Introduction to Docker Certified Infrastructure for Microsoft Azure

Docker Enterprise is the only enterprise-ready container platform that enables organizations to choose how to cost-effectively build and manage their entire application portfolio at their own pace, without fear of architecture and infrastructure lock-in. *Docker Certified Infrastructure* is Docker's prescriptive approach to deploying Docker Enterprise on a range of infrastructure choices. This Docker Certified Infrastructure Reference Architecture documents our best practice guidance for running the Docker container platform on your selected infrastructure.

In conjunction with the Reference Architecture we also publish Ecosystem Solution Briefs (https://success.docker.com/article/certified-infrastructures-aws#dockersolutionbriefs) to help you integrate Docker Enterprise with popular 3rd party tools used in conjunction with our container platform.

Microsoft Azure is a **public cloud** platform composed of a number of cloud services that developers and IT professionals can use to build, deploy and manage infrastructure and applications across a global network of datacenters.

Microsoft Azure is a popular option for enterprises deploying services on a public cloud provider. A broad number of services are available, which help make infrastructure and applications reliable and failure-resilient. These services can be integrated with on-premises systems to support **hybrid cloud** scenarios.

Many organizations strive for a scalable and resilient container platform that can also sit harmoniously within the Microsoft Azure public cloud.

Reference Architecture

What You Will Learn

This reference architecture is designed to provide a best practice scenario and architecture considerations when provisioning a new environment or modifying an existing environment on which Docker Enterprise is deployed within Microsoft Azure public cloud.

It describes:

- How to correctly architect and deploy Docker Enterprise on Microsoft Azure
- How to take advantage of features around compute/storage and networking available on Microsoft Azure to provide the best experience of Docker Enterprise

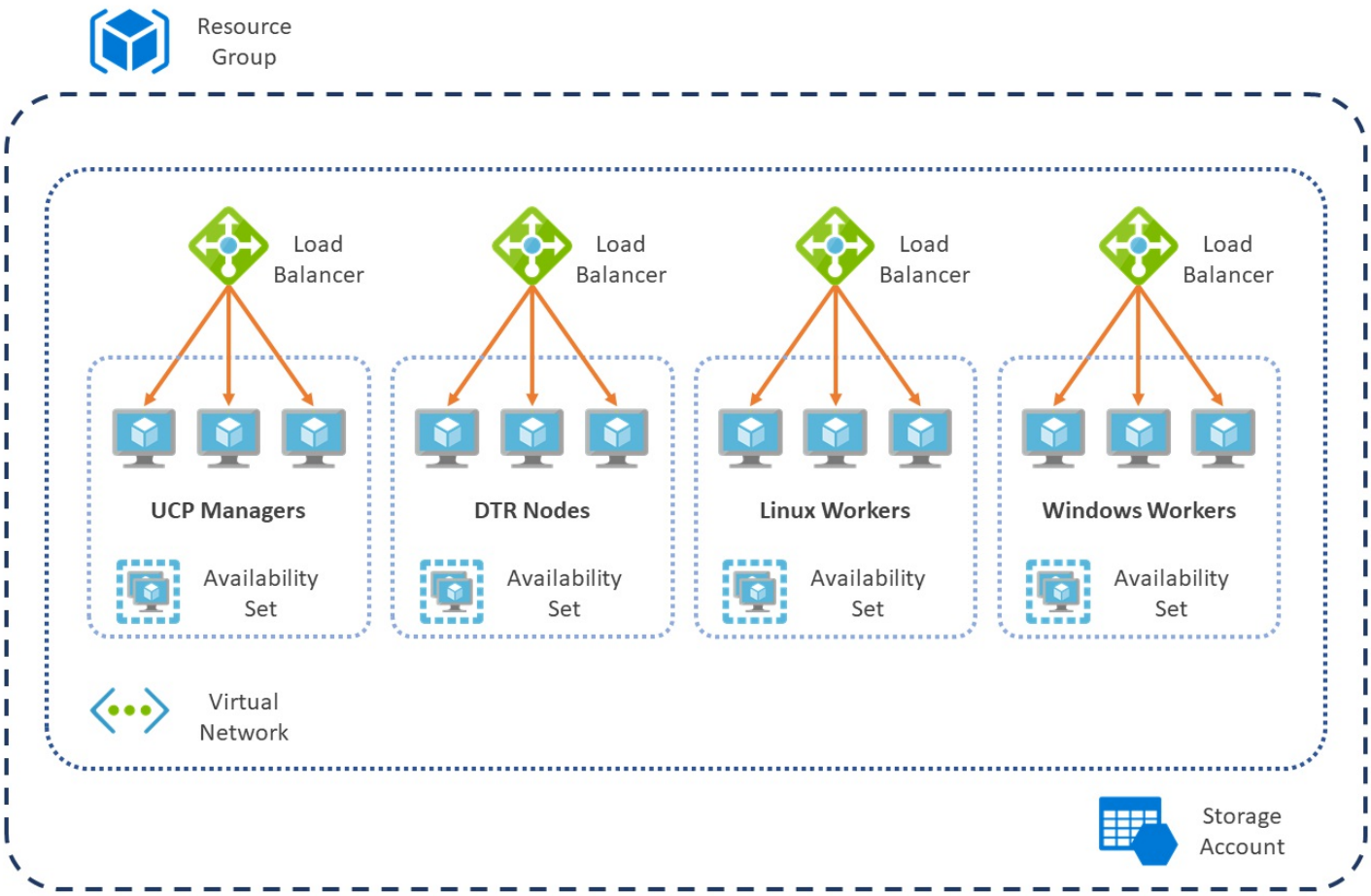
It provides a checklist of tasks and procedures to ensure that your platform is configured correctly **before** the installation begins. It details all of the components in both Microsoft Azure and the Docker Enterprise environment to ensure your platform is both understood and built from supported components. Additionally there are a number of sections that describe further configuration such as the use of storage/networking

plugins to provide expanded functionality from Microsoft Azure. Finally, this reference architecture explains how to scale the platform as application requirements grow and provides a number of troubleshooting procedures should there be issues deploying the Docker Enterprise platform.

Installation of Docker Enterprise is not covered in this reference architecture, but detailed installation instructions are provided for each supported operating system at docs.docker.com (https://docs.docker.com/ee/supported-platforms/#on-premises).

Architecture Overview

This Reference Architecture provides a solution architected for Docker Enterprise that is to be hosted on Microsoft Azure as shown in the following architecture diagram:



All components are deployed in the same Azure Region.

The deployment makes use of a number of key Azure resource types. These are the resources you will need to spend time customizing:

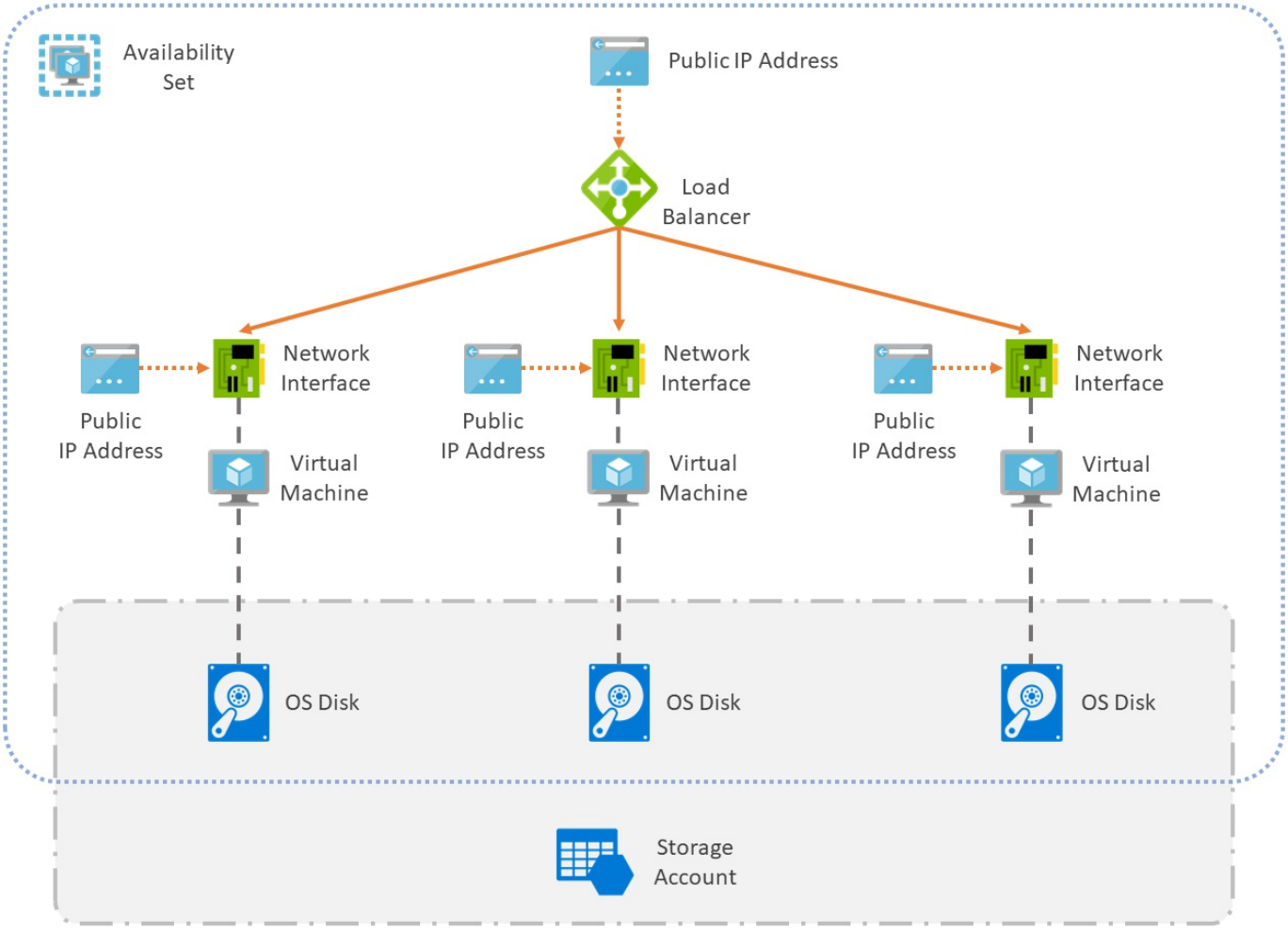
- **Load balancer:** A Load Balancer endpoint is created for each of UCP, DTR, Linux workers and Windows workers. Each is pre-configured with a Load Balancing rule that balances traffic from the Load Balancer's frontend IP address to the backend pool VMs for that group (e.g. the set of UCP managers).
- **Storage account:** A Single Storage account is created. This is a Standard (not Premium) type configured to use Locally-redundant storage (LRS) replication. The OS disk for each VM is stored in this Storage account.

- **Virtual machine:** Each node in a cluster is deployed as a virtual machine, including UCP Managers, DTR nodes, Linux workers and Windows workers.
- **Virtual network:** All virtual machines are deployed onto a single virtual network. This is composed of a single subnet, in which all UCP Managers, DTR nodes, Linux workers and Windows workers are deployed.

In addition, the following Azure resource types are also present in the cluster. Although less configuration is necessary, it is useful to become familiar with them.

- **Availability set:** Each type of node (e.g. UCP managers or Linux workers) is deployed inside an availability set. This ensures the availability of each role type is not impacted by planned or unplanned downtime. For more information, review the guide on managing the availability of Windows virtual machines in Azure (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability).
- **Disk:** Each virtual machine has a single OS disk, of varying sizes. These are configured to use Premium SSD disk types.
- **Network interface:** Each virtual machine has a single Network interface, attached to the single virtual network.
- **Network security group:** A single Network security group is created and applied to the Network Interfaces for all Windows worker nodes. It is configured with inbound and outbound rules allowing Ansible to connect through WinRM.
- **Public IP address:** A Public IP address endpoint is created for DTR, UCP and Linux workers. Each virtual machine also has its own Public IP address. Although these are useful for initial setup and configuration, you may wish to restrict external access to Public IP addresses depending on the purpose of your platform (e.g. when hosting internal applications).

The following diagram illustrates how the Networking and Storage components come together for each set of virtual machines inside an availability set:



Infrastructure Configuration

Deploying Docker Enterprise on Microsoft Azure makes use of the compute, networking, and storage resources available in the Azure public cloud platform. This Reference Architecture is designed for a cloud-only deployment with no Docker Enterprise components deployed outside Microsoft Azure.

To provide an environment that is suitable for high availability and guarantee the performance of the platform, consider the following design choices:

- Deploy to enough Fault Domains (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability) to ensure that there is N+1 resiliency in the event of unplanned hardware failure or planned platform maintenance. This is done automatically by deploying all virtual machines inside availability sets.
- Deploy to the Azure Region (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/regions-and-availability) that offers the lowest latency to the most common users of the platform. You can use tools such as Azure Speed (http://azurespeed.com/) to discover the closest Azure regions to your users.
- Create logical separation between management and application compute, network, and storage resources. This can be achieved in different ways depending on the resource type.
- Filter network traffic into and within the virtual network you deploy Docker Enterprise in through Network Security Groups (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg).

> Consider also the recommendations available at the Azure Architecture Center (https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/).

For scenarios where higher availability or performance is necessary, you may wish to make use of more advanced features in Microsoft Azure. Instructions to configure these advanced Azure services can be found in the Docker Solution Briefs (https://success.docker.com/article/certified-infrastructures-azure#dockersolutionbriefs)

Managed Disks are an example of an advanced feature providing higher reliability for virtual machines. Managed disks provide better reliability for VMs inside availability sets by isolating the VM disks as well as the VMs themselves, avoiding single points of failure. You can upgrade your VMs to use Managed disks by following the Azure Premium Storage Solution Brief (https://success.docker.com/article/azure-premium-storage). Each Azure region supports a different number of fault domains (two or three). Ensure you select a region where the number of fault domains matches your availability needs. You can find the list of fault domains per region available on docs.microsoft.com (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability#use-managed-disks-for-vms-in-an-availability-set).

Node Sizing Recommendations

A node is a machine in the cluster (virtual machines in Microsoft Azure) with Docker Engine running on it. Each node is assigned a role: UCP controller, DTR registry, or worker node.

The sizing requirements (e.g. CPU and RAM available) for a Docker Enterprise node are based upon having sufficient resources for both the Docker Enterprise management platform (including load balancers and additional management tooling) and then the requirements for the containerized applications that are hosted on the Docker Enterprise platform.

To define the appropriate size for each kind of node, refer to the recommendations included in the Docker Enterprise Best Practices and Design Considerations Reference Architecture (https://success.docker.com/article/docker-ee-best-practices). Where possible, node sizes should be determined by experimentation and testing actual workloads, through input from applications and operations teams, and through iterative refinement.

Gathering Inventory from Microsoft Azure

There are two options to gather information from Microsoft Azure. This may be done for example to validate a deployment has been completed successfully.

First, you can visit the Azure Portal (https://azure.microsoft.com/en-us/). Use the search bar above the dashboard to search for Resource Groups or Resources, or navigate the menu to the left to create a Resource Group or Resource, or view existing ones. Once your Docker Enterprise cluster is deployed, you may choose to create and share a custom dashboard (https://docs.microsoft.com/en-us/azure/azure-portal/azure-portal-dashboards) outlining the resources deployed.

The second option is to use Azure CLI or Azure PowerShell to view and modify the Docker Enterprise environment.

- Azure CLI (https://docs.microsoft.com/en-us/cli/azure/?view=azure-cli-latest) provides you with commands to view and manage Azure resources.
- Azure PowerShell (https://docs.microsoft.com/en-us/powershell/azure/overview) provides you with cmdlets to view and manage Azure resources.

> **Note:** Both Azure CLI and Azure PowerShell can be installed on Mac OS, Linux, and Windows, so you can choose the option that you or your team are most familiar with.

Azure Resources

The Microsoft Azure public cloud platform provides a flexible set of cloud resources that can be deployed and configured to help that workloads perform effectively and reliably. This section covers the key Microsoft Azure resources that should be configured to provide the best experience for running Docker Enterprise on Microsoft Azure.

Microsoft Azure Prerequisites

Before Docker Enterprise can be successfully deployed onto Microsoft Azure there are a number of architectural choices that need to be considered. These range from providing a networking topology that provides either a shared or physically separated network to ensuring that storage profiles provide the performance and capacity needed for the Docker Enterprise platform.

Subscriptions

You need access to an Azure Subscription to deploy Docker Enterprise on Azure. You can log into https://account.azure.com/Subscriptions (https://account.azure.com/Subscriptions) with your Microsoft or Organizational account. If using an account belonging to your organization, make sure you are familiar with any usage policies defined by your organization.

Note that Azure subscriptions have a number of default limits, for example 20 cores per subscription. You may reach these limits when provisioning large Docker Enterprise environments (for example, each UCP manager uses 4 cores by default), resulting in a deployment error. If you plan to provision large Docker Enterprise environments (3 UCP managers or more), you must first review the current limits applied to your subscription, and, where appropriate, raise a request with Microsoft to increase these.

For more information see Azure subscription and service limits, quotas, and constraints (https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits).

Access Management

Microsoft Azure supports Role-Based Access Control (https://docs.microsoft.com/en-us/azure/active-directory/role-based-access-control-configure) and Resource Locks (https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-lock-resources) for resources deployed inside Resource Groups. When working in a shared Azure environment, you should make use of this feature to limit and protect resources. Some examples:

- If other teams, third parties or automation/monitoring tools need access to Azure resources, set RBAC permissions to provide only the necessary permissions (e.g. Read but not Write) to the relevant accounts.
- Use Resource Locks (https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-lock-resources) to prevent business critical resources from being accidentally deleted, such as Docker Enterprise environments hosting services in Production.

Review more examples on how to configure Azure to fit your Enterprise's governance requirements in Azure enterprise scaffold - prescriptive subscription governance (https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-subscription-governance).

Microsoft Azure Networking

When designing the networking topology for Docker Enterprise there is the design option to either logically separate the control plane and data plane through the use of overlay networks, or physically separate by binding management and data traffic to separate interfaces (https://docs.docker.com/engine/Swarm/networking/#use-a-separate-interface-for-control-and-data-traffic).

Storage

Microsoft Azure provides a number of options for storage, including Managed Disks (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/managed-disks-overview) and Premium Storage (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/premium-storage).

You should consider the following when selecting storage capabilities:

- By default, virtual machines are deployed with Premium Storage disks (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/premium-storage#scalability-and-performance-targets). Ensure you are familiar with the performance available to Premium Disks.
- Increase virtual machine disk sizes for workloads where default values may cause bottlenecks.
- Managed Disks (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/managed-disks-overview) are a good option if you expect to scale your cluster size considerably. Managed Disks are not deployed by default, but you can convert VMs to use managed disks (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/convert-unmanaged-to-managed-disks) after deployment.
- If any virtual machines will host data that is not replicated externally, you may consider upgrading the default locally redundant storage (LRS) to geo-redundant storage (GRS). In practice, this should mainly be considered for DTR storage. See Azure Storage Replication (https://docs.microsoft.com/en-gb/azure/storage/common/storage-redundancy) for more information. This option is not possible if using Managed Disks.

> **Note:** For recommendations on using Premium Storage, refer to the Azure Premium Storage Solution Brief for Docker Enterprise 17.06 (https://success.docker.com/article/azure-premium-storage).

Virtual Machine Images

One of the **key** requirements for providing a platform that Docker Enterprise can be deployed upon is pre-built virtual machine images. Azure provides a number of pre-built images, though you may wish to use your own.

There are a number of requirements to these images that need to be observed for a successful Docker Enterprise deployment.

Docker Enterprise is validated and supported to work on the operating system distributions found on the [compatibility matrix](https://success.docker. com/article/compatibility-matrix)

Often, the public SKU provided by an OS vendor on Azure is sufficient to proceed. If you're planning on utilizing your own Azure image, be sure that the following requirements are met.

Required Image Configuration

Ideally a Virtual machine image for Docker Enterprise should be based upon a minimal package deployment (dependent on Distribution) this should ensure that no GUI tooling or programming toolchains are present within the template. Keeping the template as simple as possible provides both an efficient template for multiple deployments as well as a virtual machine with the smallest attack vector. It is also recommended that

virtual machine templates are tested against the Center for Internet Security (https://www.cisecurity.org) benchmarks, which provides a report of detected issues that can be corrected before the template is deemed production ready.

Further image requirements are listed below:

- At least 40GB for the operating system. Additional space may be required depending on the use case for the virtual machine deployed from the template, for example, DTR image storage.
- The **openSSH** daemon installed and keys created for a user with the following permissions:
  - Modify firewall rules
  - Modify filesystem (mount, create filesystems)
  - Install packages
  - Enable system services (`init.d` / `systemd`)
- Python installed (optional)
- Password-less sudo (optional)
  - Generate a key-pair and add the public key in the template's `authorized_keys` file (`/etc/ssh/authorized_keys` or `.ssh/authorized_keys`).
  - An SSH agent helps to avoid retyping passwords: `$ ssh-add <path to the private key>`
  - See ssh-keygen (https://man.openbsd.org/ssh-keygen), ssh-agent (https://man.openbsd.org/ssh-agent), and ssh-add (https://man.openbsd.org/ssh-add) for complementary information.
- Programs or agents that perform operating system monitoring or data collection should be configured to ensure they don't impact the Docker Engine/UCP/DTR. One example is configuring anti-virus optimization for Windows containers (https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/anti-virus-optimization-for-windows-containers)

Recommended APIs to Monitor

To programmatically integrate Docker UCP monitoring with existing solutions there are a number of API endpoints that can be accessed in order to determine both the state of the running services along with the Docker Enterprise platform itself.

Determine the state of the platform

To get a simple overview of the platform the URI endpoint `/_ping` will return the HTTP status code of the Docker UCP components.

HTTP Status Code	Reason
200	**Success**, manager healthy
500	**Failure**, manager unhealthy
Success, manager healthy	default

Return a list of events

To return a list of events that have happened on the Docker Enterprise platform the endpoint `/events` returns a parsable JSON response that can be evaluated to determine what has happened on the Docker Enterprise platform.

Examine a particular service

The endpoint `/services/{id}` allows an end-user to programmatically evaluate the health and status of a particular service that is running on the Docker Enterprise platform.

Further information about the API endpoints are available at docs.docker.com (https://docs.docker.com/datacenter/ucp/2.2/reference/api/#/).

Log Collection and Frequency

Part of deploying Docker Enterprise is configuring logging. Logging provides visibility into the health and performance of both the platform and the services that run on top of it. It is recommended that you send logs to a log aggregator that provides ease of search as well as levels of observability that allow for quick troubleshooting of instability issues.

A reference architecture that details logging design and best practices is available in Docker Logging Design and Best Practices (https://success.docker.com/article/docker-reference-architecture-docker-logging-design-and-best-practices).

Additionally, Docker provides a number of solutions briefs for integrating Docker logging with other platforms:

- Splunk Enterprise Solution Brief for Docker Enterprise (https://success.docker.com/article/splunk-logging/)
- Logging with Elasticsearch, Logstash, and Kibana Solution Brief for Docker Enterprise (https://success.docker.com/article/elasticsearch-logstash-kibana-logging/)

Upgrading Individual Components of Docker Enterprise

> Before upgrading, make sure you create a backup (https://success.docker.com/article/backup-restore-best-practices). This makes it possible to recover if anything goes wrong during the upgrade.

Docker Enterprise has the capability to "dial home" to docker.com and determine if a new version of Docker UCP or DTR is available; this requires Internet access. In the event that an update is available, a banner will appear in the UI alerting the user to the newly released version of a particular component of Docker Enterprise.

When a new version of Docker Enterprise is available, an end-user has the option of upgrading directly through the user interface by clicking the banner alert displaying the new version and then following the prompts to update. Alternatively a user can download the offline bundle(s) and follow the CLI steps to perform a command line update to Docker Enterprise components. Use the documentation for the method you prefer:

- Perform updates in the UI (https://docs.docker.com/ee/upgrade/) Perform updates from the command line (https://docs.docker.com/datacenter/ucp/2.2/reference/cli/upgrade/)

> **Note:** Before starting any upgrade tasks, it is recommended that you study the upgrade matrix to ensure that the correct upgrade path is followed, the matrix is available on success.docker.com (https://success.docker.com/article/compatibility-matrix).

Monitoring Docker Enterprise with Prometheus

Currently only the Docker Engines can be monitored with Prometheus and not the management platform as a whole. To monitor the Docker Enterprise engines, their configuration will need modifying so that the engines will expose the metrics. The instructions for enabling Prometheus metrics are available on docs.docker.com

(https://docs.docker.com/config/thirdparty/prometheus/) as well as in Grafana/Prometheus Monitoring Solution Brief for Docker Enterprise (https://success.docker.com/article/grafana-prometheus-monitoring/).

Persistent Storage

The majority of applications have some component that requires data to persist for a myriad of reasons, such as application resiliency in the event of application failure. A user of Microsoft Azure has a number of options that can be considered to provide a persistent storage option for services and applications that run on the Docker Enterprise platform.

Locally Presented Storage

A virtual machine that is going to host Docker workloads can be configured to have a larger virtual disk assigned to it in order to provide more local persistent storage that can be mapped into a container when started. However, creating a second virtual disk and attaching it to the virtual machine to be consumed as persistent storage has the following benefits:

- The application disk can be placed on an alternative datastore that has faster backing storage (SSD/NVMe)
- In the event that the application disk reaches capacity, it won't stop the operating system from functioning (allows easier problem diagnosis).
- It also simplifies the resizing of the disk as it isn't shared with other filesystem partitions, etc.
- In the event of operating system corruption, it's quick and easy to remap the application disk to a healthy Docker node and reschedule the workload to run there.

For recommendations on using Premium Storage, refer to the Azure Premium Storage Solution Brief for Docker Enterprise 17.06 (https://success.docker.com/article/azure-premium-storage).

What is Cloudstor?

Cloudstor is a modern volume plugin built by Docker. Docker Swarm mode tasks and regular Docker containers can use a volume created with Cloudstor to mount a persistent data volume. The volume stays attached to the Swarm tasks no matter which Swarm node they get scheduled on or migrated to. Cloudstor relies on shared storage infrastructure provided by Azure (specifically File Storage shares exposed over SMB) to allow Swarm tasks to create/mount their persistent volumes on any node in the Swarm.

> **Note:** Direct attached storage, which is used to satisfy very low latency / high IOPS requirements, is not yet supported.

You can share the same volume among tasks running the same service, or you can use a unique volume for each task.

After initializing or joining a Docker Swarm cluster, connect to any Swarm manager using SSH. Verify that the Cloudstor plugin is already installed and configured for the stack or resource group:

```
$ docker plugin ls

ID                  NAME                DESCRIPTION                         ENABLED
f416c95c0dcc        cloudstor:azure     cloud storage plugin for Docker     true
```

The following examples show how to create Swarm services that require data persistence using the --mount flag and specifying Cloudstor as the volume driver.

Share the Same Volume Among Tasks

If you specify a static value for the `source` option to the `--mountflag`, a single volume is shared among the tasks participating in the service.

Cloudstor volumes can be created to share access to persistent data across all tasks in a Swarm service running in multiple nodes. Example:

```
$ docker service create \
  --replicas 5 \
  --name ping1 \
  --mount type=volume,volume-driver=cloudstor:azure,source=sharedvol1,destination=/shareddata \
  alpine ping docker.com
```

In this example, all replicas/tasks of the service `ping1` share the same persistent volume `sharedvol1` mounted at `/shareddata` path within the container. Docker takes care of interacting with the Cloudstor plugin to ensure that the common backing store is mounted on all nodes in the Swarm where service tasks are scheduled. Your application needs to be designed to ensure that tasks do not write concurrently on the same file at the same time, to protect against data corruption.

You can verify that the same volume is shared among all the tasks by logging into one of the task containers, writing a file under `/shareddata/`, and logging into another task container to verify that the file is available there as well.

Use a Unique Volume per Task

You can use a templatized notation with the `docker service create` CLI to create and mount a unique Cloudstor volume for each task in a Swarm service.

It is possible to use the templatized notation to indicate to Docker Swarm that a unique Cloudstor volume be created and mounted for each replica/task of a service. This may be useful if the tasks write to the same file under the same path which may lead to corruption in case of shared storage. Example:

```
$ docker service create \
  --replicas 5 \
  --name ping2 \
  --mount type=volume,volume-driver=cloudstor:azure,source={{.Service.Name}}-{{.Task.Slot}}-
vol,destination=/mydata \
  alpine ping docker.com
```

A unique volume is created and mounted for each task participating in the `ping2` service. Each task mounts its own volume at `/mydata/` and all files under that mount point are unique to the task mounting the volume.

If a task is rescheduled on a different node after the volume is created and mounted, Docker interacts with the Cloudstor plugin to create and mount the volume corresponding to the task on the new node for the task.

It is highly recommended that you use the `.Task.Slot` template to ensure that task `N` always gets access to volume `N`, no matter which node it is executing on/scheduled to.

Volume Options

Cloudstor creates a new File Share in Azure File Storage for each volume and uses SMB to mount these File Shares. SMB has limited compatibility with generic Unix file ownership and permissions-related operations. Certain containers, such as `jenkins` and `gitlab`, define specific users and groups which perform different file operations. These types of workloads require the Cloudstor volume to be mounted with the UID/GID of the user

specified in the Dockerfile or setup scripts. Cloudstor allows for this scenario and provides greater control over default file permissions by exposing the following volume options that map to SMB parameters used for mounting the backing file share.

Option	Description	Default
`uid`	User ID that owns all files on the volume.	`0 = root`
`gid`	Group ID that owns all files on the volume.	`0 = root`
`filemode`	Permissions for all files on the volume.	`0777`
`dirmode`	Permissions for all directories on the volume.	`0777`
`share`	Name to associate with file share so that the share can be easily located in the Azure Storage Account.	MD5 hash of volume name

This example sets `uid` to `1000` and `share` to `sharedvol` rather than an md5 hash:

```
$ docker service create \
  --replicas 5 \
  --name ping1 \
  --mount type=volume,volume-driver=cloudstor:azure,source=sharedvol1,destination=/shareddata,volume-
opt=uid=1000,volume-opt=share=sharedvol \
  alpine ping docker.com
```

List or Remove Volumes Created by Cloudstor

Use `docker volume ls` on any node to enumerate all volumes created by Cloudstor across the Swarm.

Use `docker volume rm [volume name]` to remove a Cloudstor volume from any node. If you remove a volume from one node, make sure it is not being used by another active node, since those tasks/containers in another node lose access to their data.

Use a Different Storage Endpoint

To use a different storage endpoint, such as `core.cloudapi.de`, specify it when you install the plugin:

```
$ docker plugin install --alias cloudstor:azure \
  --grant-all-permissions docker4x/cloudstor:1.0.0 \
  CLOUD_PLATFORM=AZURE \
  AZURE_STORAGE_ACCOUNT_KEY="$SA_KEY" \
  AZURE_STORAGE_ACCOUNT="$SWARM_INFO_STORAGE_ACCOUNT" \
  AZURE_STORAGE_ENDPOINT="core.cloudapi.de" \
  DEBUG=1
```

By default, the Public Azure Storage endpoint is used.

Storage Ecosystem

Docker Enterprise storage capabilities can be extended through the use of plugins. There are a number of Docker storage plugins that support third party storage devices and external management platforms.

Microsoft Azure provides a number of options for storage, including Managed Disks (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/managed-disks-overview) and Premium Storage (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/premium-storage).

For recommendations on using Premium Storage, refer to the Azure Premium Storage Solution Brief for Docker Enterprise 17.06 (https://success.docker.com/article/azure-premium-storage).

Networking Ecosystem

The Docker engine provides the capability to extend the networking functionality through the use of plugins. There are a number of Docker networking plugins that support third party networking devices and external management platforms, these plugins can be found in the Docker Store (https://store.docker.com/search?category=network&q=&type=plugin).

> Ensure you are familiar with the best practices for Designing Scalable, Portable Docker container Networks (https://success.docker.com/article/networking).

Azure Native Networking Options

When you deploy the Docker Enterprise platform to Azure you should consider using native Azure Networking components. These include:

- Virtual Networks and Subnets Load Balancers Network Security Groups

You may customize and extend these capabilities to meet networking requirements for your deployment. Here are some examples:

- You might wish to create additional subnets for the virtual network, and split the nodes into different subnets. Although this may be useful for separating roles (e.g. by applying different Network Security Groups to different subnets), you will need to configure your Swarm networking further to reflect the new network structure. Ensure you are familiar with the recommendations for Designing Scalable, Portable Docker container Networks (https://success.docker.com/article/networking) first.
- To restrict access to nodes, you should consider creating Network Security Groups (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg) to restrict access to key nodes such as UCP managers.
- In some scenarios, your Docker Enterprise platform will need to be directly connected to your on-premises network. One example is where public internet access to the platform is not permitted. In this case, you can link the virtual network in which the Docker Enterprise platform is deployed to your on-premises network by either deploying a VPN Gateway (https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways) or an ExpressRoute Circuit (https://docs.microsoft.com/en-gb/azure/expressroute/expressroute-introduction).
- When deploying multiple Docker Enterprise platforms across several regions, you could make use of Azure Traffic Manager (https://docs.microsoft.com/en-gb/azure/traffic-manager/traffic-manager-overview) to distribute traffic to the right region. An example scenario is when provisioning a Primary cluster and a Failover cluster. Traffic Manager can redirect users to the Failover cluster should the region in which the Primary cluster is located become inaccessible.
- Azure supports a number of network virtual appliances (https://azure.microsoft.com/en-gb/solutions/network-appliances/) from a number of vendors. Before including one of these appliances in your design, consult with the vendor to understand any supportability requirements for your scenario.

Scaling Considerations

Deploying Docker Enterprise on a Public Cloud provider such as Microsoft Azure makes it possible to rapidly scale by providing more compute, networking, and storage resources to the Docker Enterprise platform.

Best practices for running Docker Enterprise at scale are available in Running Docker Enterprise at Scale (https://success.docker.com/article/running-docker-ee-at-scale).

You may wish to extend the deployment to further automate deployment by deploying some node types (e.g. Linux or Windows worker nodes) within a Virtual Machine Scale Set (https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview). This would allow scaling each type of node individually, e.g. by scaling the number of Windows nodes up if more Windows services are deployed. Since the current templates do not set up and configure Virtual Machine Scale Sets, you will need to create and configure your own, and set the appropriate autoscale routes (https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview).

Troubleshooting Docker Enterprise

The following articles describe common troubleshooting issues with Docker Enterprise:

- Swarm Troubleshooting Methodology (https://success.docker.com/article/Swarm-troubleshooting-methodology)
- Troubleshooting container Networking (https://success.docker.com/article/troubleshooting-container-networking)
- Best Docker Support Resources for Troubleshooting (https://success.docker.com/article/best-support-resources)
- Monitor the Swarm status (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-troubleshoot/)
- Troubleshoot UCP node states (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-troubleshoot/troubleshoot-node-messages/)
- Troubleshoot your Swarm (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-troubleshoot/troubleshoot-with-logs/)
- Troubleshoot Swarm configurations (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-troubleshoot/troubleshoot-configurations/)
- Troubleshooting External Certificates for UCP/DTR (https://success.docker.com/article/troubleshooting-external-certificates-for-ucp-dtr)

Troubleshooting Docker Enterprise Environment on Azure

The following articles can help troubleshoot issues that may occur with the Docker Enterprise infrastructure deployed on Azure:

- Troubleshoot attached VHDs on Azure Windows virtual machines (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/troubleshoot-vhds)
- Troubleshoot deploying Linux virtual machine issues in Azure (https://docs.microsoft.com/en-us/azure/virtual-machines/linux/troubleshoot-deploy-vm)
- Troubleshoot SSH connections to an Azure Linux VM that fails, errors out, or is refused (https://docs.microsoft.com/en-us/azure/virtual-machines/linux/troubleshoot-ssh-connection)
- Troubleshoot Remote Desktop connections to an Azure virtual machine (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/troubleshoot-rdp-connection)
- Troubleshooting connectivity problems between Azure VMs (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-troubleshoot-connectivity-problem-between-vms)
- Monitor, diagnose, and troubleshoot Microsoft Azure Storage (https://docs.microsoft.com/en-us/azure/storage/common/storage-monitoring-diagnosing-troubleshooting)

- Troubleshoot Azure Load Balancer (https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-troubleshoot)
- Troubleshoot Network Security Groups using the Azure Portal (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-nsg-troubleshoot-portal)
- Troubleshoot routes using the Azure Portal (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-routes-troubleshoot-portal)
- Network Watcher Connection Troubleshoot (https://azure.microsoft.com/en-us/blog/network-watcher-connection-troubleshoot-now-generally-available/)
- Diagnose and resolve issues with Azure Troubleshooting (https://azure.microsoft.com/en-gb/blog/azure-troubleshoot-diagonse-resolve-issues/)
- Understanding Azure troubleshooting and support (https://azure.microsoft.com/en-gb/blog/understanding-azure-troubleshooting-and-support/)
  ve..

The azure-disk provisioner accepts a `storageaccounttype` and `kind` parameters determine the type of volume. Other values like `Premium_LRS` for `storageaccounttype` or `dedicated` for `kind` may be specified. If you choose to use a different parameters type, be sure to update the metadata name, so the type of volume is clear to consumers of this storage class. Additional parameters and requirements are defined in the documentation for the [New Azure Disk Storage Class] (https://kubernetes.io/docs/concepts/storage/storage-classes/#azure-disk)

The reclaimPolicy defines how the volumes will be reclaimed by Kubernetes, when the pod is destroyed. The value can be either `Delete`, `Recycle`, or `Retain`. If no reclaimPolicy is specified when a StorageClass object is created, it will default to `Delete`. For more details on the lifecycle of a volume and claim, check out the Kubernetes documentation (https://kubernetes.io/docs/concepts/storage/persistent-volumes/#lifecycle-of-a-volume-and-claim)

Kubernetes PersistentVolumeClaim for Azure Disk

Below is an example of a `PersistentVolumeClaim` for dynamically creating volumes:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim-one-gig
spec:
  storageClassName: azure-disk-standard-lrs
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

The kind of object for this Kubernetes resource is `PersistentVolumeClaim`. The `apiVersion` is set to `v1` since this will be using the built-in APIs. Additionally, this `PersistentVolumeClaim` will be referenced by a metadata name: `claim-one-gig`.

The `spec` for the `PersistentVolumeClaim` references the `storageClassName` to use for creating volumes. The accessModes indicate how many nodes may read and write to the volume. Azure Disk supports the `ReadWriteOnce` access mode, only. Therefore, the volumes created by this `PersistentVolumeClaim` can be read from and written to by a single node. Finally, the `PersistentVolumeClaim` will request 1-gigabyte of storage from Azure Disk.

There are many options available, and for more details on them review the Kubernetes Documentation on Persistent Volume Claims (https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims)

Kubernetes Pod for Azure Disk

Below is an example of a `Pod` which will use the `PersistentVolumeClaim`:

```
kind: Pod
apiVersion: v1
metadata:
  name: example-pod
spec:
  volumes:
    - name: one-gigabyte
      persistentVolumeClaim:
       claimName:  claim-one-gig
  containers:
    - name: example-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: one-gigabyte
```

First, the kind of object for this Kubernetes resource is `Pod`. The `apiVersion` is set to `v1` since this will be using the built-in APIs. Additionally, this `Pod` will be named `example-pod`. We'll describe the `spec` for the `example-pod` in two parts, the `volumes` and the `containers`

The `volumes` for the `example-pod` references the `PersistentVolumeClaim` `claim-one-gig`, to enable the `Pod` to request 1-gigabyte volumes from Azure Disk. The `volumeMounts` section of the `containers` `spec` will describe where the newly created volume will be mounted.

The `containers` for the `example-pod` will create a container named `example-container`, which will contain the nginx `image` and expose port 80. At last, we see that the `volume` named `one-gigabyte` will be mounted to the container at the `mountPath` inside the container: `"/usr/share/nginx/html"`.

Networking Ecosystem

The Docker engine provides the capability to extend the networking functionality through the use of plugins. There are a number of Docker networking plugins that support third party networking devices and external management platforms, these plugins can be found in the Docker Store (https://store.docker.com/search?category=network&q=&type=plugin).

> Ensure you are familiar with the best practices for Designing Scalable, Portable Docker container Networks (https://success.docker.com/article/networking).

Azure Native Networking Options

When you deploy the Docker Enterprise platform to Azure you should consider using native Azure Networking components. These include:

- Virtual Networks and Subnets Load Balancers Network Security Groups

You may customize and extend these capabilities to meet networking requirements for your deployment. Here are some examples:

- You might wish to create additional subnets for the virtual network, and split the nodes into different subnets. Although this may be useful for separating roles (e.g. by applying different Network Security Groups to different subnets), you will need to configure your Swarm networking further to reflect the new network structure. Ensure you are familiar with the recommendations for Designing Scalable, Portable Docker container Networks (https://success.docker.com/article/networking) first.

- To restrict access to nodes, you should consider creating Network Security Groups (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg) to restrict access to key nodes such as UCP managers.

- In some scenarios, your Docker Enterprise platform will need to be directly connected to your on-premises network. One example is where public internet access to the platform is not permitted. In this case, you can link the virtual network in which the Docker Enterprise platform is deployed to your on-premises network by either deploying a VPN Gateway (https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways) or an ExpressRoute Circuit (https://docs.microsoft.com/en-gb/azure/expressroute/expressroute-introduction).

- When deploying multiple Docker Enterprise platforms across several regions, you could make use of Azure Traffic Manager (https://docs.microsoft.com/en-gb/azure/traffic-manager/traffic-manager-overview) to distribute traffic to the right region. An example scenario is when provisioning a Primary cluster and a Failover cluster. Traffic Manager can redirect users to the Failover cluster should the region in which the Primary cluster is located become inaccessible.

- Azure supports a number of network virtual appliances (https://azure.microsoft.com/en-gb/solutions/network-appliances/) from a number of vendors. Before including one of these appliances in your design, consult with the vendor to understand any supportability requirements for your scenario.

Scaling Considerations

Deploying Docker Enterprise on a Public Cloud provider such as Microsoft Azure makes it possible to rapidly scale by providing more compute, networking, and storage resources to the Docker Enterprise platform.

Best practices for running Docker Enterprise at scale are available in Running Docker Enterprise at Scale (https://success.docker.com/article/running-docker-ee-at-scale).

You may wish to extend the deployment to further automate deployment by deploying some node types (e.g. Linux or Windows worker nodes) within a Virtual Machine Scale Set (https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview). This would allow scaling each type of node individually, e.g. by scaling the number of Windows nodes up if more Windows services are deployed. Since the current templates do not set up and configure Virtual Machine Scale Sets, you will need to create and configure your own, and set the appropriate autoscale routes (https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview).

Troubleshooting Docker Enterprise

The following articles describe common troubleshooting issues with Docker Enterprise:

- Swarm Troubleshooting Methodology (https://success.docker.com/article/Swarm-troubleshooting-methodology)
- Troubleshooting container Networking (https://success.docker.com/article/troubleshooting-container-networking)
- Best Docker Support Resources for Troubleshooting (https://success.docker.com/article/best-support-resources)

- Monitor the Swarm status (https://docs.docker.com/datacenter/ucp/3.0/guides/admin/monitor-and-troubleshoot/)
- Troubleshoot UCP node states (https://docs.docker.com/datacenter/ucp/3.0/guides/admin/monitor-and-troubleshoot/troubleshoot-node-messages/)
- Troubleshoot your Swarm (https://docs.docker.com/datacenter/ucp/3.0/guides/admin/monitor-and-troubleshoot/troubleshoot-with-logs/)
- Troubleshoot Swarm configurations (https://docs.docker.com/datacenter/ucp/3.0/guides/admin/monitor-and-troubleshoot/troubleshoot-configurations/)
- Troubleshooting External Certificates for UCP/DTR (https://success.docker.com/article/troubleshooting-external-certificates-for-ucp-dtr)

Troubleshooting Docker Enterprise Environment on Azure

The following articles can help troubleshoot issues that may occur with the Docker Enterprise infrastructure deployed on Azure:

- Troubleshoot attached VHDs on Azure Windows virtual machines (https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/troubleshoot-vhds)
- Troubleshoot deploying Linux virtual machine issues in Azure (https://docs.microsoft.com/en-us/azure/virtual-machines/linux/troubleshoot-deploy-vm)
- Troubleshoot SSH connections to an Azure Linux VM that fails, errors out, or is refused (https://docs.microsoft.com/en-us/azure/virtual-machines/linux/troubleshoot-ssh-connection)
- Troubleshoot Remote Desktop connections to an Azure virtual machine (https://docs.microsoft.com/en-us/azure/virtual-machines/windows/troubleshoot-rdp-connection)
- Troubleshooting connectivity problems between Azure VMs (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-troubleshoot-connectivity-problem-between-vms)
- Monitor, diagnose, and troubleshoot Microsoft Azure Storage (https://docs.microsoft.com/en-us/azure/storage/common/storage-monitoring-diagnosing-troubleshooting)
- Troubleshoot Azure Load Balancer (https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-troubleshoot)
- Troubleshoot Network Security Groups using the Azure Portal (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-nsg-troubleshoot-portal)
- Troubleshoot routes using the Azure Portal (https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-routes-troubleshoot-portal)
- Network Watcher Connection Troubleshoot (https://azure.microsoft.com/en-us/blog/network-watcher-connection-troubleshoot-now-generally-available/)
- Diagnose and resolve issues with Azure Troubleshooting (https://azure.microsoft.com/en-gb/blog/azure-troubleshoot-diagonse-resolve-issues/)
- Understanding Azure troubleshooting and support (https://azure.microsoft.com/en-gb/blog/understanding-azure-troubleshooting-and-support/)

lb-1 : ok=42 changed=20 unreachable=0 failed=0
MTA-mgr-1 : ok=96 changed=42 unreachable=0 failed=0
MTA-mgr-2 : ok=82 changed=36 unreachable=0 failed=0
MTA-mgr-3 : ok=82 changed=36 unreachable=0 failed=0
MTA-ucp-lb-1 : ok=42 changed=20 unreachable=0 failed=0
MTA-wrk-1 : ok=68 changed=29 unreachable=0 failed=0
MTA-wwrk-1 : ok=40 changed=16 unreachable=0 failed=0

```
In the event of any Ansible tasks failing, please consult the [Troubleshooting section]
(https://success.docker.com/article/certified-infrastructures-vmware-vsphere#troubleshooting).
```

```
Upon completion, accessing the IP address of any Universal Control Plane virtual machine should present
the user interface, allowing you to confirm a successful deployment.

## Upgrading a Docker EE Deployment

To update Docker EE, simply update the variables you want (e.g. `docker_ee_version`), then run
`ansible-playbook update.yml`.

If you want to update a specific part (e.g DTR), simply add `-t dtr`. Available tags are:

- `engine`
- `docker-ee` (alias for engine)
- `ucp`
- `dtr`

Make sure versions are compatible by consulting the [compatibility matrix]
(https://success.docker.com/article/Compatibility_Matrix).

## Validate Docker EE Deployment

Once Docker Enterprise Edition has been successfully deployed the deployment can be validated in a
number of ways as shown in the next sections.

### Validate with UCP

To validate the Docker EE deployment, connect to UCP/DTR through the address of either the load
balancer or any of the UCP nodes. Once logged in, any issues with the platform will be displayed on the
dashboard interface.

### Validate with the CLI

Using the `ucp-bundle` a user can easily validate the deployment through the following commands:

- `docker node ls` - Ensure that all nodes are present
- `docker ps | grep unhealthy` - Ensure that no containers are unhealthy

### Example Acceptance into Service (AIS) Tests

This table provides a number of example AIS tests that should be performed to ensure that the Docker
Enterprise Edition platform has been deployed correctly.

| AIS Test                         | Expected Outcome                          |
|:---------------------------------|:------------------------------------------|
| UCP VM Reboot                    | Node rejoins cluster                      |
| DTR VM Reboot                    | Node rejoins cluster                      |
| Docker EE VM Engine Reboot       | Node rejoins cluster                      |
| UCP Cluster all powered off/on   | UCP becomes available as VMs restart      |
| DTR Cluster all powered off/on   | DTR becomes available as VMs restart      |
| all Docker EE VMs powered off/on | Docker EE becomes availabe once VMs restart |

## Operational Management

Now that Docker EE is deployed on Azure, the following sections provide guidance on managing your
deployment.

### Connecting to your manager nodes using SSH
```

```
First, you obtain the public IP address for a manager node. Any manager node can be used for
administrating the swarm. You can find the Public IP and SSH ports for each Manager nodes by opening
the relevant virtual machine on Azure.

> Follow this [Troubleshooting Guide](https://docs.microsoft.com/en-us/azure/virtual-
machines/linux/troubleshoot-ssh-connection) if you have trouble connecting to a virtual machine on
Azure.

Obtain the public IP and/or port for the manager node use the SSH key you provided in
```terraform.tfvars``` to begin administrating your swarm and the unique port associated with a
manager:
```

```
$ ssh -i <path-to-ssh-key> -p <ssh-port> docker@
Welcome to Docker!
```

```
You can also tunnel the Docker socket over SSH to remotely run commands on the cluster (requires
[OpenSSH 6.7](https://lwn.net/Articles/609321/) or later):
```

```
$ ssh -i <path-to-ssh-key> -p <ssh-port> docker@
Welcome to Docker!
```

```
Once you are logged into the container you can run Docker commands on the swarm:
```

```
$ docker info
$ docker node ls
```

```
You can also tunnel the Docker socket over SSH to remotely run commands on the cluster (requires
[OpenSSH 6.7](https://lwn.net/Articles/609321/) or later):
```

```
$ ssh -i <path-to-ssh-key> -p <ssh-port> -fNL localhost:2374:/var/run/docker.sock docker@<ssh-host>
$ docker -H localhost:2374 info
```
```

```
If you don’t want to pass ```-H``` when using the tunnel, you can set the ```DOCKER_HOST``` environment variable to point to the localhost tunnel opening.

### Connecting to your Linux worker nodes using SSH

The Linux worker nodes have SSH enabled. SSH access is possible to the worker nodes from the public Internet directly. Follow the instructions above to connect to a Linux worker as you would to a manager.

If you set any Network Security Groups that prohibit SSH connections from the public Internet into Linux workers, you will need to access worker nodes by first connecting to a manager node (see above), or a separate "jump box", and then ```ssh``` to the worker node, over the private network. Make sure you have SSH agent forwarding enabled (see below). If you run the docker node ls command you can see the full list of nodes in your swarm. You can then ```ssh docker@<worker-host>``` to get access to that node.

SSH agent forwarding allows you to forward along your ssh keys when connecting from one node to another. This eliminates the need for installing your private key on all nodes you might want to connect from.

You can use this feature to SSH into worker nodes from a manager node without installing keys directly on the manager.

If your haven’t added your ssh key to the ```ssh-agent``` you also need to do this first.

To see the keys in the agent already, run:
```

```
$ ssh-add -L
```

```
If you don’t see your key, add it like this.
```

```
$ ssh-add ~/.ssh/your_key
```

```
On Mac OS X, the ```ssh-agent``` forgets this key, once it gets restarted. But you can import your SSH key into your Keychain like this so your key can survive restarts.
```

```
$ ssh-add -K ~/.ssh/your_key
```

```
You can then enable SSH forwarding per-session using the ```-A``` flag for the ssh command.

Connecting to the Manager.
```

```
$ ssh -A docker@
```

```
To always have it turned on for a given host, you can edit your ssh config file (```/etc/ssh_config```, ```~/.ssh/config```, etc) to add the ```ForwardAgent yes``` option.

Example configuration:
```

Host manager0
HostName
ForwardAgent yes

```
To SSH in to the manager with the above settings:
```

$ ssh docker@manager0

```
### Connecting to your Windows worker nodes using RDP ###

The Windows worker nodes have RDP enabled. RDP access is possible to the worker nodes from the public Internet. Follow [these instructions](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/connect-logon) to connect to a Windows worker on Docker EE using RDP.

If you set any Network Security Groups that prohibit RDP connections from the public Internet into Windows workers, you will need to access worker nodes by first connecting to a manager node (see above), or a separate "jump box" over ```ssh```, establish a SSH tunnel and then use RDP to connect to the worker node over the SSH tunnel.

To get started, first login to a manager node and determine the private IP address of the Windows worker VM:
```

$ docker node inspect <windows-worker-node-id> | jq -r ".[0].Status.Addr"

```
Next, in your local machine, establish the SSH tunnel to the manager for the RDP connection to the worker. The local port can be any free port and typically a high value, such as 9001.
```

$ ssh -L <local-port>:<windows-worker-ip>:3389 -p <manager-ssh-port> docker@<manager-ip>

```
Finally you can use a RDP client on your local machine to connect to `local-host:local-port` and the connection is forwarded to the RDP server running in the Windows worker node over the SSH tunnel created above.

### Installation of Docker EE Licenses

The licenses for Docker Enterprise Edition can be applied automatically through the use of the templated installation procedure described above, however if licenses aren't available during the initial deployment they can be applied afterwards. In order to license the installation go to <https://store.docker.com/my-content>, and download your Docker license key file.

In the UCP web UI, log in with administrator credentials and navigate to the **Admin Settings** page.
```

```
In the left pane, click **License** and click **Upload License**. The license refreshes immediately,
and you don't need to click Save.

![](https://docs.docker.com/datacenter/ucp/2.2/guides/images/license-ucp-2.png)

### Setup of Authentication

Docker supports the use of LDAP (Lightweight Directory Access Protocol) to provide integration with
existing authentication platforms such as Active Directory. Docker UCP integrates with LDAP directory
services, so that you can manage users and groups from your organization's directory and it will
automatically propagate that information to UCP and DTR.

If you enable LDAP, UCP uses a remote directory server to create users automatically, and all logins
are forwarded to the directory server.

When you switch from built-in authentication to LDAP authentication, all manually created users whose
usernames don't match any LDAP search results are still available.

You control how UCP integrates with LDAP by creating searches for users. You can specify multiple
search configurations, and you can specify multiple LDAP servers to integrate with. Searches start with
the `Base DN`, which is the distinguished name of the node in the LDAP directory tree where the search
starts looking for users.

Access LDAP settings by navigating to the **Authentication & Authorization** page in the UCP web UI.
There are two sections for controlling LDAP searches and servers.

- **LDAP user search configurations:** This is the section of the Authentication & Authorization page
where you specify search parameters, like `Base DN`, `scope`, `filter`, the `username` attribute, and
the `full name` attribute. These searches are stored in a list, and the ordering may be important,
depending on your search configuration.
- **LDAP server:** This is the section where you specify the URL of an LDAP server, TLS configuration,
and credentials for doing the search requests. Also, you provide a domain for all servers but the first
one. The first server is considered the default domain server. Any others are associated with the
domain that you specify in the page.

### Recommended APIs to Monitor

To programmatically integrate Docker UCP monitoring with existing solutions there are a number of API
endpoints that can be accessed in order to determine both the state of the running services along with
the Docker EE platform itself.

**Determine the state of the platform**

To get a simple overview of the platform the URI endpoint `/_ping` will return the HTTP status code of
the Docker UCP components.


| HTTP Status Code         | Reason                         |
|:-------------------------|:-------------------------------|
| 200                      | **Success**, manager healthy   |
| 500                      | **Failure**, manager unhealthy |
| Success, manager healthy | default                        |

**Return a list of events**

To return a list of events that have happened on the Docker EE platform the endpoint `/events` returns
```

```
a parsable JSON response that can be evaluated to determine what has happened on the Docker EE
platform.

**Examine a particular service**

The endpoint `/services/{id}` allows an end-user to programmatically evaluate the health and status of
a particular service that is running on the Docker EE platform.

Further information about the API endpoints are available at [docs.docker.com]
(https://docs.docker.com/datacenter/ucp/2.2/reference/api/#/).

### Log Collection and Frequency

Part of deploying Docker EE is configuring logging. Logging provides visibility into the health and
performance of both the platform and the services that run on top of it. It is recommended that you
send logs to a log aggreagator that provides ease of search as well as levels of observibility that
allow for quick troubleshooting of instability issues.

A reference architecture that details logging design and best practices is available in  [Docker
Logging Design and Best Practices](https://success.docker.com/article/docker-reference-architecture-
docker-logging-design-and-best-practices).

Additionally, Docker provides a number of solutions briefs for integrating Docker logging with other
platforms:

- [Splunk Enterprise Solution Brief for Docker EE 17.06](https://success.docker.com/article/splunk-
logging/)
- [Logging with Elasticsearch, Logstash, and Kibana Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/elasticsearch-logstash-kibana-logging/)

### Upgrading Individual Components of Docker EE

> Before upgrading, make sure you [create a backup](https://success.docker.com/article/backup-restore-
best-practices). This makes it possible to recover if anything goes wrong during the upgrade.

Docker Enterprise Edition has the capability to "dial home" to docker.com and determine if a new
version of Docker UCP or DTR is available; this requires Internet access. In the event that an update
is available, a banner will appear in the UI alerting the user to the newly released version of a
particular component of Docker EE.

When a new version of Docker Enterprise Edition is available, an end-user has the option of upgrading
directly through the user interface by clicking the banner alert displaying the new version and then
following the prompts to update. Alternatively a user can download the offline bundle(s) and follow the
CLI steps to perform a command line update to Docker EE components. Use the documentation for the
method you prefer:

- [Perform updates in the UI](https://docs.docker.com/ee/upgrade/)
- [Perform updates from the command line]
(https://docs.docker.com/datacenter/ucp/2.2/reference/cli/upgrade/)

> **Note:** Before starting any upgrade tasks, it is recommended that you study the upgrade matrix to
ensure that the correct upgrade path is followed, the matrix is available on [success.docker.com]
(https://success.docker.com/article/compatibility-matrix).

### Monitoring Docker EE with Prometheus

Currently only the Docker Engines can be monitored with Prometheus and not the management platform as a
```

```
whole. To monitor the Docker EE engines, their configuration will need modifying so that the engines
will expose the metrics. The instructions for enabling Prometheus metrics are available on
[docs.docker.com](https://docs.docker.com/config/thirdparty/prometheus/) as well as in
[Grafana/Prometheus Monitoring Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/grafana-prometheus-monitoring/).

## Persistent Storage

The majority of applications have some component that requires data to persist for a myriad of reasons
such as application resiliency in the event of application failure to the size of data to lookup. A
user of Microsoft Azure has a number of options that can be considered to provide a persistent storage
option for services and applications that run on the Docker Enterprise Edition platform.

### Locally Presented Storage

A virtual machine that is going to host Docker workloads can be configured to have a larger virtual
disk assigned to it in order to provide more local persistent storage that can be mapped into a
container when started. However, creating a second virtual disk and attaching it to the virtual machine
to be consumed as peristent storage has the following benefits:

- Can place the application disk on an alternative datastore that has faster backing storage (SSD/NVMe)
- In the event that the application disk reaches capacity, it won't stop the operating system from
functioning (allows easier problem diagnosis). It also simplifies the resizing of the disk as it isn't
shared with other filesystem partitions, etc.
- In the event of operating system corruption, it's quick and easy to remap the application disk to a
healthy Docker node and reschedule the workload to run there.

For recommendations on using Premium Storage, refer to the [Azure Premium Storage Solution Brief for
Docker EE 17.06](https://success.docker.com/article/azure-premium-storage).

### Cloudstor

Cloudstor is a modern volume plugin built by Docker. It comes pre-installed and pre-configured in
Docker Swarms deployed on Docker Certified Infrastructure for Microsoft Azure. Docker swarm mode tasks
and regular Docker containers can use a volume created with Cloudstor to mount a persistent data
volume. The volume stays attached to the swarm tasks no matter which swarm node they get scheduled on
or migrated to. Cloudstor relies on shared storage infrastructure provided by Azure (specifically File
Storage shares exposed over SMB) to allow swarm tasks to create/mount their persistent volumes on any
node in the swarm.

> **Note:** Direct attached storage, which is used to satisfy very low latency / high IOPS
requirements, is not yet supported.

You can share the same volume among tasks running the same service, or you can use a unique volume for
each task.

After initializing or joining a swarm on Docker Certified Infrastructure for Microsoft Azure, connect
to any swarm manager using SSH. Verify that the CloudStor plugin is already installed and configured
for the stack or resource group:
```

```
$ docker plugin ls

ID NAME DESCRIPTION ENABLED
f416c95c0dcc cloudstor:azure cloud storage plugin for Docker true
```

The following examples show how to create swarm services that require data persistence using the --mount flag and specifying Cloudstor as the volume driver.

Share the same volume among tasks
If you specify a static value for the ```source``` option to the ```--mount flag```, a single volume is shared among the tasks participating in the service.

Cloudstor volumes can be created to share access to persistent data across all tasks in a swarm service running in multiple nodes. Example:

```
$ docker service create
--replicas 5
--name ping1
--mount type=volume,volume-driver=cloudstor:azure,source=sharedvol1,destination=/shareddata
alpine ping docker.com
```

In this example, all replicas/tasks of the service ```ping1``` share the same persistent volume ```sharedvol1``` mounted at ```/shareddata``` path within the container. Docker takes care of interacting with the Cloudstor plugin to ensure that the common backing store is mounted on all nodes in the swarm where service tasks are scheduled. Your application needs to be designed to ensure that tasks do not write concurrently on the same file at the same time, to protect against data corruption.

You can verify that the same volume is shared among all the tasks by logging into one of the task containers, writing a file under ```/shareddata/```, and logging into another task container to verify that the file is available there as well.

Use a unique volume per task
You can use a templatized notation with the ```docker service create``` CLI to create and mount a unique Cloudstor volume for each task in a swarm service.

It is possible to use the templatized notation to indicate to Docker Swarm that a unique Cloudstor volume be created and mounted for each replica/task of a service. This may be useful if the tasks write to the same file under the same path which may lead to corruption in case of shared storage. Example:

```
$ docker service create
--replicas 5
--name ping2
--mount type=volume,volume-driver=cloudstor:azure,source={{.Service.Name}}-{{.Task.Slot}}-
vol,destination=/mydata
alpine ping docker.com
```

A unique volume is created and mounted for each task participating in the ```ping2``` service. Each task mounts its own volume at ```/mydata/``` and all files under that mountpoint are unique to the task mounting the volume.

If a task is rescheduled on a different node after the volume is created and mounted, Docker interacts with the Cloudstor plugin to create and mount the volume corresponding to the task on the new node for the task.

It is highly recommended that you use the ```.Task.Slot``` template to ensure that task ```N``` always gets access to volume ```N```, no matter which node it is executing on/scheduled to.

Volume options
Cloudstor creates a new File Share in Azure File Storage for each volume and uses SMB to mount these File Shares. SMB has limited compatibility with generic Unix file ownership and permissions-related operations. Certain containers, such as ```jenkins``` and ```gitlab```, define specific users and groups which perform different file operations. These types of workloads require the Cloudstor volume to be mounted with the UID/GID of the user specified in the Dockerfile or setup scripts. Cloudstor allows for this scenario and provides greater control over default file permissions by exposing the following volume options that map to SMB parameters used for mounting the backing file share.

Option	Description	Default
``uid```	User ID that owns all files on the volume.	```0 = root```
``gid```	Group ID that owns all files on the volume.	```0 = root```
``filemode```	Permissions for all files on the volume.	```0777```
``dirmode```	Permissions for all directories on the volume.	```0777```
``share```	Name to associate with file share so that the share can be easily located in the Azure Storage Account.	MD5 hash of volume name

This example sets uid to 1000 and ```share``` to ```sharedvol``` rather than a md5 hash:

```
$ docker service create
--replicas 5
--name ping1
--mount type=volume,volume-driver=cloudstor:azure,source=sharedvol1,destination=/shareddata,volume-opt=uid=1000,volume-opt=share=sharedvol
alpine ping docker.com
```

```
#### List or Remove Volumes Created by Cloudstor ####
You can use ```docker volume ls``` on any node to enumerate all volumes created by Cloudstor across the swarm.

You can use ```docker volume rm [volume name]``` to remove a Cloudstor volume from any node. If you remove a volume from one node, make sure it is not being used by another active node, since those tasks/containers in another node lose access to their data.

#### Use a different storage endpoint ####
If you need to use a different storage endpoint, such as core.cloudapi.de, you can specify it when you install the plugin:
```

```
$ docker plugin install --alias cloudstor:azure
--grant-all-permissions docker4x/cloudstor:17.06.1-ce-azure1
CLOUD_PLATFORM=AZURE
AZURE_STORAGE_ACCOUNT_KEY="$SA_KEY"
AZURE_STORAGE_ACCOUNT="$SWARM_INFO_STORAGE_ACCOUNT"
AZURE_STORAGE_ENDPOINT="core.cloudapi.de"
DEBUG=1
```

```
By default, the Public Azure Storage endpoint is used.

### Storage Ecosystem

Docker EE storage capabilities can be extended through the use of plugins. There are a number of Docker storage plugins that support third party storage devices and external management platforms.

Microsoft Azure provides a number of options for storage, including [Managed Disks](https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/managed-disks-overview) and [Premium Storage](https://docs.microsoft.com/en-gb/azure/virtual-machines/windows/premium-storage).

For recommendations on using Premium Storage, refer to the [Azure Premium Storage Solution Brief for Docker EE 17.06](https://success.docker.com/article/azure-premium-storage).

## Container Networking

All virtual machines are connected through the use of a virtual network.
### Batteries Included, Networking Options

Docker provides an out-of-the box experience – it comes with everything needed to provide the best set of networking features for most workloads. The majority of features should just work with no configuration changes, however there are some networking features that require a security configuration change in order to utilize.

**Bridge**

The bridge driver creates a private network internal to the host so containers on this network can communicate. External access is granted by exposing ports to containers. Docker secures the network by managing rules that block connectivity between different Docker networks.

Behind the scenes, the Docker Engine creates the necessary Linux bridges, internal interfaces, iptables rules, and host routes to make this connectivity possible. In the example highlighted below, a Docker bridge network is created and two containers are attached to it. With no extra configuration the Docker
```

Engine does the necessary wiring, provides service discovery for the containers, and configures security rules to prevent communication to other networks. A built-in IPAM driver provides the container interfaces with private IP addresses from the subnet of the bridge network.

Overlay

The built-in Docker overlay network driver radically simplifies many of the complexities in multi-host networking. It is a swarm scope driver, which means that it operates across an entire Swarm or UCP cluster rather than individual hosts. With the overlay driver, multi-host networks are first-class citizens inside Docker without external provisioning or components. IPAM, service discovery, multi-host connectivity, encryption, and load balancing are built right in. For control, the overlay driver uses the encrypted Swarm control plane to manage large scale clusters at low convergence times.

The overlay driver utilizes an industry-standard VXLAN data plane that decouples the container network from the underlying physical network (the underlay). This has the advantage of providing maximum portability across various cloud and on-premises networks. Network policy, visibility, and security is controlled centrally through the Docker Universal Control Plane (UCP).

MACVLAN

The macvlan driver is the newest built-in network driver and offers several unique characteristics. It's a very lightweight driver, because rather than using any Linux bridging or port mapping, it connects container interfaces directly to host interfaces. Containers are addressed with routable IP addresses that are on the subnet of the external network.

As a result of routable IP addresses, containers communicate directly with resources that exist outside a Swarm cluster without the use of NAT and port mapping. This can aid in network visibility and troubleshooting. Additionally, the direct traffic path between containers and the host interface helps reduce latency. macvlan is a local scope network driver which is configured per-host. As a result, there are stricter dependencies between MACVLAN and external networks, which is both a constraint and an advantage that is different from overlay or bridge.

The macvlan driver uses the concept of a parent interface. This interface can be a host interface such as eth0, a sub-interface, or even a bonded host adaptor which bundles Ethernet interfaces into a single logical interface. A gateway address from the external network is required during MACVLAN network configuration, as a MACVLAN network is a L2 segment from the container to the network gateway. Like all Docker networks, MACVLAN networks are segmented from each other – providing access within a network, but not between networks.

Azure Native Networking Options

When you deploy the Docker EE platform to Azure using Docker Certified Infrastructure, native Azure Networking components are used. These include:

- Virtual Networks and Subnets
- Load Balancers
- Network Security Groups

You may customize and extend these capabilities to meet networking requirements for your deployment. Here are some examples:

- You might wish to create additional subnets for the virtual network, and split the nodes into different subnets. Although this may be useful for separating roles (e.g. by applying different Network Security Groups to different subnets), you will need to configure your Swarm networking further to reflect the new network stucture. Ensure you are familiar with the recommendations for Designing Scalable, [Portable Docker Container Networks](https://success.docker.com/article/networking) first.

- The Docker Certified Infrastructure deployment deploys a small number of Network Security Groups, mainly applying to Windows virtual machines. To further restrict access to nodes, you should consider creating additional [Network Security Groups](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg) to restrict access to key nodes such as UCP managers.

- In some scenarios, your Docker EE platform will need to be directly connected to your on-premises network. One example is where public internet access to the platform is not permitted. In this case, you can link the virtual network in which the Docker EE platform is deployed to your on-premises network by either deploying a [VPN Gateway](https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways) or an [ExpressRoute Circuit](https://docs.microsoft.com/en-gb/azure/expressroute/expressroute-introduction).

- When deploying multiple Docker EE platforms across several regions, you could make use of [Azure Traffic Manager](https://docs.microsoft.com/en-gb/azure/traffic-manager/traffic-manager-overview) to distribute traffic to the right region. An example scenario is when provisioning a Primary cluster and a Failover cluster. Traffic Manager can redirect users to the Failover cluster should the region in which the Primary cluster is located become inaccessible.

- Azure supports a number of [network virtual appliances](https://azure.microsoft.com/en-gb/solutions/network-appliances/) from a number of vendors. Before including one of these appliances in your design, consult with the vendor to understand any supportability requirements for your scenario.

Swarm Networking Considerations

Making the most of your Docker Enterprise Edition platform involves designing and maintaining a Swarm networking environment. Ensure you are familiar with the best practices for [Designing Scalable, Portable Docker Container Networks](https://success.docker.com/article/networking).

Networking Ecosystem

The Docker engine provides the capability to extend the networking functionality through the use of plugins. There are a number of Docker networking plugins that support third party networking devices and external management platforms, these plugins can be found in the [Docker Store](https://store.docker.com/search?category=network&q=&type=plugin).

Scaling Considerations

Deploying Docker EE on a Public Cloud provider such as Microsoft Azure makes it possible to rapidly scale by providing more compute, networking, and storage resources to the Docker EE platform.

The Docker certified infrastructure templates provide a simple and stable **method for scaling your Docker Enterprise Edition platform**. **In order to grow and extend your existing Docker EE platform quite simply do the following**:

1. Modify the `terraform.vars` and increase the amount of workers required (either windows or linux workers).
2. Use terraform with a `terraform apply` to provision the new worker virtual machines and update the inventory file.
3. Ansible can now use the updated inventory to provide the final configuration work to scale the Docker EE platform with the `ansible-playbook --private-key=<private key> -i inventory install.yml` command.

For further details on the commands and files used, consult the [Docker EE Infrastructure Provisioning](https://success.docker.com/api/asset/.%2Fpublish%2Fcertified-infrastructures-azure%2Fdockereeinfra) section. In addition, best practices for running Docker EE at scale are available in [Running Docker Enterprise Edition at Scale](https://success.docker.com/article/running-docker-ee-at-scale).

```
You may wish to extend the deployment to further automate deployment by deploying some node types (e.g. Linux or Windows worker nodes) within a [Virtual Machine Scale Set](https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview). This would allow scaling each type of node individually, e.g. by scaling the number of Windows nodes up if more Windows services are deployed. Since the current temlates do not set up and configure Virtual Machine Scale Sets, you will need to create and configure your own, and set the appropriate [autoscale routes](https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview).

## Troubleshooting

This section of the reference architecture contains a number of sections that can help when there are issues either during or after the deployment of Docker Enterprise Edition.

### Rolling Back an Installation

To uninstall Docker EE but keep the infrastructure in place, run the following:
```

ansible-playbook uninstall.yml

```
To completely destroy the Azure infrastructure, run:
```

terraform destroy

```
This will delete all infrastructure associated with your Docker EE deployment, including the **Resource Group** it is deployed in and all resources inside it.

Before running these command, ensure you have backed up any docker images you need to a Docker Trusted Registry outside the cluster. In addition, you may want [export an Azure Resource Manager template](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-export-template) to document and recreate Azure resources, particularly if you modified the default deployment in any way and may wish to do so again at a later point.

> To avoid accidental destruction of your cluster, you may wish to set [Azure resource locks](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-lock-resources) using the **CanNotDelete** lock level.

### Troubleshooting the Docker EE Installation Templates

In the event that there are any issues deploying Docker Enterprise, both Terraform and Ansible will display diagnostic output on the command line. This will identify where in the deployment a task has failed, which can then be used to inspect the failed task.

When running terraform and encountering an error, it will **pause** and exit at the failed task. If that happens,fix the error and re-run `terraform apply`. If the error is fixed, Terraform will continue on to the next task. Terraform errors are usually fairly descriptive as to why it failed.

Similarly, when an ansible playbook is run, it will verify whether a step needs to run before executing it (e.g. check whether Docker is installed before attempting to install). Therefore you can examine why an error occur, fix the error and re-run the ansible playbook, where it will resume.

Common errors to check when deployments fail on Azure include:

- Ensure you have created the Resource Group you are trying to deploy to.
- Ensure you have created a Service Principal and given it Contributor permissions to the Resource Group.
- Ensure you have chosen an Azure Region for deployment that is allowed for your subscription.
- Ensure your subscription has not gone over its allocated [limits](https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits)

Connectivity to the Docker EE virtual machines can be achieved by pinging every single node in the cluster, which can be achieved by running:
```

ansible linux -m ping all

```
and
```

ansible windows -m win_ping all

```
Additionally, logs are fetched after execution and will be stored under `$PWD/.logs/`. For example:
```

```
.logs/
├── host-1
│   ├── dtr
│   ├── engine.log
│   └── ucp
│   ├── ucp-controller.err.log
│   ├── ucp-reconcile.err.log
│   ├── ucp-swarm-manager.err.log
│   └── ucp-swarm-manager.out.log
└── host-2
├── dtr
│   ├── dtr-api-821e45555c0a.err.log
│   ├── dtr-garant-821e45555c0a.err.log
│   ├── dtr-garant-821e45555c0a.out.log
│   └── dtr-registry-821e45555c0a.err.log
├── engine.log
└── ucp
└── ucp-reconcile.err.log
```

```
You can fetch logs at any time by running:
```

ansible-playbook logs.yml

```
In the event that there are issues during the deployment of virtual machines through Terraform, it may
be required to log into VMware vCenter to determine the cause of the failure.

### Troubleshooting Docker EE

The following articles describe common troubleshooting issues with Docker EE:

- [Swarm Troubleshooting Methodology](https://success.docker.com/article/swarm-troubleshooting-
methodology)
- [Troubleshooting Container Networking](https://success.docker.com/article/troubleshooting-container-
networking)
- [Troubleshooting a UCP 2.2.x Cluster](https://success.docker.com/article/troubleshooting-a-ucp-22x-
cluster)
- [Troubleshooting a DTR 2.3.x Cluster](https://success.docker.com/article/troubleshooting-a-dtr-23x-
cluster)
- [Best Docker Support Resources for Troubleshooting](https://success.docker.com/article/best-support-
resources)
- [Monitor the swarm status](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-
troubleshoot/)
- [Troubleshoot UCP node states](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-
troubleshoot/troubleshoot-node-messages/)
- [Troubleshoot your swarm](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-and-
troubleshoot/troubleshoot-with-logs/)
- [Troubleshoot swarm configurations](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/monitor-
and-troubleshoot/troubleshoot-configurations/)
- [Troubleshooting External Certificates for UCP/DTR]
(https://success.docker.com/article/troubleshooting-external-certificates-for-ucp-dtr)

### Troubleshooting Docker EE Environment on Azure
```

```
The following articles can help troubleshoot issues that may occur with the Docker EE infrastrucutre
deployed on Azure:

- [Troubleshoot attached VHDs on Azure Windows virtual machines](https://docs.microsoft.com/en-
gb/azure/virtual-machines/windows/troubleshoot-vhds)
- [Troubleshoot deploying Linux virtual machine issues in Azure](https://docs.microsoft.com/en-
us/azure/virtual-machines/linux/troubleshoot-deploy-vm)
- [Troubleshoot SSH connections to an Azure Linux VM that fails, errors out, or is refused]
(https://docs.microsoft.com/en-us/azure/virtual-machines/linux/troubleshoot-ssh-connection)
- [Troubleshoot Remote Desktop connections to an Azure virtual machine](https://docs.microsoft.com/en-
us/azure/virtual-machines/windows/troubleshoot-rdp-connection)
- [Troubleshooting connectivity problems between Azure VMs](https://docs.microsoft.com/en-
us/azure/virtual-network/virtual-network-troubleshoot-connectivity-problem-between-vms)
- [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](https://docs.microsoft.com/en-
us/azure/storage/common/storage-monitoring-diagnosing-troubleshooting)
- [Troubleshoot Azure Load Balancer](https://docs.microsoft.com/en-us/azure/load-balancer/load-
balancer-troubleshoot)
- [Troubleshoot Network Security Groups using the Azure Portal](https://docs.microsoft.com/en-
us/azure/virtual-network/virtual-network-nsg-troubleshoot-portal)
- [Troubleshoot routes using the Azure Portal](https://docs.microsoft.com/en-us/azure/virtual-
network/virtual-network-routes-troubleshoot-portal)
- [Network Watcher Connection Troubleshoot](https://azure.microsoft.com/en-us/blog/network-watcher-
connection-troubleshoot-now-generally-available/)
- [Diagnose and resolve issues with Azure Troubleshooting](https://azure.microsoft.com/en-
gb/blog/azure-troubleshoot-diagonse-resolve-issues/)
- [Understanding Azure troubleshooting and support](https://azure.microsoft.com/en-
gb/blog/understanding-azure-troubleshooting-and-support/)

## Docker Solution Briefs

Docker Solution Briefs are short articles that aim to provide basic installation and validation details
for specific 3rd-party ecosystem solutions with the Docker Enterprise Edition (EE) platform. The
information on each of the following solutions is provided by Docker as a known, working configuration
at the time of publishing. Docker does not support these products.  Please contact the specific vendor
using their approved support methods if you have any questions with the particular solution.

- [Azure Premium Storage Solution Brief for Docker EE 17.06](https://success.docker.com/article/azure-
premium-storage)
- [Azure Load Balancer Solution Brief for Docker EE 17.06](https://success.docker.com/article/azure-
load-balancer)
- [Azure Application Gateway Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/azure-application-gateway)
- [Microsoft Operations Management Suite Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/microsoft-operations-management-suite)
- [Azure Resource Manager Solution Brief for Docker EE 17.06](https://success.docker.com/article/azure-
resource-manager)
- [Grafana/Prometheus Monitoring Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/grafana-prometheus-monitoring/)
- [Splunk Enterprise Solution Brief for Docker EE 17.06](https://success.docker.com/article/splunk-
logging)
- [Splunk Windows Containers: Metrics and Logs Forwarding Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/splunk-windows-monitoring-logging)
- [NGINX Solution Brief on Docker EE 17.06](https://success.docker.com/article/nginx-load-balancer/)
- [IBM Security Access Manager v9.0.4 Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/ibm-isam-security/)
- [Logging with Elasticsearch, Logstash, and Kibana Solution Brief for Docker EE 17.06]
(https://success.docker.com/article/elasticsearch-logstash-kibana-logging/)
```

```
## Appendix A. Manual Installation of Docker EE

This appendix contains generic instructions for deploying Docker EE on Microsoft Azure.

### Prerequisites

- Permissions to configure DNS, setup firewall rules, and sign-up for Docker EE.

- Existing infrastructure stack with 8 VMs available. All VMs must be able to
  reach each other over the network, but regular Docker clients will only need
  access to `ucp-lb` and `dtr-01`.

    - 3 managers (`manager-01`, `manager-02`, `manager-03`) with 1GB memory or more
    - 3 workers (`worker-01`, `worker-02`, `worker-03`)
    - 1 DTR node (`dtr-01`)
    - 1 load balancer node or external load balancer (`ucp-lb`)

- Domain name and TLS certificates for `ucp-lb` and `dtr-01`.

- Shell access to each VM (e.g. `ssh` or console access)

- (Optional) client with Docker installed (for testing the registry)

### Recommended: Prepare IPs, Domains, and Certificates

To avoid certificate warnings when connecting to UCP for the first time, an IP address, domain, and SSL
certificate should be obtained.

For testing, a self-signed certificate and a temporary domain can be used.

In your DNS system, you need at least a host for UCP and DTR. These typically point to `dtr-01` and the
UCP load balancer `ucp-lb`.
```

DOMAIN_NAME=
UCP_HOSTNAME = ucp.${DOMAIN_NAME}
DTR_HOSTNAME = dtr.${DOMAIN_NAME}

```
For the `UCP_HOSTNAME` and the `DTR_HOSTNAME`, you need to create a SSL certificate.  You can use your
normal process for issuing X.509 server certificates.

### Get Docker EE

Go to <https://store.docker.com/my-content> and download the license key file.
Also note the `DOCKER_EE_URL` (`https://storebits.docker.com/ee/...`).

### Installation
The following steps guide you through setting up Docker EE.

Be sure you have the following variables defined
* `DOCKER_EE_URL`
* `UCP_HOSTNAME`
* `DTR_HOSTNAME`

Docker EE installation is made easy through package managers.  The next two sections describe how to
setup Docker EE on Ubuntu or Red Hat Enterprise Linux.

This example shows Docker EE on each of the 8 nodes. 3 are managers, 3 are workers, 1 runs DTR, and 1
is the UCP load balancer.  All of them run Docker EE.

#### Install Docker EE on Ubuntu 16.04 / 17.06
```

curl -fsSL ${DOCKER_EE_URL}/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] ${DOCKER_EE_URL}/ubuntu
$(https://success.docker.com/api/asset/.%2Fpublish%2Fcertified-infrastructures-azure%2Flsb_release -cs)
stable-17.06"
sudo apt-get update
sudo apt-get install docker-ee -y

```
#### Install Docker EE on RHEL 7
```

sudo mkdir /etc/docker
sudo cp /tmp/daemon.json /etc/docker/daemon.json
sudo -E sh -c 'echo "${DOCKER_EE_URL}" > /etc/yum/vars/dockerurl'
sudo sh -c 'echo 7 > /etc/yum/vars/dockerosversion'
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
sudo yum-config-manager --enable rhel-7-server-extras-rpms
sudo -E yum-config-manager --add-repo "${DOCKER_EE_URL}/rhel/docker-ee.repo"
sudo yum -y install docker-ee
sudo systemctl enable docker

```
#### Create UCP Manager Nodes

From the set of 7 nodes created, use three of them to serve as managers. On the first manager run the
following command:
```

docker container run --rm -it --name ucp
-v /var/run/docker.sock:/var/run/docker.sock
docker/ucp:2.2.4 install
--san ${UCP_HOSTNAME}
--host-address <node-ip-address>
--interactive

```
The installer will prompt you for a username and password for the
administrator. The instructions in this runbook assumes that the chosen
username is `admin`, but this is not required.

Once UCP is installed on the first manager, use a web browser to connect to
the IP address of `manager-01` using HTTPS (e.g. `https://manager-01`). Then
log into the UCP dashboard with the admin user.

A new TLS certificate can be uploaded in **Admin settings** -> **Certificates**.

To add two more managers:

1. Go to the **Nodes** section.
2. Click on **Add Node** in the UI.
3. Select **Manager** to get the join command.
4. Run the command on each manager node.

The UCP dashboard should now show **3 Manager Nodes**.

#### Create Worker Nodes

From the set of 7 nodes created, use three of them to serve as workers.

1. Go to `https://manager-01` (replace `manager-01` with IP or domain name of the first manager).
2. Go to the **Nodes** section.
3. Click on **Add Node** in the UI.
4. Select **Worker** to get the join command.
5. Run the command on each worker node.

The UCP dashboard should now show **3 Worker Nodes**.

#### Setup Load Balancing

`${UCP_HOSTNAME}` should point to a load balancer that balances traffic to
`manager-01`, `manager-02` and `manager-03`. This runbook uses `nginx` for load
balancing in a VM, but a different or existing load balancer could also be used. For further examples,
such as `haproxy`, see the [UCP
documentation](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-a-load-
balancer/#load-balancing-ucp-and-dtr).

On node `ucp-lb`, create `nginx.conf` with the configuration below.
Replace `manager-01`, `manager-02` and `manager-03` with IPs on an interface
that can be reached from the `ucp-lb` node.
```

user nginx;
worker_processes 1;

```
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
worker_connections 1024;
}

stream {
upstream ucp_443 {
server manager-01:443 max_fails=2 fail_timeout=30s;
server manager-02:443 max_fails=2 fail_timeout=30s;
server manager-03:443 max_fails=2 fail_timeout=30s;
}
server {
listen 443;
proxy_pass ucp_443;
}
}
```

```
To start the load balancer, run:
```

```
docker run --detach
--name ucp-lb
--restart=unless-stopped
--publish 443:443
--volume ${PWD}/nginx.conf:/etc/nginx/nginx.conf:ro
nginx:stable-alpine
```

```
If `${UCP_HOSTNAME}` is configured to point to the external interface `ucp-lb`
you should now be able to go to `https://${UCP_HOSTNAME}` to use the UCP
dashboard.

#### Create DTR Node

Create at least one DTR node. The node used as the DTR node has to first be added to the swarm as a
worker or manager (see instructions above).

After the node has been added to the swarm, run the following command on `dtr-01`:
```

```
docker container run -it --rm
docker/dtr:2.3.4 install
--ucp-node dtr-01
--ucp-url=https://${UCP_HOSTNAME}
--ucp-insecure-tls
--dtr-external-url ${DTR_HOSTNAME}
```

```
Enter the UCP username and password when prompted.

If you get `failed to choose ucp node`, just try again.
If it still doesn't work, add `--debug` to the command to find out why.

You should now be able to access DTR at `https://${DTR_HOSTNAME}`.
The username and password are the same as for UCP.

Go to `Settings` and set the `Load Balancer/Public Address` to the DNS name (e.g. `DTR_HOSTNAME`).

Log out and log back in, using the `Use Single Sign-On` button.
After confirming that this works, go back to the Settings back and turn on
`Automatically redirect users to ucp for login`.

A TLS certificate can be uploaded in `Settings`, `Domain & Proxies`.

#### Test the Registry

> **Note:** This step requires that DTR has been configured to use a signed
certificate. For testing with self-signed certificates or insecure connections,
see <https://docs.docker.com/registry/insecure/>.

Use the DTR web interface to create a new private repository called
`admin/test`. If you used a different username than `admin` when you installed
DTR/UCP, replace `admin` with the correct name or manually create an `admin`
organisation in the web UI first.

Then try pushing an image to it:
```

```
docker login ${DTR-HOSTNAME}
docker pull busybox
docker tag busybox ${DTR-HOSTNAME}/admin/test
docker push ${DTR-HOSTNAME}/admin/test
```

```
The image should now appear in the web UI.
You can also go to the DTR Settings and turn on image scanning to scan it.

#### Download Client Bundle

**WARNING**: The hostname in the UCP bundle comes from the `Host` field sent by the browser.
Therefore, you *must* access the site via the correct DNS name when downloading the
bundle. Otherwise, it will contain an IP address and the certificate will be rejected.

Go to `My profile` in the UCP web interface and create a new client bundle.
Check that you can use this bundle to run a container. For example:
```

```
mkdir bundle
cd bundle/
unzip ~/Downloads/ucp-bundle-admin.zip
source env.sh
docker run --rm hello-world
Hello from Docker!
```

```
#### Firewall

Depending on the environment is recommended to ensure the firewall enables the
minimal set of ports required. Docker clients will need access to the external
interface of `ucp-lb` and `dtr-01` so they can be reached via
`${UCP_HOSTNAME}` and `${DTR_HOSTNAME}`.

## Appendix B. Docker EE Overview

This appendix describes the three main components of Docker EE – Universal Control Plane, Docker
Trusted Registry, and Docker engine.

Refer to [Docker Reference Architecture: Docker EE Best Practices and Design Considerations]
(https://success.docker.com/article/docker-ee-best-practices) for detailed information.

### Universal Control Plane

There are three ways to interact with UCP: the **web UI**, the **API**, or the **CLI**.

You can use the UCP web UI to manage your swarm, grant and revoke user permissions, deploy, configure,
manage, and monitor your applications.

![](https://docs.docker.com/v17.09/datacenter/images/ucp.png)

Docker UCP secures your swarm by using role-based access control. From the browser, administrators can:

- Manage swarm configurations
- Manage the permissions of users, teams, and organizations
- See all images, networks, volumes, and containers
- Grant permissions to users for scheduling tasks on specific nodes (with the Docker EE Advanced
license)

UCP also exposes the standard Docker API, so you can continue using existing tools like the Docker CLI
client. Since UCP secures your cluster with role-based access control, you need to configure your
Docker CLI client and other client tools to authenticate your requests using client certificates that
you can download from your UCP profile page.

Docker UCP secures your swarm by using role-based access control, so that only authorized users can
perform changes to the cluster.

For this reason, when running Docker commands on a UCP node, you need to authenticate your request with
client certificates. When trying to run Docker commands without a valid certificate, you get an
authentication error:
```

$ docker ps

x509: certificate signed by unknown authority

```
There are two different types of client certificates:

- Admin user certificate bundles: allow users to run Docker commands on the Docker Engine of any node
- User certificate bundles: only allow users to run Docker commands through a UCP manager node
```

To download a client certificate bundle, log into the UCP web UI and navigate to your **My Profile** page. In the left pane, click **Client Bundles**, and click **New Client Bundle** to download the certificate bundle. Further information is available in the [Docker docs](https://docs.docker.com/v17.09/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/).

There is also the option of interacting with the Universal Control plane through its API, allowing programmatic access to swarm resources that are managed by UCP. The API is secured with role-based access control so that only authorized users can make changes and deploy applications to your Docker swarm.

The UCP API is accessible in the same IP addresses and domain names that you use to access the web UI. It's the same API that the UCP web UI uses, so everything you can do on the UCP web UI from your browser, you can also do programmatically.

The full API documentation is available on [docs.docker.com](https://docs.docker.com/datacenter/ucp/2.2/reference/api/).

Docker Trusted Registry

Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker providing secure storage and management of the Docker images you use in your applications.

There are two ways to interact with DTR: the **web UI** or the **CLI**.

You can use DTR as part of your continuous integration, and continuous delivery processes to build, ship, and run your applications.

DTR has a web based user interface that allows authorized users in your organization to browse docker images. It provides information about who pushed what image at what time. It even allows you to see what dockerfile lines were used to produce the image and, if security scanning is enabled, to see a list of all of the software installed in your images.

DTR is **highly available** through the use of multiple replicas of all containers and metadata such that if a machine fails, DTR continues to operate and can be repaired.

DTR uses the same authentication mechanism as Docker Universal Control Plane. Users can be managed manually or synched from LDAP or Active Directory. DTR uses Role Based Access Control (RBAC) to allow you to implement fine-grained access control policies for who has access to your Docker images.

DTR also has a built in security scanner that can be used to discover what versions of software are used in your images. It scans each layer and aggregates the results to give you a complete picture of what you are shipping as a part of your stack. Most importantly, it co-relates this information with a vulnerability database that is kept up to date through periodic updates. This gives you unprecedented insight into your exposure to known security threats.

Docker DTR should be configured to be the default registry for all Docker engines so when an engine needs to pull an image it will automatically pull from the trusted registry. Details on how to configure the Docker engine to communicate
correctly with Docker Trusted Registry can be found in the [Docker docs](https://docs.docker.com/datacenter/dtr/2.4/guides/user/access-dtr/).

Docker Engine

The Docker Enterprise Edition engine is deployed on all hosts that are part of the Docker EE platform. The deployment of applications can be done through the Universal Control Plane or through the CLI of

```
the Docker hosts that are part of the management plane. The Client Bundle is required to authenticate
with the Docker EE platform from the CLI. Details around using the client bundle can be found in the
Universal Control Plane section.
```