documentation at https://docs.docker.com/enterprise/backup/. The solution follows the recommendations in the Docker best practices document at https://success.docker.com/article/backup-restore-best-practices.

Note

It is important that you make copies of the backed up data and that you store the copies in a separate physical location. You must also recognize that the backed up data contains sensitive information such as private keys and so it is important to restrict access to the generated files. However, the playbooks do not backup the sensitive information in your group_vars/vault file so you should make sure to keep track of the credentials for the UCP Administrator.

Warning

The restore procedures do not restore swarm data. You should follow infrastructure as code (IaC) guidelines and maintain your service, stack and network definitions using source code or configuration management tools. You must also ensure that you safely manage the credentials of your administration accounts, as existing UCP Client bundles will not work when you restore UCP on a new swarm.

Backup UCP and DTR

The playbooks support backing up the swarm, UCP, DTR metadata and DTR images.

Backup configuration variables

Table 27 shows the variables related to backing up UCP and DTR. All these variables are defined in the file group_vars/backups. All the data that is backed up is streamed over an SSH connection to the backup server. Currently, the playbooks only support the use of the Ansible box as the backup server.

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backup_server</td>
<td>group_vars/backups</td>
<td>Currently, the playbooks only support the use of the Ansible box as the backup server.</td>
</tr>
<tr>
<td>backup_dest</td>
<td>group_vars/backups</td>
<td>This variable should point to an existing folder on your Ansible box where the root user has write access. All the backups will be stored in this folder. For example, /root/backups.</td>
</tr>
<tr>
<td>#swarm_offline_backup</td>
<td>group_vars/backups</td>
<td>This variable is commented out by default. More information on this variable is provided below.</td>
</tr>
</tbody>
</table>

Backing up the swarm

When you back up the swarm, your services and stack definitions are backed up together with the networks definitions. However, Docker volumes or their contents will not be backed up. (If Docker volumes are defined in stacks, they will be re-created when you restore the stacks, but their content will be lost.) You can back up the swarm using the playbook named backup_swarm.yml which is located in the playbooks folder on your Ansible server. The playbook is invoked as follows:

```bash
# ansible-playbook -i vm_hosts playbooks/backup_swarm.yml
```

This playbook creates two archives in the folder specified by the variable backup_dest in group_vars/backups. By default, the file is named using the following pattern:
<backup_dest>/backup_swarm_<vmname>_<timestamp>.tgz
<backup_dest>/backup_swarm_<vmname>_<timestamp>.vars.tgz

<vmname> is the name of the host (in the inventory) that was used to take the backup, and <timestamp> is the time at which the backup was taken. The file with the extension .vars.tgz contains information regarding the system that was backed up.

You can override the generated file name by defining the variable backup_name on the command line when running the playbook. In the example below:

# ansible-playbook -i vm_hosts playbooks/backup_swarm.yml -e backup_name=my_swarm_backup

The generated files won't have <vmname> or <timestamp> appended:
<backup_dest>/my_swarm_backup.tgz
<backup_dest>/my_swarm_backup.vars.tgz

**Warning**

**Online versus offline backups:** By default, the playbook performs online backups. You can take offline backups by setting the variable swarm_backup_offline to "true". The playbook will then stop the Docker daemon on the machine used to take the backup (a manager or UCP node). Before it does so, the playbook will verify that enough managers are running in the cluster to maintain the quorum. If this is not the case, the playbook will exit with an error. For more information, see the Docker documentation at https://docs.docker.com/engine/swarm/admin_guide/#recover-from-disaster

### Backing up the Universal Control Plane (UCP)

When you backup UCP, you save the data/metadata outlined in Table 28:

<table>
<thead>
<tr>
<th>Table 28. UCP data backed up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
</tr>
<tr>
<td>Configurations</td>
</tr>
<tr>
<td>Access control</td>
</tr>
<tr>
<td>Certificates and keys</td>
</tr>
<tr>
<td>Metrics data</td>
</tr>
<tr>
<td>Organizations</td>
</tr>
<tr>
<td>Volumes</td>
</tr>
</tbody>
</table>

To make a backup of UCP, use playbook/backup_ucp.yml as follows:

# ansible-playbook -i vm_host playbooks/backup_ucp.yml

This playbook creates two archives in the folder specified by the variable backup_dest in group_vars/backups. By default, the files are named using the following pattern:

<backup_dest>/backup_ucp_<ucpid>_<vmname>_<timestamp>.tgz
<backup_dest>/backup_ucp_<ucpid>_<vmname>_<timestamp>.vars.tgz

<ucpid> is the ID of the UCP instance, <vmname> is the name of the host (in the inventory) that was used to take the backup, and <timestamp> is the time at which the backup was taken. The file with the extension .vars.tgz contains information regarding the system which was backed up.

You can override the generated file name by defining the variable backup_name on the command line when running the playbook. In the example below:

# ansible-playbook -i vm_hosts playbooks/backup_ucp.yml -e backup_name=my_ucp_backup
The generated files won't have `<vmname>` or `<timestamp>` appended:

```
<backup_dest>/my_ucp_backup.tgz
<backup_dest>/my_ucp_backup.vars.tgz
```

**Warning**

To create a consistent backup, the backup command **temporarily stops the UCP containers running on the node where the backup is being performed**. User resources, such as services, containers, and stacks are not affected by this operation and will continue to operate as expected. Any long-lasting `docker exec`, `docker logs`, `docker events`, or `docker attach` operations on the affected manager node will be disconnected.

For more information on UCP backup, see the Docker documentation at https://docs.docker.com/datacenter/ucp/2.2/guides/admin/backups-and-disaster-recovery/

**Backing up the Docker Trusted Registry (DTR)**

When you backup DTR, you save the data/metadata outlined in Table 29.

**Table 29. DTR data backed up**

<table>
<thead>
<tr>
<th>Data</th>
<th>Backed up?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td>yes</td>
<td>DTR settings</td>
</tr>
<tr>
<td>Repository metadata</td>
<td>yes</td>
<td>Metadata like image architecture and size</td>
</tr>
<tr>
<td>Access control to repos and images</td>
<td>yes</td>
<td>Data about who has access to which images</td>
</tr>
<tr>
<td>Notary data</td>
<td>yes</td>
<td>Signatures and digests for images that are signed</td>
</tr>
<tr>
<td>Scan results</td>
<td>yes</td>
<td>Information about vulnerabilities in your images</td>
</tr>
<tr>
<td>Certificates and keys</td>
<td>yes</td>
<td>TLS certificates and keys used by DTR</td>
</tr>
<tr>
<td>Image content</td>
<td>no</td>
<td>Needs to be backed up separately, depends on DTR configuration</td>
</tr>
<tr>
<td>Users, orgs, teams</td>
<td>no</td>
<td>Create a UCP backup to backup this data</td>
</tr>
<tr>
<td>Vulnerability database</td>
<td>no</td>
<td>Can be re-downloaded after a restore</td>
</tr>
</tbody>
</table>

To make a backup of DTR metadata, use `playbook/backup_dtr_metadata.yml`

```
# ansible-playbook -i vm_host playbooks/backup_dtr_metadata.yml
```

This playbook creates two archives in the folder specified by the variable `backup_dest` in `group_vars/backups`. By default, the file is named using the following pattern:

```
<backup_dest>/backup_dtr_meta_<replica_id>_<vmname>_<timestamp>.tgz
<backup_dest>/backup_dtr_meta_<replica_id>_<vmname>_<timestamp>.vars.tgz
```

 `<replica_id>` is the ID of the DTR replica that was backed up, `<vmname>` is the name of the host (in the inventory) that was used to take the backup, and `<timestamp>` is the time at which the backup was taken. The file with the extension `.vars.tgz` contains information regarding the system that was backed up.

You can override the generated file name by defining the variable `backup_name` on the command line when running the playbook. In the example below:

```
# ansible-playbook -i vm_hosts playbooks/backup_dtr_metadata.yml -e backup_name=my_dtr_metadata_backup
```

The generated files won't have `<vmname>` or `<timestamp>` appended:

```
<backup_dest>/my_dtr_metadata_backup.tgz
<backup_dest>/my_dtr_metadata_backup.vars.tgz
```
For more information on DTR backups, see the Docker documentation at https://docs.docker.com/datacenter/dtr/2.4/guides/admin/backups-and-disaster-recovery/.

**Backing up DTR data** (images)

To make a backup of the images that are inventoried in DTR and stored on the NFS server, use `playbooks/backup_dtr_images.yml`

```bash
# ansible-playbook -i vm_host playbooks/backup_dtr_images.yml
```

This playbook creates two archives in the folder specified by the variable `backup_dest` in `group_vars/backups`. By default, the files are named using the following pattern:

- `<backup_dest>/backup_dtr_data_<replica_id>_<vmname>_<timestamp>.tgz`
- `<backup_dest>/backup_dtr_data_<replica_id>_<vmname>_<timestamp>.vars.tgz`

`<replica_id>` is the ID of the DTR replica that was backed up, `<vmname>` is the name of the host (in the inventory) that was used to take the backup, and `<timestamp>` is the time at which the backup was taken.

You can override the generated file names by defining the variable `backup_name` on the command line when running the playbook, as shown in the example below:

```bash
# ansible-playbook -i vm_hosts playbooks/backup_dtr_images.yml -e backup_name=my_dtr_data_backup
```

The generated files won't have `<vmname>` or `<timestamp>` appended:

- `<backup_dest>/my_dtr_data_backup.tgz`
- `<backup_dest>/my_dtr_data_backup.vars.tgz`

For more information on DTR backups, see the Docker documentation at https://docs.docker.com/datacenter/dtr/2.4/guides/admin/backups-and-disaster-recovery/.

**Backing up other metadata, including passwords**

The backup playbooks do not backup the sensitive data in your `group_vars/vault` file. The information stored in the `.vars.tgz` files includes backups of the following files:

- **vm_hosts**, a copy of the `vm_hosts` file at the time the backup was taken
- **vars**, a copy of your `group_vars/vars` file at the time the backup was taken
- **meta.yml**, a generated file containing information pertaining to the backup

The `meta.yml` file contains the following information:

```yml
backup_node="<node that took the backup>"
replica_id="<ID of DTR replica if DTR backup>"
backup_source=""
ucp_version="<UCP version if UCP backup>"
dtr_version="<DTR version of DTR backup>"
```
Backup Utility

The script `backup.sh` can be used to take a backup of the swarm, UCP, DTR metadata and the DTR images in one go. You can pass this script an argument (tag) that will be used to prefix the backup filenames, thereby overriding the default naming. Table 30 shows the file names produced by `backup.sh` based on the argument passed in the command line.

Table 30. Backup utility

<table>
<thead>
<tr>
<th>Example</th>
<th>Command line</th>
<th>Generated filenames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>./backup.sh</code></td>
<td>backup_swarm_&lt;vmname&gt;<em>&lt;timestamp&gt;.tgz, backup_ucp</em>&lt;ucpid&gt;<em>&lt;vmname&gt;</em>&lt;timestamp&gt;.tgz, backup_dtr_meta_&lt;replica_id&gt;<em>&lt;vmname&gt;</em>&lt;timestamp&gt;.tgz, backup_dtr_data_&lt;replica_id&gt;<em>&lt;vmname&gt;</em>&lt;timestamp&gt;.tgz, and the corresponding .vars.tgz files</td>
</tr>
<tr>
<td>Custom</td>
<td><code>./backup.sh my_backup</code></td>
<td>my_backup_swarm.tgz, my_backup_ucp.tgz, my_backup_dtr_meta.tgz, my_backup_dtr_data.tgz, and the corresponding .vars.tgz files</td>
</tr>
<tr>
<td>Date</td>
<td><code>./backup.sh $(date '+%Y_%m_%d_%H%M%S')</code></td>
<td>&lt;date&gt;_swarm.tgz, &lt;date&gt;_ucp.tgz, &lt;date&gt;_dtr_meta.tgz, &lt;date&gt;_dtr_data.tgz, and the corresponding .vars.tgz files</td>
</tr>
</tbody>
</table>

In addition, the `backup.sh` script accepts an optional switch that will let you specify the location of the password file that will be passed to the `ansible-playbook` commands in the script. This is required if you have encrypted the `group_vars/vault` file. The general syntax for this script is as follows:

```
./backup.sh [ -v <Vault Password File> ] [ tag ]
```

Related playbooks

- `playbooks/backup_swarm.yml` is used to back up the swarm data
- `playbooks/backup_ucp.yml` is used to back up UCP
- `playbooks/backup_dtr_meta.yml` is used to back up DTR metadata
- `playbooks/backup_dtr_images.yml` is used to back up DTR images

Restoring your cluster after a disaster

The playbooks address a disaster recovery scenario where you have lost your entire cluster and all the VMs. Other scenarios and how to handle them are described in the Docker documentation including the following scenarios:

- You have lost one UCP instance but your cluster still has the quorum. The easiest way is to recreate the missing UCP instance from scratch.
- You have lost the quorum in your UCP cluster but there is still one UCP instance running.
- You have lost one instance of DTR but still have a quorum of replicas. The easiest way is to recreate the missing DTR instance from scratch.
- You have lost the quorum of your DTR cluster but still have one DTR instance running.

Before you restore

**Step 1.** Retrieve the backup files using your chosen backup solution and save them to a folder on your Ansible server. If you have used timestamps in the naming of your backup files, you can use them to determine the chronological order. If you used the `backup.sh` script specifying a date prefix, you can use that to identify the matching set of backup files. You should choose the files in the following reverse chronological order, from the most recent to the oldest file. Make sure you restore both the *.tgz and the *.vars.tgz files.

1. DTR images backup
2. DTR metadata backup
3. UCP backup
4. Swarm backup

In this example, we will assume a set of backup files stored in `/root/restore` that were created specifying a date prefix. These will have names like `2018_04_17_151734_swarm.tgz, 2018_04_17_151734_ucp.tgz, etc` and the corresponding `.vars.tgz files.`
Step 2: Retrieve the DTR replica ID, the DTR version and the UCP version

To retrieve the ID of the replica that was backed up, as well as the version of DTR, you need to extract the data from the .vars.tgz file associated with the archive of the DTR metadata. You can retrieve this as follows:

[root@hpe2-ansible ops]# tar -Oxf /root/restore/2018_04_17_151734_dtr_meta.vars.tgz meta.yml
backup_node="hpe-dtr01"
replica_id="ad5204e8a4d0"
backup_source=""
dtr_version="2.5.3"

[root@hpe2-ansible ops]# tar -Oxf /root/restore/2018_04_17_151734_ucp.vars.tgz meta.yml
backup_node="hpe-ucp01"
replica_id="" 
backup_source=""
ucp_version="3.0.4"
dtr_version=""

Take note of the replica ID (ad5204e8a4d0), the version of DTR (2.5.3) and the version of UCP (3.0.4).

Step 3: Populate the group_vars/backups file

backup_swarm: "/root/restore/2018_04_17_151734_swarm.tgz"
backup_ucp: "/root/restore/2018_04_17_151734_ucp.tgz"
backup_dtr_meta: "/root/restore/2018_04_17_151734_dtr_meta.tgz"
backup_dtr_data: "/root/restore/2018_04_17_151734_dtr_data.tgz"
backup_dtr_id: "ad5204e8a4d0"
backup_dest: "/root/backups"
backup_server: <IP of your ansible box>

You should populate your group_vars/backups file as above, with the backup_dtr_id variable containing the value you retrieved in the preceding step as replica_id="ad5204e8a4d0".

Step 4: Verify that your group_vars/vars file specifies the correct versions of DTR and UCP.

The playbooks use the versions of UCP and DTR as specified in your group_vars/vars file to restore your backups. You must ensure that the versions specified in your current group_vars/vars file correspond to the versions in the backups as determined above.

[root@hpe2-ansible ops]# cat group_vars/vars | grep dtr_version
dtr_version: '2.5.3'

[root@hpe2-ansible ops]# cat group_vars/vars | grep ucp_version
ucp_version: '3.0.4'

Step 5: Restore UCP admin credentials if required

You must ensure that the UCP admin credentials in your current group_vars/vars file are those that were in effect when you generated the backup files. If they have changed since then, you must restore the original credentials for the duration of the restore procedure.

Step 6: Restore your inventory (vm_hosts)

Your inventory must reflect the environment that was present when the backup files were created. You can find a copy of the inventory as it was when the backup was taken in the *.vars.tgz files.
Restore UCP and DTR

Warning
This procedure is aimed at restoring a cluster after a disaster. It assumes you have lost all the VMs in your cluster and want to redeploy using data that you backed up earlier. The solution follows Docker best practice, which means the swarm artifacts are not restored. You will need to restore your Docker volumes and your applications (stacks and services) when this procedure is complete.

1. Ensure that you have completed all the preliminary steps as outlined in the section Before you restore.
2. Run the restore playbook
   \[\text{ansible-playbook -i vm_hosts restore.yml}\]
3. If you are using the image scanning functionality in DTR, you will need to re-download the vulnerability database. For more information, see the Docker documentation here.

You are now ready to restore your Docker volumes and your applications.

Restore DTR metadata and DTR images

Note
This procedure restores DTR metadata and images and assumes you have lost all the DTR VMs in your cluster. It will redeploy using the DTR data that you backed up earlier and will also restore the images if the folder exported by the NFS VM is empty.

1. Ensure that you have completed all the preliminary steps as outlined in the section Before you restore. In this scenario, you need the archives for the DTR metadata and the DTR images.
2. Ensure that all the DTR VMs listed in your inventory are destroyed, using the vSphere Web Client to delete them if required. If you want to restore the DTR images you should also delete the NFS VM.
3. Remove the DTR nodes from the swarm by running the docker node rm <DTR node> command on a UCP node for each DTR node in your cluster. The following example shows the sequence of commands to use to remove the DTR nodes:

```bash
[coot@hpe2-ansible ops]$ docker node ls
ID       HOSTNAME                     STATUS              AVAILABILITY
aiz... *   hpe-ucp02.cloudra.local      Ready               Active
gvf...     hpe-dtc01.cloudra.local      Down                Active
ic4...     hpe-ucp03.cloudra.local      Ready               Active
mwf...     hpe-dtc02.cloudra.local      Down                Active
oqy...     hpe-ucp01.cloudra.local      Ready               Active
xqe...     hpe-wocker01.cloudra.local   Ready               Active
zdu...     hpe-dtc03.cloudra.local      Down                Active

[coot@hpe2-ansible ops]$ docker node rm hpe-dtc01.cloudra.local
hpe-dtc01.cloudra.local
[coot@hpe2-ansible ops]$ docker node rm hpe-dtc02.cloudra.local
hpe-dtc02.cloudra.local
[coot@hpe2-ansible ops]$ docker node rm hpe-dtc03.cloudra.local
hpe-dtc03.cloudra.local

[coot@hpe2-ansible ops]$ docker node ls
ID       HOSTNAME
```

```bash
[coot@hpe2-ansible ops]$ docker node rm hpe-dtc01.cloudra.local
hpe-dtc01.cloudra.local
[coot@hpe2-ansible ops]$ docker node rm hpe-dtc02.cloudra.local
hpe-dtc02.cloudra.local
[coot@hpe2-ansible ops]$ docker node rm hpe-dtc03.cloudra.local
hpe-dtc03.cloudra.local

[coot@hpe2-ansible ops]$ docker node ls
ID       HOSTNAME
```
4. Run the restore script:
   
   ```bash
   ./restore_dtr.sh
   ```

5. If you are using the image scanning functionality in DTR, you will need to re-download the vulnerability database. For more information, see the Docker documentation here.

**Related playbooks**
- `playbooks/restore_swarm.yml` is used to restore the swarm data
- `playbooks/restore_dtr_meta.yml` is used to restore DTR metadata
- `playbooks/restore_dtr_images.yml` is used to restore DTR images

### Backup and restore Docker persistent volumes

There are a number of prerequisites that must be fulfilled before you backup and restore your Docker persistent volumes.

- vSphere clusters should have access to a datastore specifically for backups. This is a separate Virtual Volume created on the HPE 3PAR StoreServ and presented to all the hosts in the vSphere cluster.
- Backup software must be available. HPE Recovery Manager Central and HPE 3PAR StoreServ is recommended but other customer backup and restore solutions are acceptable.

A number of restrictions also apply:

- Volumes may not be in use when a volume is cloned. Any container that has the volume attached must be paused prior to creating the clone. The container can be resumed once the clone is complete.
- When Docker volumes need to be restored from backup, the backup datastore needs to be detached from all vSphere cluster servers prior to restoration.

#### Persistent storage backup solution

**Creating the volume**

Docker persistent volumes can be created from a worker node using the following command:

```bash
docker volume create --driver=vsphere --name=MyVolume@MyDatastore -o size=10gb
```

**Cloning the volume**

**Note**

Prior to creating a clone of a volume, any containers accessing the volume should be paused or stopped.

```bash
docker volume create --driver=vsphere --name=CloneVolume@DockerBackup -o clone-from=MyVolume@MyDatastore -o access=read-only
```

**Snapshot and back up HPE 3PAR Virtual Volumes with HPE Recovery Manager Central and HPE StoreOnce**

HPE Recovery Manager Central (RMC) software integrates HPE 3PAR StoreServ All-Flash arrays with HPE StoreOnce Systems to leverage the performance of snapshots with the protection of backups. RMC uses a direct backup model to orchestrate data protection between the array and the backup system without a backup application. When the first full backup is complete, each subsequent backup is incremental, making it...
significantly faster than traditional backup methods, particularly for higher volumes of data. Backups to HPE StoreOnce are block-level copies of volumes, de-duplicated to save space. Because RMC snapshots are self-contained, fully independent volumes, they can be restored to any HPE 3PAR array in the event of a disaster. See Figure 28 for an overview of the architecture.

HPE Recovery Manager Central enables you to replicate data from the source storage system (HPE 3PAR StoreServ) to the destination storage system (HPE StoreOnce). The replication is based on point-in-time snapshots.

HPE Recovery Manager Central is installed as a VM on VMware vSphere ESXi. It can be installed on the HPE Synergy platform on a separate (from the Docker Solution) vSphere cluster or external to the Synergy environment as long as the external server has connectivity to the HPE 3PAR StoreServ and HPE StoreOnce. HPE RMC can be installed directly on an ESXi host or can be deployed to a VMware vCenter managed environment. For this solution, the standalone “RMC only” is installed. If HPE RMC is installed in the HPE Synergy environment, iSCSI connection to the HPE 3PAR StoreServ is required.

Figure 28. HPE Recovery Manger Central and HPE StoreOnce

- The connectivity between HPE 3PAR StoreServ and HPE RMC for data traffic is over iSCSI.
- The connectivity between HPE StoreOnce and HPE RMC is over CoEthernet (Catalyst OverEthernet).
- The connectivity between HPE RMC, HPE 3PAR StoreServ, and HPE StoreOnce for management traffic is over IP.
Figure 29 illustrates the connectivity between various components.

![Figure 29. Connectivity](image)

Refer to HPE RMC User guide for detailed instructions on setup and configuration of HPE RMC and HPE StoreOnce. When RMC is installed, it can be configured with the Backup Appliance Persona. The Backup persona allows the RMC to manage snapshots and Express Protect Backups. During installation, RMC configuration should specify Data Protection of RMC Core. The initial configuration of backups can be set up using the Protection Wizard. The Protection Wizard assists with creation of a Recovery Set.

Create a Recovery Set as shown in Figure 30 and select to protect your DockerBackup volume. Once you have created your Recovery Set, the next step is to create Protection Jobs. The Auto Protection Job simplifies the initial configuration of policies. The Auto Protection Job will automatically configure the storage, define default backup policies and protection policies and will schedule snapshots or express protect jobs with the created policies.

![Figure 30. Recovery Set Overview](image)
RMC uses the Express Protect feature, as shown in Figure 31, to enable the backup of the snapshot data from the HPE 3PAR array to the HPE StoreOnce system for deduplication and long-term retention.

The Express Restore feature restores either snapshots or base volumes.

RMC leverages HPE 3PAR StoreServ SnapDiff technology to create an application-consistent snapshot. Only changed blocks are sent to the HPE StoreOnce system, which minimizes network traffic and saves disk space on the backup system.

Restoring the volume

If a Docker persistent storage volume needs to be restored from backup, the HPE 3PAR volume can be restored either from a snapshot saved on the HPE 3PAR or from a backup on HPE StoreOnce. Stop any applications using the Docker volume. Use the vSphere Web UI to unmount the datastore from the vSphere cluster. Use RMC to detach the HPE 3PAR virtual volumes prior to restoring the backup. The volume can be restored from a Recovery Set restore point as shown in Figure 32. The Express Protect restore point will restore the volume from the HPE StoreOnce system. A Snapshot Set restore point will restore an HPE 3PAR StoreServ snapshot.

Once the HPE 3PAR virtual volume is restored, the volume must be reattached to the vSphere cluster from RMC. After the volume is reattached, the datastore must be mounted. Applications can then access the restored docker volume.

Integrate UCP and DTR backup with HPE RMC and HPE StoreOnce

You can take advantage of HPE Recovery Manager Central and HPE StoreOnce to provide scheduled snapshots and backup protection for the data generated by the backup procedure for Docker UCP and DTR.

1. Create a datastore from the Backup virtual volume you created and present it to all hosts in the vSphere cluster. This backup datastore is used for storing copies of Docker persistent volumes as well as backups of DTR and UCP.

2. The Ansible server is used to create backup and restore files for DTR and UCP on the local hard drive. The backup files should be copied to the DockerBackup datastore which can be automatically configured for snapshots and offsite backup.
3. Edit the Ansible server configuration from vCenter. Add a new hard disk and specify the location as the Docker Backup datastore as shown in Figure 33.

![Reference Architecture](image)

**Figure 33.** Add new hard disk

4. After the hard disk is added, it is visible from the Linux operating system. From the Ansible server:
   ```bash
   # ls /dev/sd*
   
   The newly added storage should appear as /dev/sdb. Now, make a filesystem, ignoring any warnings:
   ```bash
   # mkfs -t ext4 /dev/sdb
   
   5. Create a mount point for the new disk:
      ```bash
      # mkdir /dockerbackup
      
      6. Edit the `/etc/fstab` file and add the following line:
         ```bash
         /dev/sdb /dockerbackup ext4 defaults 0 0
         ```
      
      7. After saving the change, mount the new volume using:
         ```bash
         # mount -a
         ```

      Each time you backup Docker UCP and DTR using the `backup.sh` script, you should copy the generated files from the `/root/backups` folder to `/dockerbackup`. You may wish to add a command to the backup script to automate this process.

The virtual volume used to host the DockerBackup datastore can be scheduled for snapshot and backup protection with HPE Recovery Manager Central and HPE StoreOnce as described in the section Backup and restore Docker persistent volumes. Data backed up to HPE StoreOnce can be restored to the HPE 3PAR StoreServ and attached to the Ansible host for recovery.
Summary

This document has described how to architect and deploy a Docker CaaS platform on HPE Synergy and HPE 3PAR storage, using Ansible playbooks to quickly install and deploy a production ready container environment. This deployment includes a highly available container cluster with backup services and persistent data support. This solution is ideal for customers looking to run containers on VMs to take advantage of the resource efficient usage of virtual machines for Docker containers, and having the ability to run legacy and new container applications side-by-side. Customers deploying Docker containers on a large scale, on Linux and Microsoft Windows, should consider HPE Synergy as the deployment infrastructure.

Appendix A: Bill of materials

The following BOMs contain electronic license to use (E-LTU) parts. Electronic software license delivery is now available in most countries. HPE recommends purchasing electronic products over physical products (when available) for faster delivery and for the convenience of not tracking and managing confidential paper licenses. For more information, please contact your reseller or an HPE representative.

Note

Part numbers are at time of publication and subject to change. The bill of materials does not include complete support options or other rack and power requirements. If you have questions regarding ordering, please consult with your HPE Reseller or HPE Sales Representative for more details. hpe.com/us/en/services/consulting.html.

Table 31. Bill of materials

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BW908A</td>
<td>HPE 42U 600x1200mm Enterprise Shock Rack</td>
</tr>
<tr>
<td>4</td>
<td>AF522A</td>
<td>HPE Intelligent 8.6kVA/L15-30P/NA/J PDU</td>
</tr>
<tr>
<td>1</td>
<td>HC790A</td>
<td>HPE Integration Center Routg Service FIO</td>
</tr>
<tr>
<td>1</td>
<td>BW932A</td>
<td>HPE 600mm Rack Stabilizer Kit</td>
</tr>
<tr>
<td>1</td>
<td>BW909A</td>
<td>HPE 42U 1200mm Side Panel Kit</td>
</tr>
<tr>
<td>1</td>
<td>JG505A</td>
<td>HPE 59xx CTO Switch Solution</td>
</tr>
<tr>
<td>2</td>
<td>JG510A</td>
<td>HPE 5900AF 48G 4XG 2QSFP+ Switch</td>
</tr>
<tr>
<td>4</td>
<td>JD096C</td>
<td>HPE X240 10G SFP+ SFP+ 1.2m DAC Cable</td>
</tr>
<tr>
<td>2</td>
<td>JC680A</td>
<td>HPE 58x0AF 650W AC Power Supply</td>
</tr>
<tr>
<td>2</td>
<td>JC682A</td>
<td>HPE 58x0AF 8ck(pwr) Frt(prt) Fan Tray</td>
</tr>
<tr>
<td>3</td>
<td>797740-B21</td>
<td>HPE Synergy12000 CTO Frame 1xFLM 10x Fan</td>
</tr>
<tr>
<td>3</td>
<td>798096-B21</td>
<td>HPE Synergy 12000F 6x 2650W AC Ti FIO PS</td>
</tr>
<tr>
<td>2</td>
<td>804353-B21</td>
<td>HPE Synergy Composer</td>
</tr>
<tr>
<td>3</td>
<td>804942-B21</td>
<td>HPE Synergy Frame Link Module</td>
</tr>
<tr>
<td>3</td>
<td>804938-B21</td>
<td>HPE Synergy 12000 Frame Rack Rail Option</td>
</tr>
<tr>
<td>1</td>
<td>804943-B21</td>
<td>HPE Synergy 12000 Frame 4x Lift Handle</td>
</tr>
<tr>
<td>18</td>
<td>TK738A</td>
<td>HPE 2.0m 250V 16A C19-C20 Sgl IPD Jpr Crd</td>
</tr>
<tr>
<td>2</td>
<td>804937-B21</td>
<td>HPE Synergy Image Streamer</td>
</tr>
<tr>
<td>Quantity</td>
<td>Part number</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>15</td>
<td>871942-B21</td>
<td>HPE SY 480 Gen10 CTO Premium Cmpt Mdl (64Gb Memory)</td>
</tr>
<tr>
<td>30</td>
<td>873388-B21</td>
<td>HPE SY 480 Gen10 6130 Kit</td>
</tr>
<tr>
<td>30</td>
<td>873388-L21</td>
<td>HPE SY 480 Gen10 6130 Kit</td>
</tr>
<tr>
<td>30</td>
<td>815100-B21</td>
<td>HPE Q4 2166-24P Kit</td>
</tr>
<tr>
<td>15</td>
<td>871573-B21</td>
<td>HPE Smart Array P416ie-m</td>
</tr>
<tr>
<td>15</td>
<td>875242-B21</td>
<td>HPE Smart Array P416ie-m SAS Cable Kit</td>
</tr>
<tr>
<td>15</td>
<td>777430-B21</td>
<td>HPE SY 3820C 10/20Gb CAN</td>
</tr>
<tr>
<td>15</td>
<td>875242-B21</td>
<td>HPE 96W Smart Stor Battery 260mm Cbl Kit</td>
</tr>
</tbody>
</table>

**HPE SY 480 Gen10 compute components**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>794502-B23</td>
<td>HPE VC SE 40Gb F8 Module</td>
</tr>
<tr>
<td>4</td>
<td>779218-B21</td>
<td>HPE SY 20Gb Interconnect Link Mod</td>
</tr>
<tr>
<td>6</td>
<td>755985-B21</td>
<td>HPE SY 12Gb SAS Connection Module</td>
</tr>
</tbody>
</table>

**HPE SY 480 Gen10 Fabric Components**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>835386-B21</td>
<td>HPE SY 3940 CTO Storage Module</td>
</tr>
<tr>
<td>3</td>
<td>757323-B21</td>
<td>HPE SY 3940 IO Adapter</td>
</tr>
<tr>
<td>30</td>
<td>785067-B21</td>
<td>HPE 300GB 12G SAS 10K 2.5in SC ENT HDD</td>
</tr>
</tbody>
</table>

**Cables and Transceivers**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>804101-B21</td>
<td>HPE SY Interconnect Link 3m AOC</td>
</tr>
<tr>
<td>2</td>
<td>720199-B21</td>
<td>HPE BLc 40G QSFP+ 3m DAC Cable</td>
</tr>
<tr>
<td>8</td>
<td>720193-B21</td>
<td>HPE BLc QSFP+ to SFP+ Adapter</td>
</tr>
<tr>
<td>8</td>
<td>455883-B21</td>
<td>HPE BLc 10G SFP+ SR Transceiver</td>
</tr>
<tr>
<td>8</td>
<td>AJ837A</td>
<td>HPE 15m Multi-mode OM3 LC/LC FC Cable</td>
</tr>
<tr>
<td>9</td>
<td>861412-B21</td>
<td>HPE CAT6A 6ft Cbl</td>
</tr>
<tr>
<td>2</td>
<td>838327-B21</td>
<td>HPE SY Dual 10GBASE-T QSFP+ 30m RJ45 Transceiver</td>
</tr>
</tbody>
</table>

**HPE SY 480 Gen10 Composable Storage Components (not used in this solution)**

**Table 32. Bill of materials HPE 3PAR**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K2Q36B</td>
<td>HPE 3PAR 8200 2N+SW Storage Field Base</td>
</tr>
<tr>
<td>2</td>
<td>H6Z10A</td>
<td>HPE 3PAR 8000 2-pt 10Gb iSCSI/FCoE Adptr</td>
</tr>
<tr>
<td>8</td>
<td>K2PB88</td>
<td>HPE 3PAR 8000 480GB+SW Non-AFC SFF SSD</td>
</tr>
<tr>
<td>1</td>
<td>HA114A1</td>
<td>HPE Installation and Startup Service</td>
</tr>
<tr>
<td>1</td>
<td>HA114A1 SXU</td>
<td>HPE Startup 3PAR 8200 2N Fld Int Bas SVC</td>
</tr>
<tr>
<td>1</td>
<td>K2R29A</td>
<td>HPE 3PAR StoreServ RPS Service Processor</td>
</tr>
<tr>
<td>1</td>
<td>H1K92A3</td>
<td>HPE 3Y Proactive Care 24x7 Service</td>
</tr>
<tr>
<td>1</td>
<td>H1K92A3 W3G</td>
<td>HPE 3PAR 8200 2N+SW Storage Base Support</td>
</tr>
<tr>
<td>8</td>
<td>H1K92A3 X8G</td>
<td>HPE 3PAR 8000 480GB+SW LFF SSD Supp</td>
</tr>
<tr>
<td>1</td>
<td>H1K92A3 YNW</td>
<td>HPE 3PAR StoreServ RPS Service Proc Supp</td>
</tr>
<tr>
<td>2</td>
<td>H1K92A3 YTN</td>
<td>HPE 3PAR 8000 2-pt 10Gb FCoE Adptr Supp</td>
</tr>
<tr>
<td>1</td>
<td>L7F20AAE</td>
<td>HPE 3PAR All-in S-sys SW Current E-Media</td>
</tr>
</tbody>
</table>

**HPE 3PAR StoreServ 8200 with iSCSI adapters and accessories**
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L7F22AAE</td>
<td>HPE 3PAR All-in M-sys SW Current E-Media</td>
</tr>
<tr>
<td>1</td>
<td>C753SA</td>
<td>HPE RJ45 to RJ45 Cat5e Black M/M 7.6ft 1-pack Data Cable</td>
</tr>
<tr>
<td>8</td>
<td>H0JD6A1</td>
<td>HPE 3PAR SSD Extended Replacement SVC</td>
</tr>
<tr>
<td>1</td>
<td>HA124A1</td>
<td>HPE Technical Installation Startup SVC</td>
</tr>
<tr>
<td>1</td>
<td>HA124A1</td>
<td>HPE Startup 3PAR Vrt Cpy Lvl1 Tier 1 SVC</td>
</tr>
<tr>
<td>1</td>
<td>HA124A1</td>
<td>HPE Startup 3PAR 8K Mlt Sys PM PP RC SVC</td>
</tr>
</tbody>
</table>

**Software Licenses**

Licenses are required for the following software components:

- VMware
- Red Hat Linux
- Docker EE
- HPE Recovery Manager Central and HPE StoreOnce
- Splunk (optional software)
- Sysdig (optional software)

**Appendix B: Using customer supplied certificates for UCP and DTR**

Table 33 lists the variables used when configuring customer supplied certificates for UCP and DTR.

<table>
<thead>
<tr>
<th>Variable</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucp_certs_dir</td>
<td>group_vars/vars</td>
<td>If ucp_certs_dir is not defined, UCP is installed with self-signed certificates and DTR is installed with the --ucp-insecure-tls switch. If ucp_certs_dir is defined, this is a folder on the Ansible machine that must contain 3 files: ca.pem, the root CA certificate in PEM format; cert.pem, the server certificate optionally followed by intermediate CAs; key.pem, the private key that comes with the cert.pem certificates.</td>
</tr>
<tr>
<td>dtr_certs_dir</td>
<td>group_vars/vars</td>
<td>If dtr_certs_dir is not defined, DTR is installed with self-signed certificates. If dtr_certs_dir is defined, this is a folder on the Ansible machine that must contain 3 files: ca.pem, the root CA certificate in PEM format; cert.pem, the server certificate optionally followed by intermediate CAs; key.pem, the private key that comes with the cert.pem certificates.</td>
</tr>
</tbody>
</table>

**Note**

The installation will fail if the ca.pem, cert.pem and key.pem files cannot be found in the folders designated by dtc_certs_dir and ucp_certs_dir or if they don't constitute valid certificates.

The certificates should specify the names of the FQDNs of the load balancer and the FQDNs of the VMs themselves. This applies to both the UCP server certificate and the DTR server certificate.
Generating and testing certificates

In the example described here we have a root CA named Example root CA and an intermediate CA named Intermediate CA valid 3 years. The intermediate CA signs the server certificates for UCP and DTR.

Below is the start of the output displayed by running the `openssl x509` utility against the `ca.pem` file (the root CA certificate).

```bash
[root@ansible ucp_certs]# openssl x509 -text -noout -in ca.pem|head -14
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      0d:07:ca:ea:00:37:77:6e:25:e0:18:3e:0e:db:80:0f:11:cb:1b:3f
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: CN=Example Root CA
  Validity
    Not Before: Apr 24 20:12:01 2018 GMT
    Not After : Apr 21 20:12:30 2028 GMT
  Subject: CN=Example Root CA
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (4096 bit)
```

Here is an excerpt from the example `ca.pem` file:

```
-----BEGIN CERTIFICATE-----
MIIFJTCCAw2gAwIBAgIUDQfK6gA3d24l4Bg+DtuADxHLGz8wDDYJKoZIhvcNAQEL
BOAwGjEYMByGA1UEAxMPRXhhbXBsb290IENBMB4XDTE4MDQyNDIwMTIwMVoX
...
...
uXzYbCtU6Jt9B3fayAeWwsQv+jDSzuuA3reOM1x838iIZWoX93f4yLJWLJ17xsY
btvKBmqKDCsAqsOLFLnNj/JyYq4e9a6Xxycyn9FXNPzuEsfjENGHn+csY+w3E987T
MNvIY376xZbyAc1CV5kmmZzjU5b6kgTBD==
-----END CERTIFICATE-----
```

The `cert.pem` file should contain the server certificate itself, followed by your intermediate CA's certificate. The following example shows how to extract the intermediate CA certificate from `cert.pem` and to save it to a file named `intca.pem`. Using the `openssl x509` utility, you can display the content of the `intca.pem` file in human readable form. This certificate was signed by the example CA above (Issuer = 'Example Root CA').

```bash
[root@ansible ucp_certs]# openssl x509 -text -noout -in intca.pem|head -14
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: CN=Example Root CA
  Validity
    Not Before: Apr 24 20:12:09 2018 GMT
    Not After : Apr 23 20:12:39 2021 GMT
  Subject: CN=Intermediate CA valid 3 years
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: [4096 bit]
```

Here is an excerpt from the `incta.pem` file showing the example Intermediate CA certificate:
-----BEGIN CERTIFICATE-----
MIIFcjCCAlggAwIBAgIUuAx4MhiDP8jJSUg1duVb6kYegSRgwDDY3KoZIhvcNAQEL
BQAwGjQMBgGA1UdDwEB/woQCMAAwDQYJKoZIhvcNAQELBQAwGgYDVR0TAQH/BAI

... 

a2tL5nwR7P0iAc/kk9MIR2WrLnb4cYth7jEjspU9dBqsXgsTozzW1wqI9ybZwvL
N1i3n2andV1yqdo0aBM2M/1DNFkVwW3JeAcKvDA9j95n/BWFTjzo+Y0z9pYit6T7
16CGu3be
-----END CERTIFICATE-----

The openssl x509 utility will only decrypt the first certificate found in cert.pem, so you don't need to extract the server certificate from cert.pem. In this example, the server certificate is signed by the intermediate CA above. Note the Subject Alternate Names: hpe-ucp.cloudra.local is the FQDN of the UCP load balancer, and the other names are those of the UCP instances (hpe-ucp01.cloudra.local, hpe-ucp02.cloudra.local, hpe-ucp03.cloudra.local).

[root@ansible ucp_certs]# openssl x509 -text -noout -in server.pem
Certificate:
  Data:
    Version: 3 (0x2)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=Intermediate CA valid 3 years
    Validity
      Not Before: Apr 24 20:17:30 2018 GMT
      Not After : Apr 24 20:18:00 2019 GMT
    Subject: O=HPE, OU=CloudRA Team, CN=hpe-ucp.cloudra.local
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      CA Issuers - URI:http://localhost:8200/v1/intca
      ( portions removed )
    X509v3 Subject Alternative Name:
      DNS:hpe-ucp.cloudra.local, DNS:hpe-ucp01.cloudra.local, DNS:hpe-ucp02.cloudra.local, DNS:hpe-ucp03.cloudra.local

The following excerpts from cert.pem show the first certificate which is the server certificate itself and the second certificate which is the intermediate CA's certificate.

-----BEGIN CERTIFICATE-----
MIIFG6CCAwGgAwIBAgIUJdn4HZsdI/EhV1TyQ8xDnMivuuwwDDYJKoZIhvcNAQEL
BQAwKDEmMCQGA1UEAxMdSW50ZXJtZWRpYXRlIENBIHZhbGlkIDMgeWVhcnMwHhcN
...

sOR4I3Qnc5oNISng517wW1d4RMmwmXOgH1H50KAuJHfJXH4b9tIzXw/zGTVr4Z
l1YKbEwJcGAvvEkN+x==
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIFcjCCAlggAwIBAgIUuAx4MhiDP8jJSUg1duVb6kYegSRgwDDY3KoZIhvcNAQEL
BQAwGjQMBgGA1UdDwEB/woQCMAAwDQYJKoZIhvcNAQELBQAwGgYDVR0TAQH/BAI

... 

Ni1i3n2andV1yqdo0aBM2M/1DNFkVwW3JeAcKvDA9j95n/BWFTjzo+Y0z9pYit6T7
Finally, here is an excerpt from key.pem, the private key that goes with the server certificate.

---- BEGIN RSA PRIVATE KEY -----
MIIEpQIBAAKCAQEA5zmb52ufEBOa3cXhY2H5RZNa77/7iyXiZ+U5+Jw9BN5
dx/X3NncSEB/PvoS/ma6KHcnURBnqbu/B2umKN/tm/eSpY861YnGw+bcOgtiU
...
A0SGidSMk3hFX1Laetgq4EUBczZ07I8M5R64Ul1aMFNFyj4Xgh2mZtNe1wNBw
... 

----- END RSA PRIVATE KEY -----

Verify your certificates

The playbooks do not verify the validity of the certificate files you supply so you should verify them manually before you start your deployment.

Verify that the private and the server cert match

On the Ansible box, run the following commands:

```bash
ckcert=$(openssl x509 -noout -modulus -in cert.pem | openssl md5)
ckkey=$(openssl rsa -noout -modulus -in key.pem| openssl md5)
if [ "$ckkey" == "$ckcert" ] ; then echo "Private key and Certificate match" ; else echo "STOP! Private Key and Certificate don't match" ; fi
```

Verify that the server certificate was signed by the CA

Extract all but the first certificate from `cert.pem` (i.e. extract the certs for the intermediate CA authorities) into the file `int.pem`

```bash
sed -e '1,/-----END CERTIFICATE-----/d' cert.pem >int.pem
```

Combine `int.pem` and `ca.pem` to form `cachain.pem`:

```bash
cat int.pem ca.pem > cachain.pem
```

Finally, verify that `cert.pem` was signed by the CA or by an intermediate CA:

```bash
openssl verify -verbose -CAfile cachain.pem cert.pem
```

A successful check will generate output similar to:

```
[root@ansible ucp_certs]# cat int.pem ca.pem > cachain.pem
[root@ansible ucp_certs]# openssl verify -verbose -CAfile cachain.pem cert.pem
cert.pem: OK
```

An unsuccessful check will generate output similar to:

```
[root@ansible ucp_certs]# openssl verify -verbose -CAfile cachain.pem cert.pem
cert.pem: 0K
```

Appendix C: Enabling SSL between the universal forwarders and the Splunk indexers using your certificates

The procedure for enabling SSL between the universal forwarders and the Splunk indexers using your certificates is described below. In summary, the following steps are required:

1. Set the variable `splunk_ssl` to `yes` in `group_vars/vars`
2. Put your root CA certificate and your server certificate files in `/root/Docker-Synergy/ops/files/splunk/linux/SPLUNK_HOME/etc/mycerts`

3. Uncomment the `[sslConfig] stanza in the file `/files/splunk/linux/SPLUNK_HOME/etc/system/local/server.conf`

**Limitations**

SSL only works with Linux worker nodes. The Universal Forwarders verify that the indexers they connect to have a certificate signed by the configured root CA and that the Common Name in the certificate presented by the indexer matches its FQDN as listed by the variable `splunk_architecture_forward_servers`.

**Prerequisites**

Configure your indexers to use SSL on port 9998. The following is an example `inputs.conf` file located in `$SPLUNK_HOME/etc/system/local` that enables SSL on port 9998 and configures the certificate file for use by the indexer itself, in this instance `/opt/splunk/etc/mycerts/indexer.pem`.

```plaintext
[splunktcp-ssl://9998]
disabled=0
collection_host = ip

[SSL]
serverCert=/opt/splunk/etc/mycerts/indexer.pem
#requireClientCert = true
#sslAltNameToCheck = forwarder,forwarder.cloudra.local

[tcp://1514]
collection_host = dns
sourcetype = ucp
```

For more information, see the documentation at [https://docs.splunk.com/Documentation/Splunk/7.1.2/Security/ConfigureSplunkforwardingtousesignedcertificates](https://docs.splunk.com/Documentation/Splunk/7.1.2/Security/ConfigureSplunkforwardingtusesignedcertificates). In addition, you can see how to create your own certificates and the content of the file designated with `serverCert` at [http://docs.splunk.com/Documentation/Splunk/7.1.2/Security/Howtoself-signcertificates](http://docs.splunk.com/Documentation/Splunk/7.1.2/Security/Howtoself-signcertificates).

In this instance, the folder `mycerts` was created under `/opt/splunk/etc` and the file `indexer.pem` was copied to this folder.

Indexers are configured with the Root CA cert used to sign all certificates. This can be achieved by editing the file `server.conf` in `$SPLUNK_HOME/etc/system/local` on your indexer(s). The following code block shows the relevant portion of this file where `ssoRootCaPath` is pointing to the root CA certificate.

```plaintext
[sslConfig]
sslRootCaPath = /opt/splunk/etc/mycerts/ca.pem
```

**Note**

In order to be able to download and install additional applications, you may want to append the file `$SPLUNK_HOME/auth/appsCA.pem` to your `ca.pem` file. If you don't do this, the Splunk UI will make this suggestion when you attempt to Find more apps.

Splunk should be restarted on the indexers if you had to make these changes (see the Splunk documentation for more information).
Before you deploy
Generate the forwarder certificate and name it `forwarder.pem`. Make sure that you copy the root CA certificate to `ca.pem`.

1. Copy both the `ca.pem` and the `forwarder.pem` files to `files/splunk/linux/SPLUNK_HOME/etc/mycerts/` (overwriting any existing files).
2. Edit the file `server.conf` in the folder `files/splunk/linux/SPLUNK_HOME/etc/system/local` and uncomment the last two lines as suggested in the file itself. Your file should look like this:

```
# uncomment the section below if you want to enable SSL
#
[sslConfig]
sslRootCAPath = /opt/splunkforwarder/etc/mycerts/ca.pem
```

3. Set `splunk_ssl` to `yes` in the file `group_vars/vars`, uncommenting the line if required. Make sure that the `splunk_architecture_forward_servers` list specifies all your indexers together with the port that was configured to accept SSL:

```
monitoring_stack: splunk
splunk_ssl: yes
# splunk_architecture_forward_servers:
  - indexer1.cloudra.local:9998
  - indexer2.cloudra.local:9998
```

Hybrid environment Linux / Windows
Currently, you cannot deploy your own certificates for use by the Universal Forwarders deployed on Windows machines. If you want to have your Linux machines in a hybrid deployment to use SSL, proceed as follows.

1. Comment out the `splunk_architecture_forward_servers` variable (and its values) from `group_vars/vars`:

```
monitoring_stack: splunk
splunk_ssl: yes
# splunk_architecture_forward_servers:
  - hpe2-ansible.cloudra.local:9998
```

2. Create a file named `vms.yml` in the folder `group_vars` and specify the list of forward servers to use by the Linux servers. This list is typically the same as the one used for Windows servers but specifies a TCP port that enables SSL:

```
splunk_architecture_forward_servers:
  - hpe2-ansible.cloudra.local:9998
```

3. Edit the `group_vars/win_worker.yml` file and specify the list of forward servers to be used by the Windows servers. This list is typically the same as the one used for Linux servers but specifies a TCP port that does not enable SSL:

```
splunk_architecture_forward_servers:
  - hpe2-ansible.cloudra.local:9997
```

Appendix D: How to check that certs were deployed correctly
The following commands should return the CA certificates used by UCP / DTR. This certificates is the same as the one pointed to by the `--cacert` switch:

```
# curl --cacert <ucp_certs_dir>/ca.pem https://<your ucp fqdn>/ca
# curl --cacert <dtr_certs_dir>/ca.pem https://<your dtr fqdn>/ca
```

**Output 1:** certificates successfully deployed (content will depend on your own CA certificate)
If the deployment was not successful, `curl` will output something like **Output 2**.

**Output 2**: certificates were not successfully deployed

curl: (60) Peer's Certificate issuer is not recognized.
More details here: http://curl.haxx.se/docs/sslcerts.html

---

**Enable certs for browser (Windows 2016 example)**

Choose **Manage computer certificates** in the control panel as shown in Figure 34.

![Figure 34. Manage computer certificates](image)

---
Import the `ca.pem` for UCP into the Trusted Root Certification Authorities, as shown in Figure 35.

![Figure 35. Import the capem](image)

It should now show up in the list of certificates. You may need to restart your browser to see the green, secure lock symbol as shown in Figure 36.

![Figure 36. Secure HTTPS](image)
Resources and additional links
HPE Reference Architectures, hpe.com/info/ra
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
Docker Reference Architectures, https://success.docker.com/architectures
Sysdig Resources, https://sysdig.com/resources/

To help us improve our documents, please provide feedback at hpe.com/contact/feedback.