[](http://www.hpe.com)HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity

**Deployment Guide**

Contents

[Executive Summary 5](#_Toc532824526)

[Solution overview 5](#_Toc532824527)

[New in this release 5](#_Toc532824528)

[Solution configuration 6](#_Toc532824529)

[High availability 6](#_Toc532824530)

[Sizing considerations 7](#_Toc532824531)

[Disaster Recovery 9](#_Toc532824532)

[Security 9](#_Toc532824533)

[Solution components 9](#_Toc532824534)

[Hardware 9](#_Toc532824535)

[Software 10](#_Toc532824536)

[Application software 11](#_Toc532824537)

[Preparing the environment 11](#_Toc532824538)

[Verify prerequisites 12](#_Toc532824539)

[Enable vSphere High Availability (HA) 12](#_Toc532824540)

[Install vSphere Docker Volume Service driver on all ESXi hosts 12](#_Toc532824541)

[Create the Ansible node 13](#_Toc532824542)

[Create the Red Hat Linux template 14](#_Toc532824543)

[Configuring the solution components 15](#_Toc532824544)

[Ansible configuration 15](#_Toc532824545)

[Editing the inventory 16](#_Toc532824546)

[VMware configuration 18](#_Toc532824547)

[HPE SimpliVity configuration 18](#_Toc532824548)

[HPE SimpliVity backup configuration 20](#_Toc532824549)

[Networking configuration 21](#_Toc532824550)

[Environment configuration 21](#_Toc532824551)

[Docker configuration 21](#_Toc532824552)

[Orchestrator configuration 22](#_Toc532824553)

[Kubernetes configuration 22](#_Toc532824554)

[Protecting sensitive information 23](#_Toc532824555)

[Inventory group variables 24](#_Toc532824556)

[Overview of the playbooks 25](#_Toc532824557)

[Core components 25](#_Toc532824558)

[Optional components 25](#_Toc532824559)

[Backup and restore playbooks 25](#_Toc532824560)

[Convenience playbooks 26](#_Toc532824561)

[Convenience scripts 26](#_Toc532824562)

[Deploying the core components 26](#_Toc532824563)

[Provisioning RHEL VMs 26](#_Toc532824564)

[Provisioning load balancers for UCP and DTR 26](#_Toc532824565)

[Installing Docker UCP and DTR on RHEL VMs 28](#_Toc532824566)

[Deploying RHEL workers 28](#_Toc532824567)

[HPE SimpliVity backup playbooks 29](#_Toc532824568)

[Deploying Windows workers 29](#_Toc532824569)

[Create the Windows Template 29](#_Toc532824570)

[Playbooks for adding Windows workers 30](#_Toc532824571)

[Windows configuration 31](#_Toc532824572)

[Deploying Windows workers behind a proxy 32](#_Toc532824573)

[Windows operating system and Docker EE 33](#_Toc532824574)

[Deploying Sysdig monitoring 33](#_Toc532824575)

[Monitoring with Sysdig 34](#_Toc532824576)

[Playbooks for installing Sysdig on RHEL 34](#_Toc532824577)

[Sysdig configuration 35](#_Toc532824578)

[Registering for Sysdig trial 36](#_Toc532824579)

[Deploying Sysdig monitoring on Kubernetes 38](#_Toc532824580)

[Deploying Sysdig monitoring on Docker Swarm 38](#_Toc532824581)

[Deploying Splunk 39](#_Toc532824582)

[Monitoring with Splunk 39](#_Toc532824583)

[Playbooks for installing Splunk 41](#_Toc532824584)

[Splunk configuration 41](#_Toc532824585)

[Accessing Splunk UI 44](#_Toc532824586)

[Redeploying Splunk demo 46](#_Toc532824587)

[Deploying Prometheus and Grafana 47](#_Toc532824588)

[Monitoring with Prometheus and Grafana 47](#_Toc532824589)

[Playbooks for installing Prometheus and Grafana 48](#_Toc532824590)

[Prometheus and Grafana configuration 48](#_Toc532824591)

[Accessing Grafana UI 48](#_Toc532824592)

[Post deployment 50](#_Toc532824593)

[Installing kubectl 50](#_Toc532824594)

[Kubernetes guestbook example with Redis 50](#_Toc532824595)

[Deploying the NFS provisioner for Kubernetes 56](#_Toc532824596)

[Backup and restore 59](#_Toc532824597)

[Backup and restore UCP and DTR 59](#_Toc532824598)

[HPE SimpliVity backups 66](#_Toc532824599)

[Solution lifecycle management 74](#_Toc532824600)

[HPE SimpliVity environment 74](#_Toc532824601)

[vSphere Docker Volume Service Plug-in 75](#_Toc532824602)

[Red Hat Enterprise Linux operating system 75](#_Toc532824603)

[Docker EE Environment 76](#_Toc532824604)

[Monitoring Tools 76](#_Toc532824605)

[Summary 76](#_Toc532824606)

[Appendix A: Software Licenses 77](#_Toc532824607)

[Appendix B: Using customer supplied certificates for UCP and DTR 77](#_Toc532824608)

[Generating and testing certificates 77](#_Toc532824609)

[Verify your certificates 80](#_Toc532824610)

[Appendix C: Enabling SSL between the universal forwarders and the Splunk indexers using your certificates 80](#_Toc532824611)

[Limitations 80](#_Toc532824612)

[Prerequisites 81](#_Toc532824613)

[Before you deploy 81](#_Toc532824614)

[Hybrid environment Linux / Windows 82](#_Toc532824615)

[Appendix D: How to check that certs were deployed correctly 83](#_Toc532824616)

[Resources and additional links 85](#_Toc532824617)

# Executive Summary

HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity 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 HPE SimpliVity infrastructure and combines it with Docker’s enterprise-grade container platform and popular open source tools, along with deployment and advisory services from HPE Pointnext.

HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity 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 HPE Enterprise Containers as a Service with Docker Enterprise Edition (EE). It shows 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 Express Containers with Docker Enterprise Edition on HPE SimpliVity solution consists of a set of Ansible playbooks that run on top of a VMware virtualization platform on HPE SimpliVity 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.

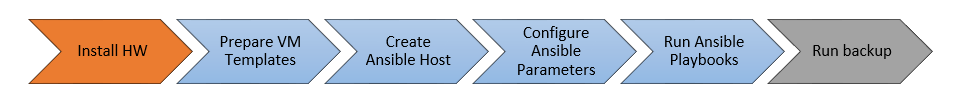


Figure 1. Solution overview

Figure 1 provides an overview of the steps used to deploy the solution. Deploying your hardware and HPE SimpliVity is specific to your environment and is not covered here. This document shows you how to:

* Prepare the VM templates
* Create the Ansible host
* Configure the Ansible parameters

Run the Ansible playbooks

Once you are up and running, you should regularly back up the system using the scripts provided as part of this solution.

## 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-SimpliVity/tree/master#release-notes>.

## Solution configuration

The Ansible playbooks are available to download at <https://github.com/HewlettPackard/Docker-SimpliVity>. By default, the playbooks are configured to set up a 3 node environment. This is the minimal starter configuration recommended by HPE and Docker for production.

### HPE SimpliVity configuration

The operational environment is comprised of three HPE SimpliVity 380 Gen10 servers. HPE recommends dual socket HPE SimpliVity systems with at least 14 CPU cores per socket (28 total cores per system) for optimal performance and support during HA failover scenarios. Since the HPE SimpliVity technology relies on VMware virtualization, the servers are managed using vCenter.

### 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. Where more than 3 physical nodes are available, the worker nodes will typically be separated onto the extra nodes. It is possible to specify that more than one worker node is 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

By default, two load balancers are deployed to increase availability of UCP and DTR and these are placed on separate physical nodes. Load balancing for applications running on worker nodes can achieved by using the playbooks to deploy additional load balancers, or by manually configuring the existing two to support your applications in addition to supporting UCP and DTR.

* 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 13 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](#_Monitoring_with_Sysdig)).

## High availability

Uptime is paramount for businesses implementing Docker containers in business critical environments. The HPE Express Containers with Docker Enterprise Edition on HPE SimpliVity solution 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 solution also deploys load balancers in the system to help with container traffic management. There are two load balancer VMs – the UCP load balancer and DTR load balancer. The playbooks can be configured to deploy one or more worker load balancers depending on the requirements of your applications. A typical load balancer architecture for applications running on Docker EE is shown in Figure 2. The playbooks now support load balancers based on VRRP, using HAproxy and keepalived. The solution can be deployed using these loadbalancers, or external load balancers, or no load balancers or the legacy version of standalone load balancers. For more information on HAproxy, see <http://www.haproxy.com/solutions/high-availability/>.

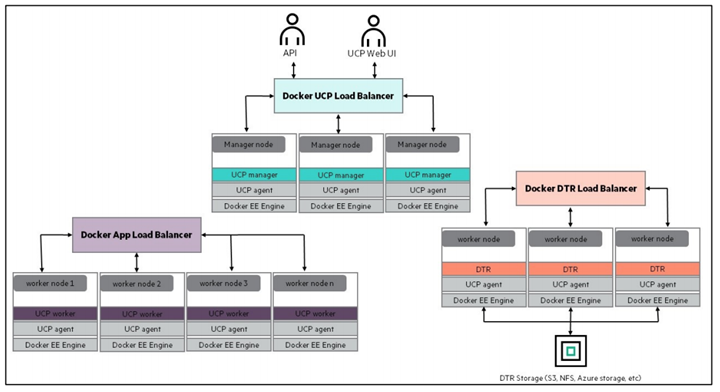


Figure 2. Load balancer architecture

## 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 fulfil the minimal requirements, for UCP 3.0, 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.

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.

Table 1. vCPU

|  |  |  |  |
| --- | --- | --- | --- |
| vCPUs | node01 | node02 | node03 |
| ucp1 | 4 |  |  |
| ucp2 |  | 4 |  |
| ucp3 |  |  | 4 |
| dtr1 | 2 |  |  |
| dtr2 |  | 2 |  |
| dtr3 |  |  | 2 |
| worker1 | 4 |  |  |
| worker2 |  | 4 |  |
| worker3 |  |  | 4 |
| win-worker1 | 4 |  |  |
| win-worker2 |  | 4 |  |
| win-worker3 |  |  | 4 |
| lb1 | 2 |  |  |
| lb2 |  | 2 |  |
| nfs |  |  | 2 |
| logger |  | 2 |  |
| Total vCPU per node | 16 | 18 | 16 |

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 Table 2.

Table 2. Memory allocation

|  |  |  |  |
| --- | --- | --- | --- |
| RAM (GB) | node01 | node02 | node03 |
| ucp1 | 16 |  |  |
| ucp2 |  | 16 |  |
| ucp3 |  |  | 16 |
| dtr1 | 16 |  |  |
| dtr2 |  | 16 |  |
| dtr3 |  |  | 16 |
| worker1 | 64 |  |  |
| worker2 |  | 64 |  |
| worker3 |  |  | 64 |
| win-worker1 | 64 |  |  |
| win-worker2 |  | 64 |  |
| win-worker3 |  |  | 64 |
| lb1 | 4 |  |  |
| lb2 |  | 4 |  |
| nfs |  |  | 4 |
| logger |  | 4 |  |
| Total RAM required (per node) | 164 | 168 | 164 |
| Available RAM | 384 | 384 | 384 |

Note

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

## 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](#_Backup_and_restore_1) for best practices, procedures and utilities for implementing disaster recovery.

## Security

The Docker Reference architecture for Securing Docker EE and Security Best Practices is available at <https://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices>

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](#_Monitoring_with_Sysdig)).

# Solution components

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

## Hardware

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

### About HPE SimpliVity

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

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

## Software

The software components used in this Reference Configuration are listed in Table 3 and Table 4.

Table 3. Third-party software

| Component | Version |
| --- | --- |
| Ansible | 2.7 |
| Docker EE | 2.0 with Docker EE Engine 17.06 (tested with UCP 3.0.6 and DTR 2.5.6) |
| Red Hat Enterprise Linux | 7.5, 7.6 |
| Microsoft Windows | Server 2016 |
| VMware | ESXi 6.5.0 and vCenter 6.5.0 |

Table 4. HPE Software

| Component | Version |
| --- | --- |
| HPE SimpliVity OmniStack | 3.7.6 |

### 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 pywinrm module which carries out actions via the Windows remote manager.

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

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

## 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 5.

Table 5. Application software

| Component | Version |
| --- | --- |
| Splunk | 7.1.2 |
| Sysdig | latest |
| Prometheus | V2.3.2 |
| Grafana | 5.2.3 |

### 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.

More information on configuring Splunk and running the relevant playbooks can be found in the section Deploying Splunk.

For more information on configuring Sysdig and running the relevant playbooks, see the section Deploying Sysdig monitoring.

### 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. See the section Deploying Prometheus and Grafana for more information on configuring and deploying the software.

# Preparing 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 yum repositories
* 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](#_Configuring_the_solution). A brief overview of the information required is presented in Table 6.

Table 6. Summary of information required

| Component | Details |
| --- | --- |
| Virtual Infrastructure | 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. |
| HPE SimpliVity Cluster | The name of the SimpliVity cluster and the names of the members of this cluster as they appear in vCenter. You will also need to know the name of the SimpliVity datastore where you want to land the various virtual machines. You may have to create this datastore if you just installed your SimpliVity cluster. In addition, you will need the IP addresses of the OmniStack virtual machines. Finally you will need credentials with admin capabilities for using the OmniStack API. These credentials are typically the same as your vCenter admin credentials. |
| L3 Network requirements | You will need one IP address for each and every VM configured in the Ansible inventory (see the section [Configuring the solution components](#_Configuring_the_solution)). The recommended minimal deployment (Linux-only) configures 13 virtual machines so you would need to allocate 13 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. |
| DNS | 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) |
| NTP Services | 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). |
| RHEL Subscription | A RHEL subscription is required to pull extra packages that are not on the DVD. |
| Docker Prerequisites | You will need a URL for the official Docker EE software download and a license file. Refer to the Docker documentation to learn more about this URL and the licensing requirements at: <https://docs.docker.com/engine/installation/linux/docker-ee/rhel/> in the section entitled “Docker EE repository URL” |
| Proxy | The playbooks pull the Docker packages from the Internet. If your environment accesses the Internet through a proxy, you will need the details of the proxy including the fully qualified domain name and the port number. |

## Enable vSphere High Availability (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.

Note

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

## Install vSphere Docker Volume Service driver on all ESXi hosts

vSphere Docker Volume Service technology enables stateful containers to access the storage volumes. 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>.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 is 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

1. Configure the following yum repositories, rhel-7-server-rpms and rhel-7-server-extras-rpms as explained in the section [Configure the yum repositories](#_Configure_the_yum). The "extras" repo can be enabled as follows:

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

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

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

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

1. Install Ansible 2.7 or higher.

# yum install ansible

1. 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 to your VM template using SSH with the credentials you configured for the root account and then 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.

# subscription-manager register --auto-attach

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

# subscription-manager config --server.proxy\_hostname=<proxy IP> --server.proxy\_port=<proxy port>

1. Verify that you don't have the issue described here: <https://access.redhat.com/solutions/3317671> by entering the following command.

# yum repolist

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

# 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.

[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/265523> and at <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 supported 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.

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.

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:

# rm /etc/ssh/ssh\_host\_\*  
# nmcli con del ens192  
# logrotate -f /etc/logrotate.conf  
# rm /var/log/\*-201?\*  
# history -c

1. Shutdown the VM

# shutdown -h now

1. 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.

# 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.

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

1. Change to the directory that you just cloned:

# cd ~/Docker-SimpliVity

Note

All subsequent file names are relative to the Docker-SimpliVity directory. For example vm\_hosts is located in ~/Docker-SimpliVity/ and group\_vars/vars corresponds to ~/Docker-SimpliVity/groups\_vars/vars.

You now need to prepare the configuration to match your own environment, prior to deploying Docker EE and the rest of the nodes. To do so, you will need to 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-SimpliVity 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.
* [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.

If you are deploying the new active-active load balancers, using floating IPs managed by keepalived:

[loadbalancer]: A group containing the UCP, DTR and any worker load balancers you are deploying.

If you are using the legacy, standalone load balancers:

* [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.

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.
* [loadbalancer:vars]: A set of variables defined for all nodes in the [loadbalancer] 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:

[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 7 below. They are all mandatory unless otherwise specified.

Table 7. Variables

| Variable | Scope | Description |
| --- | --- | --- |
| ip\_addr | Node | IP address in CIDR format to be given to a node |
| esxi\_host | Node | ESXi host where the node will be deployed. If the cluster is configured with DRS, this option will be overridden |
| cpus | Node/Group | Number of CPUs to assign to a VM or a group of VMs |
| ram | Node/Group | Amount of RAM in MB to assign to a VM or a group of VMs |
| disk2\_usage | Node/Group | Size of the second disk in GB to attach to a VM or a group of VMs. This variable is only mandatory on Docker nodes (UCP, DTR, worker) and NFS node. It is not required for the logger node or the load balancers. |
| node\_policy | Node/Group | HPE SimpliVity backup policy to assign to a VM or a group of VMs. The name has to match one of the backup policies defined in the group\_vars/vars file described in the section HPE SimpliVity backup configuration. |

## VMware configuration

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

Table 8. VMware variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| vcenter\_hostname | group\_vars/vars | IP or hostname of the vCenter appliance |
| vcenter\_username | group\_vars/vars | Username to log in to the vCenter appliance. It might include a domain, for example, 'administrator@vsphere.local'. |
| vcenter\_password | group\_vars/vault | The password corresponding to the vcenter\_username user above. |
| vcenter\_validate\_certs | group\_vars/vars | ‘no’ |
| datacenter | group\_vars/vars | Name of the datacenter where the environment will be provisioned |
| vm\_username | group\_vars/vars | 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'. |
| vm\_password | group\_vars/vault | The password for the vm\_username user above. |
| vm\_template | group\_vars/vars | Name of the RHEL VM Template to be use. Note that this is the name from a vCenter perspective, not the hostname. |
| folder\_name | group\_vars/vars | 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. |
| datastores | group\_vars/vars | 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.  Please note that from a HPE SimpliVity perspective, it is a best practice to use only one Datastore. Using more than one will not provide any advantages in terms of reliability and will add additional complexity. |
| disk2 | group\_vars/vars | UNIX® name of the second disk for the Docker VMs. Typically /dev/sdb |
| disk2\_part | group\_vars/vars | UNIX name of the partition of the second disk for the Docker VMs. Typically /dev/sdb1 |
| vsphere\_plugin\_version | group\_vars/vars | 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. |
| vm\_portgroup | group\_vars/vars | 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. |

## HPE SimpliVity configuration

Variables related to your HPE SimpliVity deployment are mandatory and are described in Table 9.

Table 9. SimpliVity variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| simplivity\_username | group\_vars/vars | Username to log in to the SimpliVity Omnistack appliances. It might include a domain, for example, [administrator@vsphere.local](mailto:administrator@vsphere.local).Note: The corresponding password is stored in the variable named simplivity\_password. |
| simplivity\_password | group\_vars/vault | The password for the simplivity\_username user. |
| omnistack\_ovc | group\_vars/vars | List of Omnistack hosts to be used, in list format, i.e. [‘omni1.local’,’onmi2.local’...] If your OmniStack virtual machines do not have their names registered in DNS, you can use their IP addresses. |

### VM placement and number of HPE SimpliVity servers in the cluster

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

1. If DRS is not enabled, the placement of the VMs is specified in the Ansible inventory file vm\_hosts

If DRS is enabled, the placement of the VMs is outside the control of the playbooks

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

#### Using more than three nodes when DRS is not enabled

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

[ucp]   
hpe-ucp01 ip\_addr='10.10.174.112/22' esxi\_host='simply01.am2.cloudra.local'  
hpe-ucp02 ip\_addr='10.10.174.113/22' esxi\_host='simply02.am2.cloudra.local'  
hpe-ucp03 ip\_addr='10.10.174.114/22' esxi\_host='simply03.am2.cloudra.local'

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

The default vm\_hosts inventory configures three Docker worker nodes and distributes them across the three ESX hosts:

[worker]   
hpe-worker01 ip\_addr='10.10.174.122/22' esxi\_host='simply01.am2.cloudra.local'  
hpe-worker02 ip\_addr='10.10.174.123/22' esxi\_host='simply02.am2.cloudra.local'  
hpe-worker03 ip\_addr='10.10.174.124/22' esxi\_host='simply03.am2.cloudra.local'

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

[worker]   
hpe-worker01 ip\_addr='10.10.174.122/22' esxi\_host='simply01.am2.cloudra.local'  
hpe-worker02 ip\_addr='10.10.174.123/22' esxi\_host='simply02.am2.cloudra.local'  
hpe-worker03 ip\_addr='10.10.174.124/22' esxi\_host='simply03.am2.cloudra.local'  
hpe-worker04 ip\_addr='10.10.174.150/22' esxi\_host='simply04.am2.cloudra.local'

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

[nfs]  
hpe-nfs ip\_addr='10.10.174.121/22' esxi\_host='simply03.am2.cloudra.local'

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

[nfs]   
hpe-nfs ip\_addr='10.10.174.121/22' esxi\_host='simply04.am2.cloudra.local'

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

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

Note

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

#### Using more than three nodes when DRS is enabled

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

Warning

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

## HPE SimpliVity backup configuration

Variables related to HPE SimpliVity backups are described in Table 10.

Table 10. HPE SimpliVity backup variables

|  |  |
| --- | --- |
| Variable | Description |
| backup\_policies | List of dictionaries containing the different backup policies to be used along with the scheduling information. Any number of backup policies can be created and they need to match the node\_policy variables defined in the inventory. Times are indicated in minutes. All month calculations use a 30-day month. All year calculations use a 365-day year. The format is as follows:  backup\_policies:  - name: daily'  days: 'All'  start\_time: '11:30'  frequency: '1440'  retention: '43200'  - name: 'hourly'  days: 'All'  start\_time: '00:00'  frequency: '60'  retention: '2880' |
| dummy\_vm\_prefix | In order to be able to back up the Docker volumes, a number of “dummy” VMs need to spin up. This variable will set a recognizable prefix for them. |
| docker\_volumes\_policy | Backup policy to use for the Docker Volumes. |

## Networking configuration

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

Table 11. Network variables

| Variable | File | Description |
| --- | --- | --- |
| nic\_name | group\_vars/vars | Name of the device, for RHEL this is typically ens192 and it is recommended to leave it as is. |
| gateway | group\_vars/vars | IP address of the gateway to be used |
| dns | group\_vars/vars | List of DNS servers to be used, in list format, i.e. ['10.10.173.1','10.10.173.2'...] |
| domain\_name | group\_vars/vars | Domain name for your Virtual Machines |
| ntp\_server | group\_vars/vars | List of NTP servers to be used, in list format, i.e. ['1.2.3.4','[0.us.pool.net.org](http://0.us.pool.net.org/)'...] |

## Environment configuration

All Environment-related variables are described in Table 12 below.

Table 12. Environment variables

| Variable | File | Description |
| --- | --- | --- |
| 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 13.

Table 13. Docker variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| docker\_ee\_url | group\_vars/vault | 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: <https://docs.docker.com/engine/installation/linux/docker-ee/rhel/>. |
| docker\_ee\_reponame | group\_vars/vars | For Docker EE 2.0, this variable must be set to the value stable-17.06 |
| rhel\_version | group\_vars/vars | For the Docker installation, this sets the version of your RHEL OS, such as 7.5. The playbooks were tested with RHEL 7.5. |
| dtr\_version | group\_vars/vars | 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 2.5.6. |
| ucp\_version | group\_vars/vars | 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 3.0.6. |
| images\_folder | group\_vars/vars | Directory in the NFS server that will be mounted in the DTR nodes and that will host your Docker images. |
| license\_file | group\_vars/vars | Full path to your Docker EE license file on your Ansible host. The license file is available from the Docker Store |
| ucp\_username | group\_vars/vars | Username of the administrator user for UCP and DTR, typically admin. |
| ucp\_password | group\_vars/vault | The password for the ucp\_username account. |
| docker\_storage\_driver | group\_vars/vars | Storage driver for Docker nodes. Accepted values are overlay2 (the default) and devicemapper. |

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.

## WORKER  
[worker]  
hpe-worker01 ip\_addr='10.60.59.21/16' esxi\_host='esxi-hpe-1.cloudra.local'  
hpe-worker02 ip\_addr='10.60.59.22/16' esxi\_host='esxi-hpe-2.cloudra.local'  
hpe-worker03 ip\_addr='10.60.59.23/16' esxi\_host='esxi-hpe-3.cloudra.local' **orchestrator=swarm**

[worker: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/%23set-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.

## 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 a 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:

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 14.

Table 14. Kubernetes Persistent Volume variables

| Variable | File | Description |
| --- | --- | --- |
| nfs\_provisioner\_role | group\_vars/vars | Name of the role to create, for example, nfs-provisioner-runner |
| nfs\_provisioner\_name | group\_vars/vars | Name of the provisioner, for example, hpe.com/nfs |
| nfs\_provisioner\_storage\_class\_name | group\_vars/vars | Name of the storage class to create, for example, nfs |
| nfs\_provisioner\_server\_ip | group\_vars/vars | IP address (or FQDN) of your external NFS server, for example, hpe-nfs.cloudra.local |
| nfs\_provisioner\_server\_share | group\_vars/vars | Name of the NFS share where all the persistent volume data will be stored, for example, /k8s |

#### 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.

## 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:

---  
docker\_ee\_url: 'your\_url\_here'  
vcenter\_password: 'xxxx'  
vm\_password: 'xxxx'  
ucp\_password: 'zzzz'  
win\_password: 'yourpass'  
sysdig\_access\_key: 'enter\_sysdig\_access\_key'  
rhn\_orgid: "YourOrgId"  
rhn\_key: "YourActivationKey"  
#password for the splunk universal forwarder. Must meet password complexity requirement 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:

* Change the permissions so only root can read it using # chmod 600 .vault\_pass

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.

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 disk 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 an 800GB drive for Kubernetes persistent volumes data.

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-SimpliVity>. Once you have cloned the repository, change directory to /root/Docker-SimpliVity.

You can use the playbook site.yml as the day 0 playbook to deploy the solution. It is simply a wrapper around a number of required and optional playbooks that allow you to configure the deployment to your needs.

To start a deployment, use the following command:

# ansible-playbook -i vm\_hosts site.yml --vault-password-file .vault\_pass

The playbooks should run for approximately 35-40 minutes for the default deployment with 3 UCP, 3 DTR and 3 Linux worker nodes (depending on your server specifications and the size of your environment).

## Core components

The playbooks for deploying the core components are described in the following sections:

* Provisioning RHEL VMs
* Provisioning load balancers for UCP and DTR
* Installing Docker UCP and DTR on RHEL VMs

Deploying RHEL workers

## Optional components

The playbooks for deploying optional components are described in the following sections:

* Playbooks for adding Windows workers
* Playbooks for installing Sysdig on RHEL
* Playbooks for installing Splunk

Playbooks for installing Prometheus and Grafana

## Backup and restore playbooks

Best practices and procedures are described in the section [Backup and restore](#_Backup_and_restore_1). 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.

# Deploying the core components

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-SimpliVity

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](#_Backup_and_restore_1).

## Provisioning RHEL VMs

The following playbooks are used to provision RHEL VMs:

* 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.
* 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](#_Create_the_Red), you can keep this playbook commented out.

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.

## Provisioning load balancers for UCP and DTR

The playbook playbooks/loadbalancer.yml is used to deploy load balancers in an active-active configuration to provide highly-available access to UCP and DTR.

At least two nodes are specified in the [loadbalancer] group in the inventory, along with group variables defining CPU and RAM requirements. These nodes run keepalived and HAproxy.

[loadbalancer]  
hpe-lb1 ip\_addr='10.10.174.248/22' esxi\_host='simply04.am2.cloudra.local' ucp=true  
hpe-lb2 ip\_addr='10.10.174.249/22' esxi\_host='simply05.am2.cloudra.local' dtr=true  
  
[loadbalancer:vars]  
cpus='2'  
ram='4096'  
node\_policy='hpe-bronze'

The virtual IP for UCP will be handled by hpe-lb1 by default, which will split the traffic across the three UCP VMs hpe-ucp01, hpe-ucp02 and hpe-ucp03. In the case of a failure of hpe-lb1, the virtual IP for UCP will automatically move to the second load balancer node hpe-lb2 which will again distribute the traffic to the UCP VMs.

Similarly, the virtual IP for DTR will be handled by default by the load balancer hpe-lb2, splitting the traffic across the three DTR VMs hpe-dtr01, hpe-dtr02 and hpe-dtr03. In the case of a failure of hpe-lb2, the virtual IP for DTR will automatically move to the first load balancer node hpe-lb1 which will again distribute the traffic to the DTR VMs.

To configure the virtual IPs for UCP and DTR, you need to add a loadbalancers dictionary to your group\_vars/vars file as shown in the excerpt below:

loadbalancers:  
 ucp:  
 public\_interface: 'ens192'  
 public\_vip: '10.60.59.251'  
 public\_fqdn: hpe-ucpvip.cloudra.local  
 virtual\_router\_id: 54  
 dtr:  
 public\_interface: 'ens192'  
 public\_vip: '10.60.59.252'  
 public\_fqdn: hpe-dtrvip.cloudra.local  
 virtual\_router\_id: 55

Warning

If you re-run playbooks/loadbalancer.yml after a configuration change, you may need to subsequently run playbooks/reconfigure\_dtr.yml as the latter playbook configures the virtual IP address for accessing the UCP Single-Sign-On (SSO) page. If there is no virtual IP or FQDN defined for UCP in the variables file, the playbook will choose the address of the first UCP node in the [ucp] group. This scenario introduces a single point of failure and should be avoided.

Note

By default, the playbook supports ports 443 and 6443 for UCP and port 433 for DTR. If you deploy Prometheus and Grafana on Docker Swarm, the Grafana port 3000 will be handled as well.

Note

The playbook playbooks/loadbalancer.yml can be used to create one or more load balancers for applications running on your worker nodes. However, it is impossible for the playbooks to know what ports to support, so manual configuration of HAproxy and keepalived may be required. By default, the playbooks support ports 80 and 443 for worker nodes

### Legacy stand-alone load balancers

The playbook playbooks/install\_haproxy.yml is used to deploy three separate load balancers, for the UCP, DTR and worker nodes. It is recommended that you use the HAproxy and keepalived solution documented above instead of this option.

### Deploying without load balancers

If you do not want to deploy load balancers when running site.yml, you should comment out any declarations in the inventory and variables files. This includes any legacy stand-alone load balancers.

### Deploying with your own load balancers

If you are using external load balancers for UCP and DTR, you can configure UCP and DTR to use these external load balancers by specifying FQDNs in the loadbalancers dictionary in group\_vars/vars:

loadbalancers:  
 ucp:  
 public\_fqdn: external-ucpvip.am2.cloudra.local  
 dtr:  
 public\_fqdn: external-dtrvip.am2.cloudra.local

## Installing Docker UCP and DTR on RHEL VMs

The following playbooks are used to install Docker UCP and DTR on RHEL VMs.

* playbooks/config\_storage\_driver.yml prepares drives for local Docker volumes and container images. It also configures Docker with either the overlay2 storage driver (the default) or the devicemapper storage driver, depending on the value of the docker\_storage\_driver variable in group\_vars/vars. This playbook was previously called playbooks/config\_docker\_lvs.yml in earlier releases of the solution.
* 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/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. The 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\_ucp.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\_ucp.yml installs and configures additional instances of UCP on the target nodes 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 node defined by the group dtr\_main in the vm\_hosts 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.

## Deploying RHEL workers

By default, site.yml will automatically deploy any RHEL (and / or Windows) worker nodes that are declared in the inventory.

If you subsequently want additional RHEL worker nodes, add them to the inventory as appropriate and then run the playbooks for [Provisioning RHEL VMs](#_Provisioning_RHEL_VMs), followed by the specific playbooks for RHEL worker nodes outlined below:

* playbooks/scale\_workers.yml installs and configures additional Linux workers on the target nodes defined by the group worker in the vm\_hosts inventory.

A utility script scale\_worker.sh is provided to assist you in adding worker nodes after the initial deployment.

## HPE SimpliVity backup playbooks

Two playbooks are provided to support the backup of Docker volumes using HPE SimpliVity functionality. The playbooks are run by default when using site.yml to deploy the solution.

### Configure dummy VMs to backup Docker volumes

The playbook playbooks/config\_dummy\_vms\_for\_docker\_volumes\_backup.yml ensures that you can back up Docker volumes that have been created using the vSphere plugin (vDVS) in SimpliVity. There is not a straight-forward way to do this, so you need to use a workaround. Since all Docker volumes are going to be stored in the dockvols folder in the datastore(s), you need to create a ‘dummy’ VM per datastore. The vmx, vmsd and vmkd files from this VM will have to be inside the dockvols folder, so when these VMs are backed up, the volumes are backed up as well. Obviously these VMs don’t need to take any resources and you can keep them powered off.

### Configure SimpliVity backups

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

# Deploying Windows workers

The site.yml playbook will automatically deploy any Windows workers declared in the inventory. The playbooks should run for approximately 70-80 minutes with 3 Windows workers added to the default deployment (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.

This section describes the functionality and configuration of the Windows-specific playbooks. It also details how to create the initial Windows template and how to manage deploying Windows worker nodes behind a proxy.

## Create the Windows Template

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 the 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, then enter a password and 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.

## Playbooks for adding Windows workers

* playbooks/create\_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. Windows workers nodes are defined in the group win\_worker in the vm\_hosts inventory.
* playbooks/install\_docker.yml installs Docker along with all its dependencies on your Windows VMs
* playbooks/scale\_workers.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.

## Windows configuration

Window-related variables are shown in Table 15.

Table 15. Windows variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| win\_vm\_template | group\_vars/vars | Name of the Windows 2016 VM Template to use. Note that this is the name from a vCenter perspective, not the hostname. |
| win\_username | group\_vars/vars | Windows user name. The default is Administrator |
| win\_password | group\_vars/vault | The password for the Windows account. |
| windows\_vdvs\_ps | group\_vars/vars | Variable used to download the PowerShell script that is used to install vDVS for Windows. For example, https://raw.githubusercontent.com/vmware/vsphere-storage-for-docker/master/install-vdvs.ps1 |
| windows\_vdvs\_path | group\_vars/vars | 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 <windows\_vdvs\_path>\_<windows\_vdvs\_version>.zip to download the software. For example, to download version 0.21, set windows\_vdvs\_path equal to https://vmware.bintray.com/vDVS/vsphere-storage-for-docker\_windows and windows\_vdvs\_version equal to 0.21 |
| windows\_vdvs\_version | group\_vars/vars | Combined with windows\_vdvs\_path, this variable is used to generate the URL for downloading the software. |
| windows\_vdvs\_directory | group\_vars/vars | Variable used to determine where vDVS software will be unzipped and installed from. The default is C:\Users\Administrator\Downloads |
| windows\_winrm\_script | group\_vars/vars | Variable used to determine where the winrm Powershell script will be downloaded from. See the section [Deploying Windows workers behind a proxy](#_Deploying_Windows_workers) for more information. |
| docker\_ee\_version\_windows | group\_vars/vars | It is important that the version of the Docker engine running on your Windows worker nodes is the same as that running on RHEL in the rest of your cluster. You should use this variable to explicitly match up the versions. For Docker 2.0, the recommended value is '17.06'. If you do not explicitly set this value, you may end up with an incompatible newer version running on your Windows workers. |
| windows\_update | group\_vars/vars | Variable used to determine if Windows updates are automatically downloaded when installing Docker on Windows worker nodes (in the playbooks/install\_docker.yml). Defaults to true. See the section [Deploying Windows workers behind a proxy](#_Deploying_Windows_workers) for more information. |

### 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 16 for the sake of completeness.

Table 16. Advanced windows variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| ansible\_user | group\_vars/win\_worker.yml | Defaults to the Windows user account win\_username as specified in group\_vars/vars |
| ansible\_password | group\_vars/win\_worker.yml | Defaults to the Windows user password win\_password as specified in group\_vars/vault |
| ansible\_port | group\_vars/win\_worker.yml | 5986 |
| ansible\_connection | group\_vars/win\_worker.yml | winrm |
| ansible\_winrm\_server\_cert\_validation | group\_vars/win\_worker.yml | Defaults to ignore |
| ansible\_winrm\_operation\_timeout\_sec | group\_vars/win\_worker.yml | Defaults to 250 |
| ansible\_winrm\_read\_timeout\_sec | group\_vars/win\_worker.yml | Defaults to 300 |
| windows\_timezone | group\_vars/win\_worker.yml | Defaults to 15 |

## Deploying Windows workers behind a proxy

If you are deploying behind a proxy, you will need to make a local copy of the remoting script available.

### Configuring the winrm remoting script

The playbooks for deploying Windows workers rely on a Powershell script for remote access from the Ansible machine. The script ConfigureRemotingForAnsible.ps1 is available online on GitHub and you can configure the playbooks to use the script directly as follows:

windows\_winrm\_script: 'https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1'

However, if the deployment is running behind a proxy, you may need to make this script available locally.

1. Download the script:  
     
   wget https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1
2. Deploy a local HTTP server, enabling port 80, for example:  
     
   yum install httpd  
   systemctl enable httpd  
   systemctl start httpd  
   firewall-cmd --permanent --add-port 80/tcp --zone=pubpic  
   firewall-cmd –reload
3. Copy the downloaded script to the web server:  
     
   cp ConfigureRemotingForAnsible.ps1 /var/www/html

Configure the variable to point at the local web server, for example,  
  
windows\_winrm\_script: 'http://10.10.174.230/ConfigureRemotingForAnsible.ps1'

## Windows operating system and Docker EE

Docker Enterprise Edition for Windows Server (Docker EE) enables native Docker containers on Windows Server. This solution has been tested with Windows worker nodes running Windows Server 2016 and with Docker EE 17.06. More recent versions of Windows Server may work but have not been tested.

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](https://docs.docker.com/install/windows/docker-ee/).

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](https://docs.docker.com/install/windows/docker-ee/#update-docker-ee).

# 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>.

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

## 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 3 provides an overview of the Sysdig architecture.

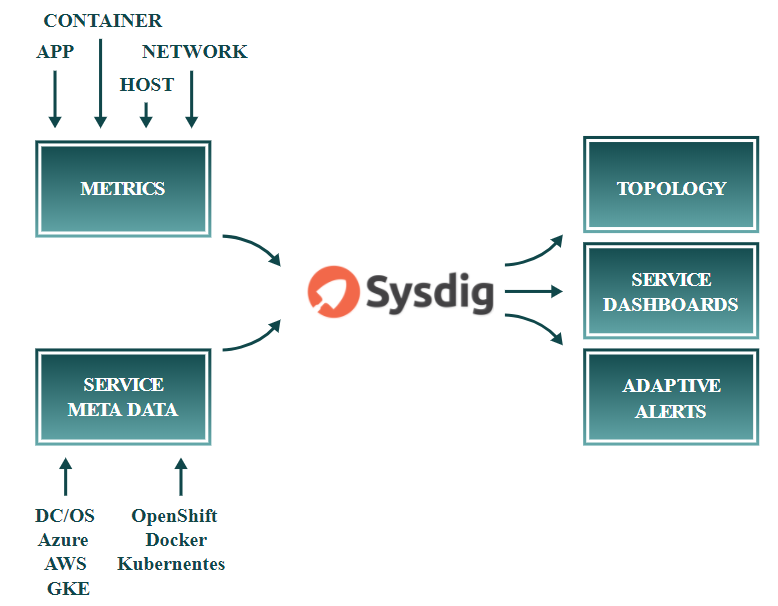


Figure 3. Sysdig architecture

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](https://www.youtube.com/watch?v=NR9XLZw0ndo&t=5s) video overview and the [Sysdig Monitor 101](https://sysdig.teachable.com/p/sysdig-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](https://www.youtube.com/watch?v=e_kdjHjK7mY) video overview and the [Sysdig Secure 101](https://sysdig.teachable.com/p/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](#_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-SimpliVity>.

## Playbooks for installing Sysdig on RHEL

The following playbooks are used when deploying Sysdig:

* 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.

playbooks/k8s-install-sysdig.yml is used to configure Sysdig for Kubernetes.

## 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 17.

Table 17. Sysdig variables for Docker swarm

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| sysdig\_access\_key | group\_vars/vault | 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. |
| sysdig\_agent | group\_vars/vars | Specifies the URL to the Sysdig Linux native install agent, for example, https://s3.amazonaws.com/download.draios.com/stable/install-agent |
| sysdig\_tags | group\_vars/vars | 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:Express Containers,owner:Customer Name' |

### 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 18.

Table 18. Sysdig variables for Kubernetes

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| sysdig\_access\_key | group\_vars/vault | 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. |
| sysdig\_restricted\_control\_role | group\_vars/vars | The Sysdig service account requires the 'Restricted Control' grant. So this value will typically be set as follows: sysdig\_restricted\_control\_role: 'Restricted Control' |
| k8s\_cluster | group\_vars/vars | This should match the cluster name displayed when you source the environment setup script, for example.  # source env.sh Cluster "ucp\_hpe-ucp.cloudra.local:6443\_admin" set. User "ucp\_hpe-ucp.cloudra.local:6443\_admin" set.  For more information, see the section on installing the UCP client bundle in the section Deploying Sysdig monitoring on Kubernetes. |

## 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 Express Containers with Docker Enterprise Edition on HPE SimpliVity solution. For more details on how to sign up, see the GitHub repository at <https://github.com/HewlettPackard/Docker-SimpliVity>.

After registering for the trial, you will be presented with options for setting up your environment, as shown in Figure 4.

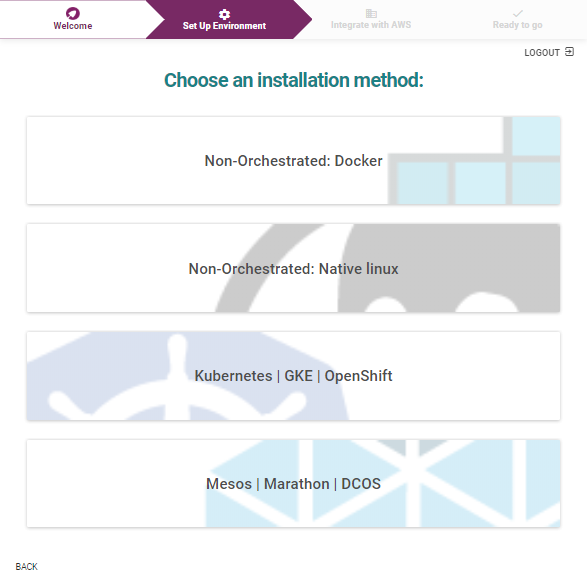


Figure 4. Sysdig Monitor set up environment

### 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 5.

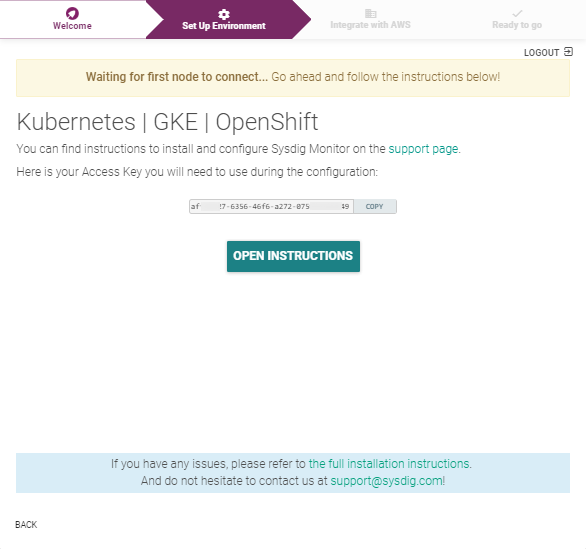


Figure 5. Sysdig Monitor access code for Kubernetes

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 6.

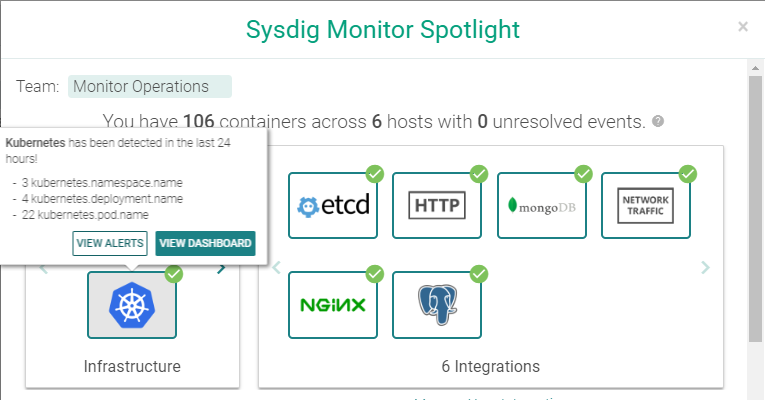


Figure 6. Sysdig Monitor Spotlight for Kubernetes

Select View Dashboard for an entry point to accessing all your monitoring data. Alternatively, you can browse to <https://app.sysdigcloud.com> 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 7.

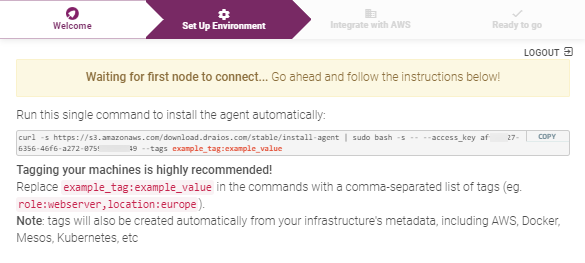


Figure 7. Sysdig Monitor download location and access code for Docker

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 relevant variables in the group\_vars/vars file.

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.

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-SimpliVity  
# 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 17.

# cd Docker-SimpliVity  
# 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.

# Deploying Splunk

This section provides an overview of Splunk, outlines how to configure and run the relevant playbooks and shows how to access the UI to see the resultant Docker and Kubernetes dashboards.

## 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. The solution architecture for Splunk is shown in Figure 8.

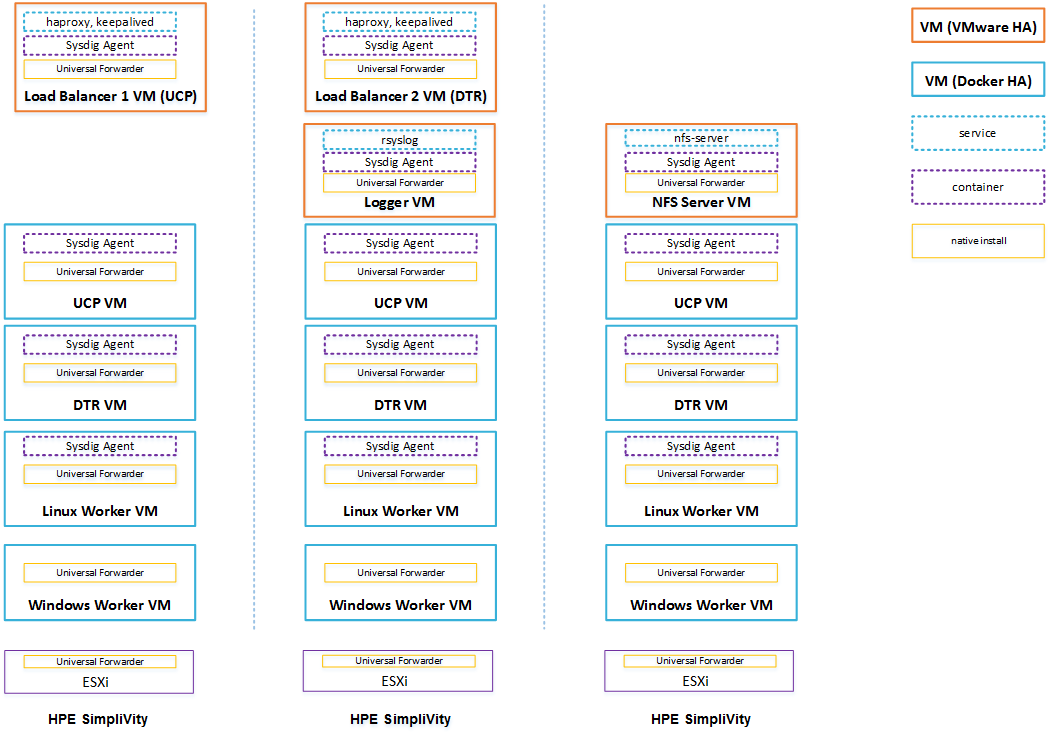


Figure 8. Solution architecture: Hybrid Linux and Windows workers with Splunk and Sysdig

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 9 shows the Splunk architecture.

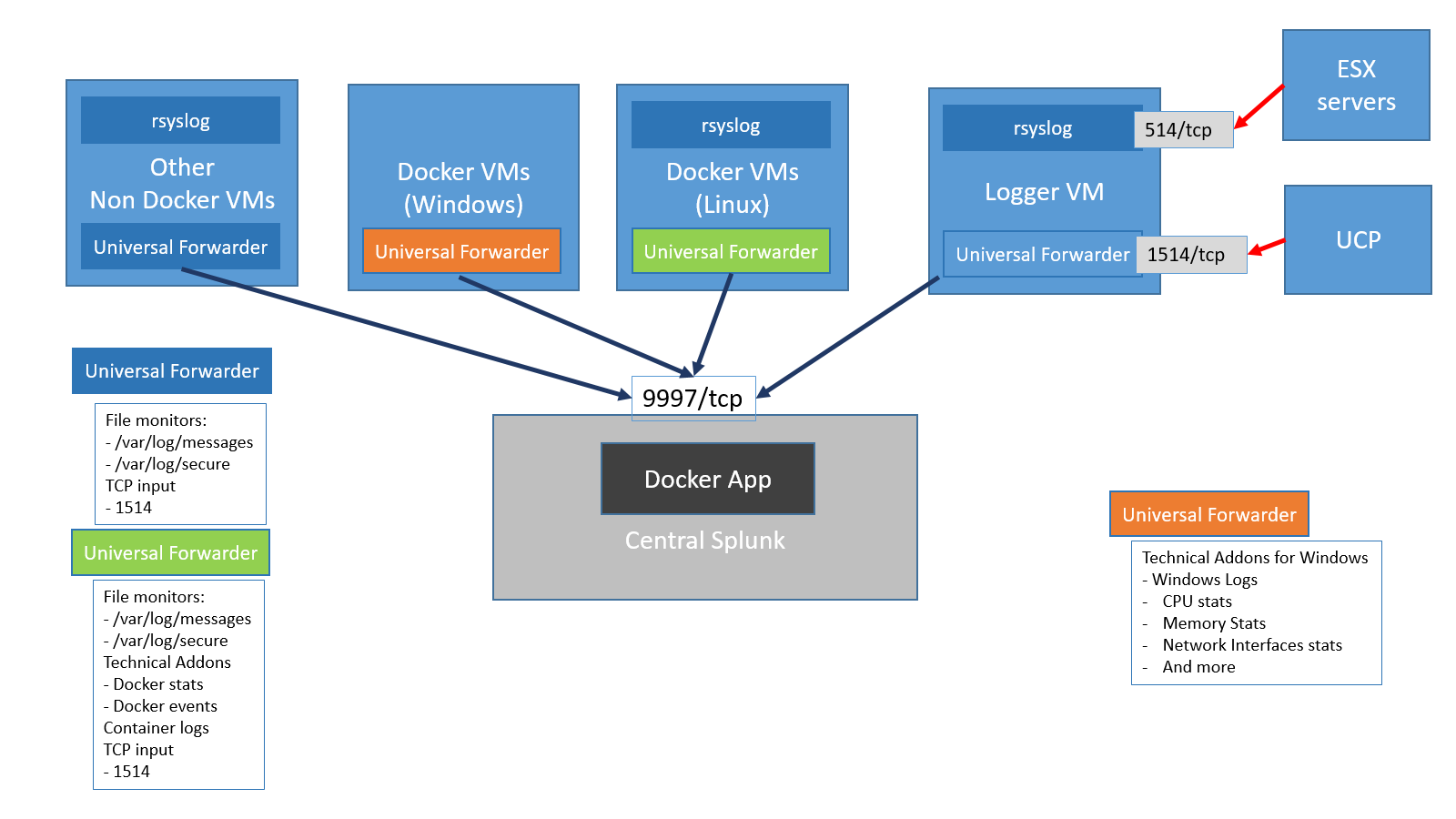


Figure 9. Splunk architecture

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](#_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.

## Playbooks for installing Splunk

The following playbooks are used to install Splunk.

* playbooks/splunk\_demo.yml installs a demo of Splunk Enterprise in the cluster (if the splunk\_demo deployment option is selected. A value of splunk is used to configure an external production Splunk deployment.)

playbooks/splunk\_uf.yml installs and configures the Splunk Universal Forwarder on each Linux and Windows node in the inventory

## 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 demo 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/Configureforwardingwithoutputs.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 10.

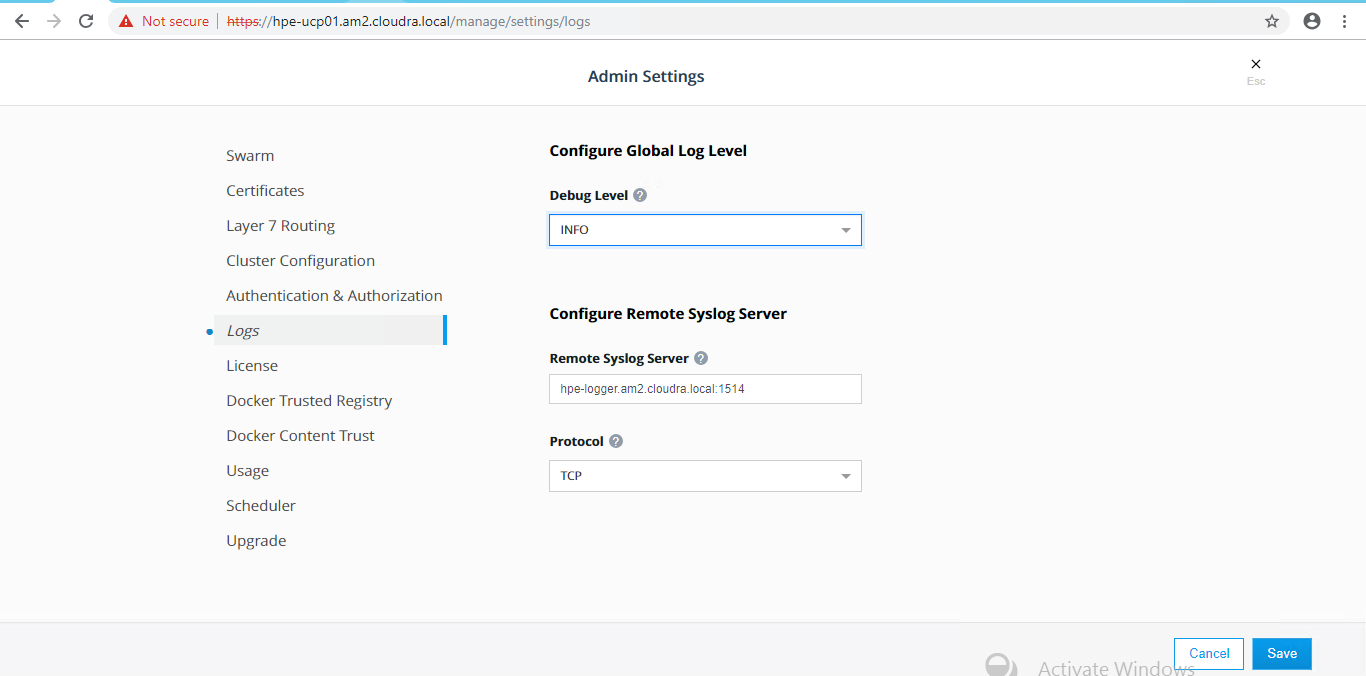


Figure 10. Configure Remote Syslog Server in UCP

### 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 11.

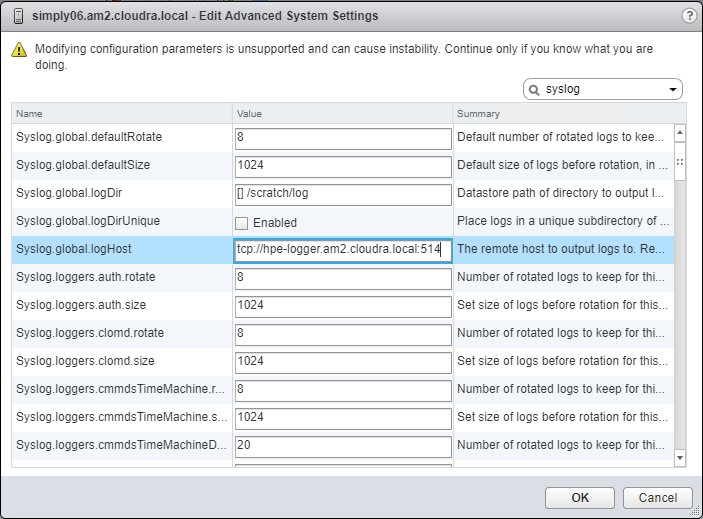


Figure 11. Configure Syslog on ESXi Hosts

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>.

### 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

## Accessing Splunk UI

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. For example:

http://hpe-ucp01.am2.cloudra.local:8000/

The default username and password for Splunk is admin / changeme.

Use the Docker App to view the Docker overview as shown in Figure 12 and the Docker stats as shown in Figure 13.

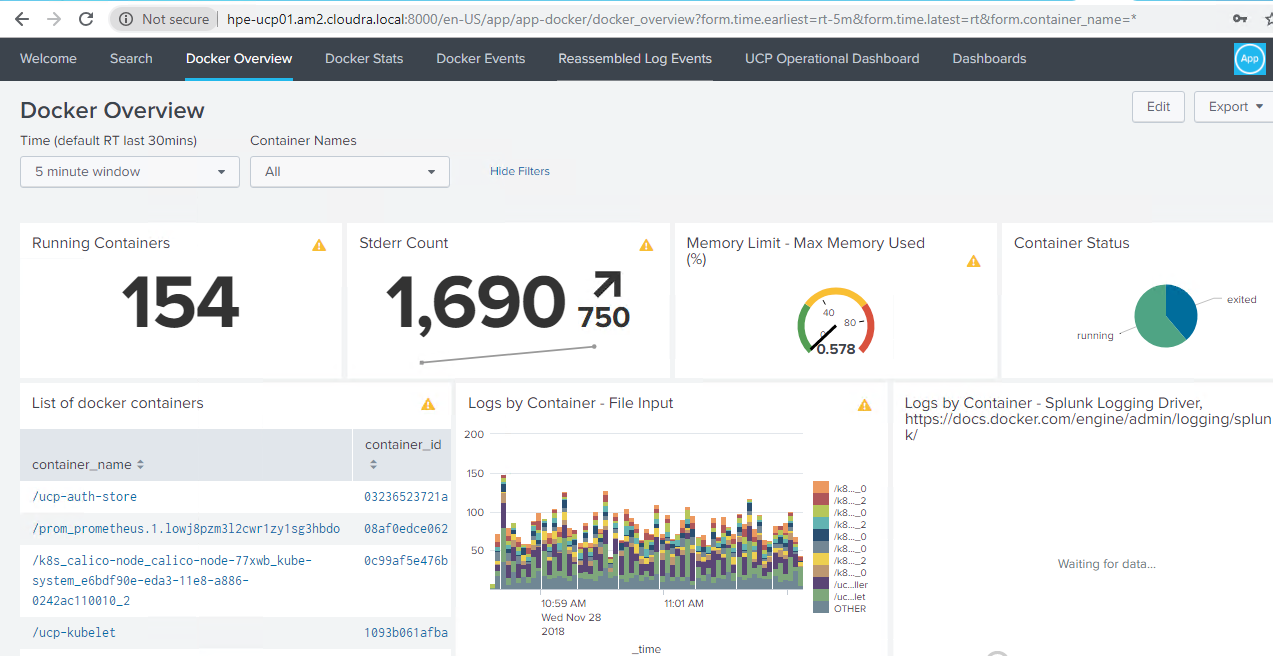


Figure 12. Docker overview

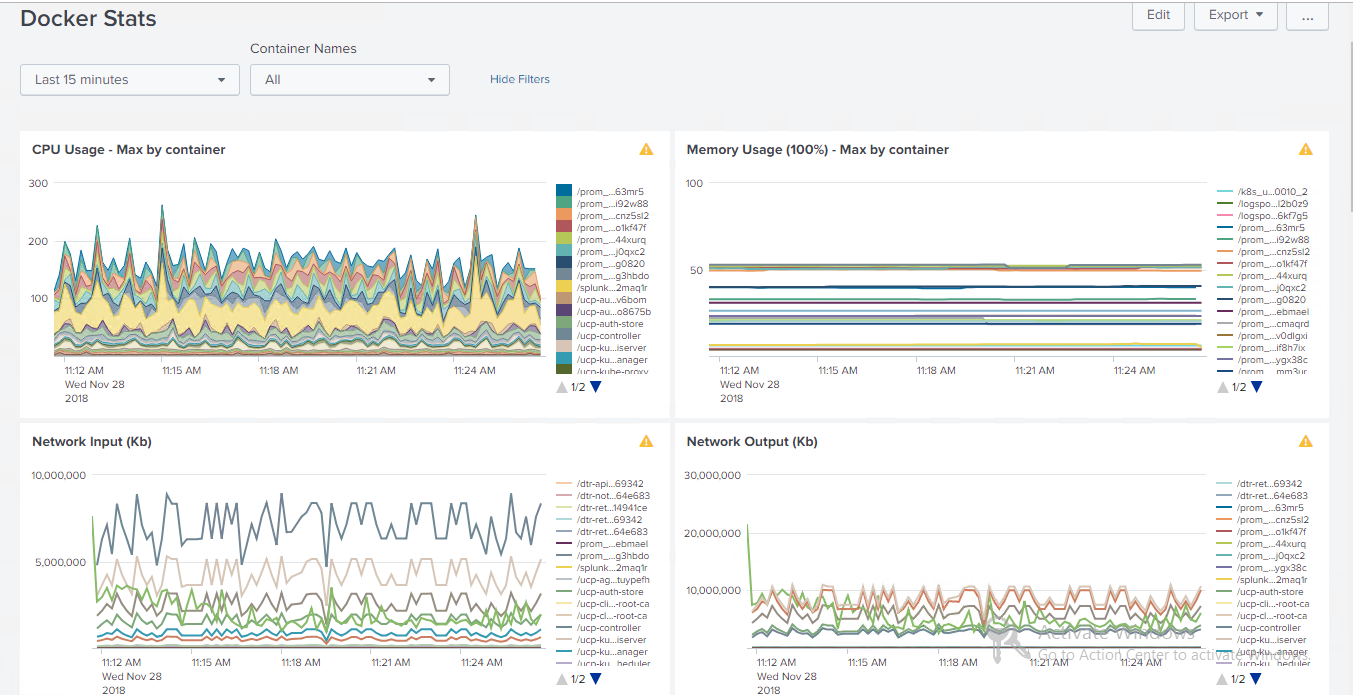


Figure 13. Docker stats

Use the k8s App to see the Kubernetes overview as shown in Figure 14 and then access the details for deployments, daemon sets, replica sets, services, etc.

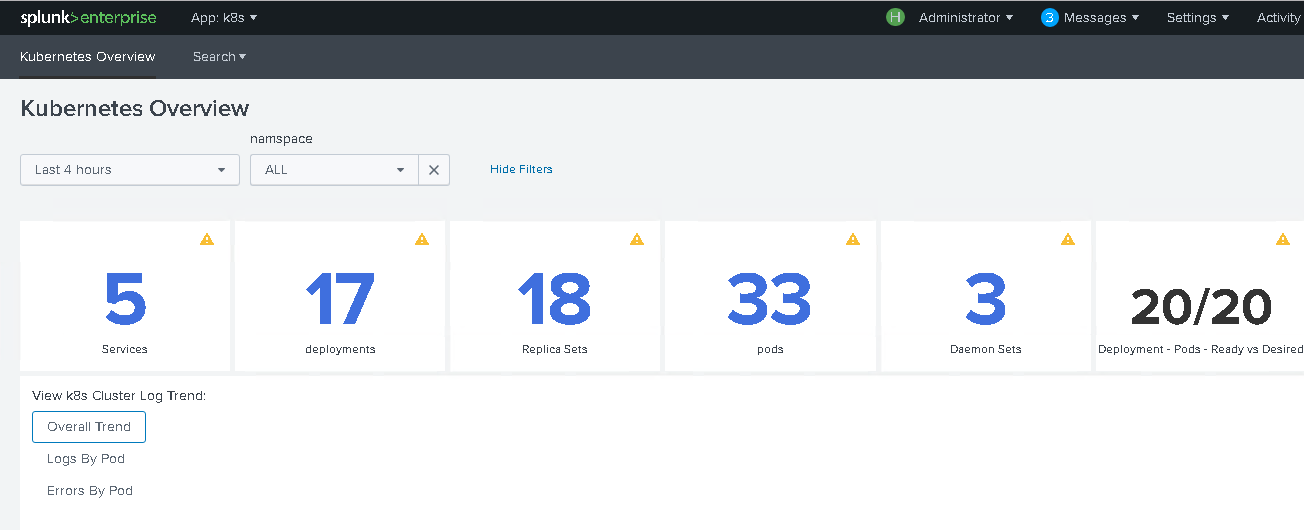


Figure 14. Kubernetes overview

## 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 hpe-ucp02  
  
# docker stack rm splunk\_demo  
Removing service splunk\_demo\_splunkenterprise  
Removing network splunk\_demo\_default  
  
# docker volume ls | grep splunk  
vsphere:latest splunk\_demo\_vsplunk-opt-splunk-etc@Docker\_HPE  
vsphere:latest splunk\_demo\_vsplunk-opt-splunk-var@Docker\_HPE  
  
# docker volume rm splunk\_demo\_vsplunk-opt-splunk-etc@Docker\_HPE  
splunk\_demo\_vsplunk-opt-splunk-etc@Docker\_HPE  
  
# docker volume rm splunk\_demo\_vsplunk-opt-splunk-var@Docker\_HPE  
splunk\_demo\_vsplunk-opt-splunk-var@Docker\_HPE

Then re-run the playbook on your ansible node.

ansible-playbook -i vm\_hosts playbooks/splunk\_demo.yml --vault-password-file .vault\_pass

# Deploying Prometheus and Grafana

## 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 15.



Figure 15. Solution architecture: Linux workers with Prometheus and Grafana

The Prometheus and Grafana services are declared in a Docker stack as replicated services with one replica each, so that 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](#_Prometheus_and_Grafana) 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.

## Playbooks for installing Prometheus and Grafana

The following playbooks are used to deploy Prometheus and Grafana on Docker RHEL nodes.

* 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.

## Prometheus and Grafana configuration

All monitoring-related variables for Prometheus and Grafana are described in Table 19. The variables determine the versions of various software tools that are used and it is recommended that the values given below are used.

Table 19. Monitoring variables

|  |  |
| --- | --- |
| Variable | Description |
| cadvisor\_version | v0.28.3 |
| node\_exporter\_version | v1.15.0 |
| prometheus\_version | V2.3.2 |
| grafana\_version | 5.2.3 |
| prom\_persistent\_vol\_name | The name of the volume which will be used to store the monitoring data. The volume is created using the vSphere Docker Volume plugin. |
| prom\_persistent\_vol\_size | 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. |

## Accessing Grafana UI

The Grafana UI is available at the UCP VIP, using HTTP on port 3000, for example,

http://hpe-ucpvip.am2.cloudra.local:3000

The default username and password for Grafana is admin/admin. The first time you login, you will be asked to reset the default admin password. Select the Docker Swarm Monitor dashboard that has already been loaded by the playbooks, as shown in Figure 16 and Figure 17.

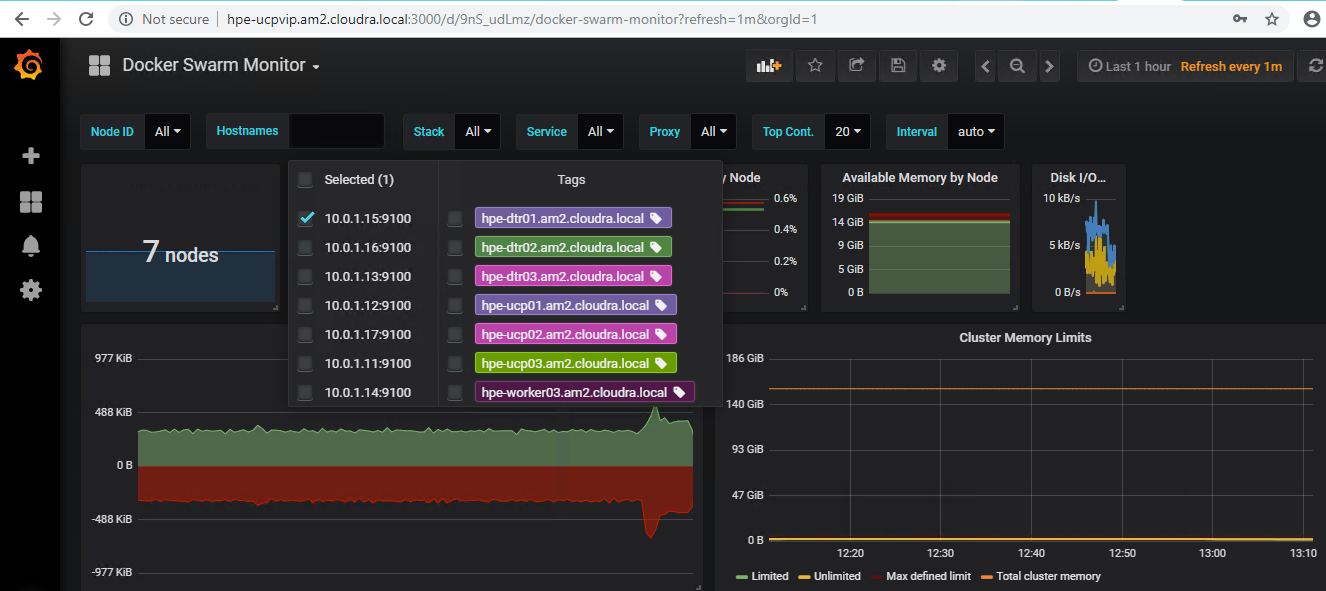


Figure 16. Docker Swarm Monitor dashboard

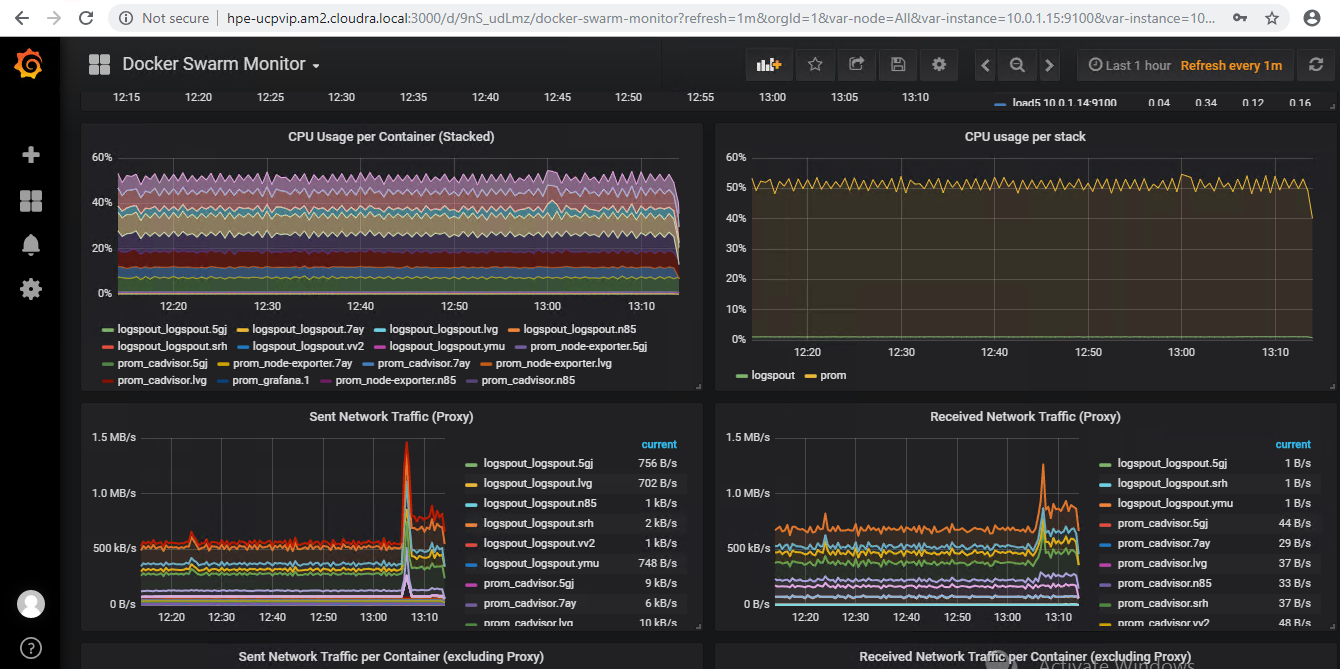


Figure 17. Docker Swarm Monitor dashboard

# 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 folder /root/Docker-SimpliVity. 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.

## 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:"5ca598b4ba5abb89bb773071ce452e33fb66339d", GitTreeState:"clean", BuildDate:"2018-06-06T08:13:03Z", GoVersion:"go1.9.3", Compiler:"gc", Platform:"linux/amd64"}  
  
Server Version: version.Info{Major:"1", Minor:"8+", GitVersion:"v1.8.11-docker-8d637ae", GitCommit:"8d637aedf46b9c21dde723e29c645b9f27106fa5", GitTreeState:"clean", BuildDate:"2018-04-26T16:51:21Z", GoVersion:"go1.8.3", 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  
 role: master  
 tier: backend  
 replicas: 1  
 template:  
 metadata:  
 labels:  
 app: redis  
 role: master  
 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:

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

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

# kubectl get pods  
NAME READY STATUS RESTARTS AGE  
**redis-master-5f8d48b75f-8ztt4** 1/1 Running 0 33s

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

# kubectl logs -f **redis-master-5f8d48b75f-8ztt4**  
 \_.\_  
 \_.-``\_\_ ''-.\_  
 \_.-`` `. `\_. ''-.\_ Redis 2.8.19 (00000000/0) 64 bit  
 .-`` .-```. ```\/ \_.,\_ ''-.\_  
 ( ' , .-` | `, ) Running in stand alone mode  
 |`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379  
 | `-.\_ `.\_ / \_.-' | PID: 1  
 `-.\_ `-.\_ `-./ \_.-' \_.-'  
 |`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|  
 | `-.\_`-.\_ \_.-'\_.-' | http://redis.io  
 `-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'  
 |`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|  
 | `-.\_`-.\_ \_.-'\_.-' |  
 `-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'  
 `-.\_ `-.\_\_.-' \_.-'  
 `-.\_ \_.-'  
 `-.\_\_.-'  
  
[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 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.  
[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.

# cat redis-master-service.yaml  
apiVersion: v1  
kind: 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

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.

# cat redis-master-service.yaml  
apiVersion: v1  
kind: 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.

# 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  
redis-slave-58fb7bb9b5-25z4m 1/1 Running 0 1m  
redis-slave-58fb7bb9b5-q8zn9 1/1 Running 0 1m

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  
 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  
kind: 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 18.

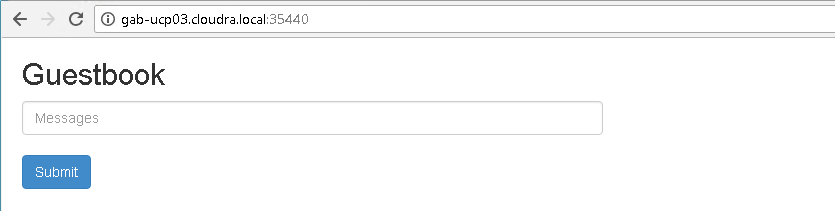


Figure 18. Guestbook UI

## Deploying the NFS provisioner for Kubernetes

NFS can be provisioned using the NFS VM for proof of concept or demo systems.

### 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](https://docs.docker.com/ee/ucp/user-access/cli/%23download-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-SimpliVity  
# 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 20.

Table 20. Configuration values

|  |  |
| --- | --- |
| Variable | Value |
| nfs\_provisioner\_role | nfs-provisioner-runner |
| nfs\_provisioner\_name | hpe.com/nfs |
| nfs\_provisioner\_storage\_class\_name | nfs |
| nfs\_provisioner\_server\_ip | hpe-nfs.cloudra.local |
| nfs\_provisioner\_server\_share | /k8s |

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-SimpliVity  
# 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.

# 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.

# 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.

# 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.

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

# 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 21 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 21. Backup variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| backup\_server | group\_vars/backups | Currently, the playbooks only support the use of the Ansible box as the backup server. |
| backup\_dest | group\_vars/backups | 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 |
| #swarm\_offline\_backup | group\_vars/backups | This variable is commented out by default. More information on this variable is provided below. |

#### 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:

# 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-disasterv>

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

When you backup UCP, you save the data/metadata outlined in Table 22.

Table 22. UCP data backed up

|  |  |
| --- | --- |
| Data | Description |
| Configurations | The UCP cluster configurations, as shown by docker config ls, including Docker EE license and swarm and client CAs |
| Access control | Permissions for team access to swarm resources, including collections, grants, and roles |
| Certificates and keys | The certificates, public keys, and private keys that are used for authentication and mutual TLS communication |
| Metrics data | Monitoring data gathered by UCP |
| Organizations | Your users, teams, and orgs |
| Volumes | All [UCP named volumes](https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/#volumes-used-by-ucp), which include all UCP component certs and data |

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/3.0/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 23.

Table 23. DTR data backed up

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

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/2.5/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

# 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:

# 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/2.5/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:

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 24 shows the file names produced by backup.sh based on the argument passed in the command line.

Table 24. Backup utility

|  |  |  |
| --- | --- | --- |
| Example | Command line | Generated filenames |
| Default | ./backup.sh | backup\_swarm\_<vmname>\_<timestamp>.tgz, backup\_ucp\_<ucpid>\_<vmname>\_<timestamp>.tgz, backup\_dtr\_meta\_<replica\_id>\_<vmname>\_<timestamp>.tgz, backup\_dtr\_data\_<replica\_id>\_<vmname>\_<timestamp>.tgz and the corresponding .vars.tgz files |
| Custom | ./backup.sh my\_backup | my\_backup\_swarm.tgz, my\_backup\_ucp.tgz, my\_backup\_dtr\_meta.tgz, my\_backup\_dtr\_data.tgz, and the corresponding .vars.tgz files |
| Date | ./backup.sh $(date '+%Y\_%m\_%d\_%H%M%S') | <date>\_swarm.tgz, <date>\_ucp.tgz, <date>\_dtr\_meta.tgz, <date>\_dtr\_data.tgz, and the corresponding .vars.tgz files |

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

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=""  
ucp\_version=""  
dtr\_version="2.4.3"

[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](#_Before_you_restore).
2. Run the restore playbook

ansible-playbook -i vm\_hosts restore.yml

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](https://docs.docker.com/datacenter/dtr/2.4/guides/admin/configure/set-up-vulnerability-scans/#get-the-security-scanning-license).

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](#_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:

[root@hpe2-ansible ops]# docker node ls  
ID       HOSTNAME                     STATUS              AVAILABILITY           
aiz… \*   hpe-ucp02.cloudra.local      Ready               Active   
gvf…     hpe-dtr01.cloudra.local      Down                Active  
ir4…     hpe-ucp03.cloudra.local      Ready               Active     
mwf…     hpe-dtr02.cloudra.local      Down                Active  
oqy…     hpe-ucp01.cloudra.local      Ready               Active    
xqe…     hpe-worker01.cloudra.local   Ready               Active  
zdu…     hpe-dtr03.cloudra.local      Down                Active

[root@hpe2-ansible ops]# docker node rm hpe-dtr01.cloudra.local  
hpe-dtr01.cloudra.local  
[root@hpe2-ansible ops]# docker node rm hpe-dtr02.cloudra.local  
hpe-dtr02.cloudra.local  
[root@hpe2-ansible ops]# docker node rm hpe-dtr03.cloudra.local  
hpe-dtr03.cloudra.local

[root@hpe2-ansible ops]# docker node ls  
ID       HOSTNAME                     STATUS              AVAILABILITY   
aiz…     hpe-ucp02.cloudra.local      Ready               Active        
ir4…     hpe-ucp03.cloudra.local      Ready               Active      
oqy… \*   hpe-ucp01.cloudra.local      Ready               Active   
xqe…     hpe-worker01.cloudra.local   Ready               Active

1. Run the restore script:

./restore\_dtr.sh

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](https://docs.docker.com/datacenter/dtr/2.4/guides/admin/configure/set-up-vulnerability-scans/#get-the-security-scanning-license).

#### 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

## HPE SimpliVity backups

HPE SimpliVity functionality can be used to backup and restore Docker persistent volumes.

### Backup and restore Docker persistent volumes

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

datastores: ['**Docker\_HPE**']

Note

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

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

* <prefix> is the value of the variable dummy\_vm\_prefix from the file group\_vars/vars

<Datastore> is the name of the datastore

For example, based on the default values in the scripts, the VM name would be hpe-VM-in-dockervols-Docker\_HPE.

#### Create a Docker volume

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

# docker volume ls | grep vsphere  
vsphere:latest prom\_hpe-db-data@Docker\_HPE  
#

To create a Docker volume named test\_01, you can use the docker volume create command specifying the vSphere driver:

# docker volume create -d vsphere test\_01  
test\_01

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

# docker volume ls | grep vsphere  
vsphere:latest prom\_hpe-db-data@Docker\_HPE  
vsphere:latest test\_01@Docker\_HPE

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

# docker run -it --rm -v test\_01:/tmp alpine sh -c "echo **some test data here** > /tmp/foo.txt"

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

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

The container will exit once the shell command has run and any unnamed volumes will be removed. However, the named volume test\_01:/tmp will persist. To check that the data is still available after the container exited, spin up a new container and try to retrieve the data:

# docker run -it --rm -v test\_01:/tmp alpine sh -c "cat /tmp/foo.txt"  
**some test data here**

#### Automated backup

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

backup\_policies:  
 - name: 'hpe-gold'  
 days: 'All'  
 start\_time: '00:00'  
 frequency: '60'  
 retention: '43200'  
dummy\_vm\_prefix: 'hpe-VM'  
docker\_volumes\_policy: 'hpe-gold'

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

#### Manual backup

Rather than waiting for an automated backup to take place, you can create a backup immediately. Right-click on the special VM, in this case hpe-VM-in-dockervols-Docker\_HPE, select All HPE SimpliVity Actions and choose Backup Virtual Machine as shown in Figure 19.

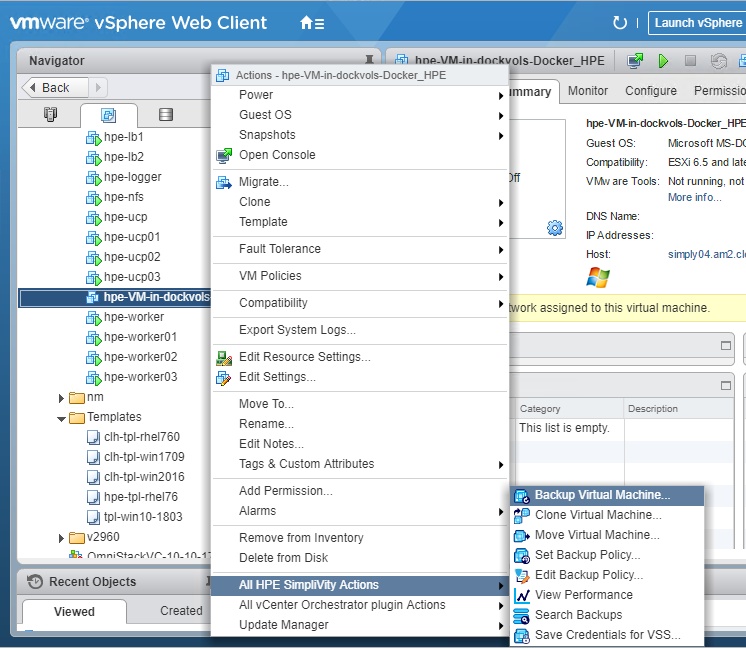


Figure 19. Backup virtual machine

You can specify a backup name, in this case manual\_backup\_test\_01, as shown in Figure 20.

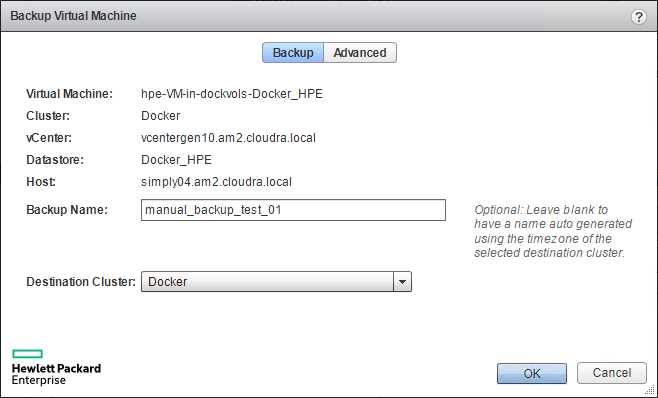


Figure 20. Backup virtual machine details

#### Restore

Right-click on the special VM, in this case hpe-VM-in-dockervols-Docker\_HPE. On the Configure tab, select HPE SimpliVity Search Backups as shown in Figure 21.

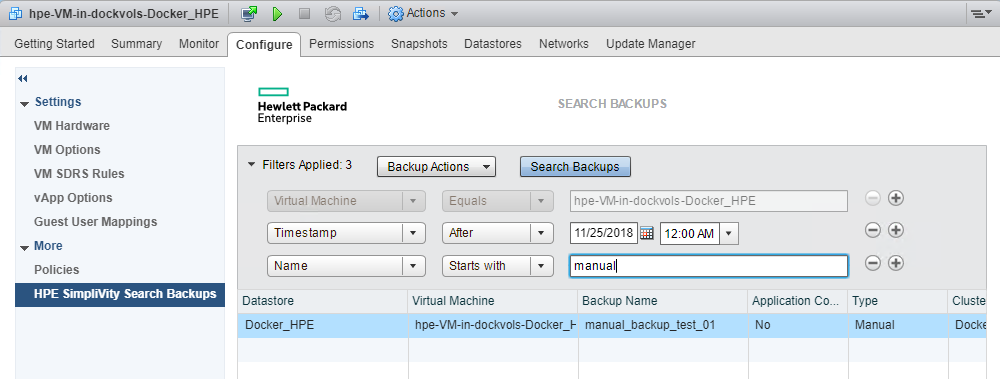


Figure 21. Search backups

You can narrow the search based on the time of the backup. If you are restoring from an automatic backup, the name will be the timestamp of the backup. If you are restoring from a manual backup, the name will be the one you specified earlier when creating the backup, in this case manual\_backup\_test\_01.

Right-click on the backup you wish to restore, as shown in Figure 22, and select Restore Virtual Machine.

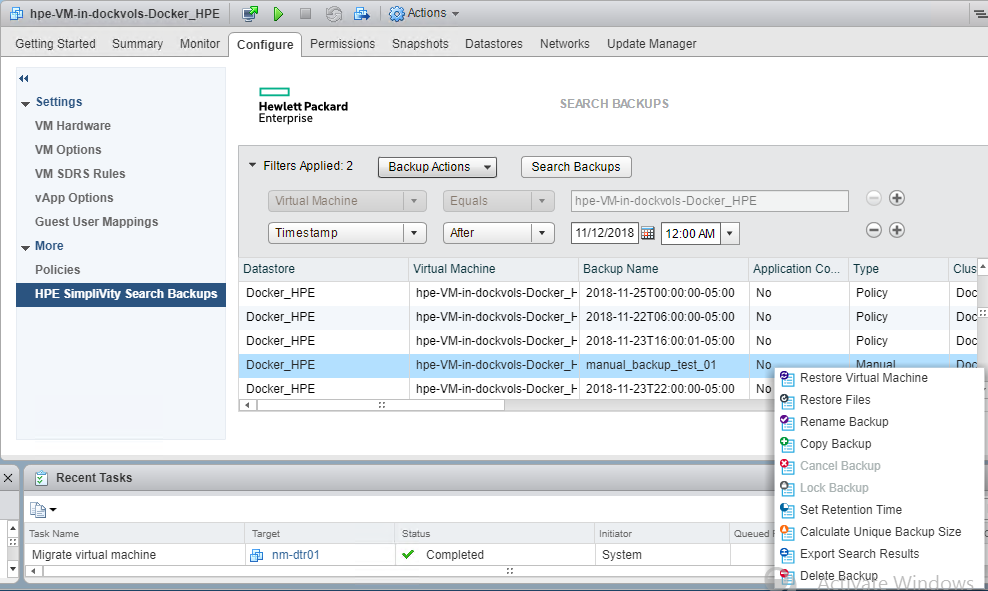


Figure 22. Restore virtual machine

In the details screen, shown in Figure 23, you can choose a name for the new virtual machine and specify the datastore.

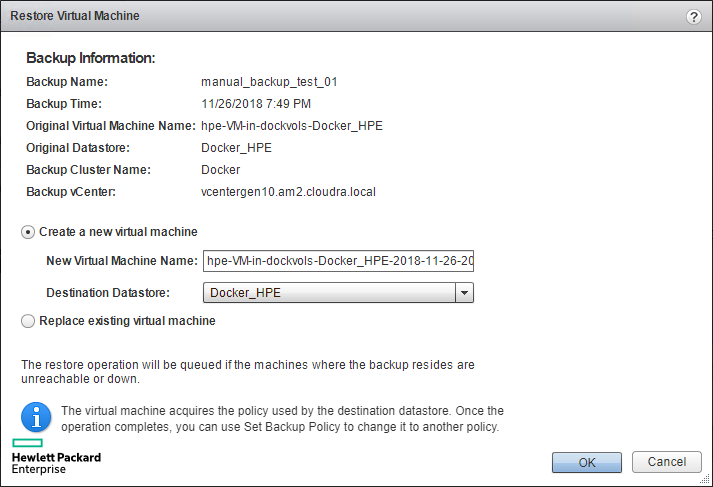


Figure 23. Restore virtual machine details

The name of the new virtual machine will default to a combination of the special VM name and a timestamp, in this instance hpe-VM-in-dockervols-Docker\_HPE-2018-11-26-20h47m01s. The datastore should be the one specified in the datastores array from the group\_vars/vars file. Click OK to restore the virtual machine.

Once the virtual machine has been restored, navigate to the datastore and locate the new VM in the file browser, as shown in Figure 24.

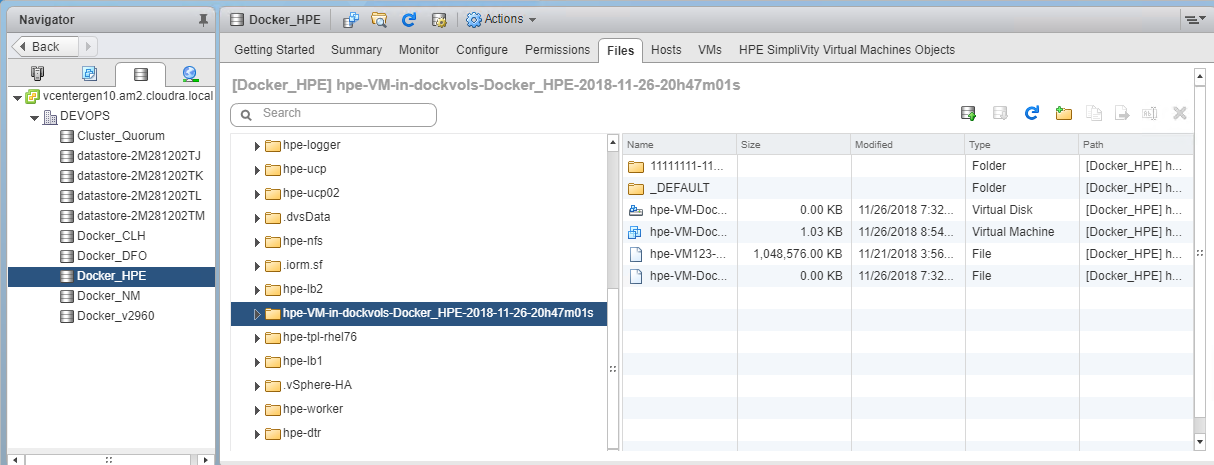


Figure 24. Browse for restored virtual machine

Navigate to the folder named 1111111-1111-1111-1111-... as shown in Figure 25. You will see files with names based on the Docker volume name that you used at the start, in this instance test\_01.vmdk and test\_01-478...f1f.vmfd

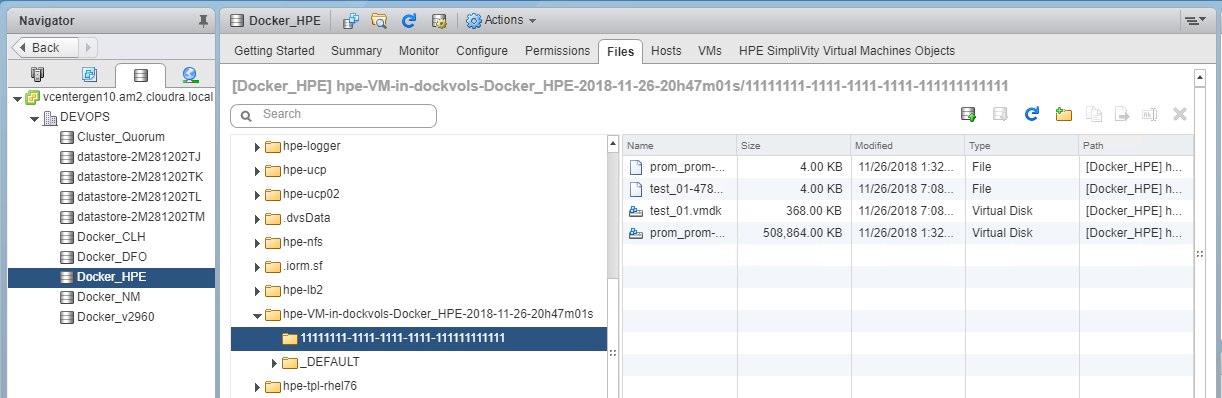


Figure 25. Locate vmdk and vmfd files

You need to move these two files to the dockvols sub-directory named 1111111-1111-1111-1111-... in the same datastore. Right click on the .vmdk file and choose Move to... as shown in Figure 26.

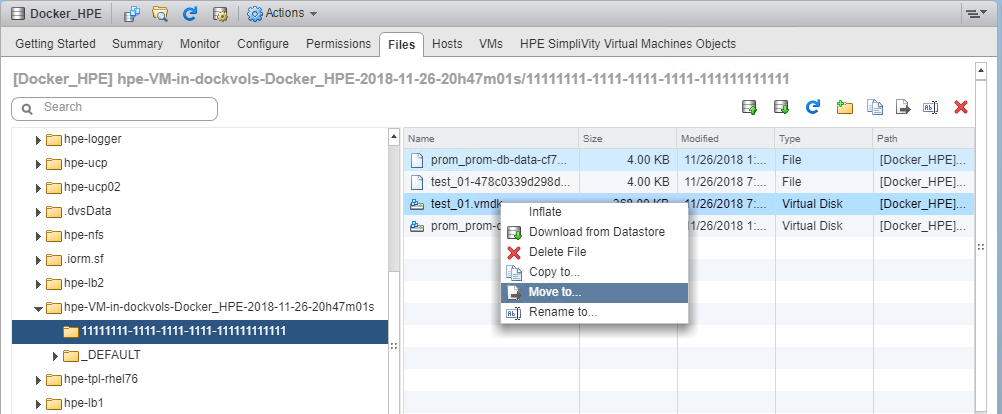


Figure 26. Move files

Set the destination folder to the dockvols sub-directory named 1111111-1111-1111-1111-... as shown in Figure 27.

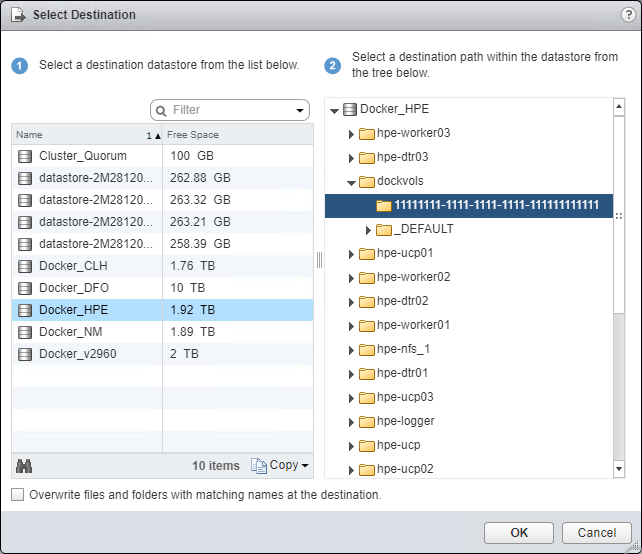


Figure 27. Move to destination

It is only necessary to move the .vmdk file as the .vmfd file will automatically follow. The dockvols sub-directory named 1111111-1111-1111-1111-... should now contain both files as shown in Figure 28.

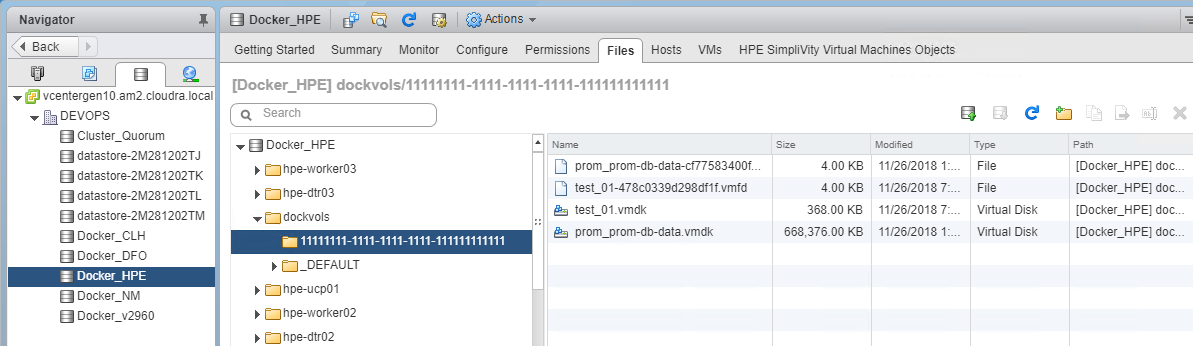


Figure 28. Files moved to destination

#### Test the restore

You can check that the volume test\_01 has been restored by using the docker volume ls command again.

# docker volume ls | grep vsphere  
vsphere:latest prom\_hpe-db-data@Docker\_HPE  
vsphere:latest test\_01@Docker\_HPE

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

# docker run -it --rm -v test\_01:/tmp alpine sh -c "cat /tmp/foo.txt"  
some test data here

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

# 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
* Virtual Machine Operating systems
* HPE SimpliVity environment

The general practice and recommendation is to follow a bottom-up approach for updating all components of the environment and making sure the dependencies are met. In our solution, we would start with HPE SimpliVity 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.

## HPE SimpliVity environment

The HPE SimpliVity environment is made up of proprietary HPE SimpliVity software, VMware software and HPE firmware. There are interdependencies between the various components that need to be accounted for and are provided in the table below. The components in Table 25 are part of the HPE SimpliVity environment that require lifecycle management.

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

Table 25. HPE SimpliVity components

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

### 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>.

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

Table 26. VMware components

|  |  |  |  |
| --- | --- | --- | --- |
| Order | Component | Dependency (compatibility) | Download/Documentation |
| 1 | VMware vCenter | 1. HPE OmniStack 2. VMware ESXi | [VMware Upgrade for SimpliVity](https://support.hpe.com/hpsc/doc/public/display?sp4ts.oid=1010292248&docLocale=en_US&docId=emr_na-a00028056en_us) |
| 2 | VMware ESXi | 1. HPE OmniStack 2. VMware vCenter |

### HPE server software

HPE SimpliVity servers are based on HPE server platforms and require a compatible firmware version to function with HPE OmniStack Software, as shown in Table 27.

Table 27. HPE server components

|  |  |  |  |
| --- | --- | --- | --- |
| Order | Component | Dependency (compatibility) | Download/Documentation |
| 1 | HPE Firmware | 1. HPE OmniStack Software | [Firmware Upgrade for SimpliVity](https://support.hpe.com/hpesc/public/home/result?qt=HPE+SimpliVity+380) |

## 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 28). 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 28. vSphere Docker Volume service components

| Order | Component | Dependency (compatibility) | Download/Documentation |
| --- | --- | --- | --- |
| 1. | Server Software | 1. VMware ESXi  2. Docker EE | [vSphere Docker Volume Service on GitHub](http://vmware.github.io/vsphere-storage-for-docker/documentation/index.html) |
| 2. | Client Software | 1. VM Operating System  2. Docker EE |

## Red Hat Enterprise Linux operating system

This solution is built using Red Hat Enterprise Linux (see Table 29) 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 29. Operating system

| Order | Component | Dependency (compatibility) | Download/Documentation |
| --- | --- | --- | --- |
| 1. | Red Hat Enterprise Linux | 1. Docker EE  2. vDVS client software plugin | [RHEL](https://access.redhat.com/articles/11258) |

## 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 30.

A banner will be displayed on the UI, as shown in Figure 29, when an update is available for UCP or DTR. You can start the upgrade process by clicking on the banner.

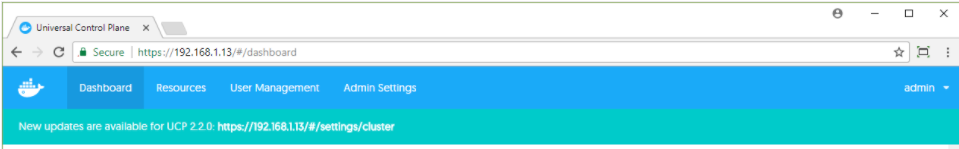


Figure 29. Docker update notification

Table 30. Docker EE components

| Order | Component | Dependency (compatibility) | Download/Documentation |
| --- | --- | --- | --- |
| 1. | Docker Daemon/Engine | 1. VM Operating System  2. vDVS plugin  3. Prometheus and Grafana | [Docker Lifecycle Maintenance](https://success.docker.com/Policies/Maintenance_Lifecycle)  [Docker Compatibility Matrix](https://success.docker.com/Policies/Compatibility_Matrix) |
| 2. | Universal Control Plane |
| 3. | Docker Trusted Registry |

## Monitoring Tools

To learn more about upgrading Splunk, see the relevant documentation at [How to upgrade Splunk Enterprise](http://docs.splunk.com/Documentation/Splunk/7.0.3/Installation/HowtoupgradeSplunk).

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

Table 31. Monitoring tools: Prometheus and Grafana

|  |  |  |  |
| --- | --- | --- | --- |
| Order | Component | Dependency (compatibility) | Download/Documentation |
| 1. | Prometheus | 1. Grafana  2. Docker EE | 1. Prometheus Images on Docker Hub  2. [Upgrading Grafana](http://docs.grafana.org/installation/upgrading/) |
| 2. | Grafana | 1. Prometheus  2. Docker EE |

# Summary

This document has described how to architect and deploy a Docker CaaS platform on HPE SimpliVity, 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 SimpliVity as the deployment infrastructure.

# Appendix A: Software Licenses

Licenses are required for the following software components:

* VMware
* Red Hat Linux
* Docker EE
* Splunk (optional software)

Sysdig (optional software)

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

Table 32 lists the variables used when configuring customer supplied certificates for UCP and DTR.

Table 32. Customer certs variables

|  |  |  |
| --- | --- | --- |
| Variable | File | Description |
| ucp\_certs\_dir | group\_vars/vars | 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 |
| dtr\_certs\_dir | group\_vars/vars | 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 |

Note

The installation will fail if the ca.pem, cert.pem and key.pem files cannot be found in the folders designated by dtr\_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).

[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+DtuADxHLGz8wDQYJKoZIhvcNAQEL  
BQAwGjEYMBYGA1UEAxMPRXhhbXBsZSBSb290IENBMB4XDTE4MDQyNDIwMTIwMVoX  
...  
...  
uXzYbCtU6Jt9B3fayAeWWswQv+jQSzuuA3re0M1x838iIZWDx93f4yLJWLJl7xsY  
btvKBmqKDCsAqsQLFLnNj/JyYq4e9a6Xxcyn9FXNpzuEsfjfNGHn+csY+w3f987T  
MNviy376xZbyAc1CV5kgmnZzjU5bDkgT8Q==  
-----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').

[root@ansible ucp\_certs]# openssl x509 -text -noout -in intca.pem|head -14  
Certificate:  
    Data:  
        Version: 3 (0x2)  
        Serial Number:  
            6b:1e:0c:86:20:cf:f0:88:d2:52:0d:5d:b9:56:fa:91:87:a0:49:18  
    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-----  
MIIFcjCCA1qgAwIBAgIUax4MhiDP8IjSUg1duVb6kYegSRgwDQYJKoZIhvcNAQEL  
BQAwGjEYMBYGA1UEAxMPRXhhbXBsZSBSb290IENBMB4XDTE4MDQyNDIwMTIwOVoX  
...  
...  
o2tL5nwR7ROiAr/kk9MIRzWrLNbc4cYth7jEjspU9dBqsXgsTozzWlwqI9ybZwvL  
Ni1JnZandVlyQdoOaB2M/1DNFfKvwW3JeArKvDA9j95n/BWFTjoZ+YOz9pYit6T7  
1GCGu3be  
-----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.clodura.local, hpe-ucp03.cloudra.local).

[root@ansible ucp\_certs]# openssl x509 -text -noout -in server.pem  
Certificate:  
    Data:  
        Version: 3 (0x2)  
        Serial Number:  
            25:d9:f8:1d:9b:1d:23:f1:21:56:54:f2:43:cc:4f:0e:73:22:be:ec  
    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-----  
MIIFGTCCAwGgAwIBAgIUJdn4HZsdI/EhVlTyQ8xPDnMivuwwDQYJKoZIhvcNAQEL  
BQAwKDEmMCQGA1UEAxMdSW50ZXJtZWRpYXRlIENBIHZhbGlkIDMgeWVhcnMwHhcN  
...  
...  
s0R4I3Qnc50oNISng5l7wW1d4RMMwmXQhG1H5QKAUjHfJXH4bNtIzKxw/zGTVr4Z  
llYKbEwJcgAvvfkn+w==  
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----  
MIIFcjCCA1qgAwIBAgIUax4MhiDP8IjSUg1duVb6kYegSRgwDQYJKoZIhvcNAQEL  
BQAwGjEYMBYGA1UEAxMPRXhhbXBsZSBSb290IENBMB4XDTE4MDQyNDIwMTIwOVoX  
...  
...  
Ni1JnZandVlyQdoOaB2M/1DNFfKvwW3JeArKvDA9j95n/BWFTjoZ+YOz9pYit6T7  
1GCGu3be  
-----END CERTIFICATE-----

Finally, here is an excerpt from key.pem, the private key that goes with the server certificate.

-----BEGIN RSA PRIVATE KEY-----  
MIIEpQIBAAKCAQEA5rmmb52ufE80a3cXhY2HSRZNazb7/fipXY1rZ+U5+rJv9BN5  
d/X3NTroSE8/PvoS/maGkHCnURGNqbu/G2umKN/tm/eSpDY861YnGWxj+bc0gtiU  
...  
...  
A0SGidSMk3hFX1Iaftgx4EUGbrzZO7I8M5RO64U1aMFNFyj4XghJ2mZTdNelwNBw  
pr/fYulyi5lYPalQHYH3OyvNqQQ3arEbTbZp8hEyY0gxtZRXmmaoqOY=  
-----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:

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

sed -e '1,/-----END CERTIFICATE-----/d' cert.pem >intca.pem

Combine intca.pem and ca.pem to form cachain.pem:

cat intca.pem ca.pem > cachain.pem

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

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

A successful check will generate output similar to:

[root@ansible ucp\_certs]# cat intca.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  certsignedbyanotherca.pem  
certsignedbyanotherca.pem: O = HPE, OU = CloudRA Team, CN = hpe-ucp.cloudra.local  
error 20 at 0 depth lookup:unable to get local issuer certificate

# 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-SimpliVity/files/splunk/linux/SPLUNK\_HOME/etc/mycerts

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.

[splunktcp-ssl://9998]  
disabled=0  
connection\_host = ip  
  
[SSL]  
serverCert=/opt/splunk/etc/mycerts/indexer.pem  
#requireClientCert = true  
#sslAltNameToCheck = forwarder,forwarder.cloudra.local  
  
[tcp://1514]  
connection\_host = dns  
sourcetype = ucp

For more information, see the documentation at <https://docs.splunk.com/Documentation/Splunk/7.1.2/Security/ConfigureSplunkforwardingtousesignedcertificates>. 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>.

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 sssRootCaPath is pointing to the root CA certificate.

[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

1. 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

1. 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

1. 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)

-----BEGIN CERTIFICATE-----  
MIIDyTCCArGgAwIBAgIUUeo+H6xGSB7/9gqq9T2SUwJPLggwDQYJKoZIhvcNAQEL  
BQAwbDELMAkGA1UEBhMCRlIxFTATBgNVBAcTDFRoZSBJbnRlcm5ldDETMBEGA1UE  
ChMKQ2hyaXN0b3BoZTEUMBIGA1UECxMLQ0EgU2VydmljZXMxGzAZBgNVBAMTEkNo  
...  
XkJ8WcsHocJO8J9J3RaWsM2BQc7wRntJc0kA7ooTH13OtQTP1jFcQp5xNdI4J3Mz  
j9BAYERjkGqu7v9tfOem99oVGUal20pu4r73eWUm1mL948xuw6PgiRSLZrXhn/RS  
uvFVnS/vPYJozOXIZA==  
-----END 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 30.

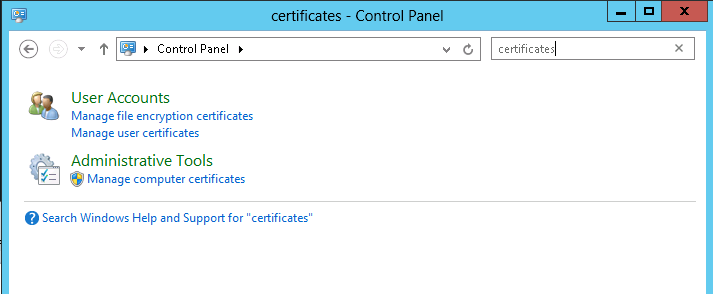


Figure 30. Manage computer certificates

Import the ca.pem for UCP into the Trusted Root Certification Authorities, as shown in Figure 31.

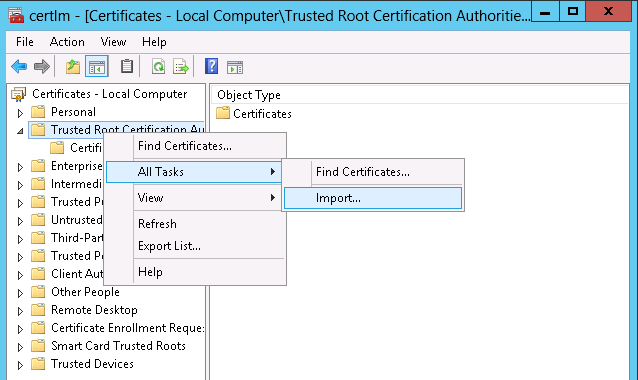


Figure 31. Import the ca.pem

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 32.

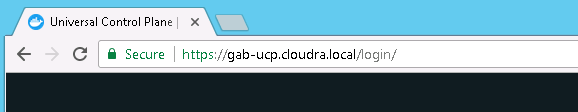


Figure 32. Secure HTTPS

# Resources and additional links

HPE Reference Architectures, [hpe.com/info/ra](http://www.hpe.com/info/ra)

HPE SimpliVity, [hpe.com/info/simplivity](file:///C:\Users\cardenje\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\88X2T2V7\hpe.com\info\simplivity)

HPE Servers, [hpe.com/servers](http://www.hpe.com/servers)

HPE Storage, [hpe.com/storage](http://www.hpe.com/storage)

HPE Networking, [hpe.com/networking](http://www.hpe.com/networking)

HPE Technology Consulting Services, [hpe.com/us/en/services/consulting.html](http://www.hpe.com/us/en/services/consulting.html)

Docker Reference Architectures, <https://success.docker.com/architectures>

Splunk Validate Architectures, <https://www.splunk.com/pdfs/white-papers/splunk-validated-architectures.pdf>

Sysdig Resources, <https://sysdig.com/resources/>

To help us improve our documents, please provide feedback at [hpe.com/contact/feedback](http://www.hpe.com/contact/feedback).