Docker Certified Associate Training

Source: https://docs.docker.com

simplilearn

Installation and Configuration of Docker Enterprise

Learning Objectives

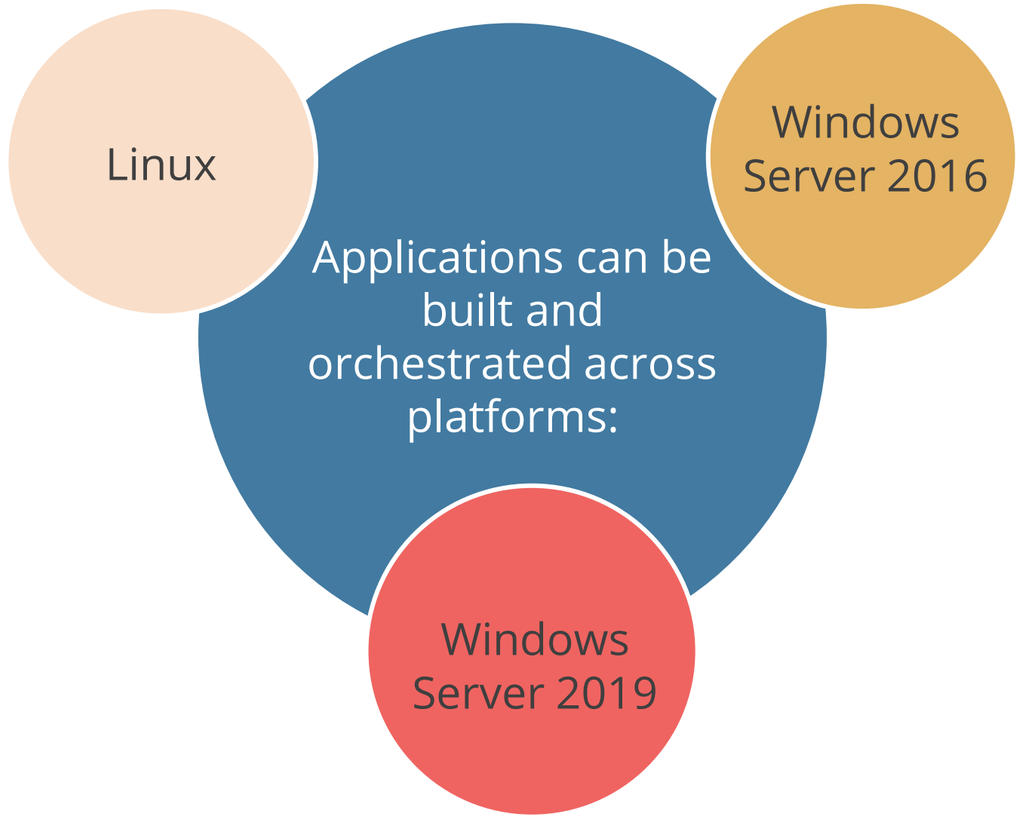
By the end of this lesson, you will be able to:

- Describe different tiers of Docker and their capabilities
- Set up the repository and install the Docker Engine Enterprise
- Install the UCP, DTR, and comprehend their architecture
- Create grants and understand how access control works
- Comprehend high availability and load balancing

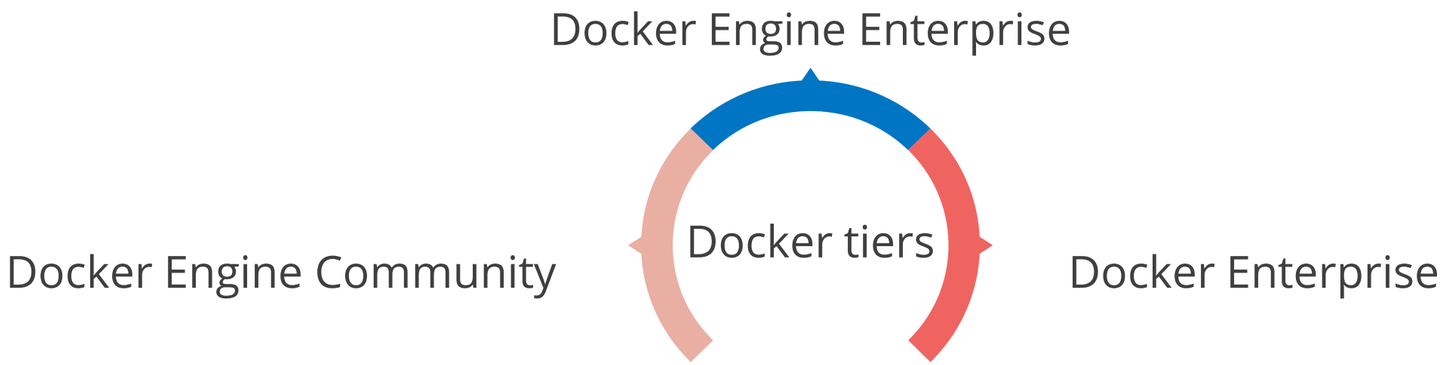
Docker Enterprise

Docker Enterprise: Overview

It is a scalable, secure, and supported container platform that helps in building and sharing legacy/modern applications. It also helps the operators to securely run them anywhere.



Docker Enterprise: Overview



Docker Enterprise: Overview

Purpose

Docker Engine Enterprise: It is meant for enterprise development of a container runtime. While developing containers the security and an enterprise grade SLA are kept in mind.

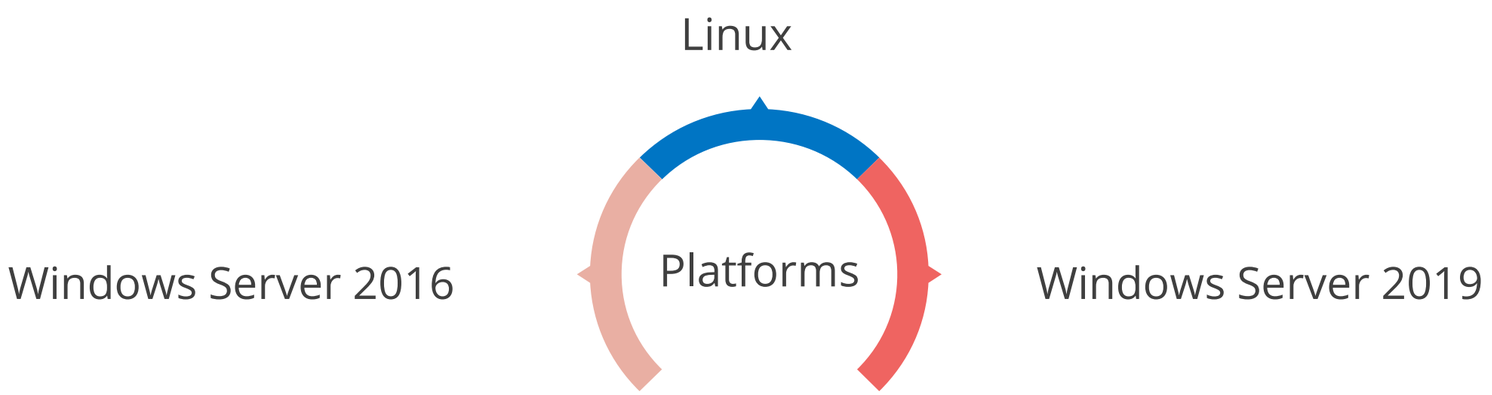
Docker Enterprise: It is meant for enterprise development as well as for IT teams who build, ship, and run business critical applications in production at scale.

Docker Enterprise: Overview

Capabilities	Docker Engine - Community	Docker Engine - Enterprise	Docker Enterprise
Container engine and built in networking, orchestration, and security	Yes	Yes	Yes
Certified infrastructure, plugins, and ISV containers	-	Yes	Yes
Image management	-	-	Yes
Container app management	-	-	Yes
Image security scanning	-	-	Yes

Docker Enterprise: Overview

Applications can be built and orchestrated across:



Docker Enterprise: Overview

Workloads are deployed using:

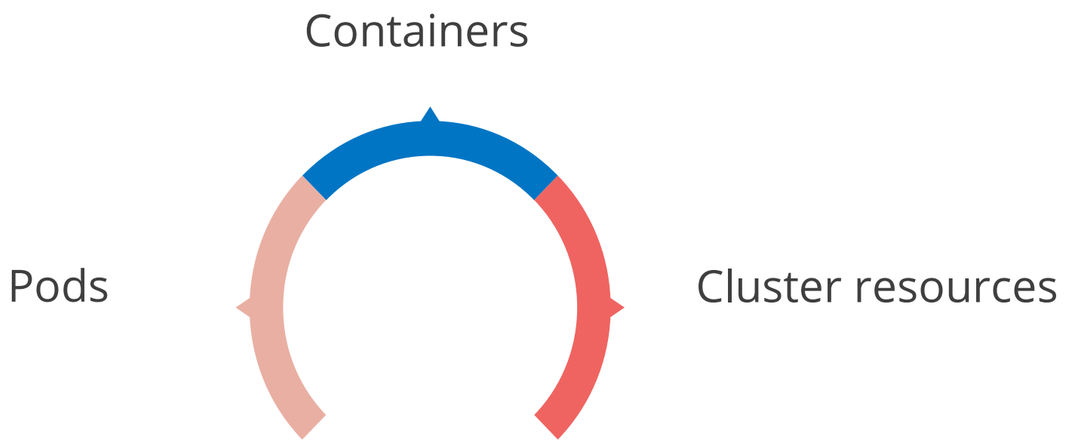
Docker Kubernetes Service

Features

Docker Swarm

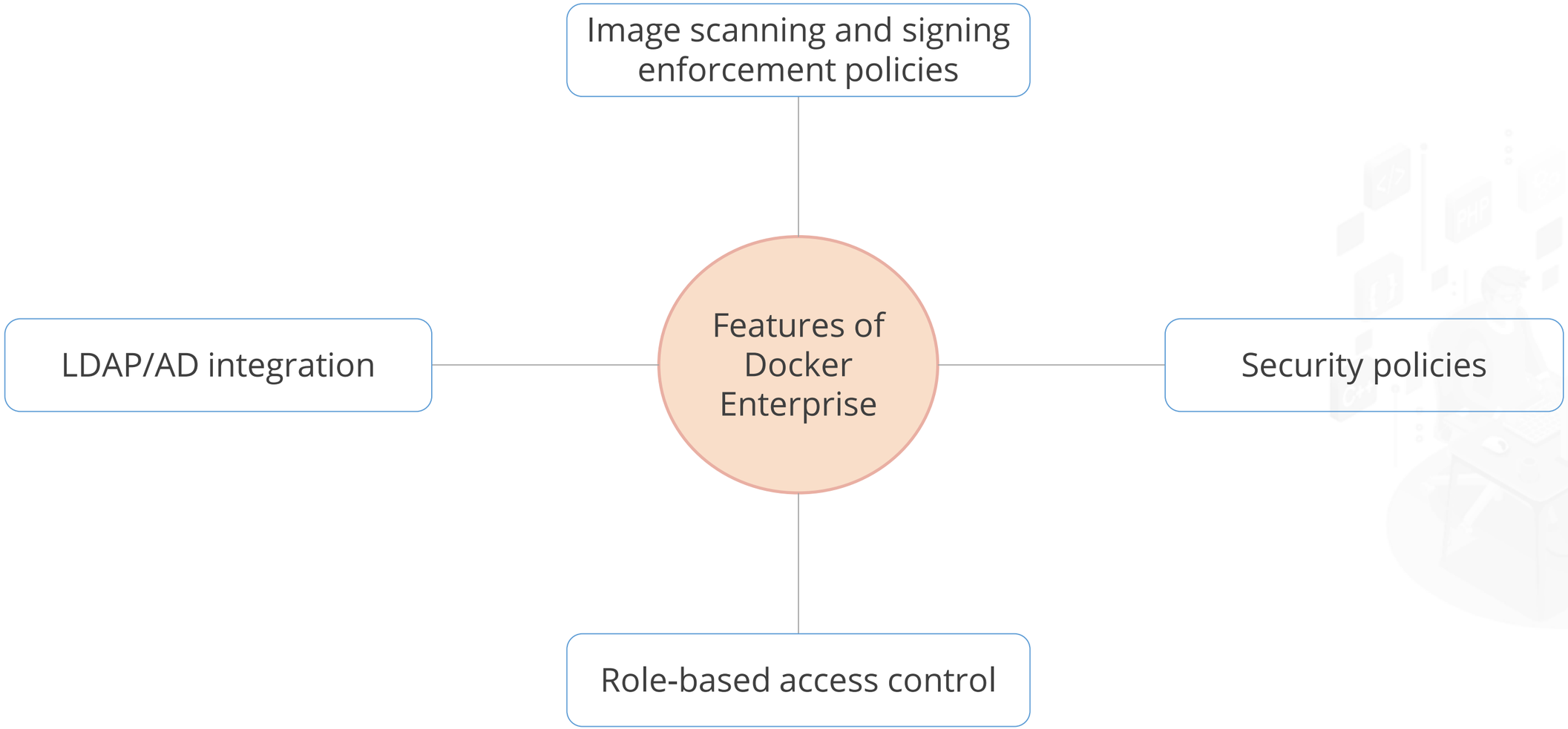
Docker Enterprise: Overview

Tasks that are required by orchestration and are automated by Docker Enterprise, like provisioning:

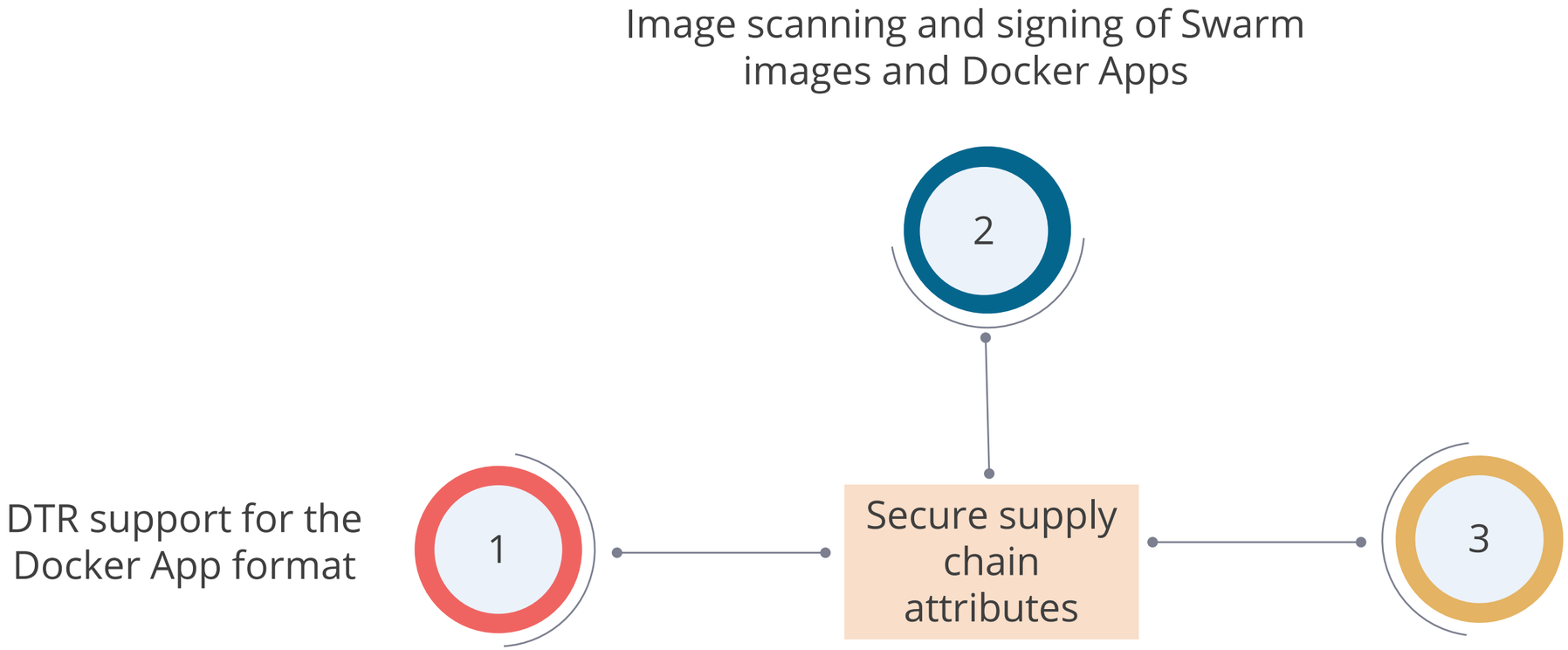


Features of Docker Enterprise

Docker Enterprise : Features



Docker Enterprise: Features



Docker Enterprise: Features

By using Docker, thousands of physical or virtual machines can be joined to create cluster.

- Docker Enterprise deploys the applications at scale.
- Docker Enterprise makes management of the cluster from a centralized place very easy.
- Docker Enterprise manages and monitors the container cluster by using a graphical web interface.

Docker Enterprise: Features

Deploy, manage, and monitor:

Docker Enterprise is used to deploy and monitor the applications and services.

Docker Enterprise manages the following infrastructure resources from a central console:

- Nodes
- Volumes
- Networks, from a central console.

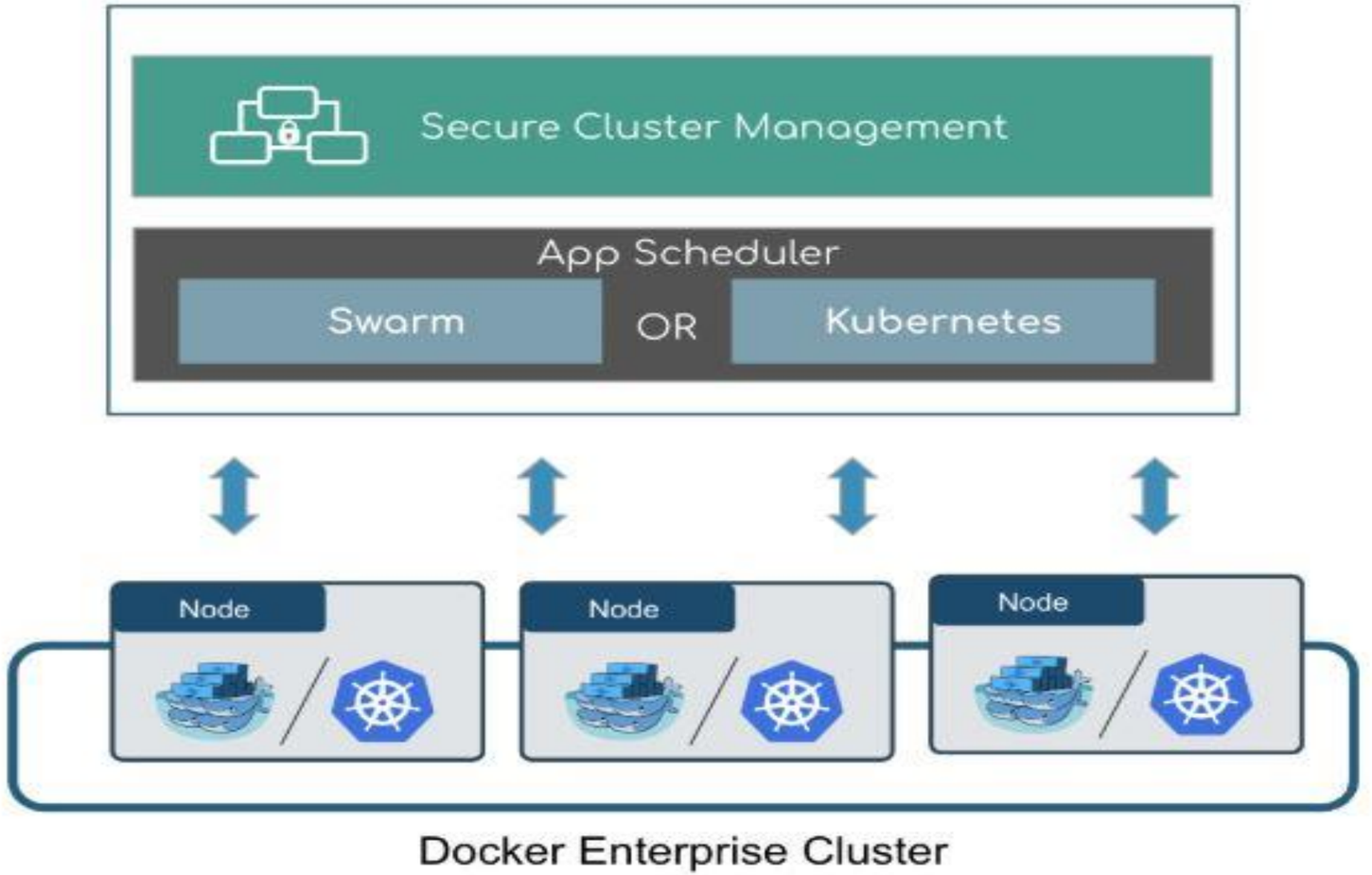
Docker Enterprise: Features

Built-in security and access control:

- Docker Enterprise has a built-in authentication mechanism with role-based access control (RBAC) in order to control access and edit permissions for cluster and applications.
- Docker Enterprise authentication gets integrated with LDAP services and supports SAML SCIM to proactively synchronize with authentication providers.

Orchestration

Docker Enterprise Orchestration:

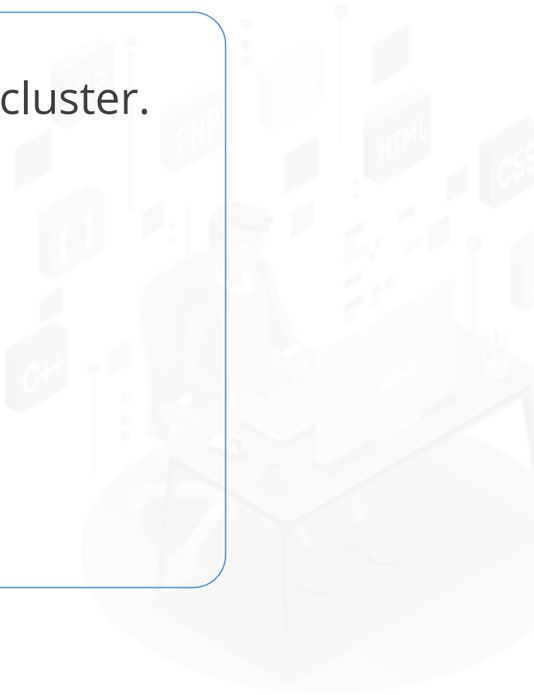


Source: https://docs.mirantis.com

Orchestration

Docker Enterprise platform allows to run both Swarm and Kubernetes in the same cluster.

- There is no need for the developers to select orchestrators.
- There is freedom to change orchestrators according to the requirement.
- The Enterprise manager nodes are enabled with both Swarm and Kubernetes.
- The worker node is both Kubernetes API- ready and Swarm API- ready.



Source: https://docs.docker.com/ee/#orchestration-platform-features

Orchestration

Orchestration platform features:

- Enabling high availability using Docker Enterprise manager nodes
- Allocating worker nodes for Swarm/Kubernetes workloads
- Monitoring apps via a single pane of glass
- Enhancing Swarm hostname routing mesh with Interlock 2.0
- One platform-wide management plane:
  - Secure software supply chain
  - Secure multi-tenancy
  - Secure and highly available node management

Universal Control Plane (UCP)

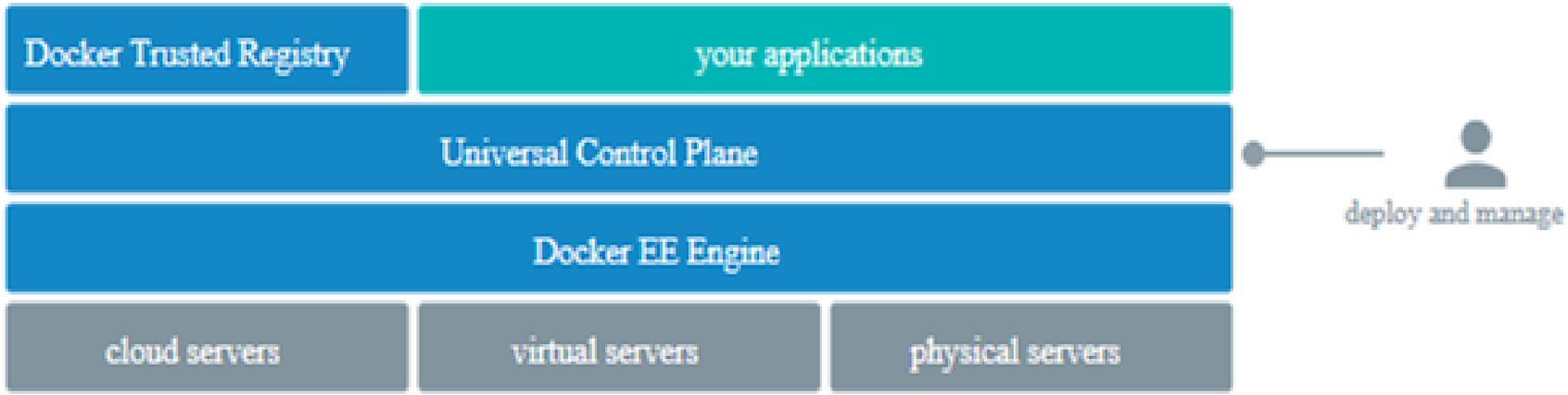
UCP: Overview

Universal Control Plane is the cluster management solution from Docker Enterprise that helps in managing the applications and Docker cluster through a single interface. It is now known as MKE (Mirantis Kubernetes Engine)

Source: https://docs.mirantis.com

UCP: Architecture

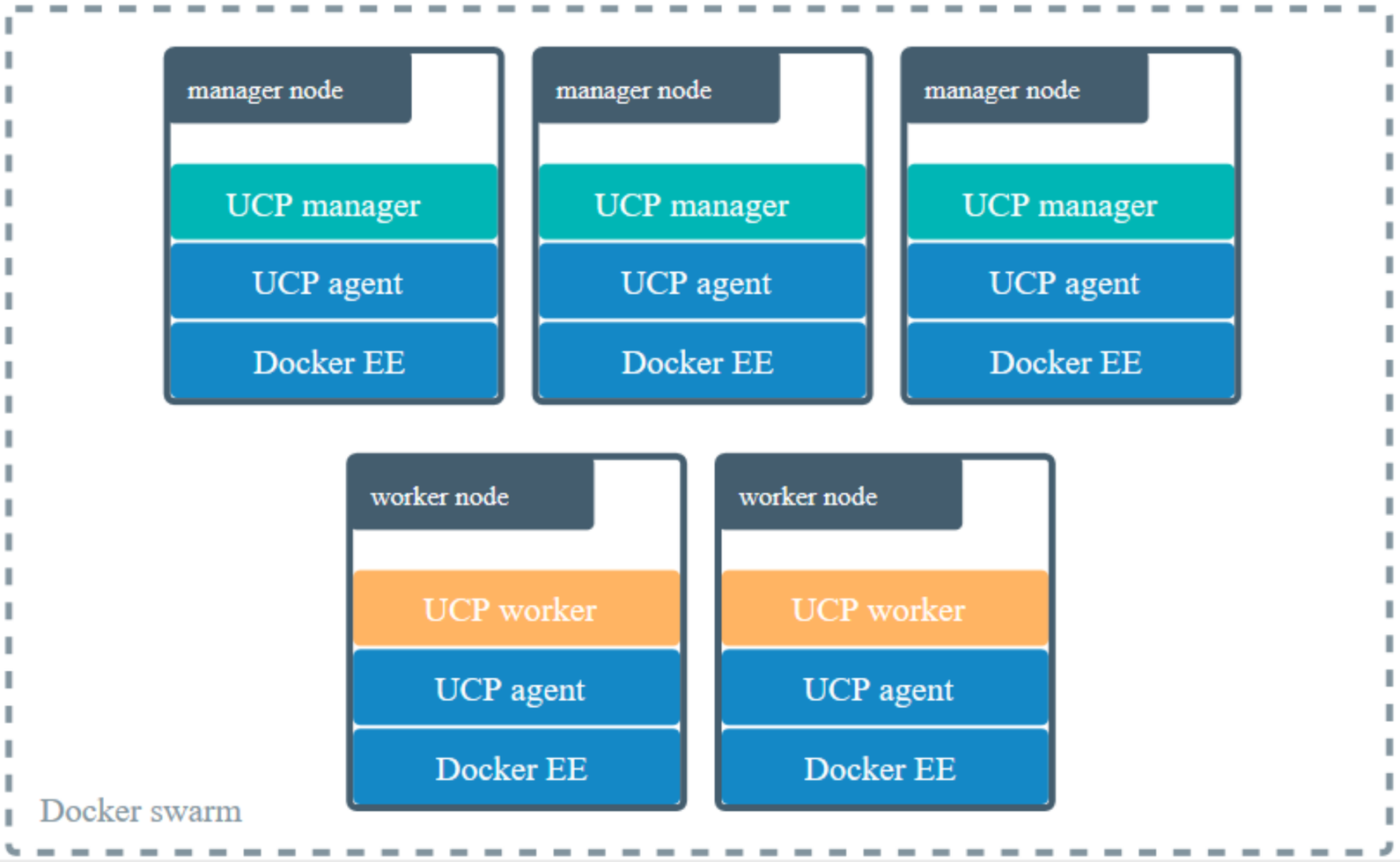
After the deployment of a UCP instance, developers and IT operations no longer interact with Docker Engine directly, but interact with UCP instead.



Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

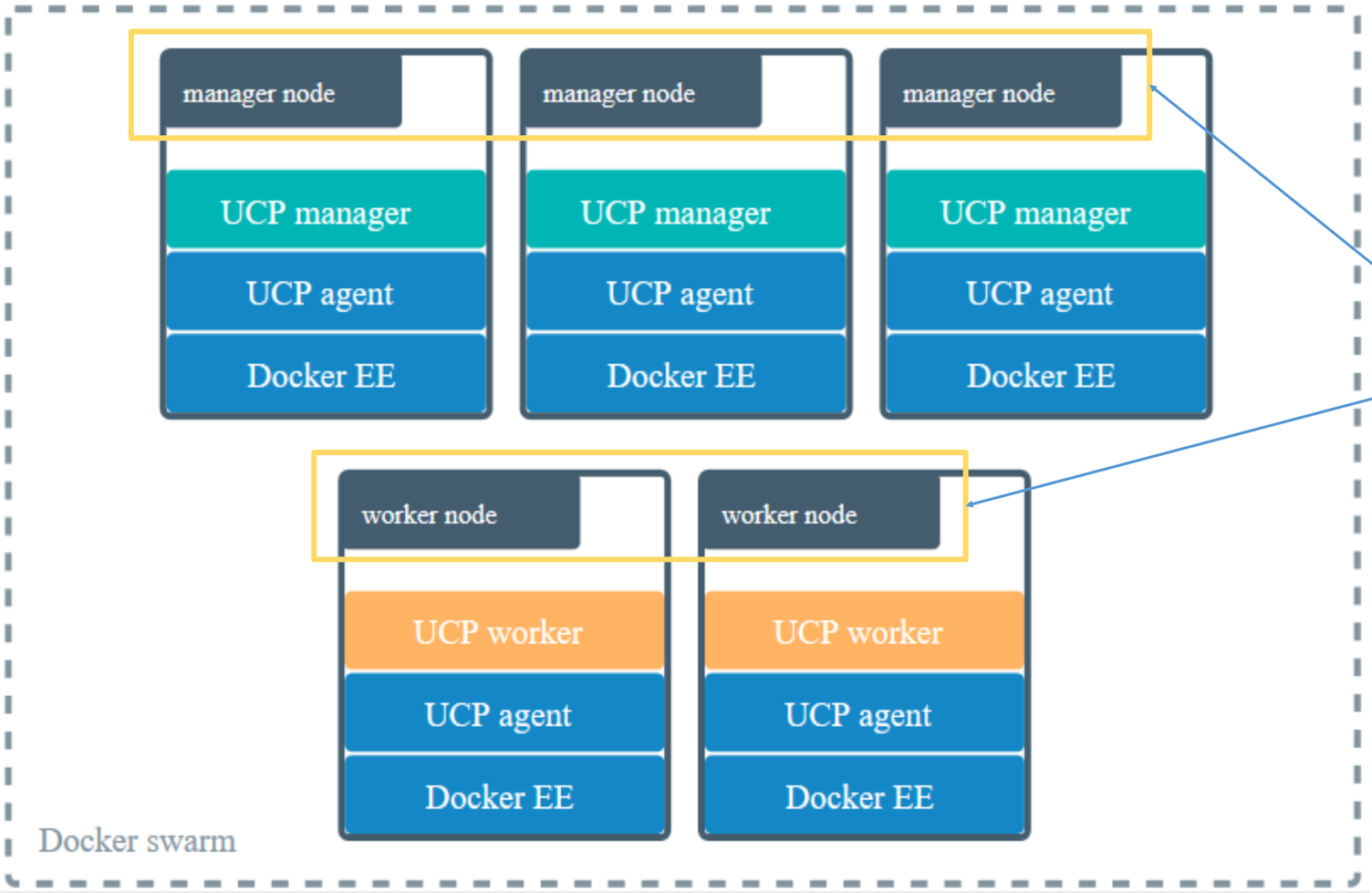
UCP: Architecture

The clustering and orchestration functionality is leveraged by Docker UCP.



Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

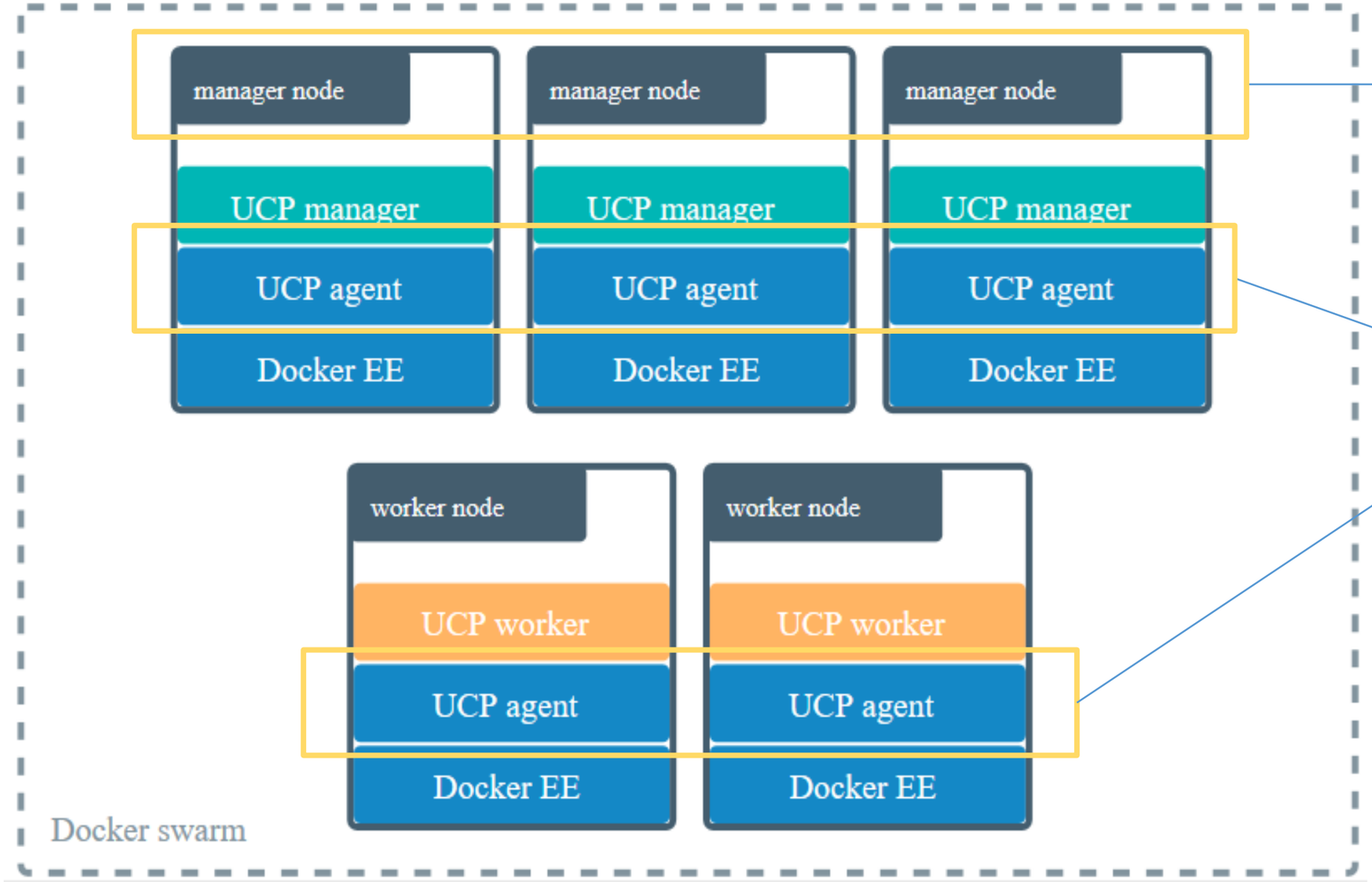
UCP: Architecture



Docker swarm is a collection of nodes. Nodes operate either as Manager or Worker nodes.

Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

UCP: Architecture

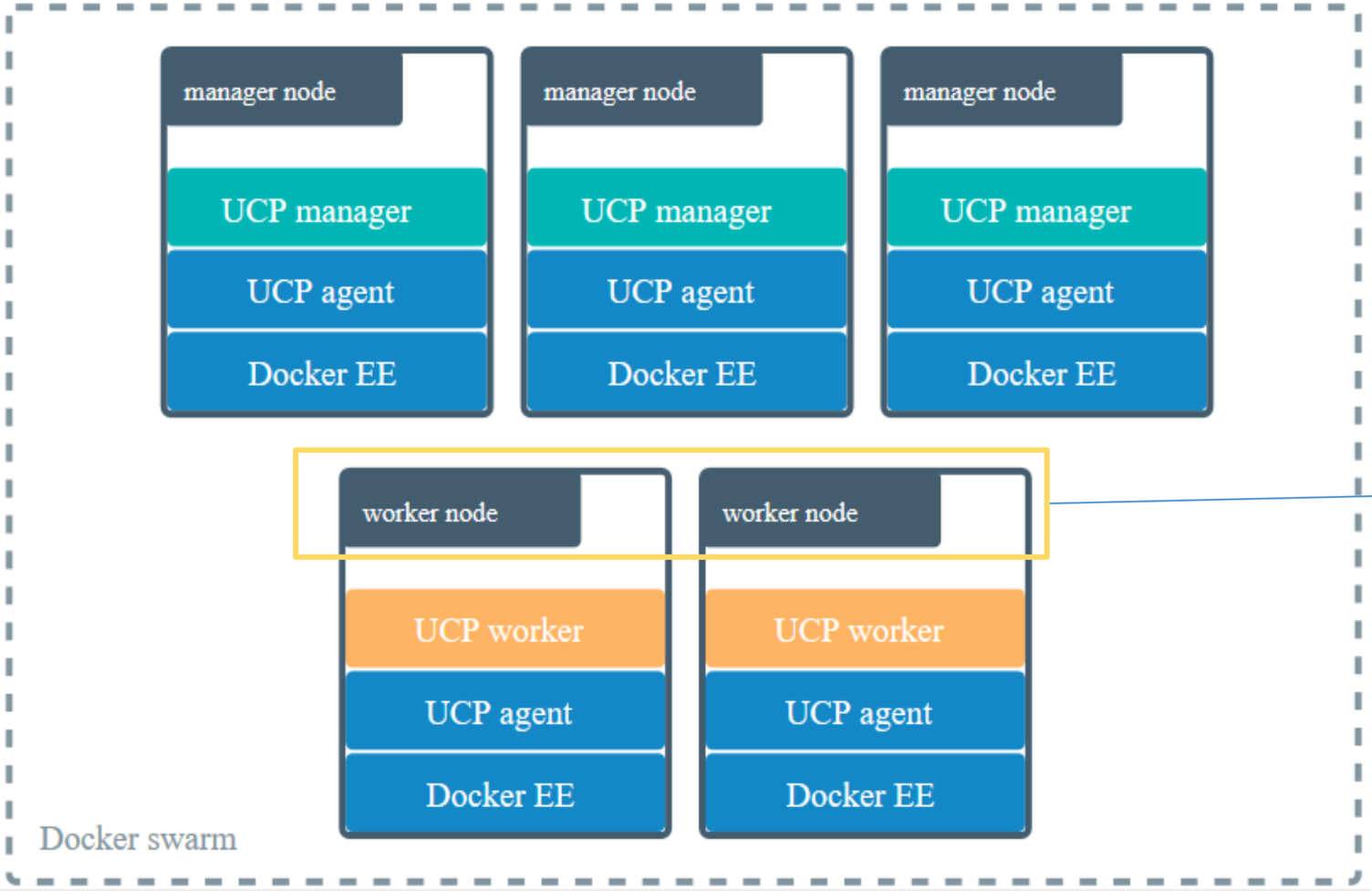


On manager nodes, the *ucp-agent* serves all the UCP components.

It is a globally scheduled service.

Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

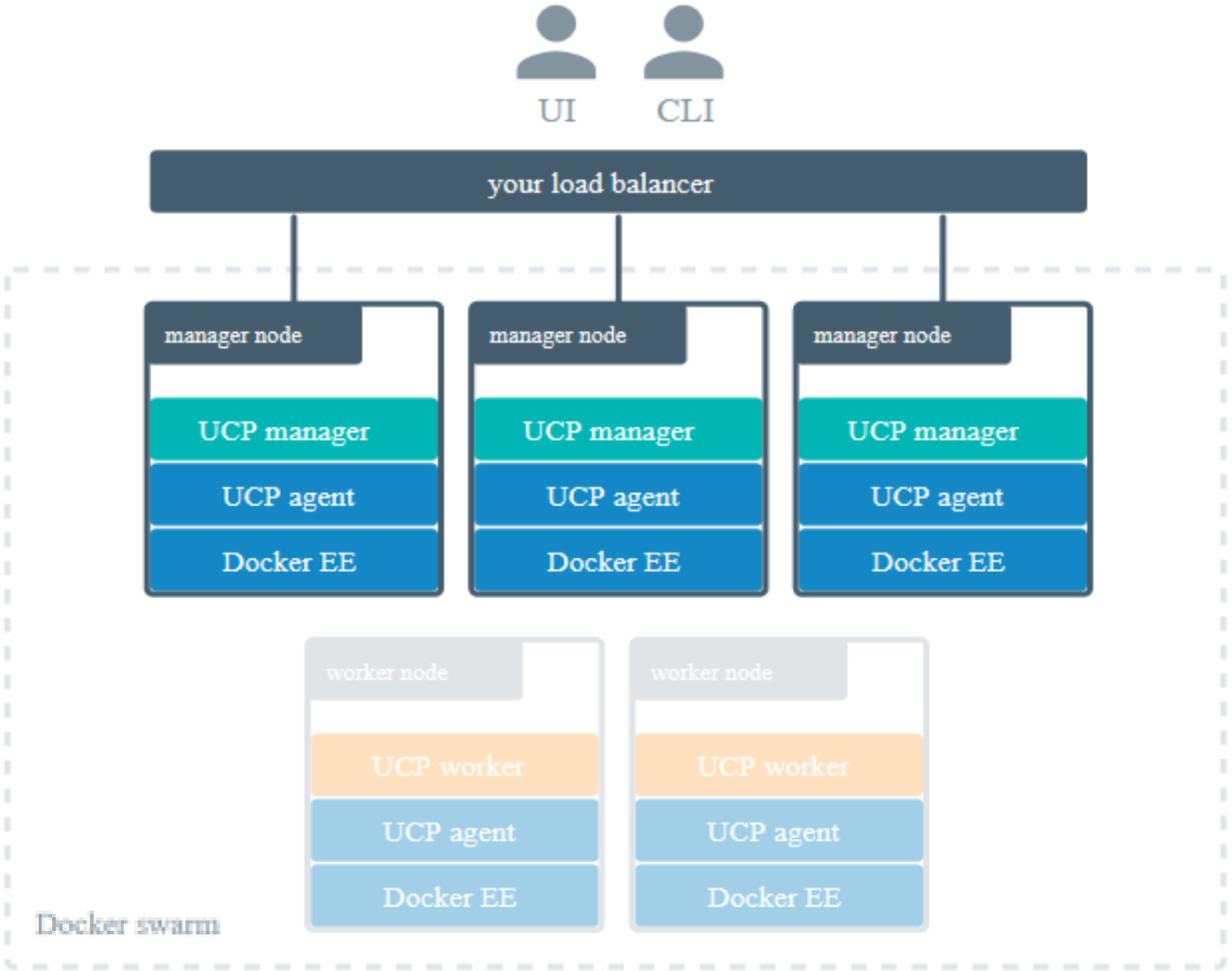
UCP: Architecture



On worker nodes, the *ucp-agent* starts serving a proxy service.

Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

Interaction with UCP



Source: https://success.mirantis.com/article/ucp-service-discovery-swarm

Docker Enterprise Installation: Mirantis Launchpad

Mirantis Launchpad: Overview

Mirantis Launchpad is a fast and easy-to-use CLI tool that deploys Docker Enterprise on public clouds, private clouds, or bare metals like Linux, Mac, or Windows machine.

- Used for installing, deploying, and updating the Docker Enterprise
- Provides full cluster lifecycle management
- Allows multi-manager, high-availability clusters defined with sufficient node capacity to move active workloads around with no downtime

Mirantis Launchpad: Installation

Prerequisites:

Minimum three servers required:

- Server 1 to download the launchpad (Docker Worker node with 4GB RAM and 10GB hard-disk)
- Server 2 to install the UCP (Docker Manager node with 8GB RAM and 10GB hard-disk)
- Server 3 to install the DTR (Docker Worker node with 4GB RAM and 10GB hard-disk)

Mirantis Launchpad: Installation

Installation Steps:

- Setting up a common bot user on all the servers and configuring an SSH connection using ssh-keygen
- Downloading and configuring the launchpad binary file
- Creating a launchpad.yml file to configure the UCP installation
- Registering the launchpad and configuring the UCP cluster

Install Mirantis Launchpad CLI

Problem Statement: Your technical manager has asked you to set up the Mirantis Launchpad CLI so that Docker Enterprise can be configured.

Steps to Perform:

1. Ensuring minimum requirements for the master and worker node's servers
2. Configuring an SSH connection between all servers using ssh-keygen
3. Downloading and configuring the launchpad binary file
4. Creating a launchpad.yml file to configure the MKE installation
5. Registering the launchpad and applying configuration settings for the MKE cluster
6. Getting Free Trial Licence for Docker Enterprise and uploading it on MKE dashboard

ASSISTED PRACTICE

Uninstall Docker Enterprise

Uninstall Docker Enterprise :

Reset launchpad to uninstall Docker EE and UCP cluster

→ *$ launchpad reset*

Upgrade Docker Enterprise

Upgrade Docker EE:

Modify the engine version in the **launchpad.yaml** file

```
apiVersion: launchpad.mirantis.com/v1
kind: DockerEnterprise
metadata:
  name: launchpad-ucp
spec:
  hosts:
  - address: 10.0.0.1
    role: manager
  engine:
    version: 19.03.12 # was previously 19.03.8
```

Apply the changes after updating the **launchpad.yaml** file

```
$ launchpad apply
```

Upgrade Docker UCP

Upgrade Docker UCP:

Modify the version tags in the **launchpad.yaml** file

```
apiVersion: launchpad.mirantis.com/v1
kind: DockerEnterprise
metadata:
  name: launchpad-ucp
spec:
  hosts:
  - address: 10.0.0.1
    role: manager
  ucp:
    version: 3.3.1
  dtr:
    version: 2.8.1
```

Apply the changes after updating the **launchpad.yaml** file

```
$ launchpad apply
```

Joining Manager Nodes

Joining Manager Nodes

Manager nodes are joined when the UCP is supposed to be highly available. Docker swarm and UCP can be made fault-tolerant and highly available by adding more manager nodes to the cluster.

Join manager nodes to the swarm:

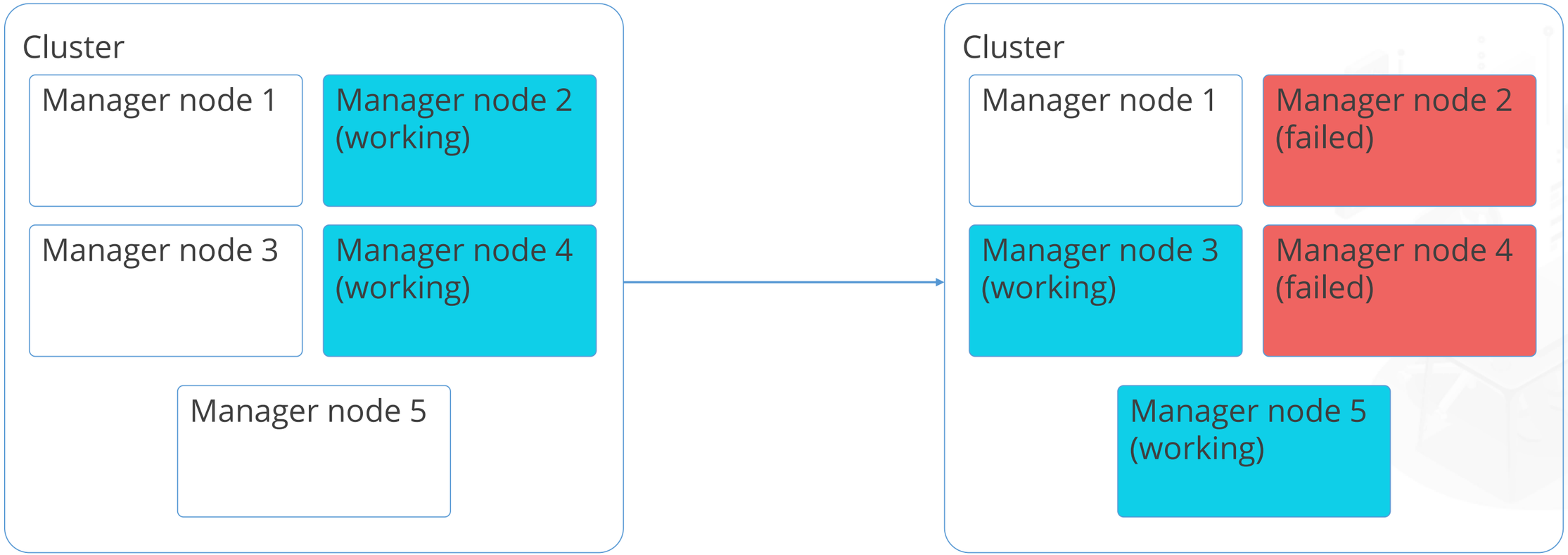
1. Navigate to the **Nodes** page in the UCP web UI and click on the **Add Node** button to add a new node
2. Select **Add node as a manager** in the Add Node page

Joining Manager Nodes

3. Click **Use a custom listen address**, and then enter the IP address and port for the node in order to listen to the inbound cluster management traffic. The format is *interface:port* or *ip:port* and the default is 0.0.0.0:2377.
4. Click **Use a custom advertise address**, and then enter the IP address and port
5. Click the **copy icon** to copy the *docker swarm join* command to add nodes to the swarm
6. Login using ***ssh*** and run the join command that was copied. The node appears on the **Nodes** page in the UCP web UI once the join command completes.

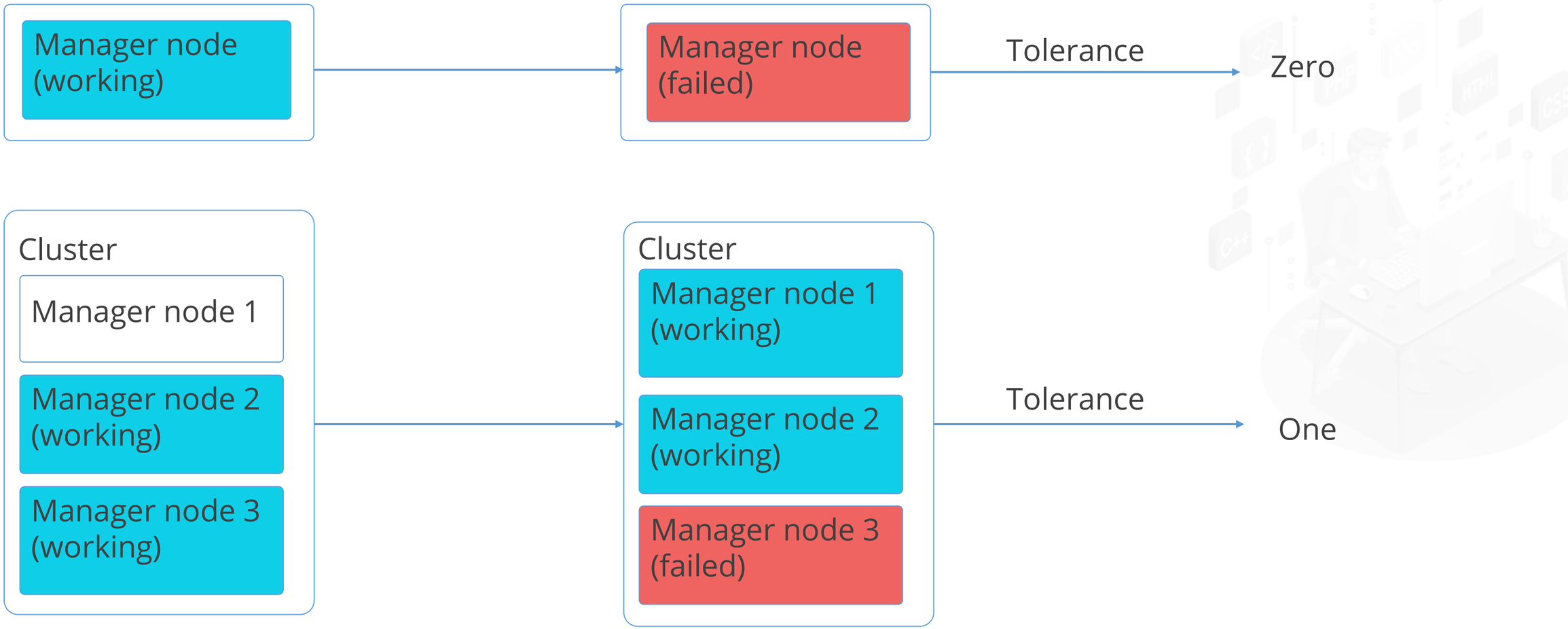
UCP: High Availability

High Availability



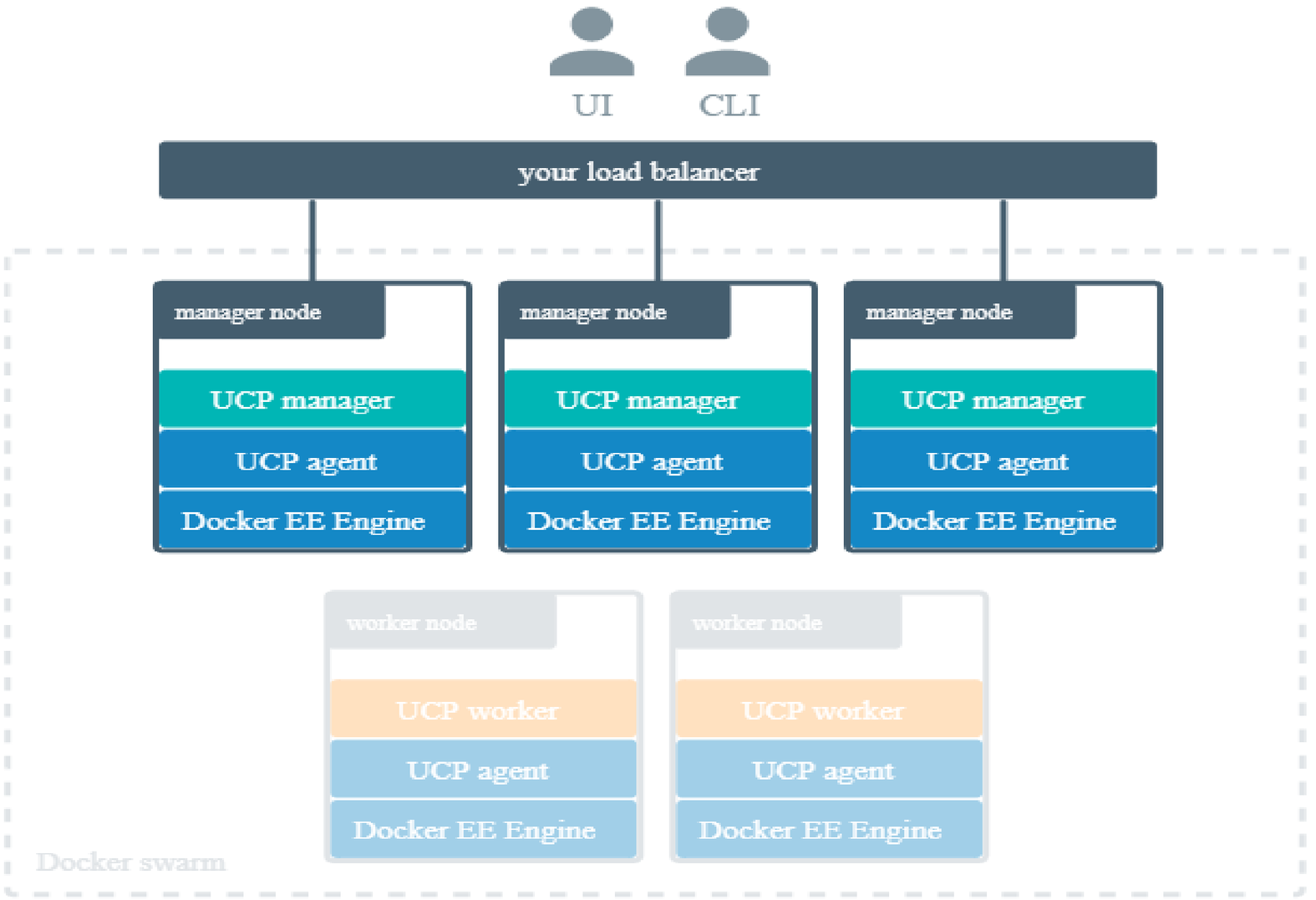
High Availability

Failure tolerance of Manager node:



UCP: Load Balancer

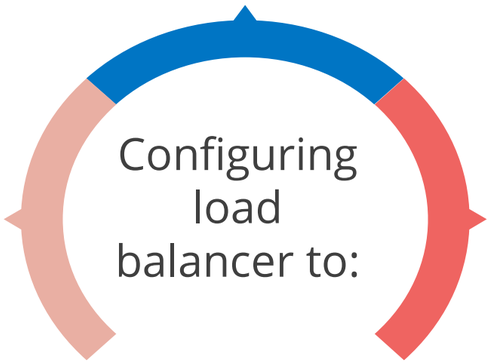
Load Balancer



Source: https://docs.mirantis.com/docker-enterprise/v3.0/dockeree-products/ucp.html

Load Balancer

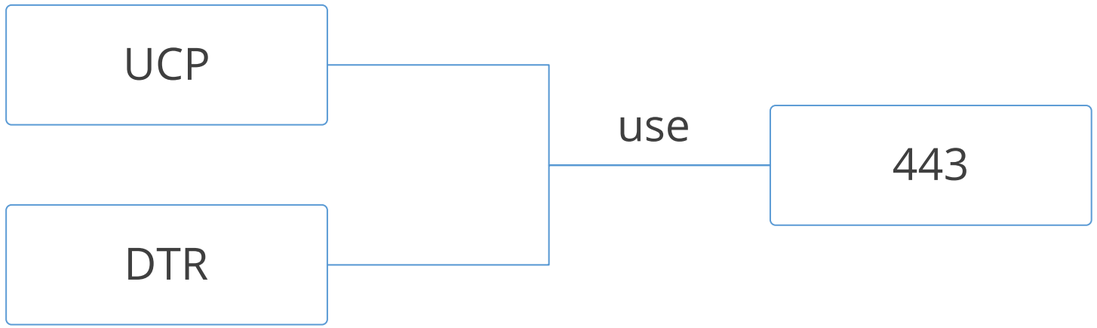
Stop termination of HTTPS connections



Load-balance the TCP traffic on ports 443 and 6443

Apply the */_ping* endpoint on all the manager node

Load Balancing UCP and DTR



How the load balancer is configured to listen on the port 443?

Load balancer can be configured :

- Using separate load balancer for UCP and DTR
- Using same load balancer but with more than one virtual IP address

Configuring Load Balancer

Example of configuring the load balancer for UCP:

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
events {
    worker_connections  1024;
}
 }
```

Contd. in the adjacent box

```
stream {
    upstream ucp_443 {
        server <UCP_MANAGER_1_IP>:443 max_fails=2
fail_timeout=30s;
        server <UCP_MANAGER_2_IP>:443 max_fails=2
fail_timeout=30s;
        server <UCP_MANAGER_N_IP>:443  max_fails=2
fail_timeout=30s;
    }
    server {
        listen 443;
        proxy_pass ucp_443;
    }
```

Deploying Load Balancer

Load balancer deployment:

Deploying a load balancer is a two-step process:

1. Create the *nginx.conf* file
2. Deploy the load balancer

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

Docker Enterprise: Swarm Service

Deploy Swarm Service Using UCP

Steps for deploying and removing a service:

1. Log in to Docker Enterprise UCP
2. Click on the option **Add Nodes** that is present at the bottom of the UCP
3. Click on the desired **Node Type:** Windows and Linux as well as **Node Role:** Manager and Worker
4. Copy the Docker CLI command and run it
5. Click on **Swarm,** and then **Services** after the addition of Node
6. Click on the **Create** button

Deploy Swarm Service Using UCP

7. Fill in the required details in order to deploy a service
8. Click on the **Create** button that is present at the bottom right corner
9. Check the status of the service
10. Choose the desired service that has to be removed
11. Click **Action**, and then **Remove**
12. Click **Confirm**

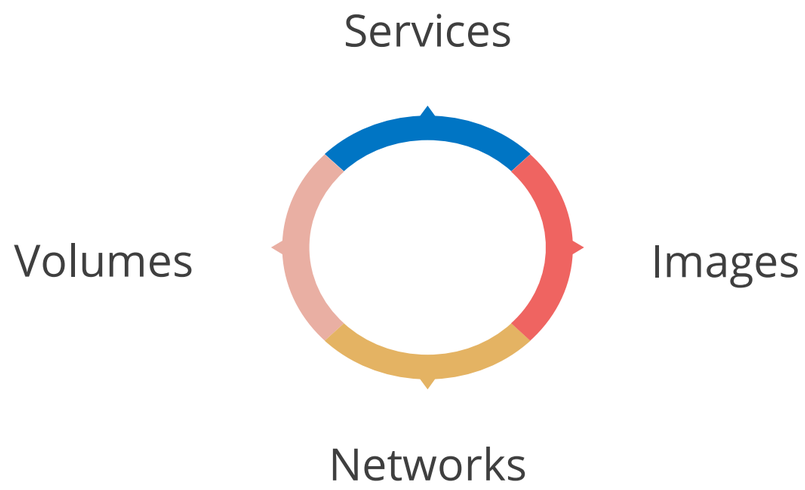
Note: There should be enough nodes to deploy a Swarm service.

Access Control

Access Control

Docker UCP provides a feature that is used to grant and manage permissions to enforce access control.

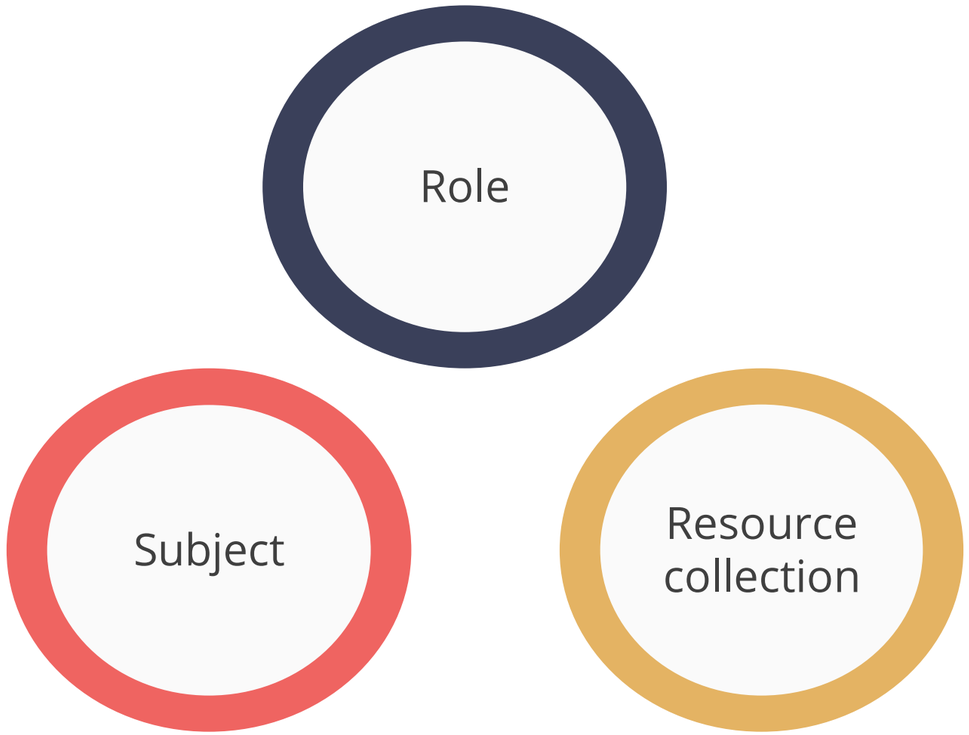
Organizations control who can create and edit container resources in the swarm, such as:



Grant

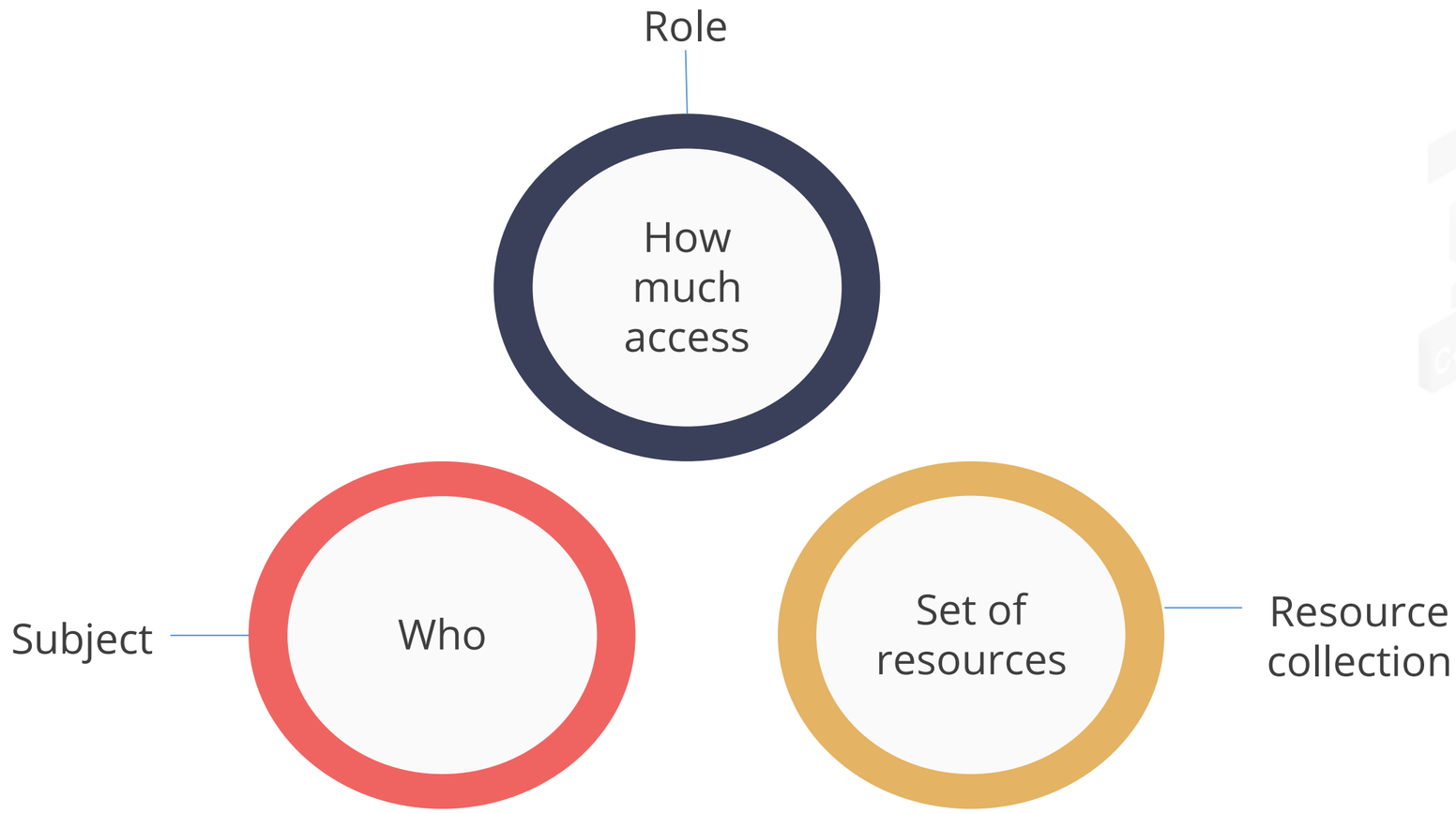
Users and organizations access swarm resources that can be controlled by creating *grants.*

Grant contains:



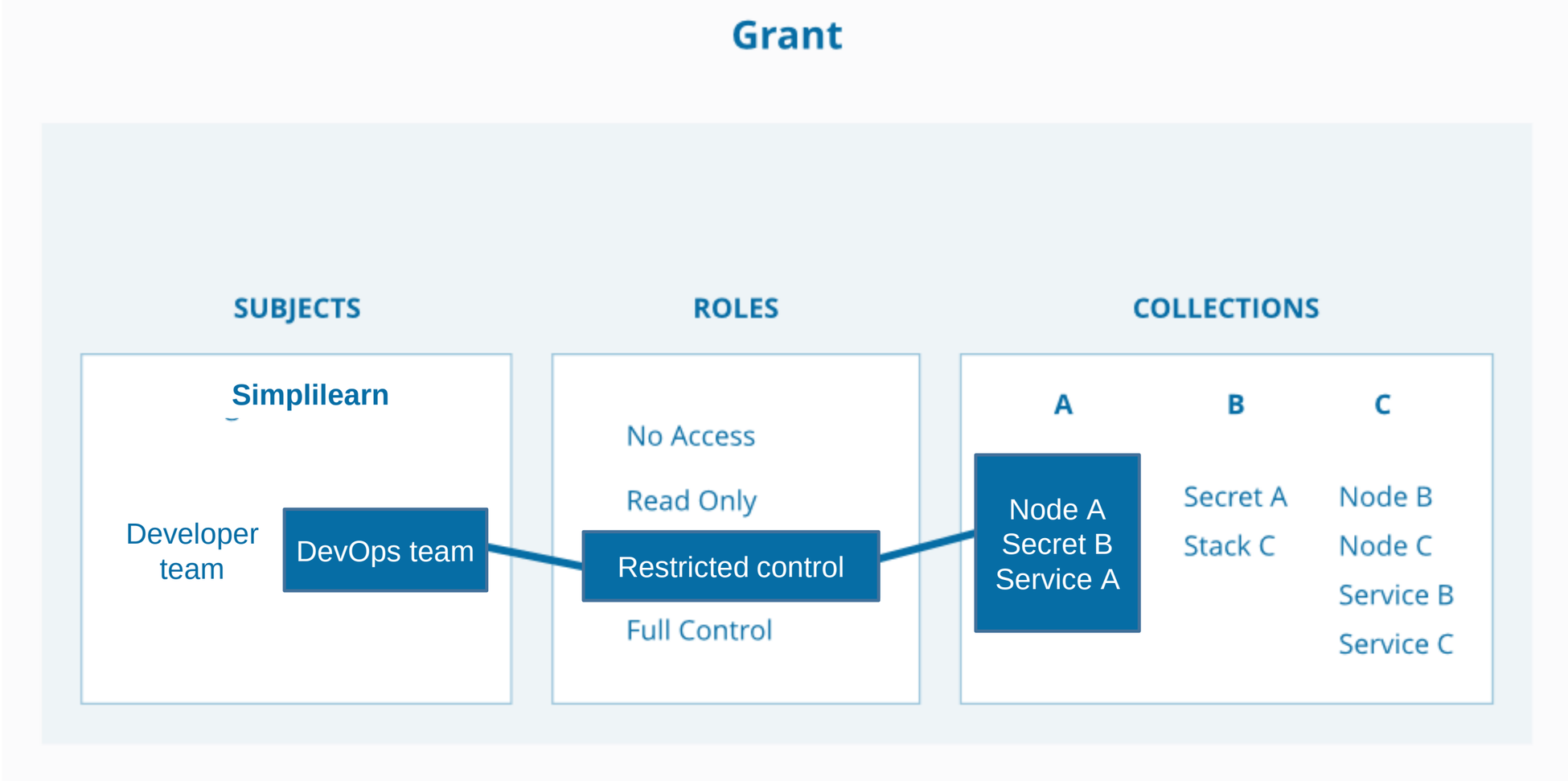
Grant

Grant defines:



Grant

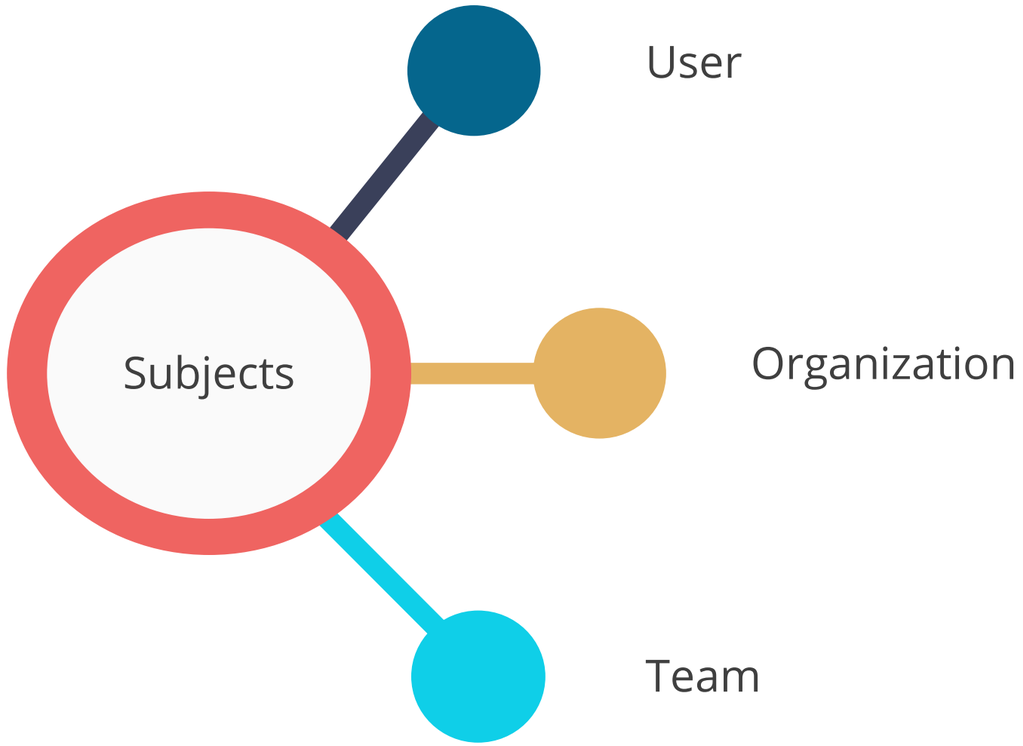
Example: DevOps team in Simplilearn have restricted control for collection A



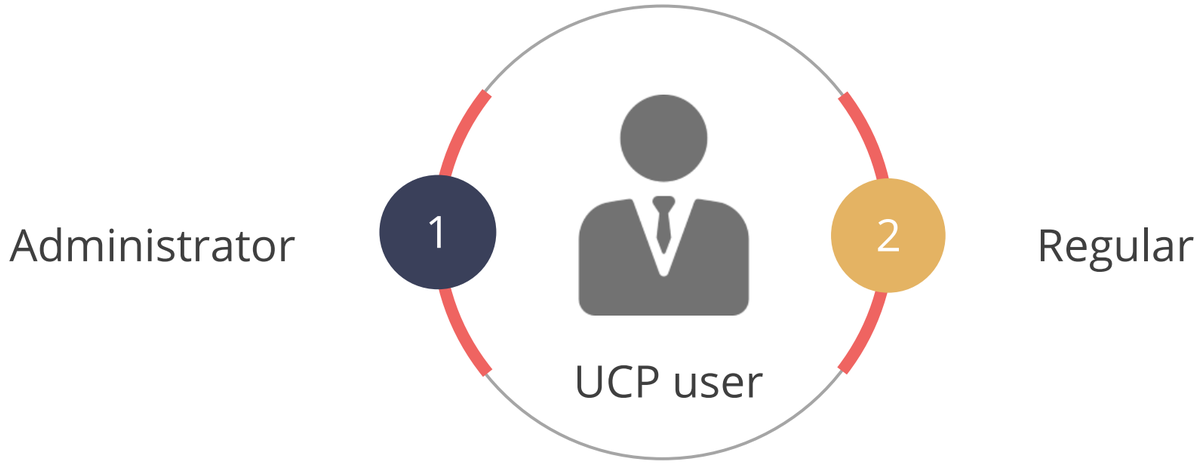
Source: https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/grant-permissions/

Grant: Subject

Subject



Subject



Regular users do not have the privilege to make changes to swarm settings.

Administrator user:

- Manages the user permissions by creating grants
- Manages the swarm configurations

Subject

Create a user:

1. Go to the UCP web user interface
2. Navigate to the **Users** page
3. Click the **Create user** button
4. Fill in the user information
5. Check the **Is a UCP admin?**
6. Click the **Create** button to create the user

Subject

Create an organization and team:

1.Go to the UCP web user interface

2.Navigate to the **Organizations** page

3.Click **Create Organization** to put a team in a new organization

4.Name the new organization “Org” (optional)

5.Click **Create** to create a organization

6.Click the organization “Org”

7.Name the team (description is optional)

8.Click **Create** to create a new team

Subject

Create an organization and team:

9.Click the new team

10.Click **Add Users** in the details pane

11.Choose the users to add it to the team

12.Click **Add Users**

13.Click **Yes** to **Enable Sync Team Members**

14.Choose **LDAP Match Method**

15.Choose **Immediately Sync Team Members**

Create and Manage Teams and Users

Problem Statement: You have been asked by your manager to create users, organization, and teams in Docker UCP that can later be used to create grants.

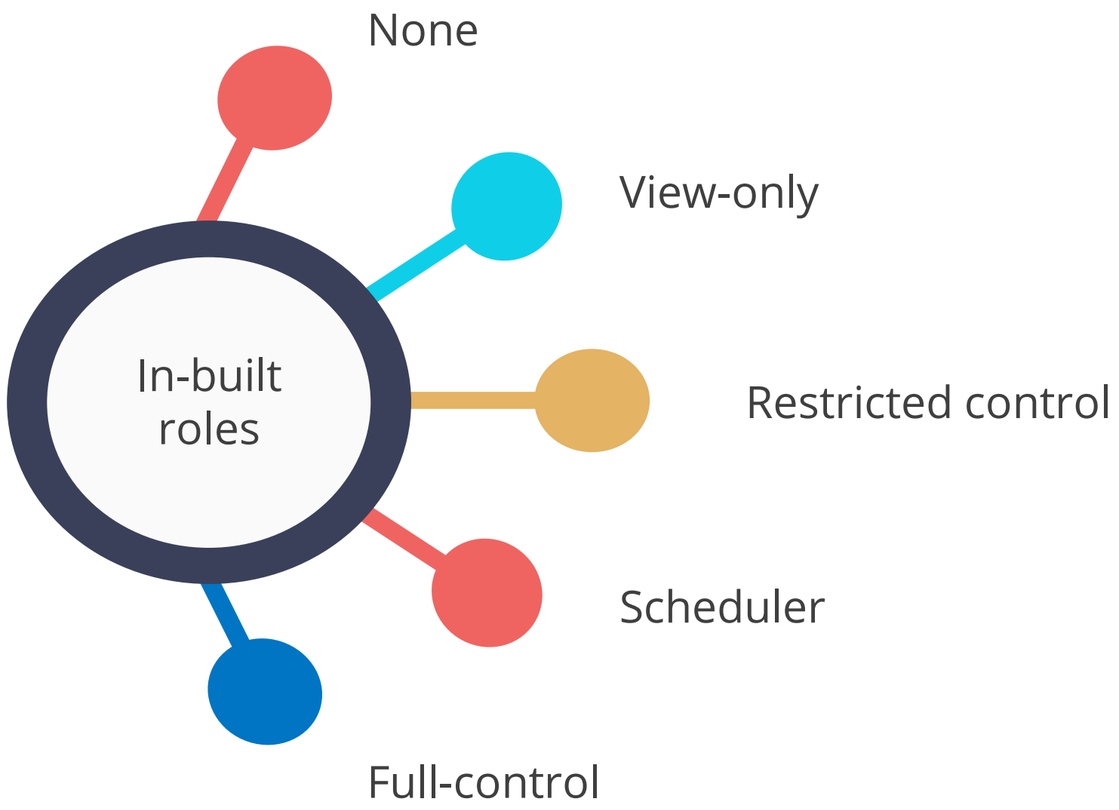
Steps to Perform:

1. Create and manage users.
2. Create and manage an organization.
3. Create and manage teams in an organization.

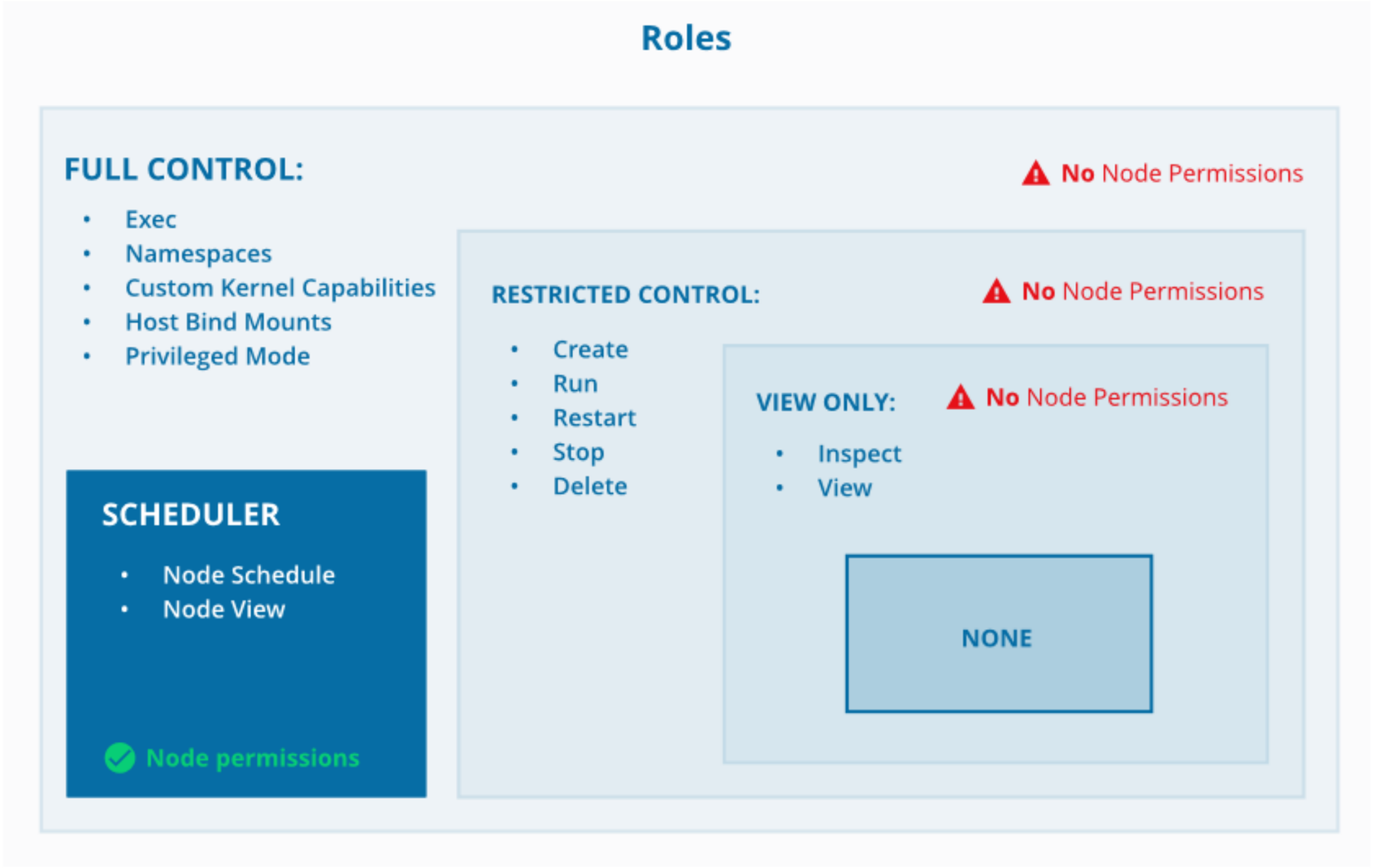
ASSISTED PRACTICE

Grant: Role

Role



Role



Source: https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/

Role

Create a custom role:

1. Go to the UCP web user interface
2. Navigate to the **Roles** page
3. Click **Create role**
4. Define the API operations that the role uses
5. Give a role a global name

Grant: Resource Collections

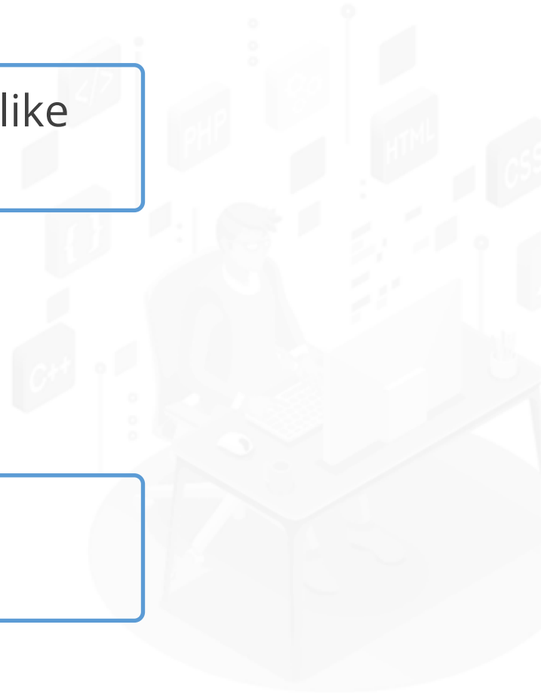
Collection

What is a collection?

The grouping of swarm cluster resources that are accessed by specifying a directory-like path is known as a collection.

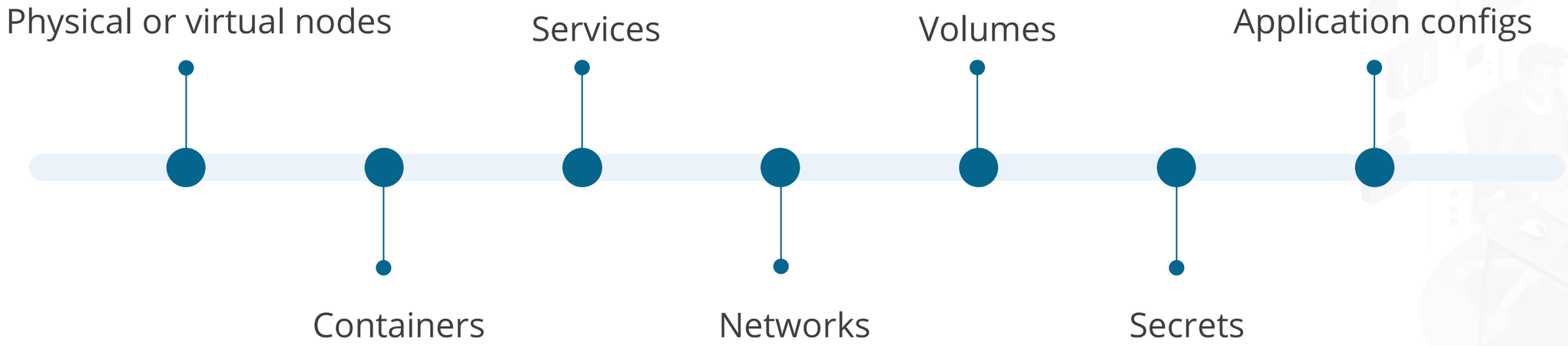
Purpose of a collection:

Collections are used to enable the controlling access to the swarm resources.



Collection

Swarm resources that can be grouped into a collection:



Collection

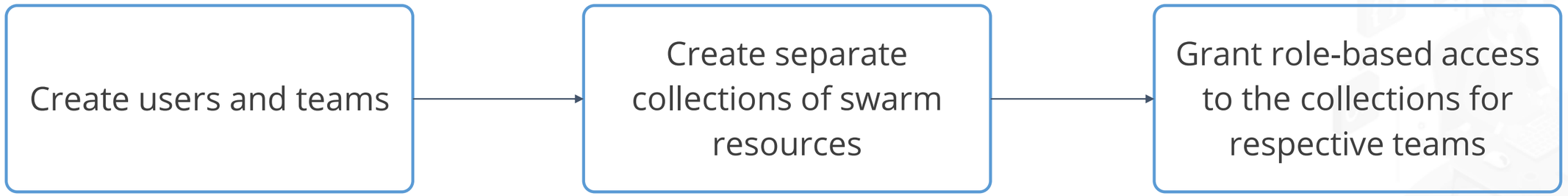
Create a collection:

1. Navigate to the **Collections** page
2. Look for **Shared** collection and click on **View children**
3. Click on **Create collection** and then name the collection as “View-O services”
4. Click **Create** to create a collection

Granting Permissions

Create a Grant

Workflow for creating a grant:



Create a Grant

1. Navigate to the **Manage Grants** page
2. Click **Create Grant**
3. Click **Select** on the desired collection
4. Click **Roles** and select a role from the dropdown list
5. Click **Subjects** and choose **All Users** or **Organizations** where a **user, team,** or **organization** is selected
6. Click **Create**

Deploying a Service

Service Deployment

Steps to deploy a service with view-only access:

1. Create an organization and a team
2. Create a collection for the view-only service
3. Deploy a service
4. Create a grant

Source: https://docs.docker.com/datacenter/ucp/3.0/guides/access-control/deploy-view-only-service/

Service Deployment

Step 1: Create an organization, a team, and add a user

1. Log in as an administrator in to UCP
2. Navigate to the **Organizations and Teams** page
3. Click **Create Organization**
4. Name the organization as “engineering” and click **Create**
5. Click **Create Team,** name the team as “Dev”, and then click **Create**
6. Add a user to the “Dev” team

Service Deployment

Step 2: Create a collection

1. Navigate to the **Collections** page
2. Look for **Shared** collection and click on **View children**
3. Click on **Create collection** and then name the collection as “View-O services”
4. Click **Create** to create a collection

Service Deployment

Step 3: Deploying a service

1. Navigate to the **Services** page
2. Create a new service and name it as “WordPress”
3. Enter “wordpress:latest” in the **Image** textbox
4. Click **Collection** on the left pane
5. Click **View children** to get the list of all the collections
6. Click **View children** in **Shared**
7. Choose **View-only services** collection
8. Click **Create** to add the service to the collection and deploy it

Service Deployment

Step 4: Create grant to give view-only access

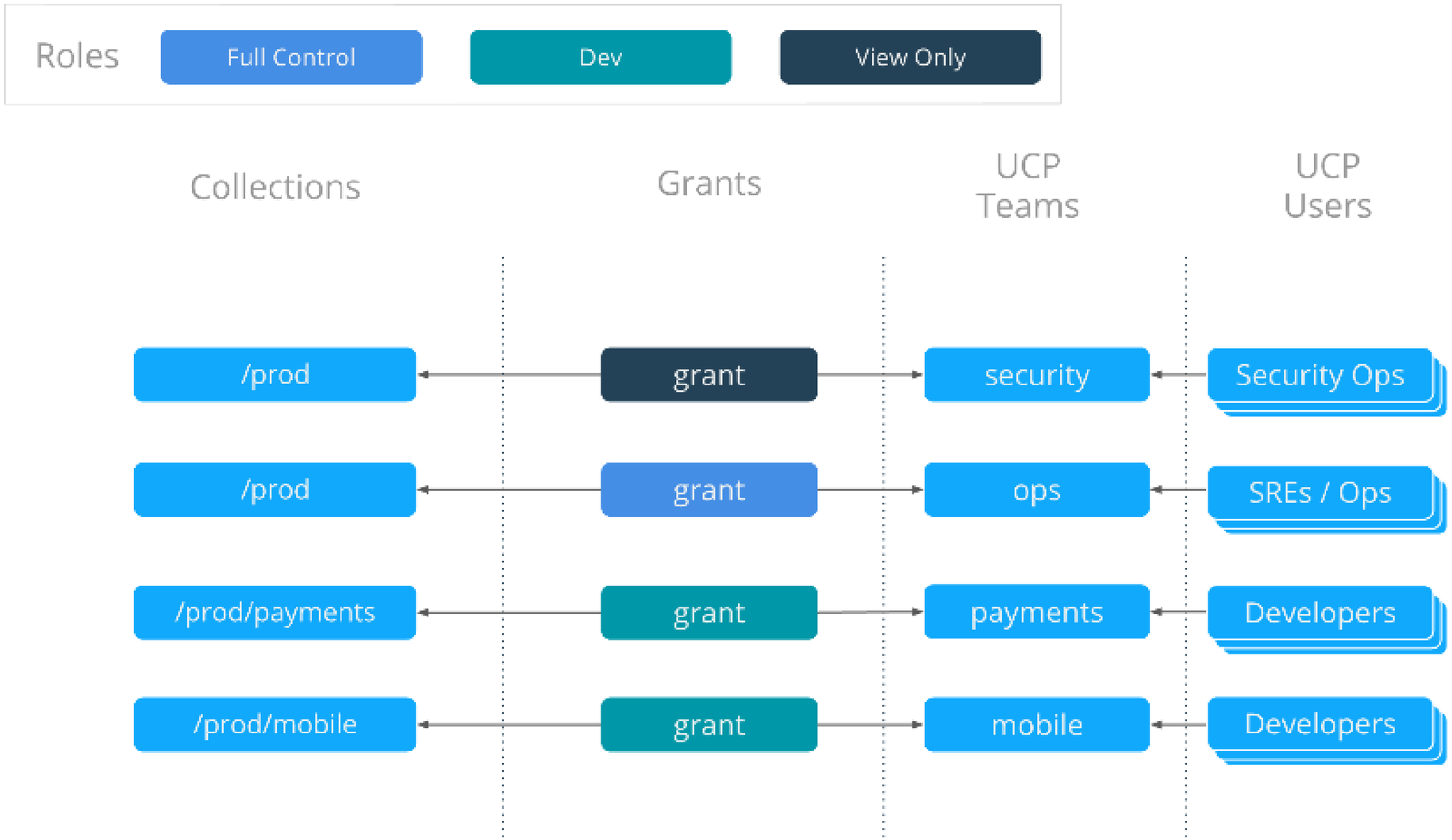
1. Navigate to the **Grants** page
2. Click on **Create Grant**
3. Click on **Collections** on the left pane
4. Navigate to **/Shared/View-O services**
5. Click on **Select Collection**
6. Click on **Roles** and select **View Only** in the dropdown
7. Click on **Subjects** and then click on **Organizations** under the **Select subject type**
8. Select **engineering** from the dropdown and then click on **Create**

Service Deployment

Step 5: Verifying the user's permissions

1. Log in as the Dev team user
2. Navigate to the **Services** page
3. Click **WordPress**
4. Confirm the service collection **/Shared/View-O services** in the details pane
5. Click on the checkbox next to the **WordPress** service
6. Click **Actions** and select **Remove**

Grant Composition



Source: https://docs.docker.com/datacenter/ucp/3.0/guides/access-control/#grant-composition

Docker Trusted Registry (DTR)

Docker Trusted Registry: Overview

Docker Trusted Registry is an image storage solution provided by the Docker. DTR is installed behind the firewall in order to securely store and manage the Docker images that are used in the applications.

Purposes served by DTR from image and job management point of view:

- DTR is used as a part of continuous integration and continuous delivery processes to:
  - Build the applications
  - Ship the applications
  - Run the applications

- DTR web user interface allows the authorized users in the organization to:
  - Browse Docker images
  - Review repository events

- DTR allows to see:
  - Which Dockerfile lines produced the image
  - Whether security scanning is enabled
  - The list of all the softwares installed in the images

Docker Trusted Registry: Overview

Availability of DTR:

It is available during the use of multiple replicas of all the containers and metadata. DTR continues to operate even if the machine fails.

Efficiency of DTR:

- DTR efficiently caches the images closer to the user. This reduces the amount of bandwidth required for pulling Docker images.
- DTR efficiently cleans up the unreferenced manifests and layers.

Image signing:

DTR ships have built-in Notary that uses Docker Content Trust to sign and verify the images.

Docker Trusted Registry: Overview

Features of built-in access control :

- The authentication mechanism used by DTR is the same as the one used by the Docker Universal Control Plane.
- Role Based Access Control (RBAC) is used by DTR in order to implement access control policies for the Docker images.

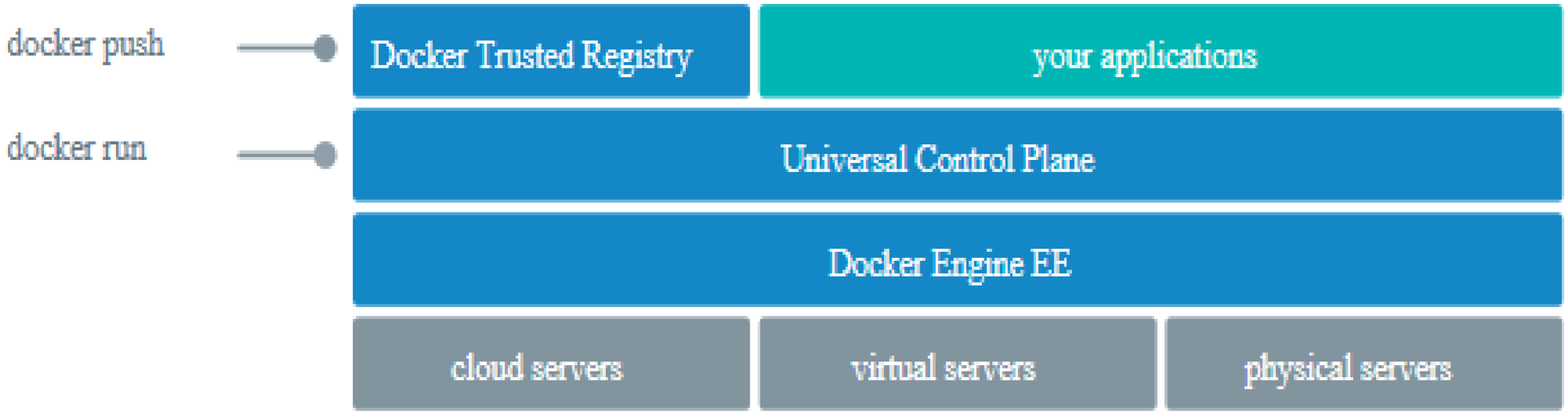
Security features of DTR:

- The built-in security scanner is used to discover the versions of software that are used in the images.
- The built-in security scans all the layers and aggregates the results for the components that are being shipped as a part of the stack.
- The built-in security correlates to the information with a vulnerability database. This provides an unmatched insight of known security threats.

DTR Architecture

Architecture

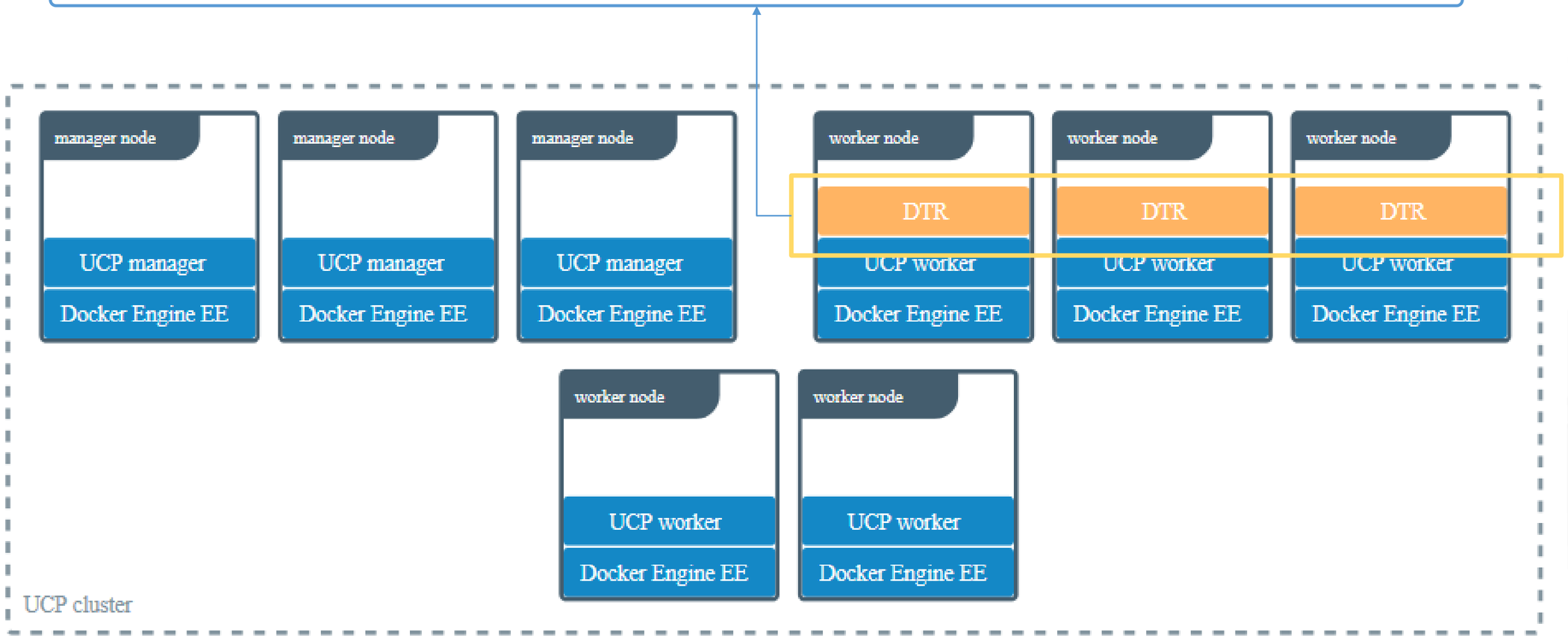
DTR is the containerized application, that on deployment uses Docker CLI client to login, push, and pull images.



Source: https://docs.mirantis.com/docker-enterprise/v3.1/dockeree-products/dtr/architecture.html

Architecture

Multiple DTR replicas are deployed on the UCP worker nodes to achieve high availability.



Source: https://docs.mirantis.com/docker-enterprise/v3.1/dockeree-products/dtr/architecture.html

DTR Components

Following containers are started after the installation of DTR:

Name	Description
dtr-api-<replica_id>	Executes the DTR business logic
dtr-garant-<replica_id>	Manages the DTR authentication
dtr-jobrunner-<replica_id>	Runs the cleanup jobs in the background
dtr-nginx-<replica_id>	Receives https/http requests and proxies them to other DTR components
dtr-notary-server-<replica_id>	Receives, serves, and validates content trust metadata
dtr-notary-signer-<replica_id>	Performs server-side timestamp and snapshot signing for content trust metadata
dtr-registry-<replica_id>	Implements the functionality for pulling and pushing Docker images
dtr-rethinkdb-<replica_id>	It's a database for persisting repository metadata
dtr-scanningstore-<replica_id>	Stores security scanning data

Networks and Volumes

Networks

Networks used:

Networks are created while installing the DTR. This makes the containers capable of communication.

Name	Type	Description
dtr-ol	overlay	It allows communication of DTR components that are running on different nodes. It also allows the replication of DTR data.

Volumes

Volumes used:

Volume name	Description
dtr-ca-<replica_id>	Root key material for the DTR root certificate authority (CA) that issues certificates
dtr-notary-<replica_id>	Certificate and keys for the Notary components
dtr-postgres-<replica_id>	Data of vulnerability scans
dtr-registry-<replica_id>	Data of Docker images (when DTR is configured to store images on the local filesystem)
dtr-rethink-<replica_id>	Repository metadata
dtr-nfs-registry-<replica_id>	Docker images data (when DTR is configured to store images on NFS)

Image Storage

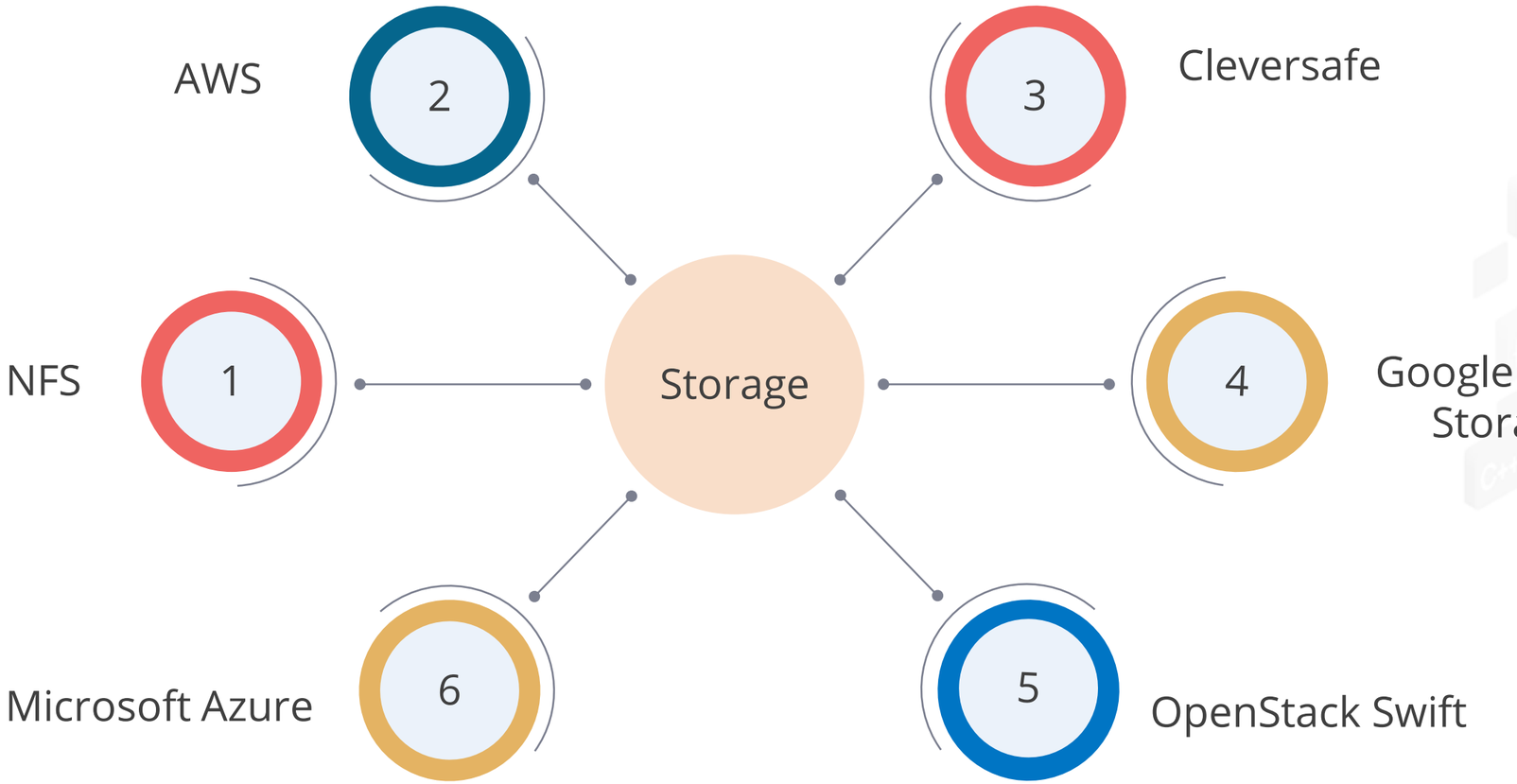
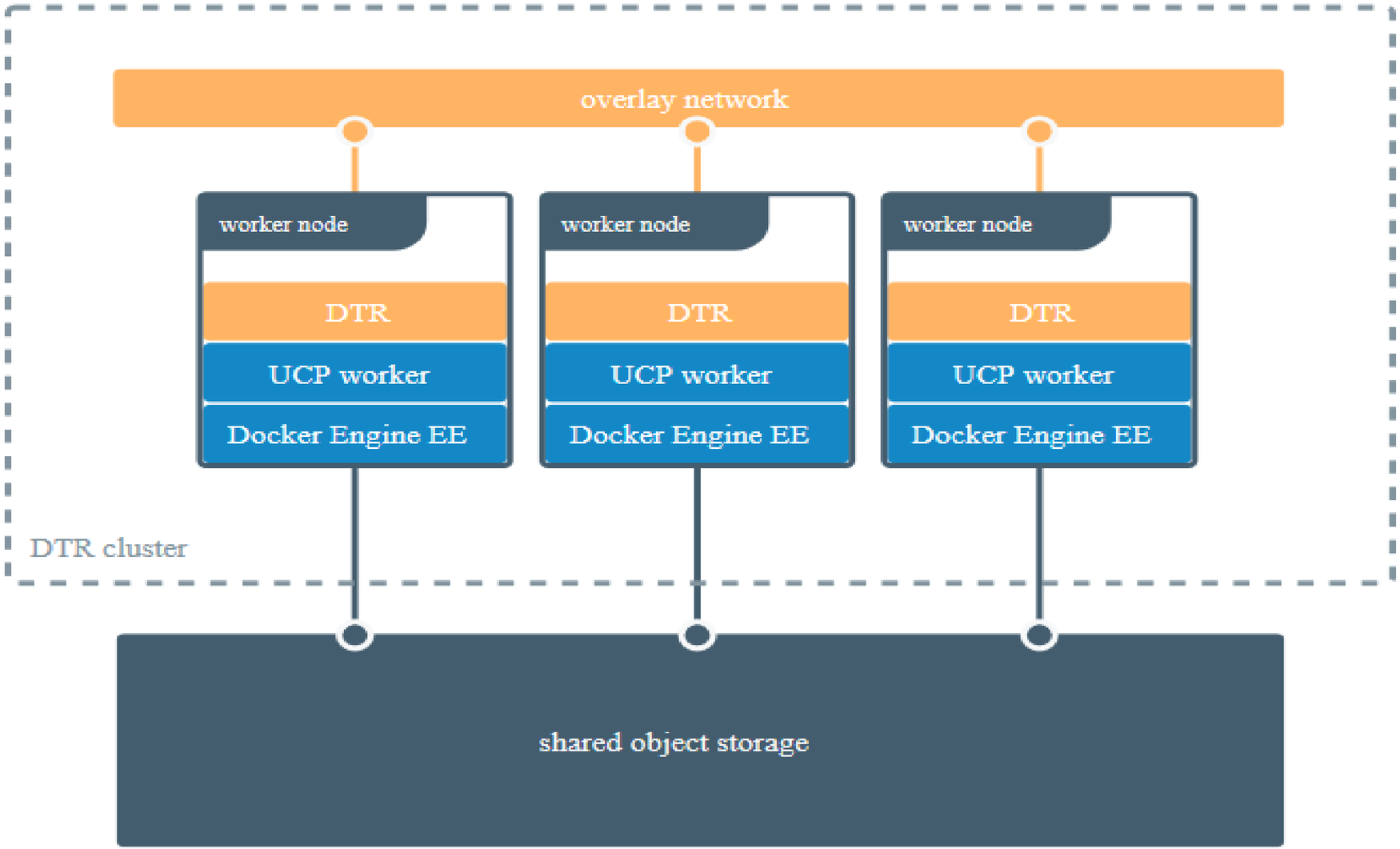


Image Storage



Source: https://docs.mirantis.com/docker-enterprise/v3.1/dockeree-products/dtr/dtr-configure/configure_external_storage.html

Installation of DTR

Installation Requirements

Hardware and software requirements:

- Nodes should always be the worker nodes and must be managed by Universal Control Plane
- Nodes must have a fixed hostname

Minimum requirements	Recommended production requirements
16GB of RAM for nodes that are running the DTR	16GB of RAM for nodes that are running the DTR
2 vCPUs for nodes that are running the DTR	4 vCPUs for nodes that are running the DTR
10GB of free disk space	25-100GB of free disk space

Installation Requirements

The following ports must remain open on the node where DTR is being installed:

Direction	Port	Purpose
in	80/tcp	Web app and API client access to DTR
in	443/tcp	Web app and API client access to DTR

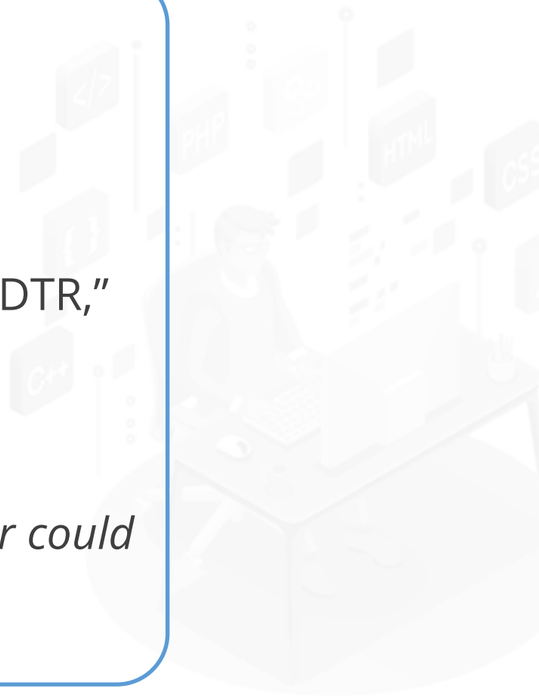
Installation Requirements

UCP configuration:

During the installation or backing up of DTR on a UCP cluster, the administrator must deploy containers on “UCP manager nodes or nodes running the DTR.”

If administrators are not deployed on the “UCP manager nodes or the nodes running DTR,” then the DTR installation or backup will fail and display the following error:

Error response from daemon: {"message":"could not find any nodes on which the container could be created"}



Installation Requirements

UCP setting: Restricting users from deploying to manager nodes:

1. Log in to the UCP web interface as an administrator
2. Navigate to the **Admin Settings** page
3. Choose **Scheduler**

Install Docker Trusted Registry

Problem Statement: Your manager has asked you to install and set up the Docker Trusted Registry (DTR) so that images can be pushed and scanned for vulnerabilities.

Steps to Perform:

1. Get DTR install command from Docker Trusted Registry tab in UCP.
2. Run the command on Worker1 node to install DTR.
3. Login to DTR with admin credentials.

ASSISTED PRACTICE

Post-Installation

Post-Installation steps:

1. Check that the DTR is running:
   - Navigate to the UCP web interface
   - Select **Shared Resources**
   - Click **Stacks**
   - Enter the DTR IP address or FQDN (Fully Qualified Domain Name) on the address bar in order to verify that the DTR is accessible from the browser

Post-Installation

Post-Installation steps:

2. Configure the certificates that are used for TLS communication:
   - Log in to the *https://<dtr-url>*
   - Select **System** and scroll down to **Domain and Proxies**
   - Enter DTR domain name and upload:
     - **Load balancer/public address**
     - **TLS private key**
     - **TLS certification chain**
     - **TLS CA**
   - Click **Save**

3. Configure the storage backend to store the Docker images:
   - Log in as an admin into the DTR web interface
   - Navigate to **System**
   - Click **Storage**

Post-Installation

Post-Installation steps:

4. Testing whether the images can be pushed or pulled:

    a. Configure the **Ubuntu/Debian** operating system to trust the certificate:

        i. Download the DTR CA certificate

        sudo curl -k https://<dtr-domain-name>/ca -o /usr/local/share/ca-certificates/<dtr-domain-name>.crt

        ii. Refresh the list of certificates to trust

        sudo update-ca-certificates

        iii. Restart the Docker daemon

        sudo service docker restart

Post-Installation

Post-Installation steps:

4. Testing whether the images can be pushed or pulled:

   b. Create an image repository:
      i. Log in to *https://<dtr-url* with UCP credentials, if image repository is being created for the first time
      ii. Select **Repositories** on left navigation pane
      iii. Click **New repository** on the upper right corner
      iv. Select the namespace and name the repository
      v. Choose the repository type
      vi. Click **Create** to create the repository

Post-Installation

Post-Installation steps:

4. Testing whether the images can be pushed or pulled:

    c. Pull an image:

        i. Access the DTR at *dtr-example.com* (DTR external URL)

        ii. Permission is granted to access the nginx and wordpress repositories

        iii. Run the following command to pull the latest tag of the wordpress image:

        docker login dtr-example.com

        docker pull dtr-example.com/library/wordpress:latest

Post-Installation

Post-Installation steps:

4. Testing whether the images can be pushed or pulled:

    d. Push an image:

        i. Tag the image:

        # Pull from Docker Hub the latest tag of the wordpress image

        docker pull wordpress:latest

        # Tag the wordpress:latest image with the full repository name we've created in DTR

        docker tag wordpress:latest dtr-example.com/library/wordpress:latest

        ii. Push the image to DTR:

        docker login dtr-example.com

        docker push dtr-example.com/library/wordpress:latest

Post-Installation

Post-Installation steps:

5. Join the replicas to the cluster:
    i. Load the UCP user bundle
    ii. Run command:
    *docker run -it --rm *
    *docker/dtr:2.7.3 join *
    *--ucp-node <ucp-node-name> *
    --ucp-insecure-tls
    iii. Check the existing replicas:
        1. Navigate to UCP web interface
        2. Select **Shared Resources**
        3. Click **Stacks**
        The replicas will be displayed on the console

Uninstallation

Uninstallation step:

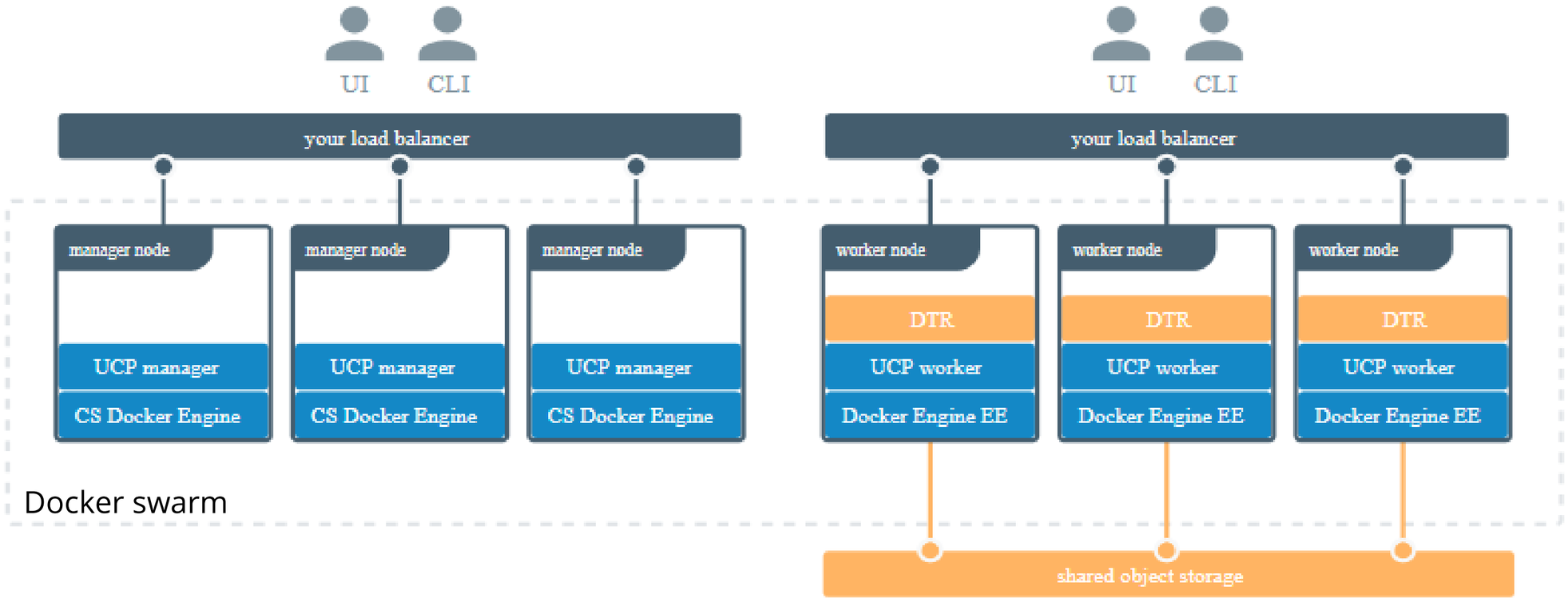
Destroy command must be run for every replica:

```
docker run -it --rm \
  docker/dtr:2.7.3 destroy \
  --ucp-insecure-tls
```

DTR: High Availability

High Availability

Docker Trusted Registry (DTR), an enterprise-grade image storage solution from Docker, is capable of scaling horizontally. As the usage increases, the replicas can be added to make the DTR scale for high availability.



DTR Failure Tolerance

Additional replicas must be added to the Docker Trusted Registry (DTR) cluster in order to make DTR tolerant to failures.

DTR replicas	Failures tolerated
1	0
3	1
5	2
7	3

Sizing DTR Installation

Thumb rules to size the DTR installation:

- Create a DTR cluster with more than two replicas
- Keep the replica online all the time
- Add enough number of replicas

The minimum number of nodes required for having high-availability on UCP and DTR:

Dedicated nodes	UCP/DTR/Containers/Applications
3	For installation of UCP with high availability
3	For installation of DTR with high availability
n	For running containers and applications

Note: Configure the DTR replicas in order to share the same object storage.

Add Replicas

Adding replicas to an existing DTR deployment:

- Use *ssh* to log in to a node that is a part of a UCP
- Run the following DTR join command:

```
docker run -it --rm \
    docker/dtr:2.7.3 join \
    --ucp-node <ucp-node-name> \
            --ucp-insecure-tls
```

- Add this DTR replica to the load balancing pool in case the load balancer is available

It is the hostname of the UCP node where the DTR replica is deployed.

It confirms the certificates used by UCP

Remove Replicas

Removing replicas from DTR deployment:

- Use *ssh* to log in to any node that is part of a UCP.

- Run the following DTR remove command:

  *docker run -it --rm *
  *docker/dtr:2.7.3 remove *
  --ucp-insecure-tls

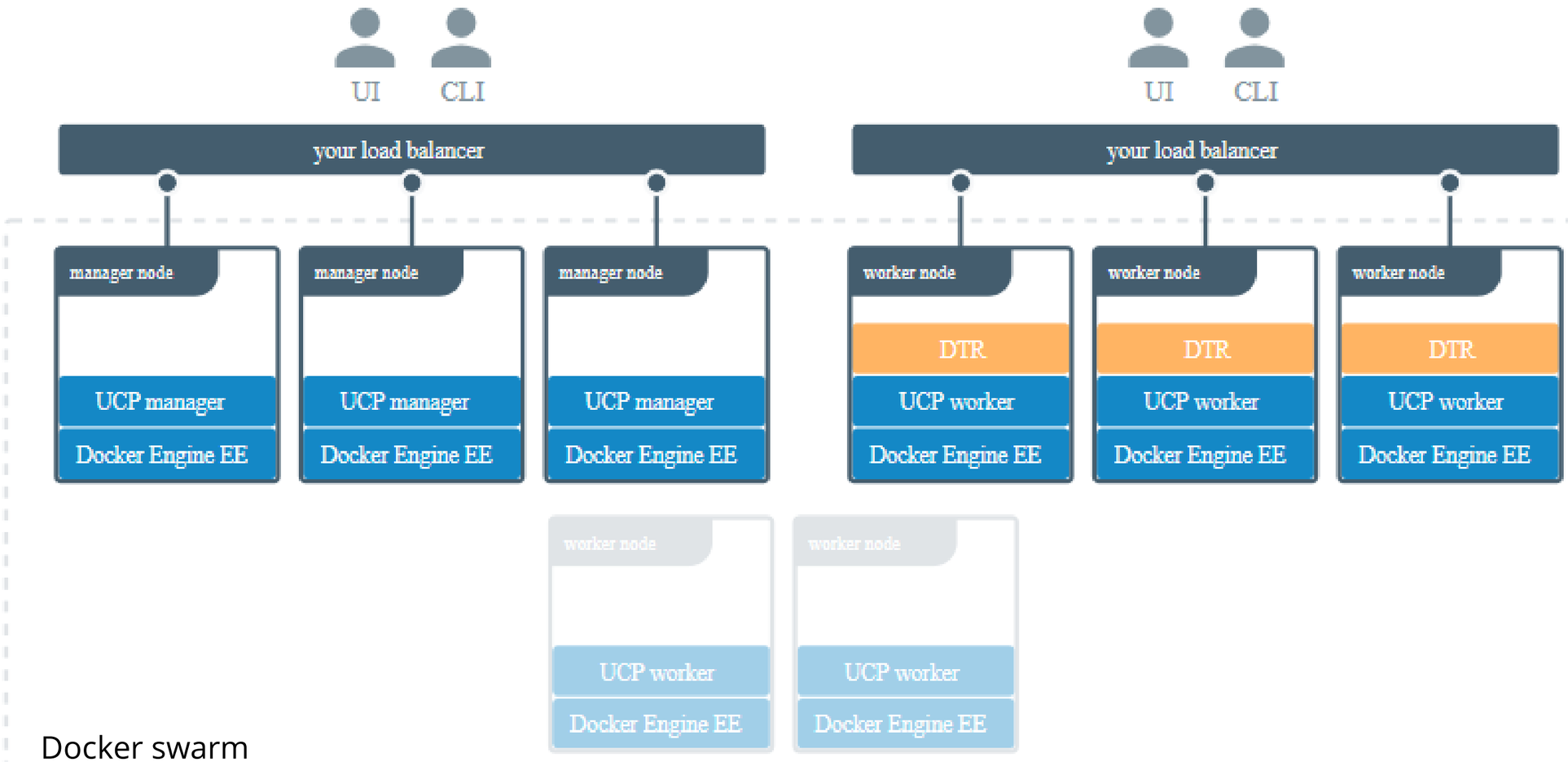
Prompts:

- Existing replica
- Replica id
- UCP username and password

DTR: Load Balancer

Load Balancer

The load balancer is configured to balance user requests across all replicas. Thus, users can access the DTR using a centralized domain name.



Load Balancer

Endpoints are exposed by DTR:

- */_ping*: Checks the health of DTR replica
- /nginx_status: Returns the number of connections being handled by the NGINX front-end used by DTR
- /api/v0/meta/cluster_status: Returns extensive information about all DTR replicas

Note: Load balancing service is not provided by DTR. On-premises or a cloud-based load balancer can be used to balance requests across multiple DTR replicas.

Configure Load Balancer

Load balancer must be configured to:

- Load balance the TCP traffic on ports 80 and 443
- Stop termination of HTTPS connections
- Stop buffer requests
- Forward the Host HTTP header correctly
- Set timeout of more than 10 minutes or set no timeout for idle connections

Health Check of Replicas

Health check:

Command:

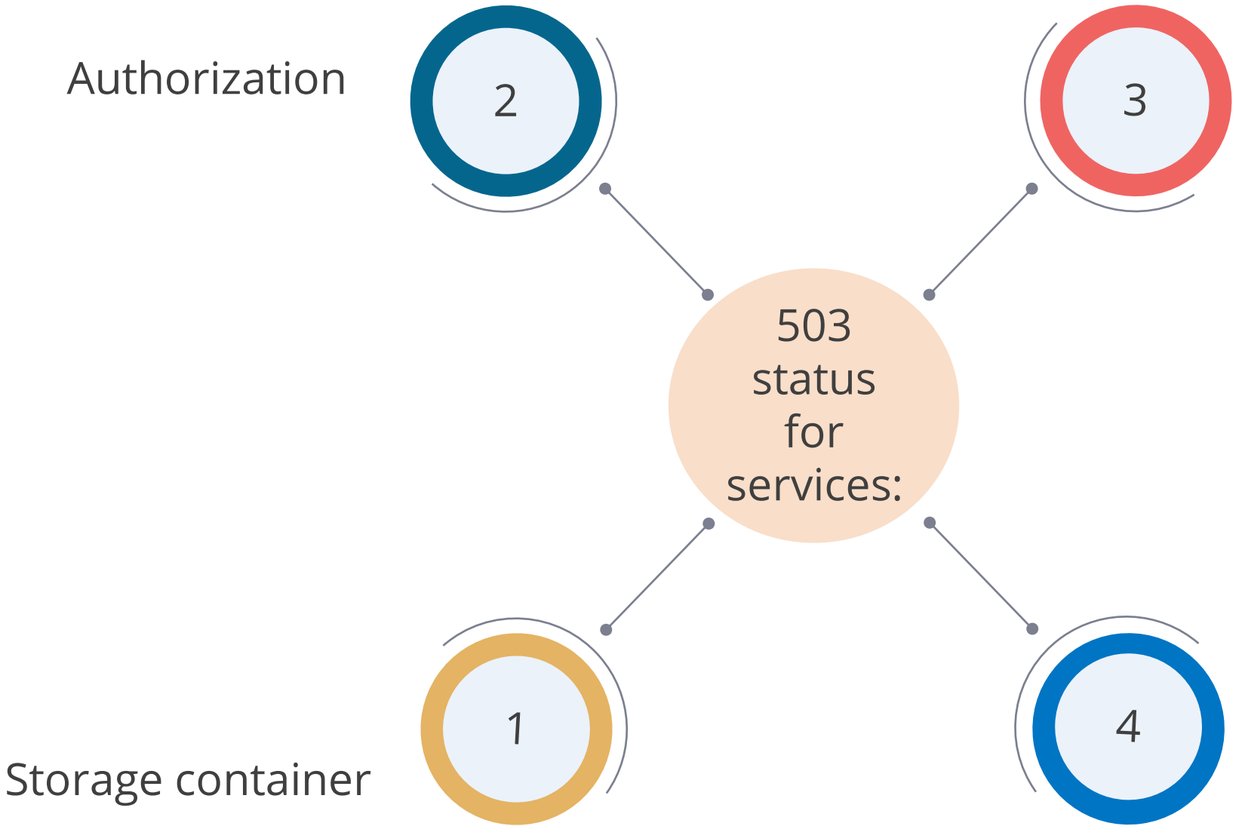
/_ping

Output:

```
{
        "Error": "error message",
        "Healthy": true
}
```

true tells that the replica is suitable for taking requests.

Health Check of Replicas



Load Balancer: Configuration and Deployment

Configure load balancer for DTR using NGINX:

```
user  nginx;
worker_processes  1;

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

events {
    worker_connections  1024;
}
stream {
    upstream dtr_80 {
        server <DTR_REPLICA_1_IP>:80  max_fails=2 fail_timeout=30s;
        server <DTR_REPLICA_2_IP>:80  max_fails=2 fail_timeout=30s;
        server <DTR_REPLICA_N_IP>:80   max_fails=2 fail_timeout=30s;
    }
```

Load Balancer: Configuration and Deployment

Configure load balancer for DTR using NGINX:

```
upstream dtr_443 {
    server <DTR_REPLICA_1_IP>:443 max_fails=2 fail_timeout=30s;
    server <DTR_REPLICA_2_IP>:443 max_fails=2 fail_timeout=30s;
    server <DTR_REPLICA_N_IP>:443  max_fails=2 fail_timeout=30s;
}
server {
    listen 443;
    proxy_pass dtr_443;
}

server {
    listen 80;
    proxy_pass dtr_80;
}
}
```

Load Balancer: Configuration and Deployment

Deploying load balancer using NGINX:

It is a two step process:

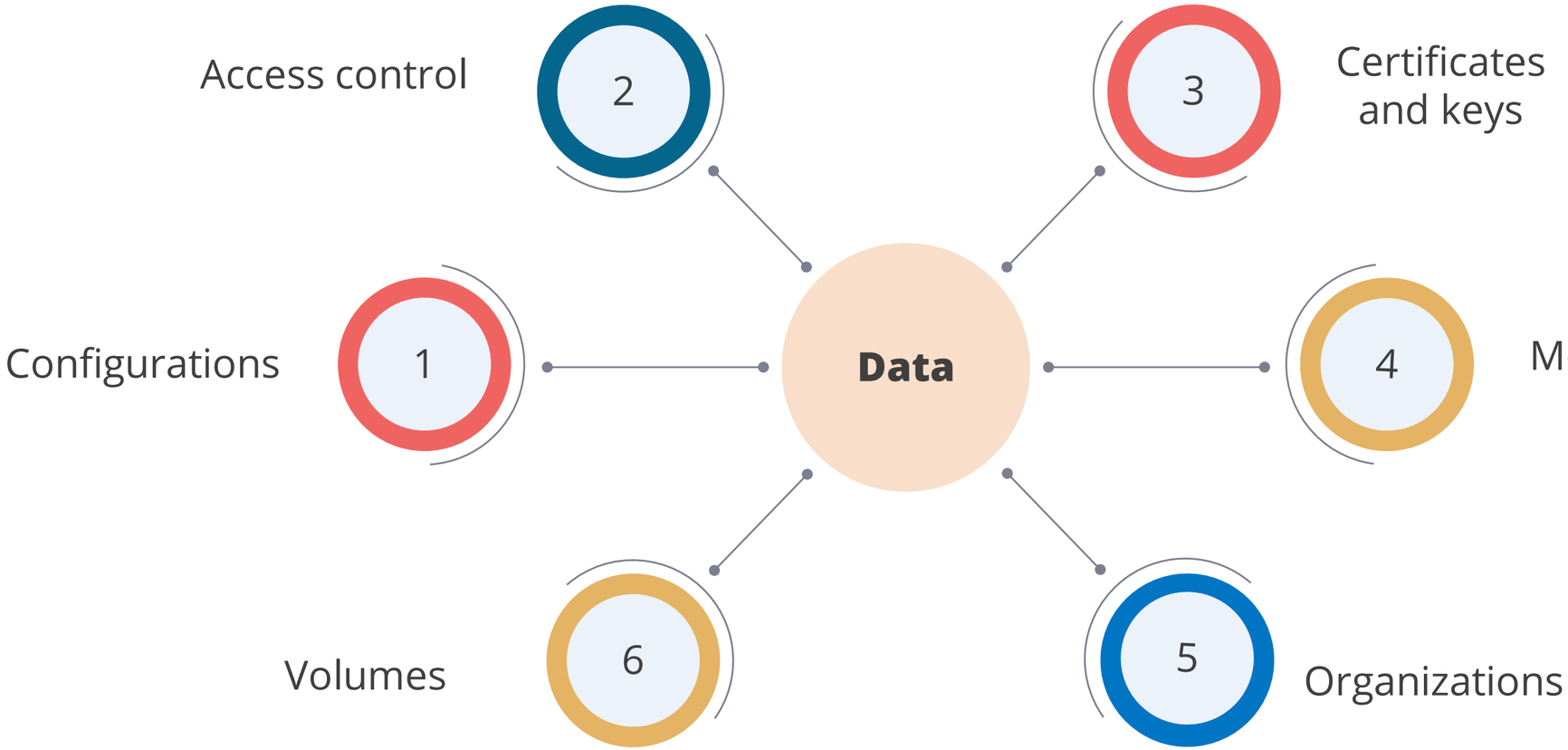
1. Creating file “nginx.conf”
2. Deploying the load balancer

Command:

```
docker run --detach \
  --name dtr-lb \
  --restart=unless-stopped \
  --publish 80:80 \
  --publish 443:443 \
  --volume ${PWD}/nginx.conf:/etc/nginx/nginx.conf:ro \
  nginx:stable-alpine
```

UCP: Backup and Restore

UCP: Backup



UCP: Backup

Procedure to create a UCP backup using CLI:

1. Run the **docker/ucp:** (latest version) backup command on a single UCP manager
2. Replace the latest version with the version the user is currently running
3. View the backup progress and error reporting
4. Verify a UCP Backup

UCP: Backup

Procedure to create a UCP backup using UI:

1. In the UCP UI, navigate to Admin Settings
2. Select Backup Admin
3. Select Backup Now to trigger an immediate backup

UCP: Restore

To restore an existing UCP installation from a backup:

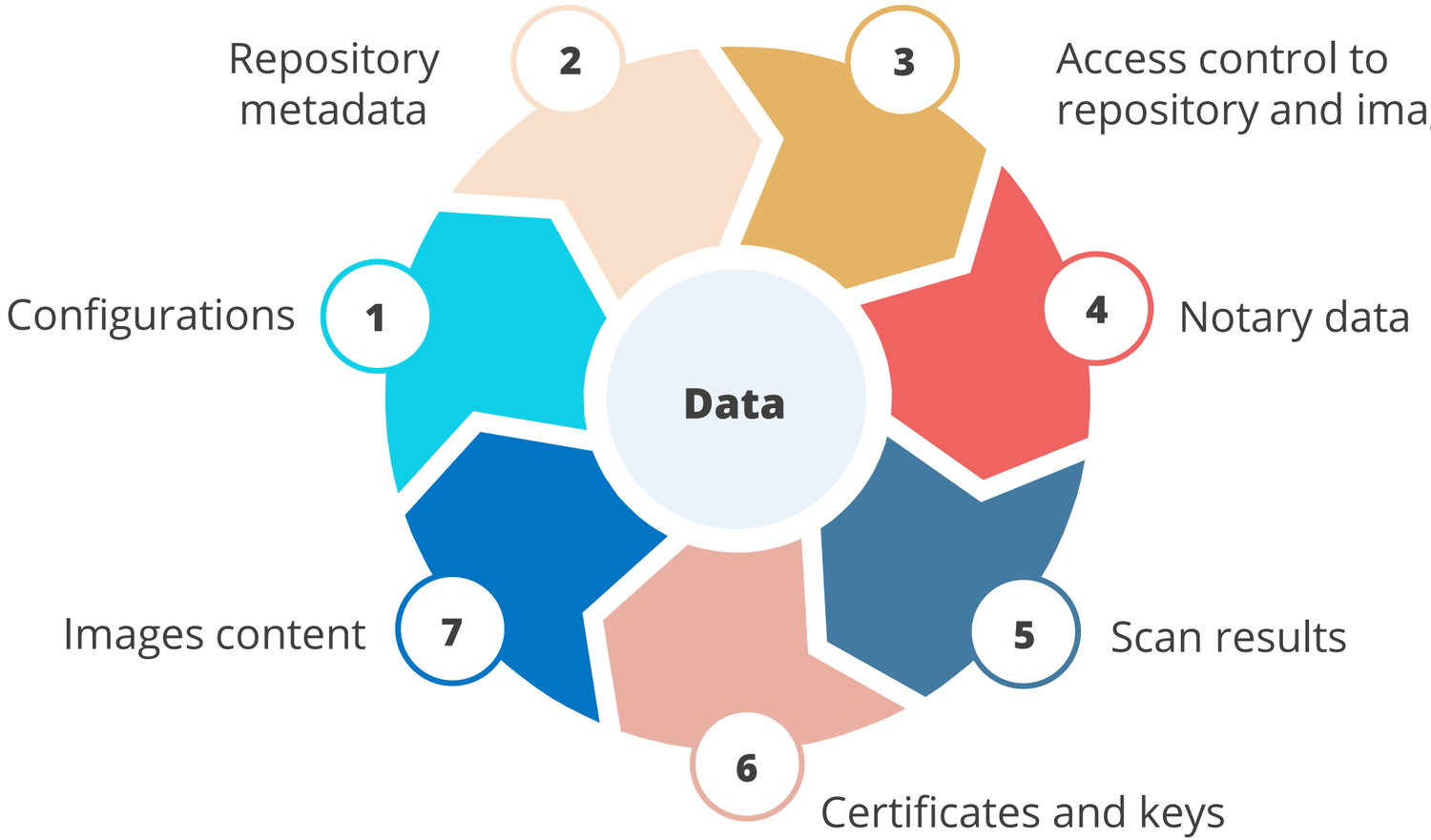
- The user needs to uninstall UCP from the swarm by using the **uninstall-ucp** command.
- The user should use the same version of the docker/ucp image that they have used to create the backup while restoring.

The example below shows how to restore UCP from an existing backup file, presumed to be located at **/tmp/backup.tar**:

```
$ docker container run --rm -i --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock  \
  docker/ucp:2.2.22 restore < /tmp/backup.tar
```

DTR: Backup and Restore

DTR: Backup



DTR: Backup

Procedure for DTR backup:

1. Run DTR Backup command
2. Back up DTR image content
3. Back up DTR metadata
4. Verify your backup

DTR: Restore

To restore DTR, the user needs to:

1. Stop any DTR containers that might be running
2. Restore the images from a backup
3. Restore DTR metadata from a backup
4. Re-fetch the vulnerability database

Disaster Recovery

UCP Disaster Recovery

Recover swarm from losing the quorum

- Uninstall UCP using the **uninstall-ucp** command, if UCP is still installed on the swarm.

- Perform restore from an existing backup on any node:
  - If there is an existing swarm, the restore operation must be performed on a manager node.
  - If no swarm exists, the restore operation will create one.

UCP Disaster Recovery

Recover a UCP cluster without an existing backup

1. Perform docker swarm init **--force-new-cluster** on one of the remaining manager nodes
2. Create a backup of the remaining manager node
3. Uninstall UCP using the **uninstall-ucp** command if UCP is still installed on the swarm
4. Perform a restore on the recovered swarm manager node
5. Log in to UCP and browse to the nodes page, or use the CLI **docker node ls** command
6. Remove the nodes from the swarm
7. Create a backup of the restored cluster

DTR Disaster Recovery

Docker Trusted Registry is a clustered application. The user can join multiple replicas for high availability.

To make a DTR cluster healthy, a majority of its replicas (n/2 + 1) need to be healthy and should be able to communicate with the other replicas. This is also known as maintaining quorum.

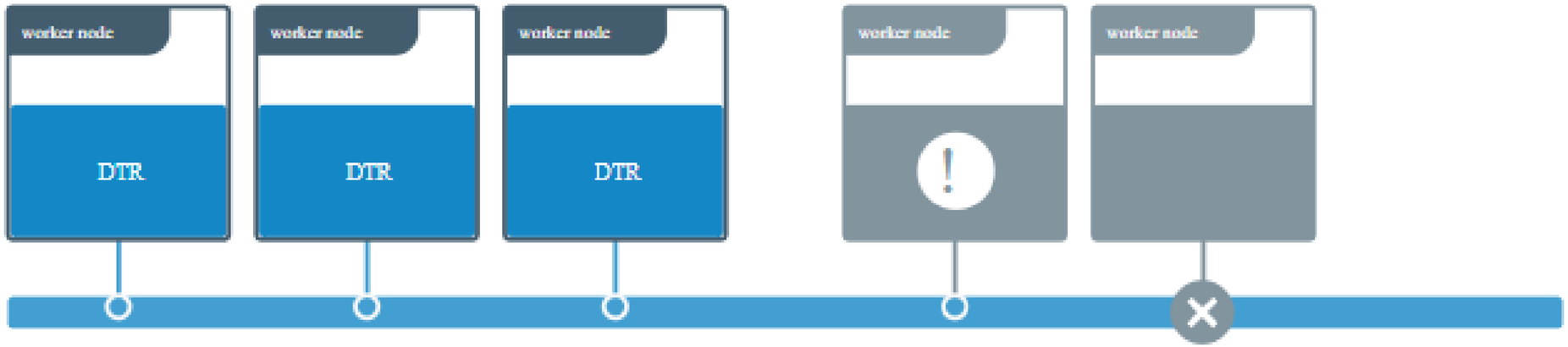
There are three failure scenarios possible:

- Replica is unhealthy but cluster maintains quorum
- The majority of replicas are unhealthy
- All replicas are unhealthy

DTR Disaster Recovery

Replica is unhealthy but cluster maintains quorum

One or more replicas are unhealthy, but the overall majority (n/2 + 1) is still healthy and are able to communicate with one another.

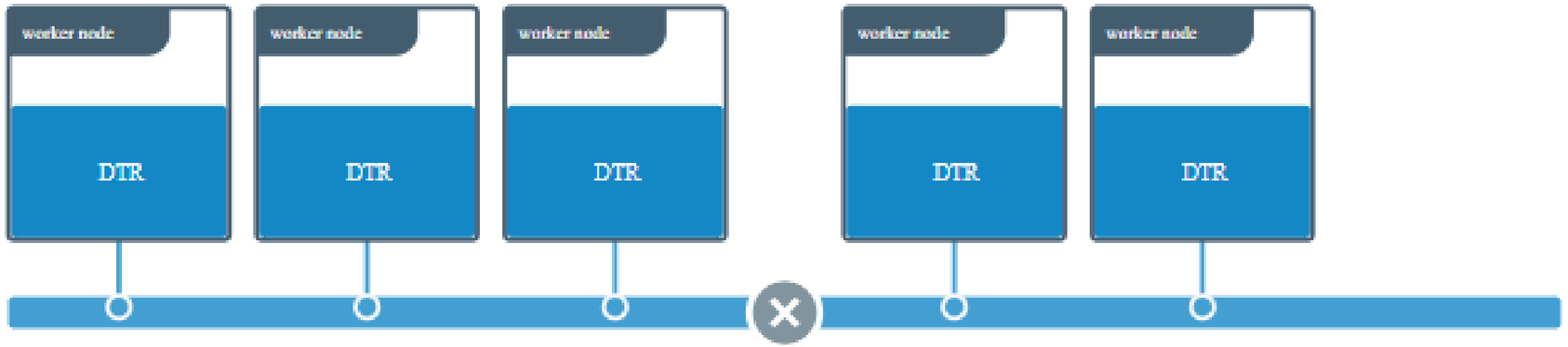


Source:: https://docs.docker.com/ee/admin/disaster-recovery/

DTR Disaster Recovery

Split-brain scenario

There are five-replicas DTR deployment, and something goes wrong with the overlay network connection the replicas, causing them to be separated in two groups.

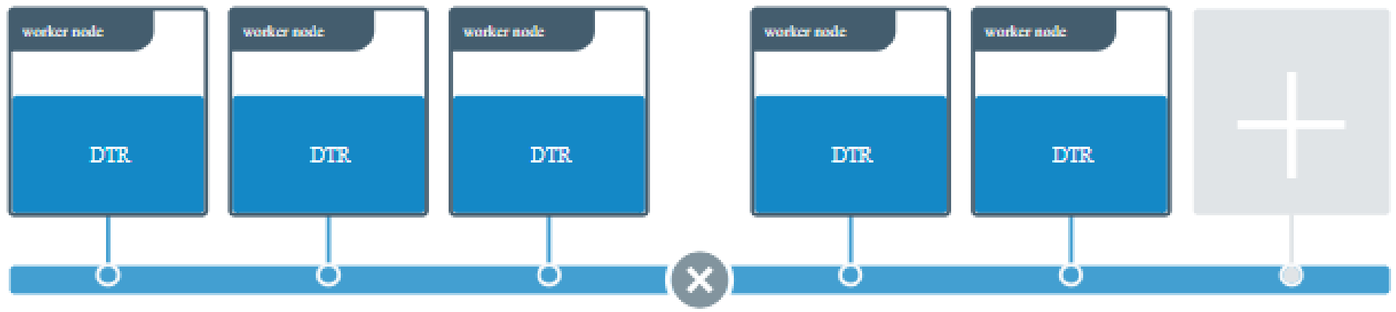


Source:: https://docs.docker.com/ee/admin/disaster-recovery/

DTR Disaster Recovery

Split-brain scenario

When you join a new replica at this point instead of fixing the network problem or removing the two replicas which have been isolated from the rest, the new replica which end up on the side of the network partition which has less replicas.



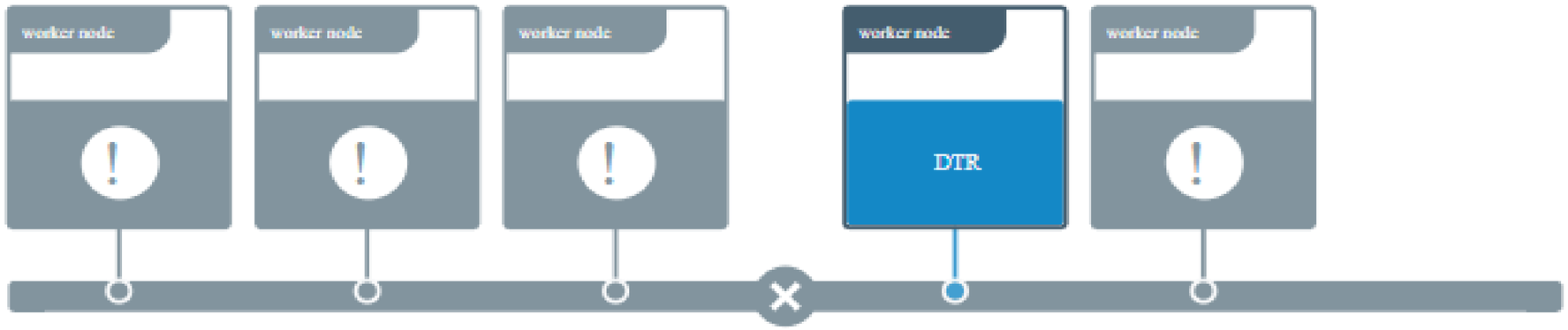
Both groups now have the minimum amount of replicas needed to establish a cluster. This is known as a split-brain scenario because both groups can now accept writes, and their histories start to diverge, effectively making the two groups two different clusters.

Source:: https://docs.docker.com/ee/admin/disaster-recovery/

DTR Disaster Recovery

The majority of replicas are unhealthy

If the majority of replicas are unhealthy, but at least one replica is still healthy, or at least the data volumes for DTR are accessible from that replica, the user can repair the cluster without having to restore from a backup. This minimizes the amount of data loss.

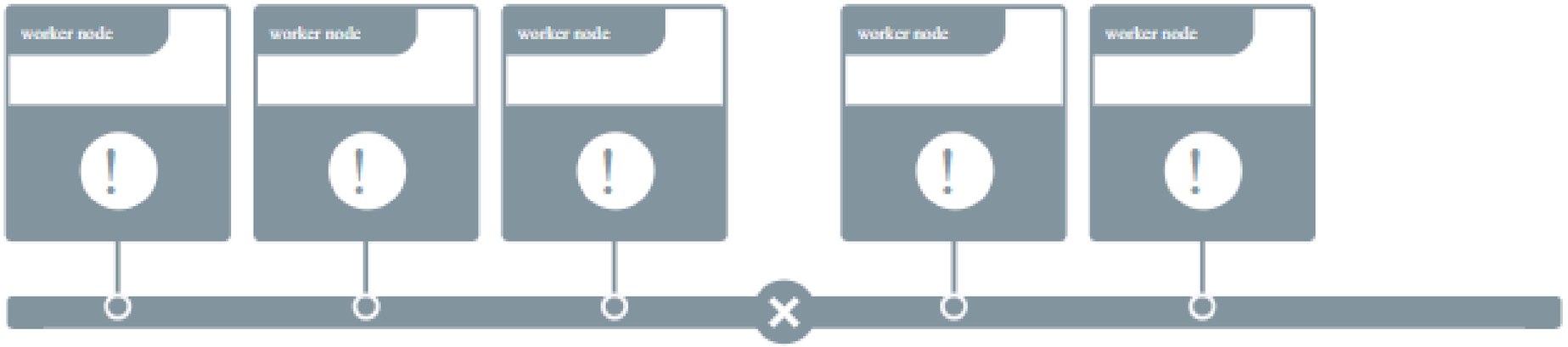


Source: https://docs.docker.com/ee/admin/disaster-recovery/

DTR Disaster Recovery

All replicas are unhealthy

This is a total disaster scenario in which all DTR replicas are lost, causing the data volumes for all DTR replicas to get corrupted or lost.



Source: https://docs.docker.com/ee/admin/disaster-recovery/

Deploy Docker EE for AWS

Deployment Options

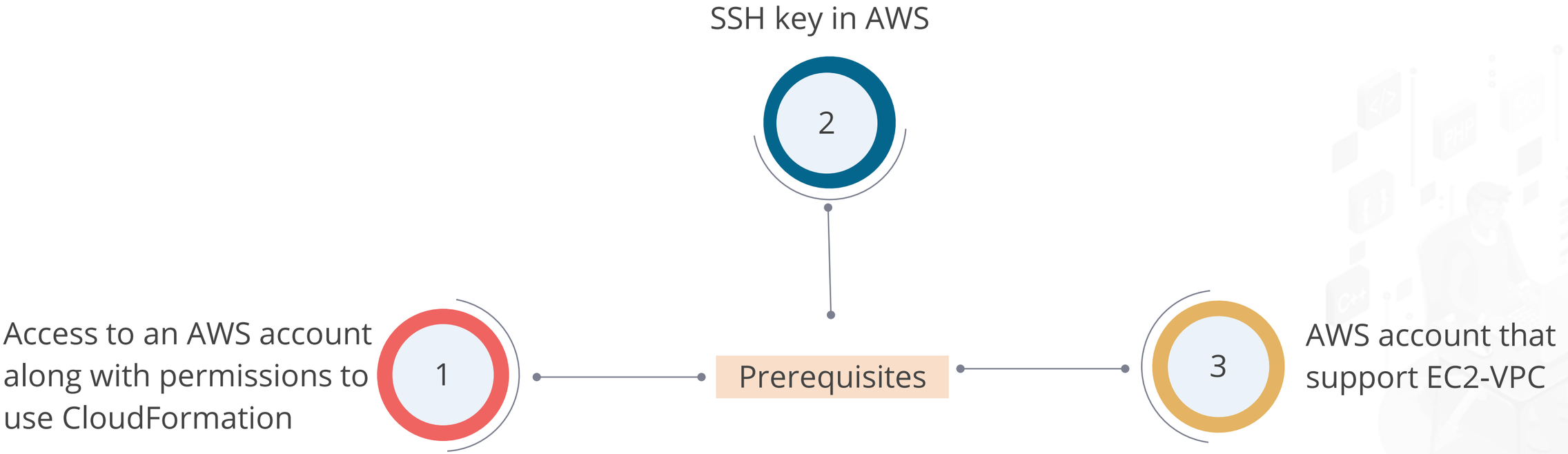
Types of Docker for AWS:

- Docker Enterprise Edition for AWS
- Docker Community Edition for AWS

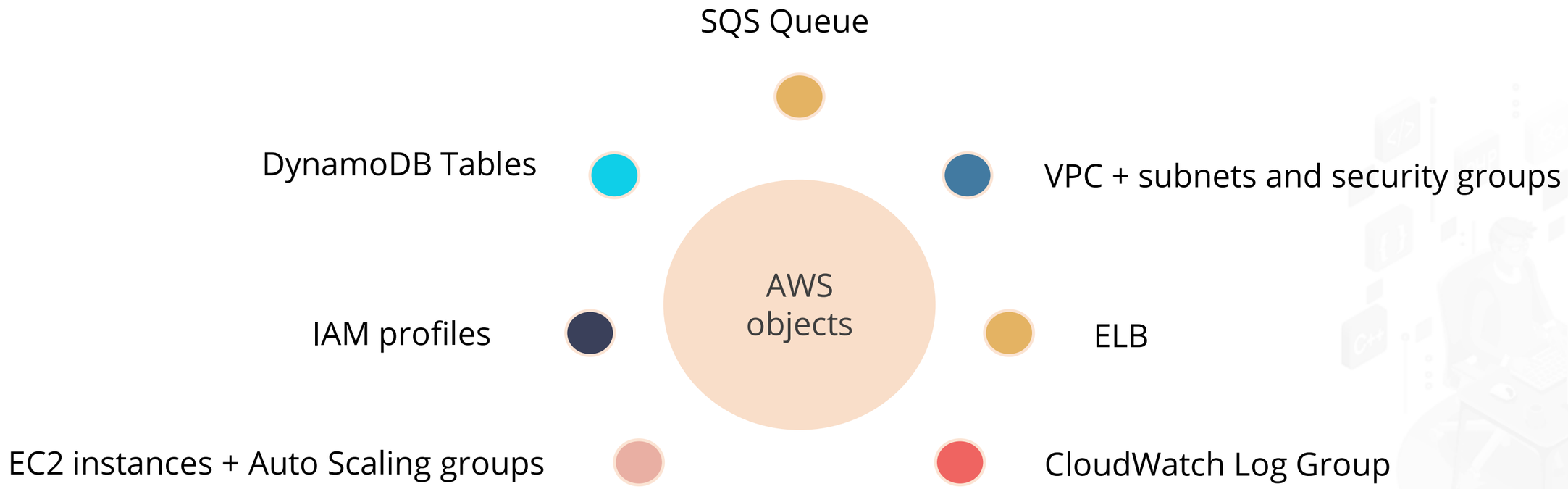
Deployment options:

- Docker for AWS can be deployed with a pre-existing VPC
- Docker for AWS can be deployed with a new VPC created by Docker

Prerequisites



Prerequisites



Mediums and Configurations

Configurations:

Docker for AWS is deployed by using two types of mediums:

- AWS Management Console
- AWS CLI

During installation the mediums must have the following configuration options:	
KeyName	InstanceType
ManagerInstanceType	ClusterSize
EnableSystemPrune	ManagerSize
EnableCloudWatchLogs	WorkerDiskSize
ManagerDiskSize	ManagerDiskType
WorkerDiskType	

Docker for AWS

Order in which the prerequisites are to be installed:

1. Install Docker EE
2. Install UCP
3. Install DTR

Docker Enterprise Edition for AWS

Adding AWS EC2 user:

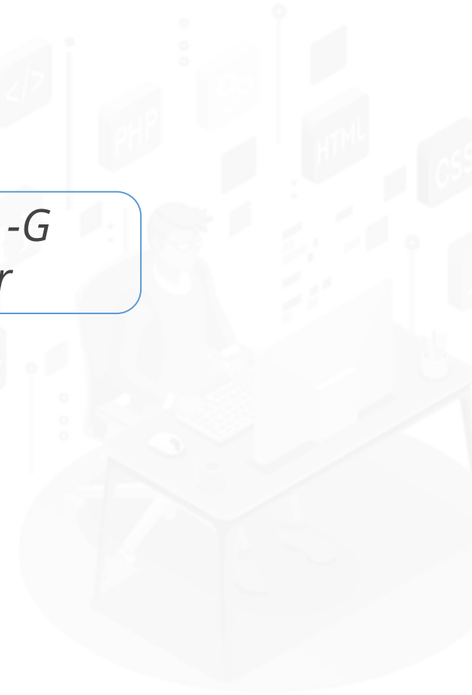
Start a Docker service 1 → *sudo service docker start*

Docker Enterprise Edition for AWS

Adding AWS EC2 user:

Start a Docker service 1

Add the ec2-user to the docker group 2 → *sudo usermod -a -G docker ec2-user*



Docker Enterprise Edition for AWS

Adding AWS EC2 user:

1. Start a Docker service
2. Add the ec2-user to the docker group
3. Log out and log back in again

Docker Enterprise Edition for AWS

Adding AWS EC2 user:

1. Start a Docker service
2. Add the ec2-user to the docker group
3. Log out and log back in again
4. Check that the ec2-user is running Docker commands without using *sudo*

→ *docker info*

Docker Enterprise Edition for AWS

Create a Docker Image:

Create a Dockerfile 1 → *touch Dockerfile*

Docker Enterprise Edition for AWS

Create a Docker Image:

Create a Dockerfile

1

Edit the Dockerfile by adding the following content

2

```
FROM ubuntu:18.04
# Install dependencies
RUN apt-get update && \
 apt-get -y install apache2
# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html
# Configure apache
RUN echo '. /etc/apache2/envvars' > /root/run_apache.sh && \
 echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh && \
 echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh
EXPOSE 80
CMD /root/run_apache.sh
```

Docker Enterprise Edition for AWS

Create a Docker Image:

1. Create a Dockerfile
2. Edit the Dockerfile by adding the following content
3. Build the Docker image from your Dockerfile

```
docker build -t hello-world .
```

Docker Enterprise Edition for AWS

Create a Docker Image:

1. Create a Dockerfile
2. Edit the Dockerfile by adding the following content
3. Build the Docker image from your Dockerfile
4. Run ***docker images*** for verification

docker images --filter reference=hello-world

Docker Enterprise Edition for AWS

Create a Docker Image:

1. Create a Dockerfile
2. Edit the Dockerfile by adding the following content
3. Build the Docker image from your Dockerfile
4. Run ***docker images*** for verification
5. Run the newly built image

docker run -t -i -p 80:80 hello-world

Docker Enterprise Edition for AWS

Create a Docker Image:

1. Create a Dockerfile
2. Edit the Dockerfile by adding the following content
3. Build the Docker image from your Dockerfile
4. Run ***docker images*** for verification
5. Run the newly built image
6. Point to the server that is running Docker

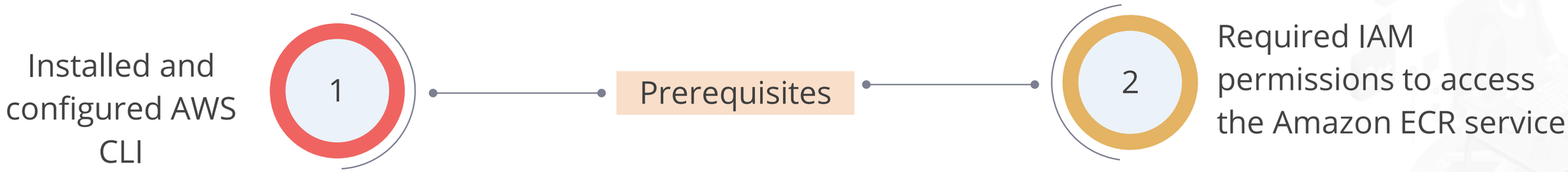
Docker Enterprise Edition for AWS

Create a Docker Image:

1. Create a Dockerfile
2. Edit the Dockerfile by adding the following content
3. Build the Docker image from your Dockerfile
4. Run ***docker images*** for verification
5. Run the newly built image
6. Point to the server that is running Docker
7. Stop the Docker container → Type *Ctrl + c*

Docker Enterprise Edition for AWS

Push an image to Amazon Elastic Container registry:

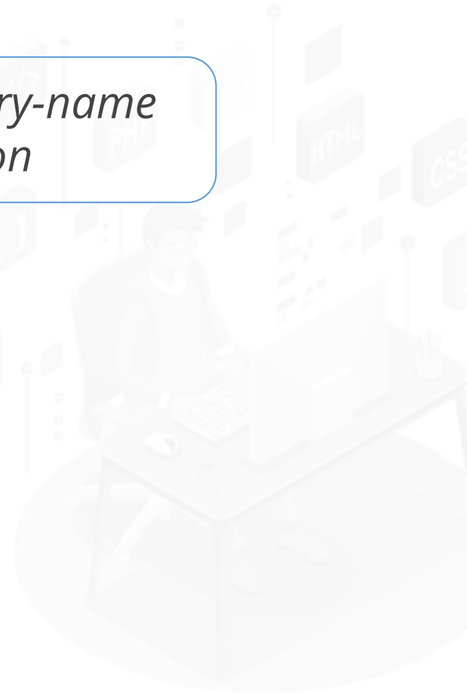


Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

Create an Amazon ECR repository 1

aws ecr create-repository --repository-name hello-repository --region region



Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

1. Create an Amazon ECR repository
2. Tag the “hello-world” image

```
docker tag hello-world
aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

1. Create an Amazon ECR repository
2. Tag the “hello-world” image
3. Get the **docker login** authentication command string

aws ecr get-login --no-include-email --region region

Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

1. Create an Amazon ECR repository
2. Tag the “hello-world” image
3. Get the **docker login** authentication command string
4. Run the **docker login** command

Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

1. Create an Amazon ECR repository
2. Tag the “hello-world” image
3. Get the **docker login** authentication command string
4. Run the **docker login** command
5. Push the image to Amazon ECR with the *repositoryUri* value

docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository

Docker Enterprise Edition for AWS

Push a “hello-world” image to Amazon Elastic Container registry:

1. Create an Amazon ECR repository
2. Tag the “hello-world” image
3. Get the **docker login** authentication command string
4. Run the **docker login** command
5. Push the image to Amazon ECR with the *repositoryUri* value
6. Delete the repository

aws ecr delete-repository --repository-name hello-repository --region region --force

Key Takeaways

- Docker EE possesses capabilities, such as image management, container app management, and image security scanning.
- UCP-agent is a globally scheduled service that starts running after the deployment of UCP. UCP provides the capability of joining multiple manager nodes to the cluster to counter the failure of the manager node.
- Docker Trusted Registry (DTR), provides image storing ability to Docker Enterprise. On increased usage, the replicas are added to make the DTR scale for high availability.
- Caching the images closer to the user helps in reducing the bandwidth required for pulling Docker images.

Using UCP to Add Subjects and Use Shared Resources

Problem Statement: Your team lead has asked you to create an organization and a team on Docker UCP and then add your team members to it. You must create a custom role with write access and group the swarm cluster resource by creating a collection. You are also required to deploy a service and create a grant for it.

Steps to Perform:

1. Create an organization, a team, and three users.
2. Create a custom role with complete access.
3. Create a collection to group the swarm cluster resources.
4. Create a service and deploy it with a container.
5. Create a grant to verify user permissions for the deployed service.

LESSON-END PROJECT