DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Installation and Configuration of Mirantis Products

Learning Objectives

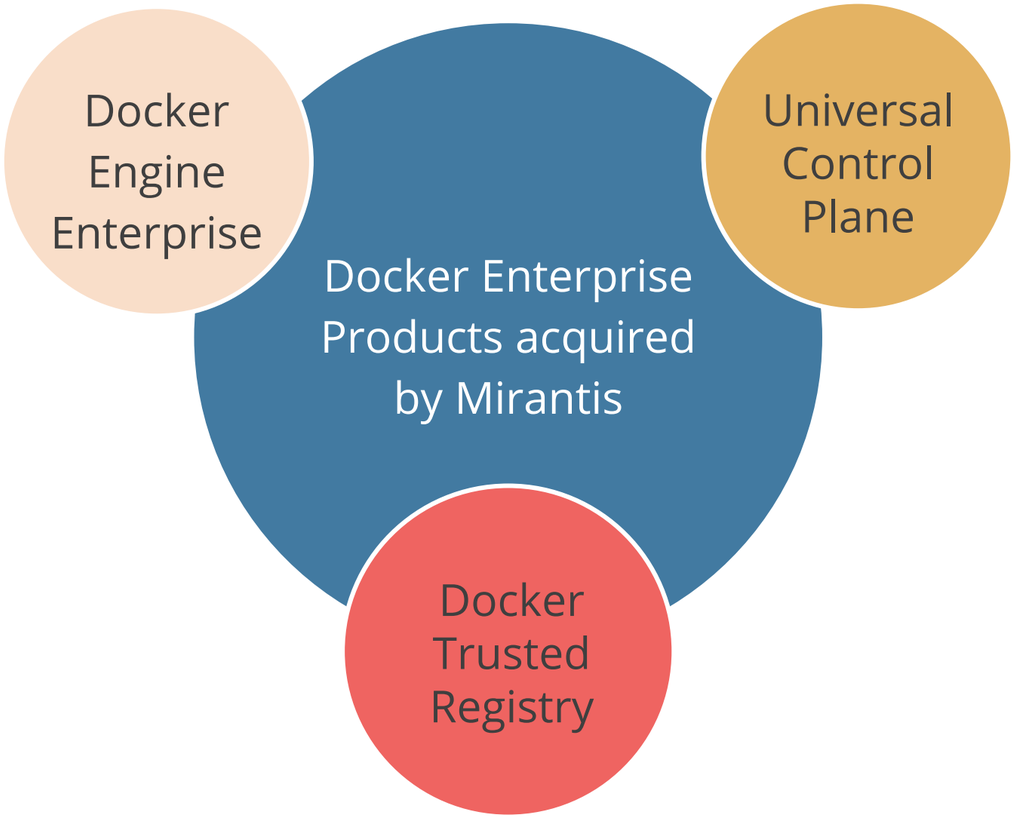
By the end of this lesson, you will be able to:

- Describe different products from Mirantis
- Install the MKE, MSR, and comprehend their architecture
- Create grants and understand how access control works
- Comprehend high availability and load balancing in MKE and MSR

Mirantis Products: Overview

Mirantis Products: Overview

Mirantis provides products that are acquired from Docker Enterprise platform to create a continuously-delivered container management platform.



Mirantis Products: Overview

Mirantis has realigned its portfolio and renamed several products. These include:

- Docker Enterprise Container Cloud (now Mirantis Container Cloud)
- Docker Universal Control Plane (now Mirantis Kubernetes Engine)
- Docker Engine- Enterprise (now Mirantis Container Runtime)
- Docker Trusted Registry (now Mirantis Secure Registry)

Mirantis Products: Overview

Mirantis Kubernetes Engine (MKE)

Mirantis Container Runtime (MCR)

Mirantis Secure Registry (MSR)

Mirantis Container Cloud (MCC)

Mirantis Products: Overview

Introduction to Mirantis Products:

- Mirantis Kubernetes Engine (MKE), Mirantis Secure Registry (MSR), and Mirantis Container Runtime (MCR) provide a standardized container platform for development and delivery of modern applications.

- They are designed for application developers and IT teams who build, share, and run business-critical applications at large scale in production.

- They offer a consistent and secure end-to-end application pipeline, choice of tools and languages, and globally consistent Kubernetes environments that run in any cloud.

Mirantis Products: Overview

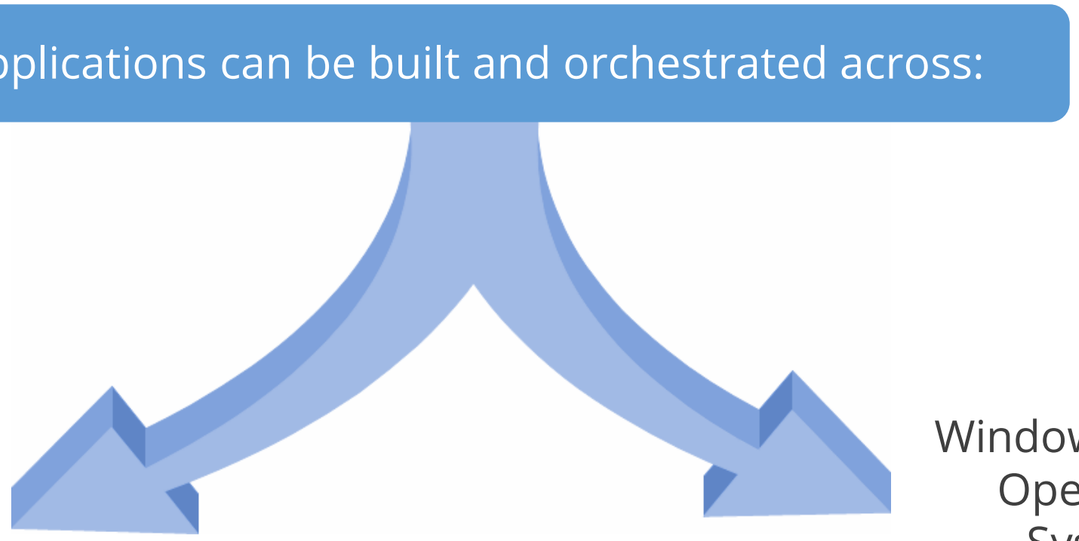
Container engine and built-in networking, orchestration, and security

Certified infrastructure, plugins, and ISV containers

Image and Container app management

Image signing and security scanning

Mirantis Products: Overview



Mirantis Container Runtime

Introduction to Mirantis Container Runtime

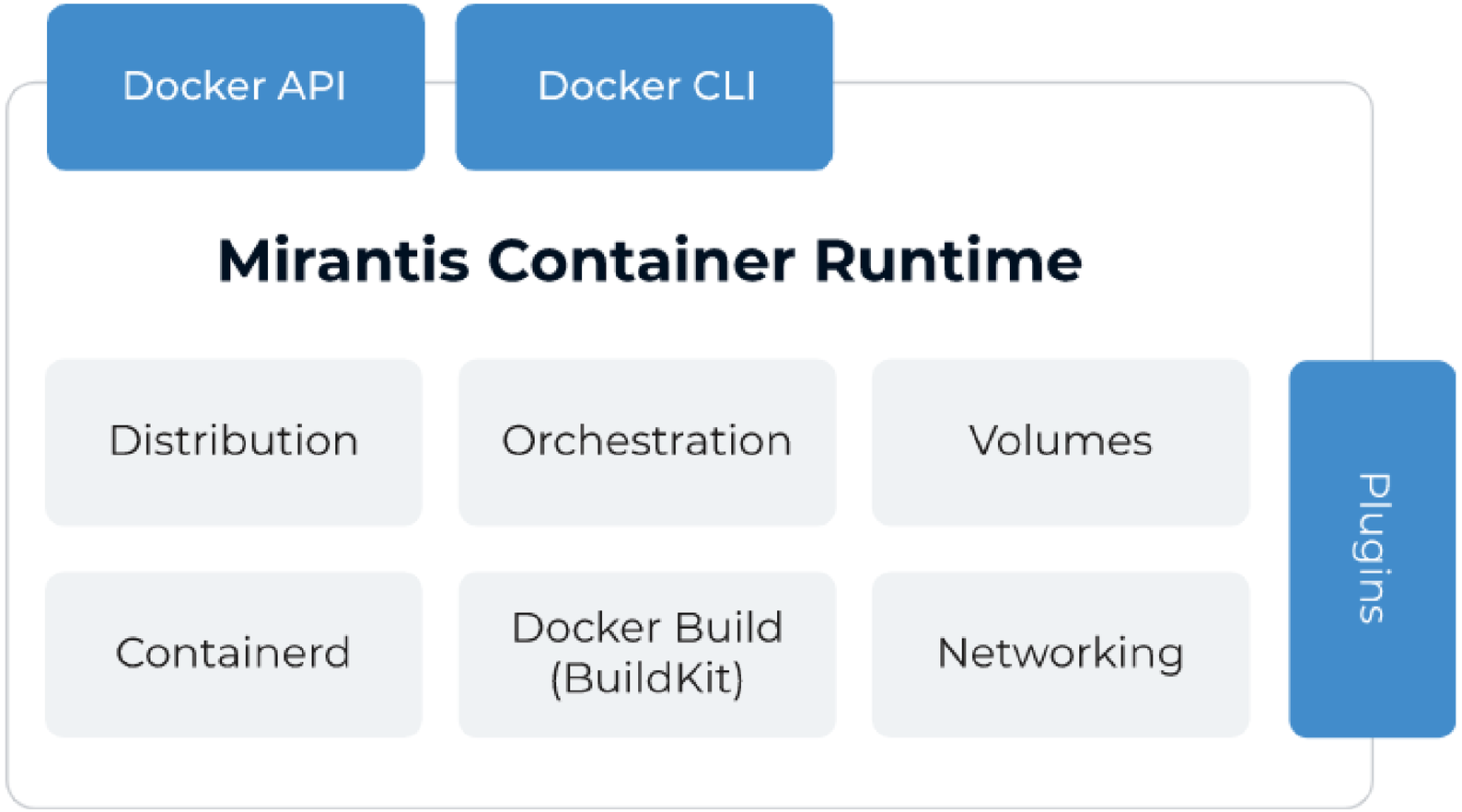
MCR Overview:

Mirantis Container Runtime (MCR), formerly known as Docker EE, is a client-server application with following major components:

- A server that is a type of long-running program called a daemon process
- A REST API that specifies interfaces which programs can use to communicate to the daemon and instruct it to perform specific tasks
- A Command Line Interface (CLI) client

Note: MCR can be installed on several linux distros as well as on Windows.

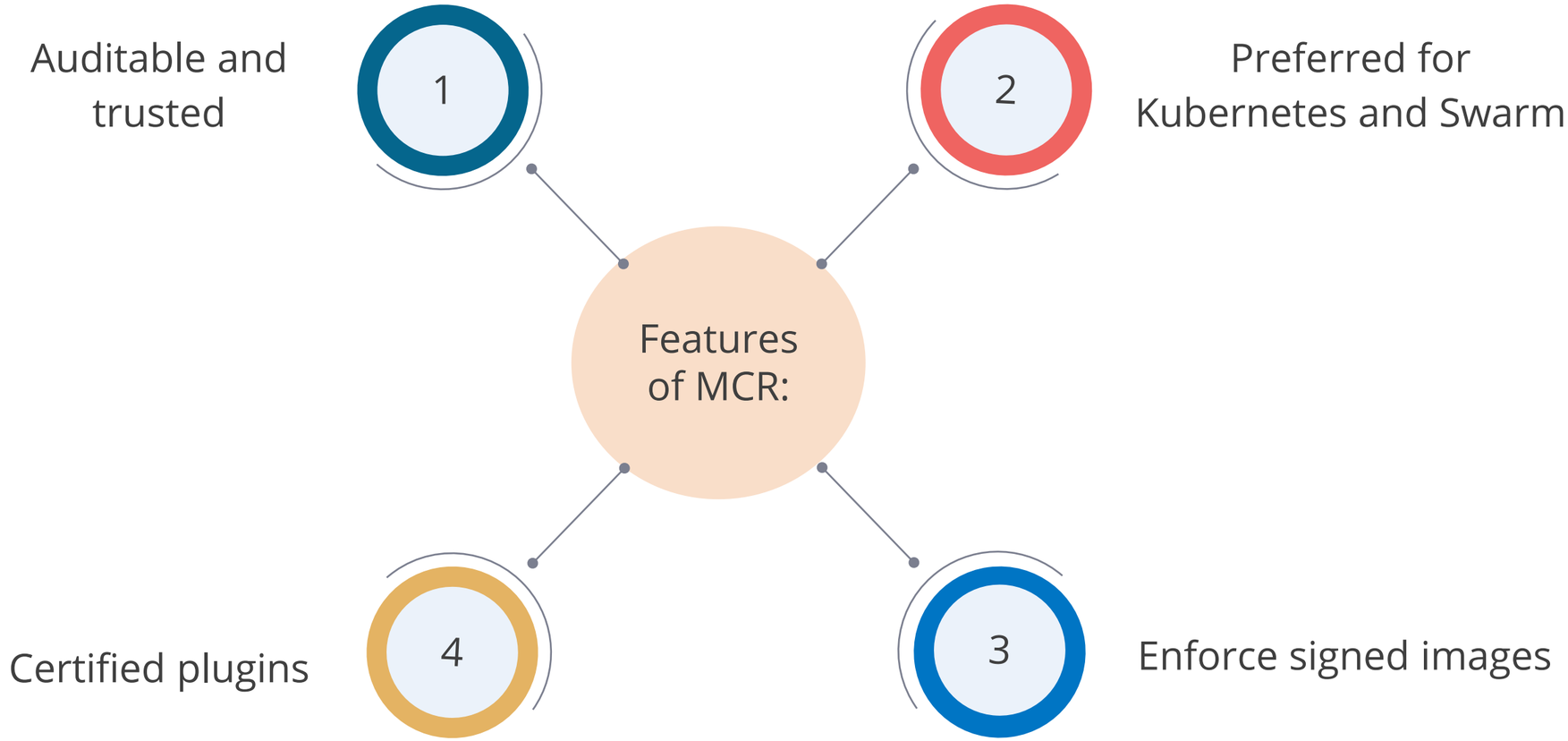
Features of MSR



Features of MCR

Source: https://www.mirantis.com/software/docker/container-runtime/

Features of MSR



MCR Installation

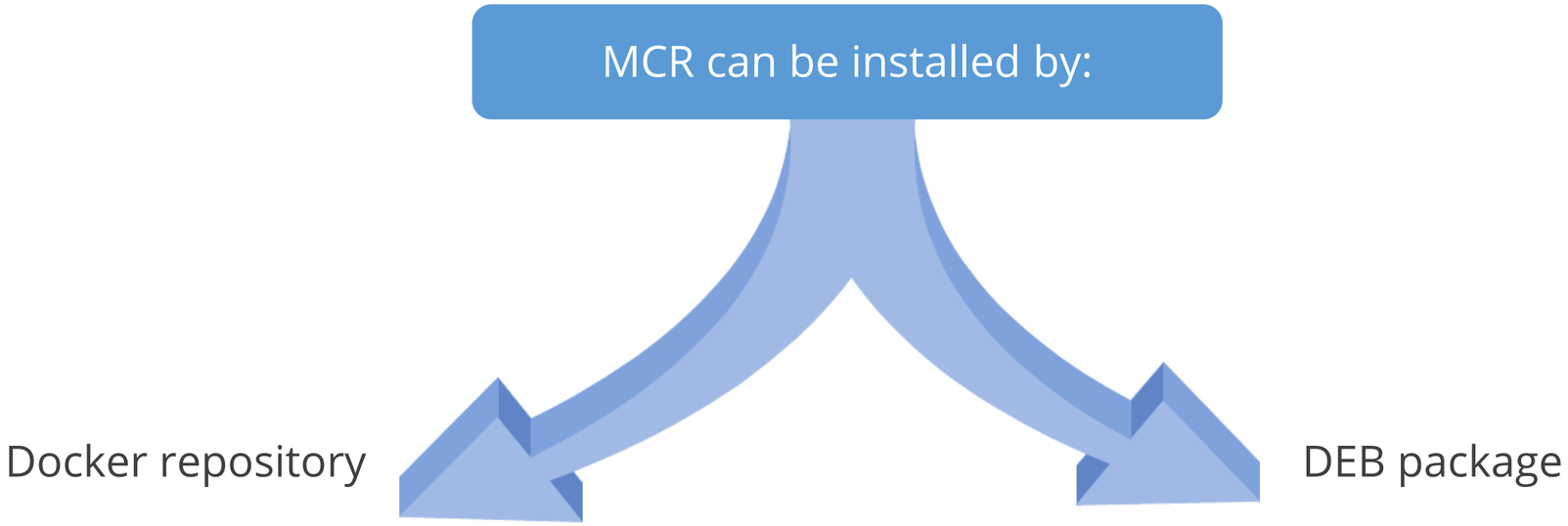
Installation Requirements

Prerequisites:

- If there is an existing version of MCR, uninstall it using the **apt-get remove** command

  sudo apt-get remove docker docker-engine docker-ce docker-ce-cli docker.io
- For Ubuntu 16.04 and higher, MCR uses *overlay2* as the default storage driver. Manually configure the default storage driver to use *aufs* instead of *overlay2*

Installation Methods



Installation Methods

Install using the Docker repository:

1. Navigate to *repos.mirantis.com* and obtain the URL for the static repository that contains the MCR software for the desired Ubuntu version and refer to it as <*MCR-Ubuntu-URL*>

1. Use the *apt-get remove* command to uninstall older versions of MCR (or docker EE)

1. Set up the Docker repository with <*DOCKER_EE_URL*> and <*DOCKER_EE_VERSION*>

1. Install MCR using the *apt-get install* command

1. Verify that MCR is installed correctly by running the *hello-world* image

Installation Methods

Install using the DEB package:

1. Open repos.mirantis.com in the web browser

1. Navigate to */ubuntu/dists/bionic/pool/stable-<VERSION>/amd64/* and download the *.deb* file

1. Install MCR by changing the path in the following command to the path where the MCR package was downloaded:

   sudo dpkg -i <path_to_downloaded_ubuntu_package_.deb>

1. Verify that the MCR is installed correctly by running the following command:

   sudo docker run hello-world

Note: Starting with **19.03**, you have to download three *.deb* files i.e. *docker-ee-cli_<version>.deb, containerd.io_<version>.deb,* and *docker-ee_<version>.deb*

Uninstall MCR

Steps to uninstall MCR:

1. Uninstall the Mirantis Container Runtime package

   sudo apt-get purge docker-ee

1. Run the following command to delete all images, containers, and volumes:

   $ sudo rm -rf /var/lib/docker

Mirantis Kubernetes Engine

Introduction to Mirantis Kubernetes Engine

Mirantis Kubernetes Engine (MKE) is the enterprise-grade cluster management solution from Docker. It can be installed on-premises or in a VPC, and it helps you in managing Docker cluster and applications through a single interface.

Introduction to Mirantis Kubernetes Engine

MKE Dashboard

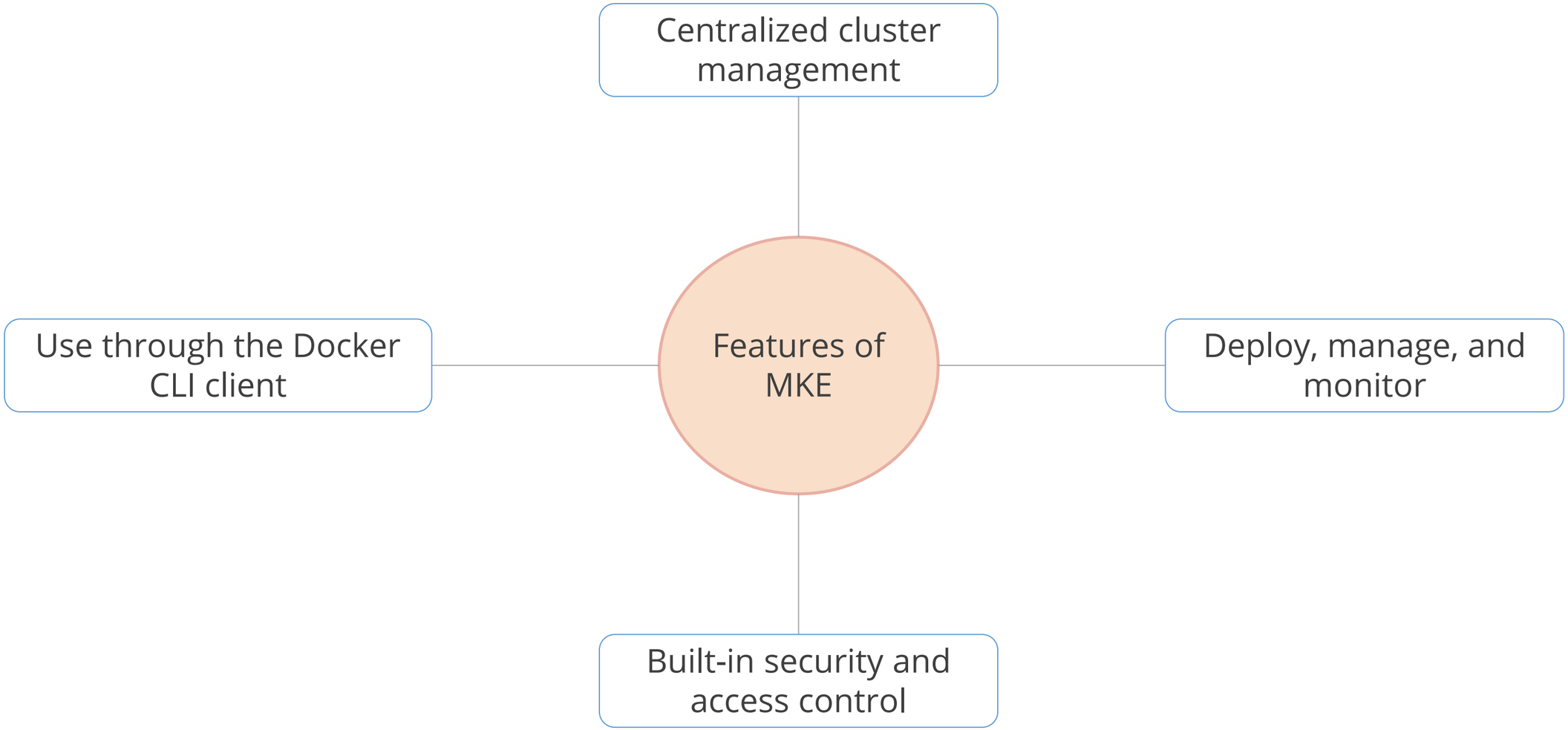
Source: https://docs.mirantis.com

Features of MKE

Features of MKE:

- Deploy highly available workloads using Docker Kubernetes Service or Docker Swarm
- Deploy applications at scale and manage clusters from a centralized place
- Automate tasks such as provisioning pods, containers, and cluster resources that are required by orchestration
- Self-healing components ensure highly available MKE clusters

Features of MKE



Features of MKE

Centralized cluster management:

- Enables you to create a cluster by joining thousands of physical or virtual machines together
- Allows you to deploy applications at a very large scale
- Makes it easier to manage clusters from a centralized place

Features of MKE

Deploy, manage, and monitor:

- Manage and monitor the clusters using a graphical UI
- Manage all the available computing resources such as nodes, volumes, and networks from a centralized place
- Deploy and monitor the applications and services

Features of MKE

Built-in security and access control:

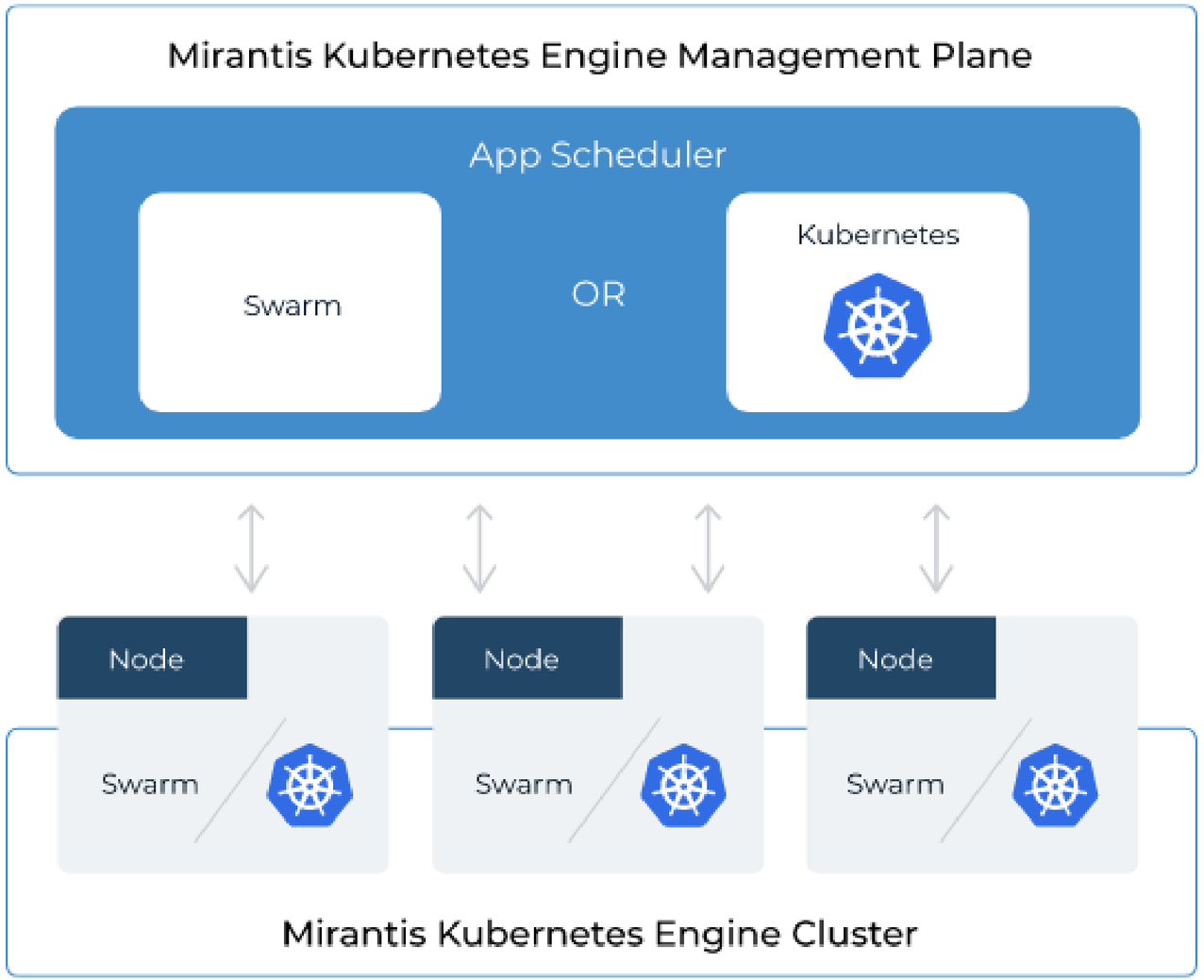
- Built-in authentication mechanism that integrates with LDAP services
- Control who can access and modify a cluster and its applications, using Role-based access control (RBAC)
- Keep the Docker images safe behind a firewall using Mirantis Secure Registry (MSR)
- Enforce security policies and only allow running applications that use trusted Docker images

Features of MKE

Use through the Docker CLI client:

- As MKE exposes the standard Docker API, you can continue using the Docker CLI client to deploy and manage your applications.
- You can use the ***docker info*** command to check the status of a cluster that is managed by MKE.

Orchestration



MKE Orchestration

Source: https://docs.mirantis.com

Orchestration

MKE Orchestration:

MKE allows to run Swarm and Kubernetes interchangeably in the same cluster:

- Applications deployed by either orchestrator can be managed through the same control plane, letting you scale more efficiently.
- Developers can choose how they want to deploy applications at runtime.
- Teams have the freedom to change orchestrators according to varying 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

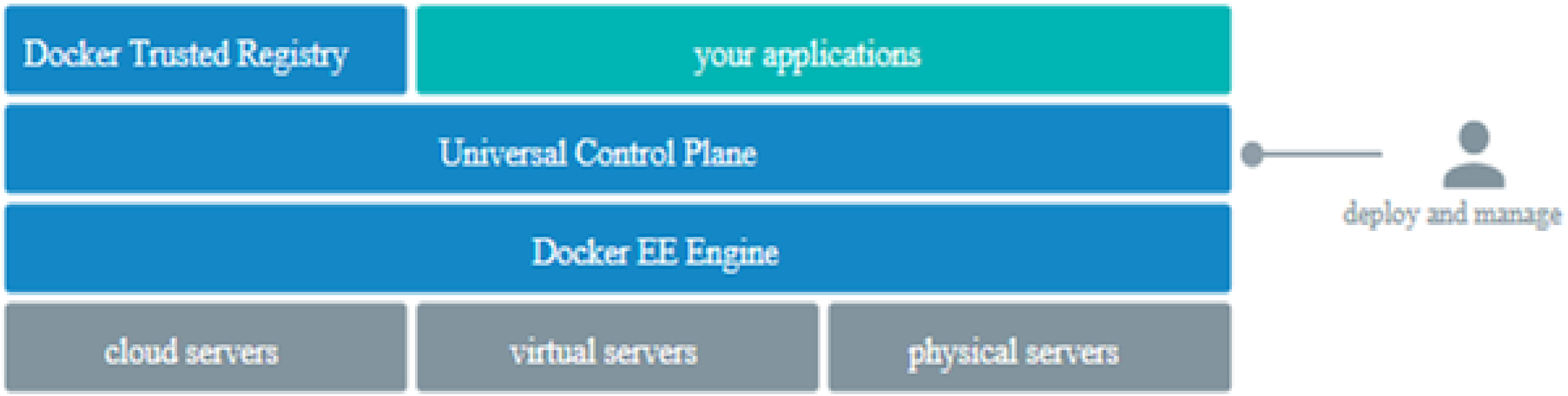
Orchestration platform features:

- Enabling high availability using MKE 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

MKE Architecture

MKE: Architecture

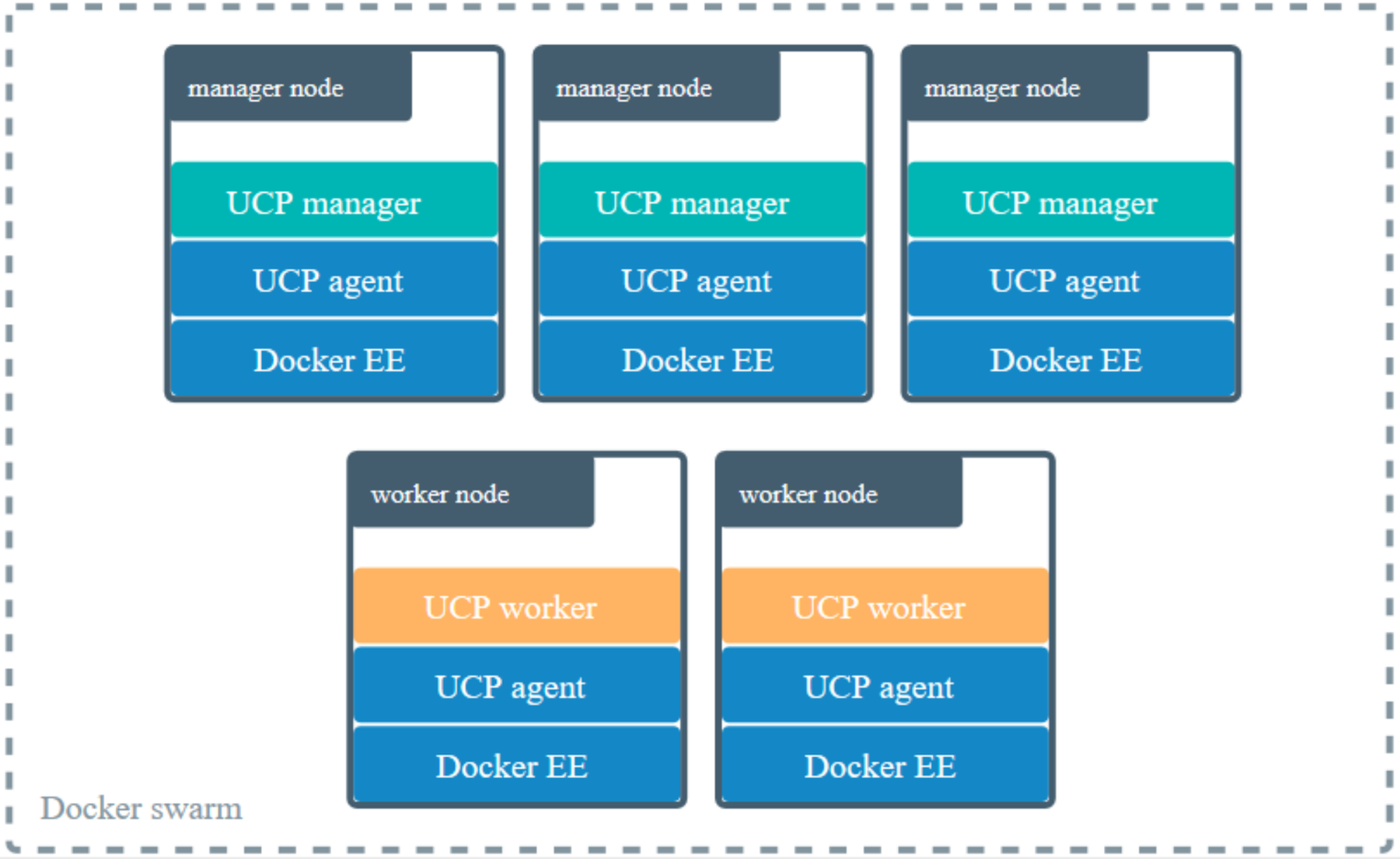
Once the MKE instance is deployed, developers and IT operations no longer interact with Mirantis Container Runtime directly, but interact with MKE instead.



Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

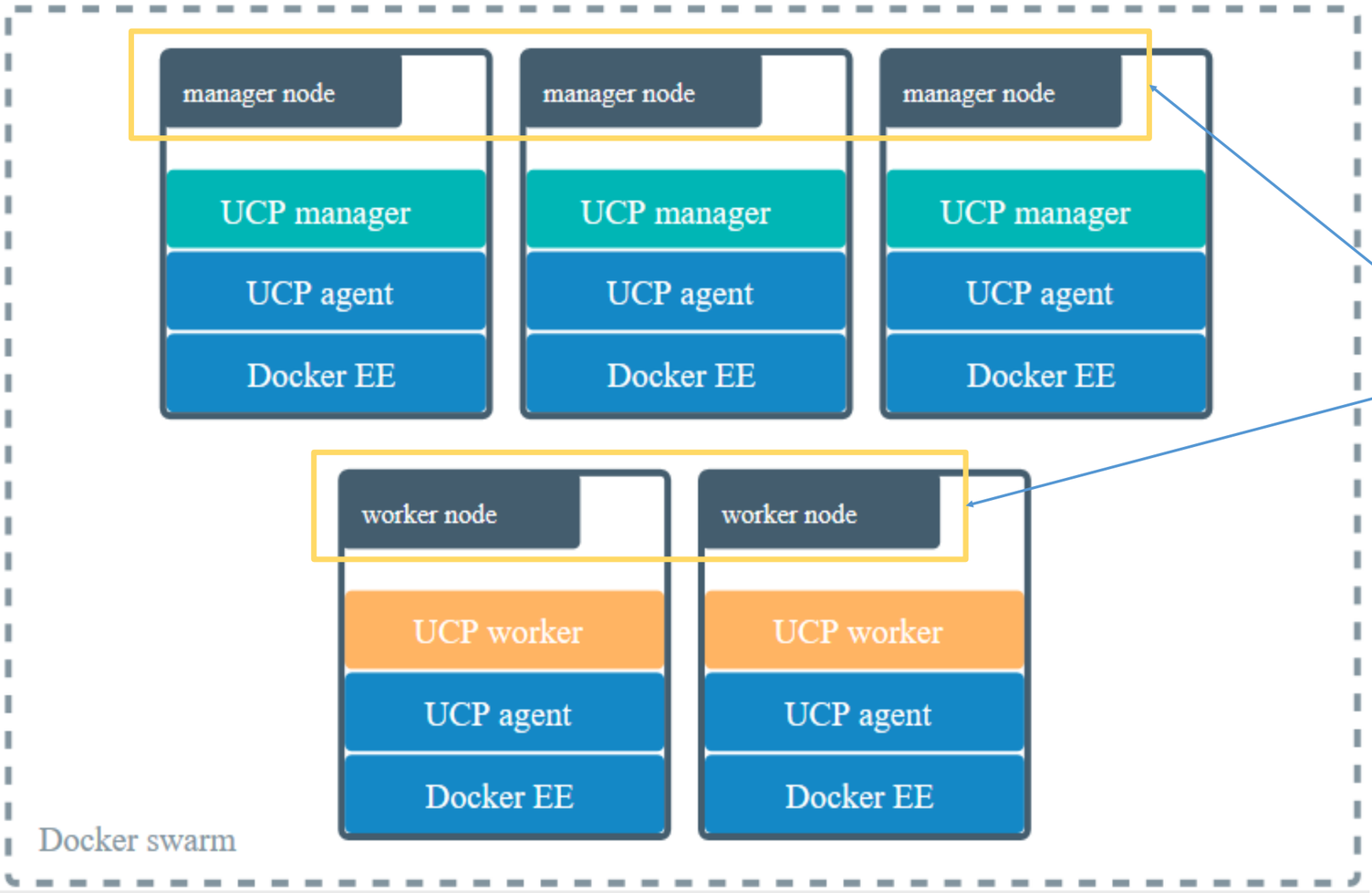
MKE: Architecture

MKE leverages the clustering and orchestration functionality provided by Docker.



Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

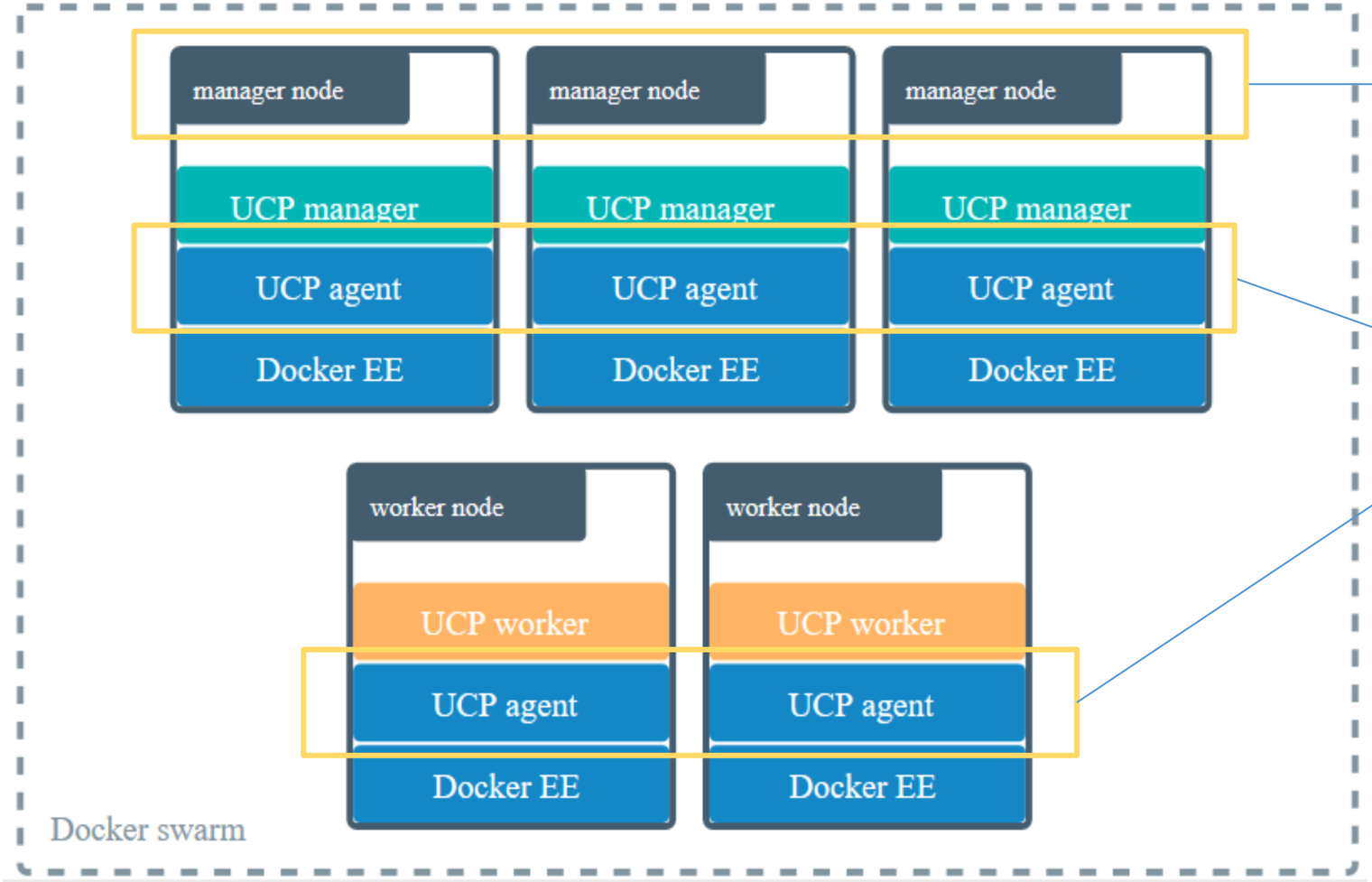
MKE: Architecture



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

Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

MKE: Architecture

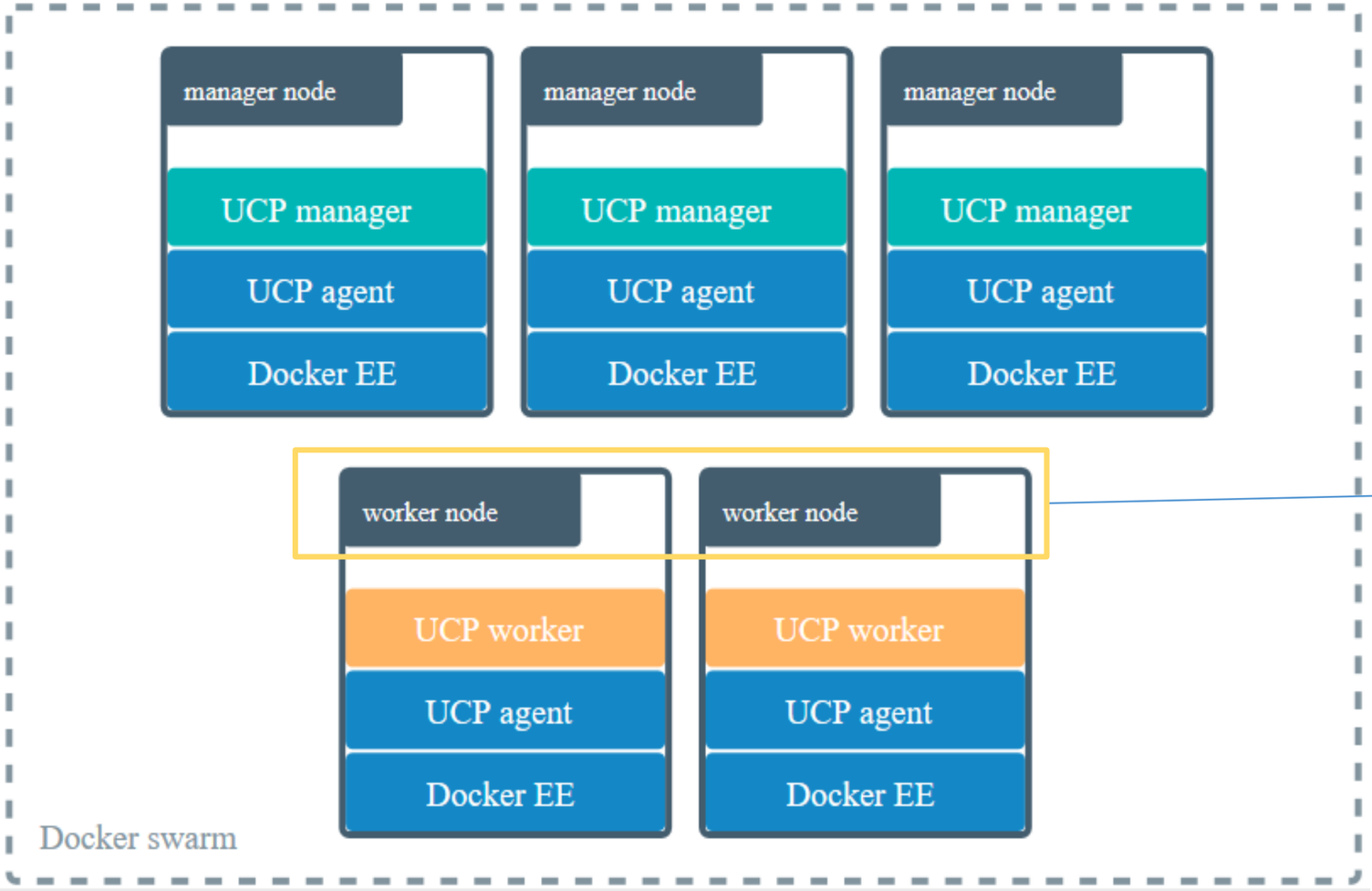


On manager nodes, the *ucp-agent* serves all the MKE components.

It is a globally scheduled service.

Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

MKE: Architecture

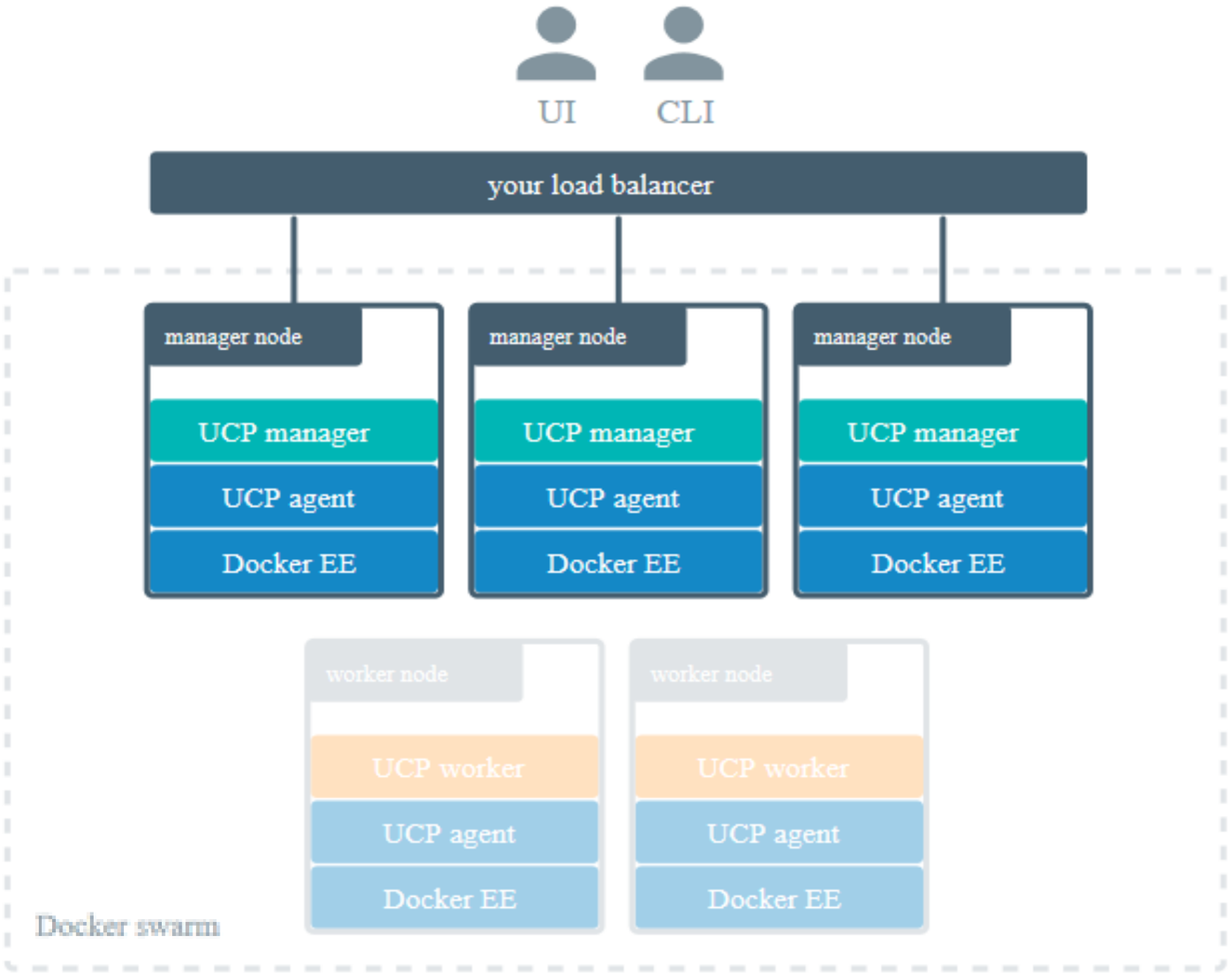


On worker nodes, the *ucp-agent* starts serving a proxy service.

Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

Interaction with MKE

Ways to interact with MKE: MKE Web UI or Docker CLI



Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/architecture.html

MKE Installation

Installation Requirements

Minimum requirements:

- 8GB of RAM is required for manager nodes
- 4GB of RAM is required for worker nodes
- 2 vCPUs are required for manager nodes
- 10GB of free disk space is required for the /var partition for manager nodes
- 500MB of free disk space is required for the /var partition for worker nodes

Recommended requirements:

- 16GB of RAM required for manager nodes
- 4 vCPUs required for manager nodes
- 25-100GB of free disk space

Installation Requirements

Hardware and software requirements:

- All nodes must be running the same version of MCR (v19.03 or higher)
- Linux kernel version 3.10 or higher is required.
- A static IP address for each node in the cluster is required
- Currently MKE does not support user namespaces for nodes

Traffic:

Scope of the port: It's the incoming traffic from a set of hosts.

Types of scope:

- External: Traffic that arrives from outside of the cluster through the end-user interaction
- Internal: Traffic that arrives from other hosts in the same cluster
- Self: Traffic that arrives to the port only from the processes that are occuring on the same host

Installation Requirements

Following ports must be kept open for incoming traffic:				
TCP 179	TCP 443	TCP 2376	TCP 2377	UDP 4789
TCP 6443	TCP 6444	TCP, UDP 7946	TCP 9099	TCP 10250
TCP 12376	TCP 12378	TCP 12379	TCP 12380	TCP 12381
TCP 12382	TCP 12383	TCP 12384	TCP 12385	TCP 12386
TCP 12388				

Disabling CLOUD_NETCONGIF_MANAGE For SUSE Linux:

1. In the network interface configuration file at */etc/sysconfig/network/ ifcfg-eth0*, set the **CLOUD_NETCONFIG_MANAGE="no"**
2. Run *service network restart*

Installation Requirements

Enable IP-in-IP traffic: Ensure IP-in-IP traffic is enabled for your cloud provider security group, while deployment is being done to the AWS or another cloud provider

Time Synchronization: Ensure all the engines are regularly synchronizing the time with a Network Time Protocol (NTP) server

Enable ESP traffic: Ensure that IP protocol 50 traffic is allowed

Timeout settings: Allow the MKE components enough time to communicate before they timeout

Installation Requirements

Timeout setting:

Component	Timeout (ms)	Configurable
Raft consensus between manager nodes	3000	no
Gossip protocol for overlay networking	5000	no
etcd	500	yes
RethinkDB	10000	no
Stand-alone cluster	90000	no

Install MKE

Install MKE using the *mirantis/ucp* image:

1. Install MCR on all nodes
2. Use ssh to log in to the node where you want to install MKE
3. Run the following commands:

```
# Pull the latest version of MKE
docker image pull mirantis/ucp:3.3.2

# Install MKE
docker container run --rm -it --name ucp \
-v /var/run/docker.sock:/var/run/docker.sock \
mirantis/ucp:3.3.2 install --host-address <node-ip-address> --interactive
```

Uninstall MKE

Uninstall MKE:

- Log in to the manager node using *ssh* in order to uninstall UCP. Run the following code after getting access:

  *docker container run --rm -it *

  *-v /var/run/docker.sock:/var/run/docker.sock *

  --name ucp mirantis/ucp:3.3.2 uninstall-ucp --interactive

Mirantis Launchpad

Mirantis Launchpad: Overview

Mirantis Launchpad is a fast and easy-to-use command-line installer that helps in getting started with MCR and MKE on public clouds, private IAAS, or bare metals like Linux, Mac, or Windows machine.

- Is used for installing, deploying, and updating the Mirantis Kubernetes Engine
- 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:

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

Assisted Practice
Install Mirantis Launchpad CLI

Problem Statement: Your technical manager has asked you to set up the Mirantis Launchpad CLI so that MKE 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 License for Docker Enterprise and uploading it on MKE dashboard

Uninstall Launchpad

Uninstall Launchpad:

Reset launchpad to uninstall it	*$ launchpad reset*

Upgrade Launchpad

Upgrade Launchpad:

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

```
apiVersion: launchpad.mirantis.com/v1
kind: MKE
metadata:
  name: launchpad-mke
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
```

Joining Manager Nodes

Joining Manager Nodes

Manager nodes are joined when the MKE is supposed to be highly available. Docker swarm and MKE 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 MKE 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. Log in using ***ssh*** and run the join command that was copied. The node appears on the **Nodes** page in the MKE web UI once the join command completes.

MKE: High Availability

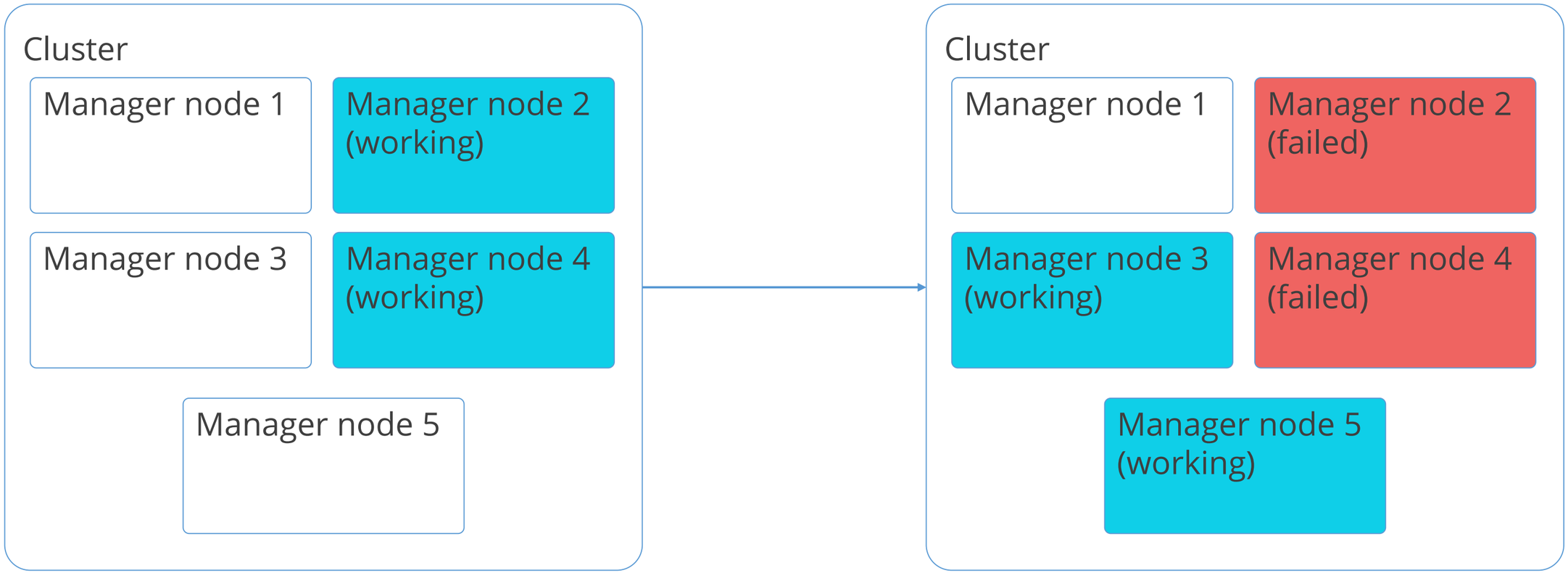
High Availability

High availability in MKE:

- MKE is designed for high availability (HA). Multiple manager nodes can be joined to a cluster, so that if one manager node fails, another can automatically replace it without impacting the cluster.

- Multiple manager nodes in a cluster help in:
  - Handling manager node failures
  - Load-balancing user requests across all manager nodes

High Availability



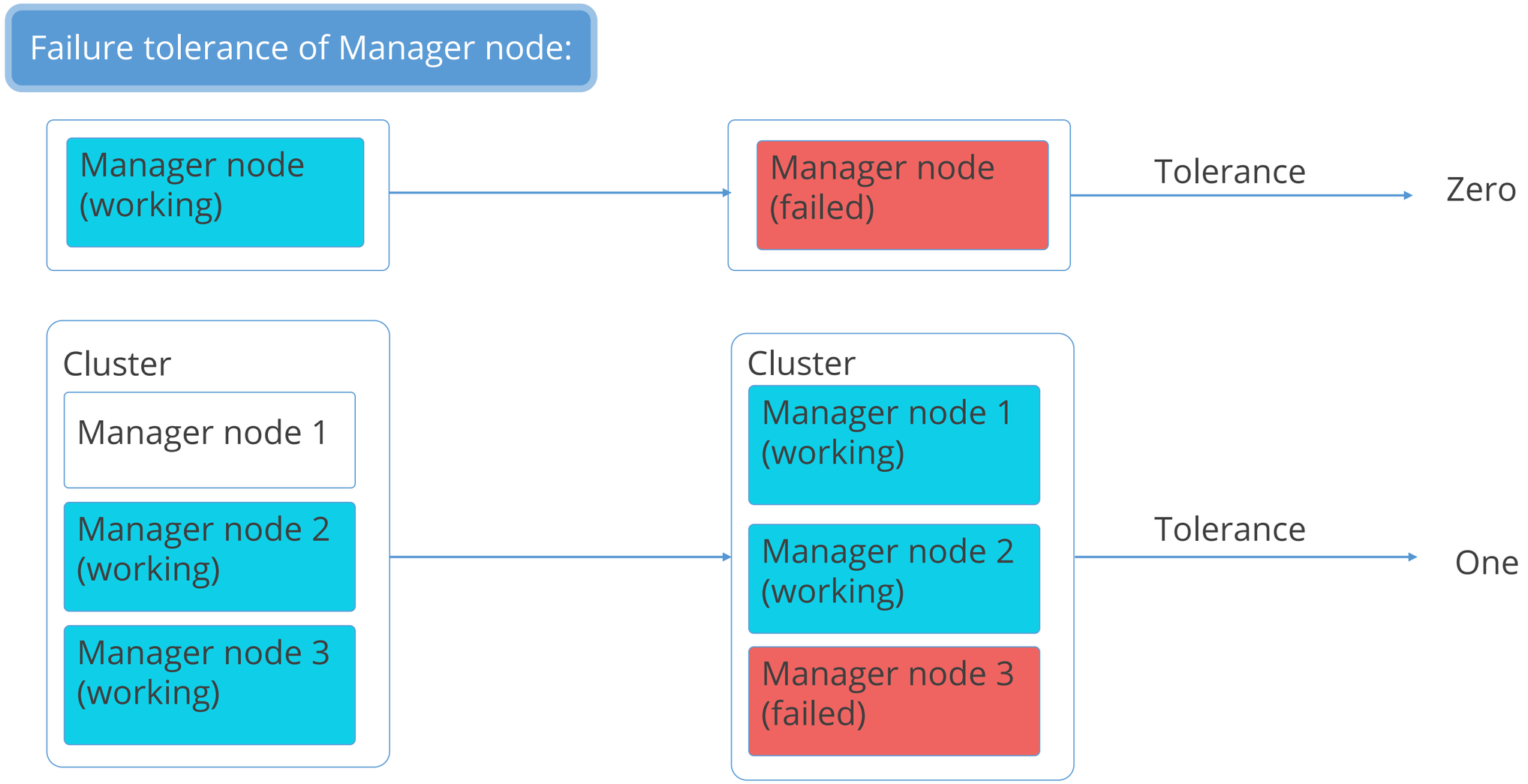
Explaining high availability through failed Manager Nodes

High Availability

Best Practices for fault tolerance:

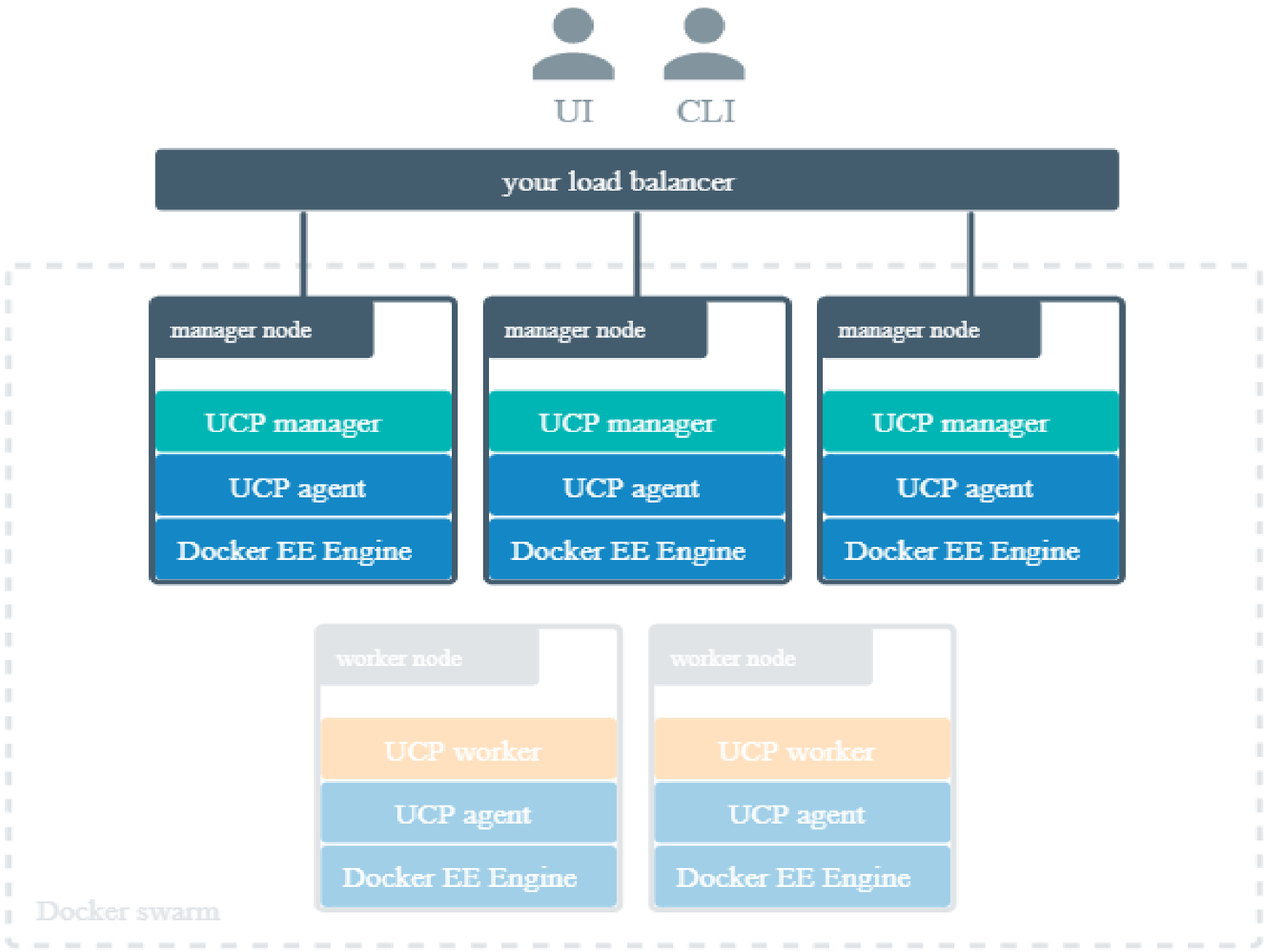
- For high availability of a cluster, the recommended number of manager nodes is between 3 to 5.
- The number of faults tolerated by a cluster decreases when a manager node fails.
- Manager nodes should be distributed across different availability zones, so that if an availability zone goes down the cluster can continue working.

High Availability



MKE: Load Balancer

Load Balancing on MKE

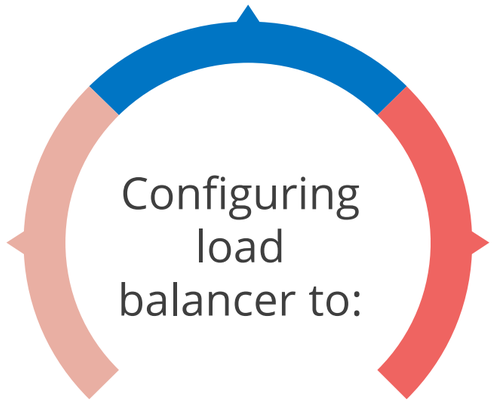


Source: https://docs.mirantis.com/docker-enterprise/v3.0/dockeree-products/ucp.html

Load Balancing on MKE

As MKE uses mutual TLS, load balancer should be configured to:

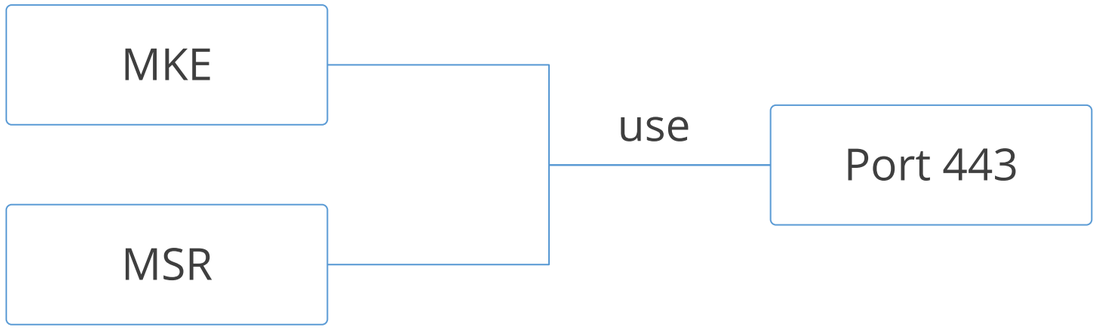
Stop termination of HTTPS connections



Load-balance the TCP traffic on ports 443 and 6443

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

Load Balancing MKE and MSR



Configuring load balancer for MKE and MSR:

- Load balancer can be configured to listen on port 443 by:
  - Using separate load balancer for MKE and MSR
  - Using same load balancer with different virtual IP addresses
- Load balancer can also be configured by exposing MKE or MSR on a port other than 443

Configuring Load Balancer

Example of configuring load balancer for MKE:

```
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 <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 using NGINX:

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
```

MKE: Swarm Service

Deploy Swarm Service Using MKE

Steps for deploying a service:

1. Log in to **MKE** web UI and click **Services** to open the **Create a Service** page
2. To configure an **NGINX** service click **Create Service**, and enter the following details:
   - Service name: **nginx**
   - Image name: **nginx:latest**
3. Click **Network** on the left pane
4. Click **Publish Port** in the **Ports** section and enter the following fields:
   - Target port: **80**
   - Protocol: **tcp**
   - Publish mode: **Ingress**
   - Published port: **8000**

Deploy Swarm Service Using MKE

5. Click **Confirm** to map the ports for the NGINX service
6. Specify the service image and ports, and click **Create** to deploy the service into the MKE cluster
7. Once the service is up and running, you can view the default NGINX page by going to **http://<node-ip>:8000**
8. In the **Services** list, click the **nginx** service, and in the details pane, click the link under **Published Endpoints**
9. Clicking the link opens a new tab that shows the default NGINX home page

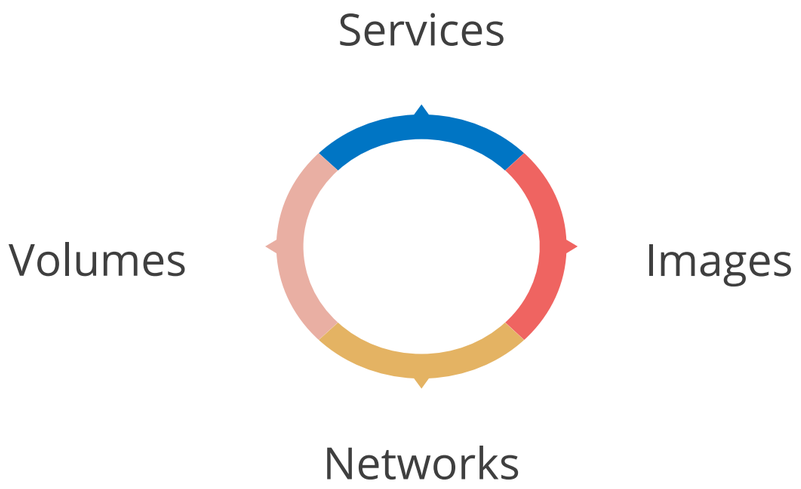
Note: There should be enough nodes to deploy a Swarm service.

MKE: Access Control

Access Control

MKE enables you to authorize users to view, edit, and use cluster resources by granting role-based permissions against resource sets.

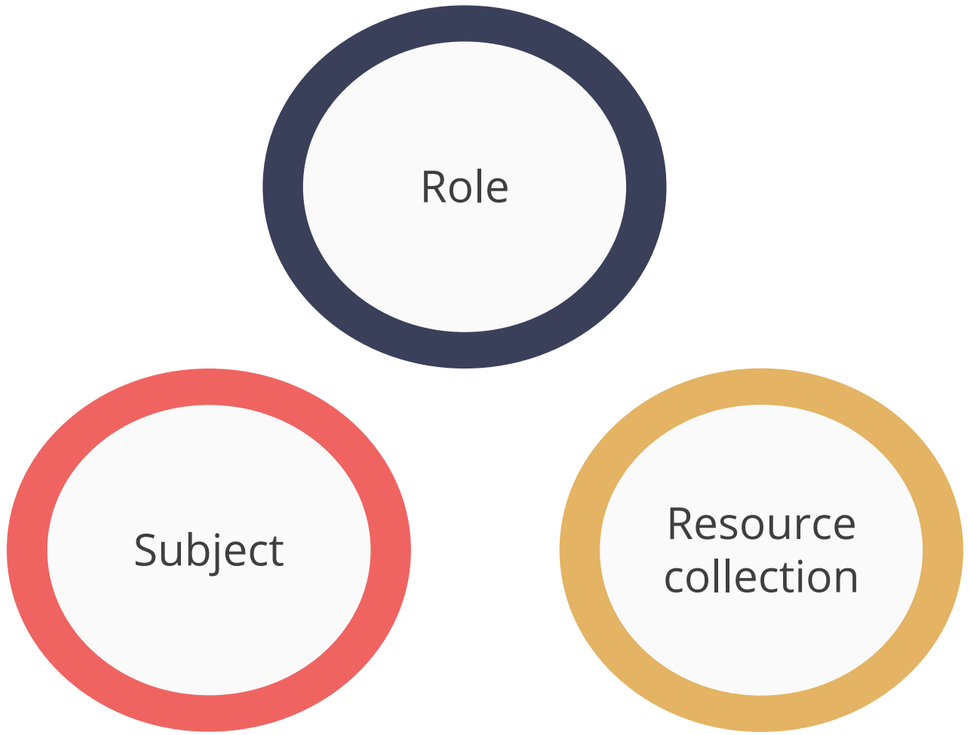
Organizations control who can create and edit the resources in a swarm such as:



Grant

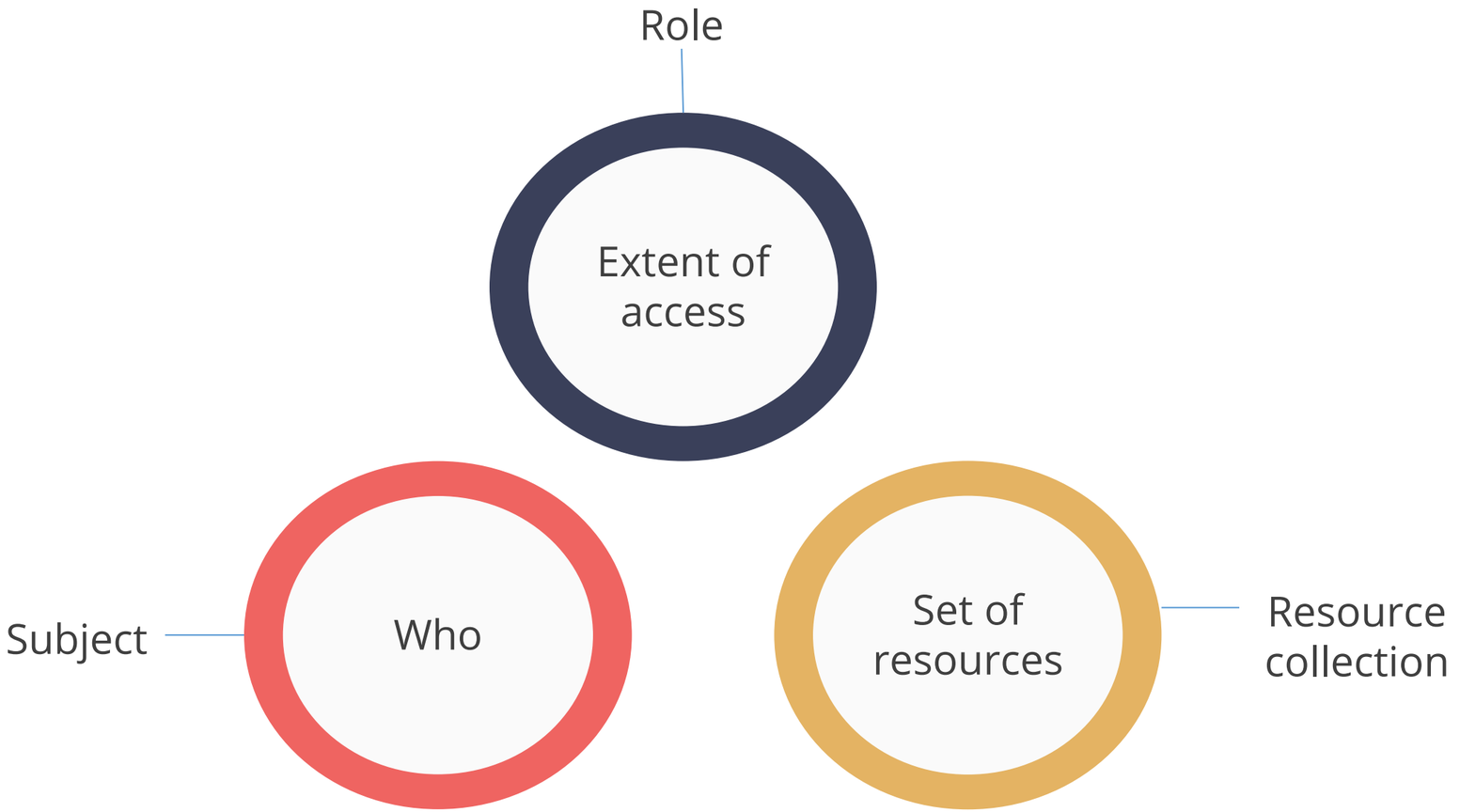
Users and organizations access swarm resources that can be controlled by creating *grants.*

A Grant contains:



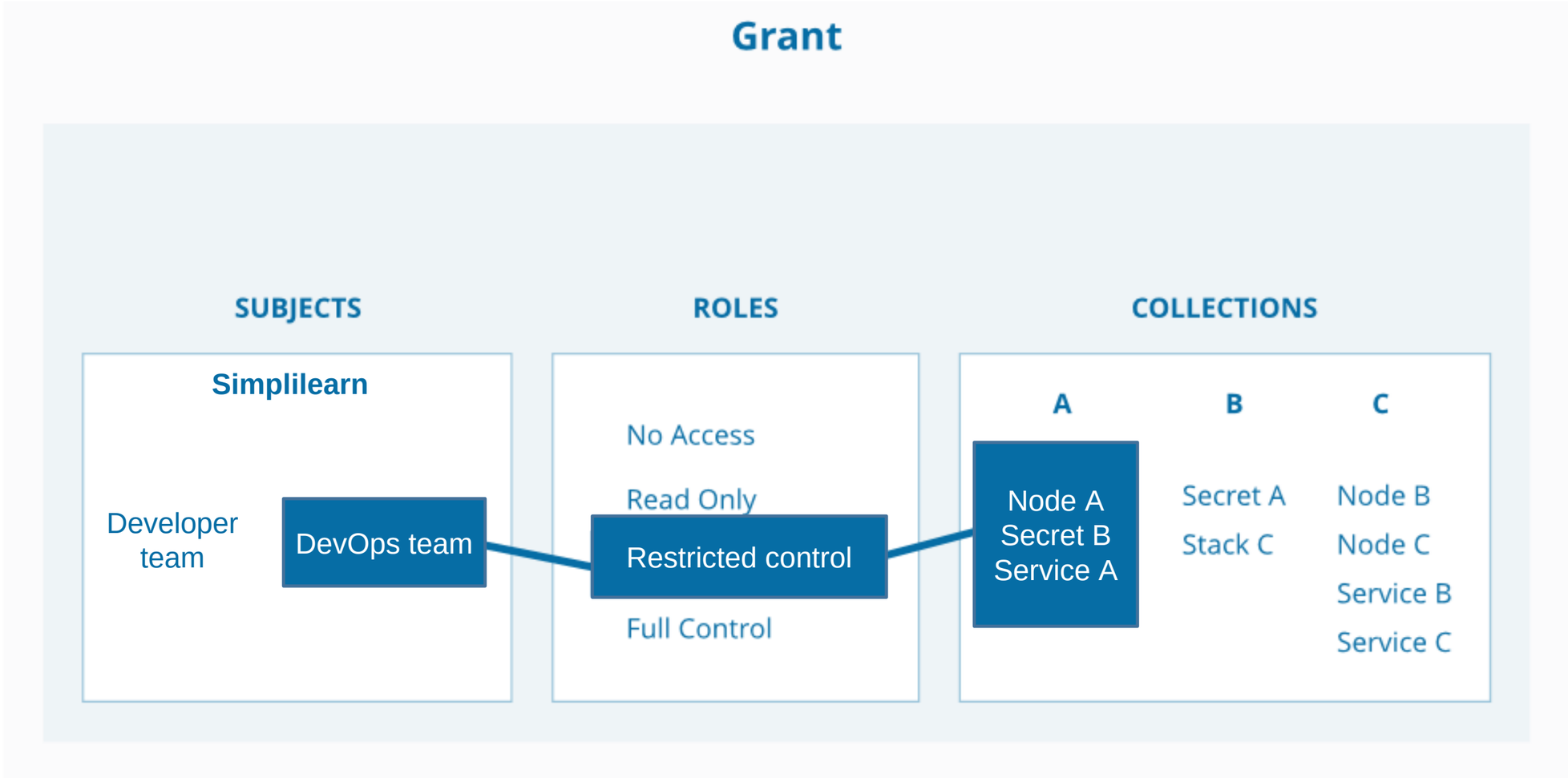
Grant

A Grant defines:



Grant

Example: The DevOps team at Simplilearn has restricted control for collection A

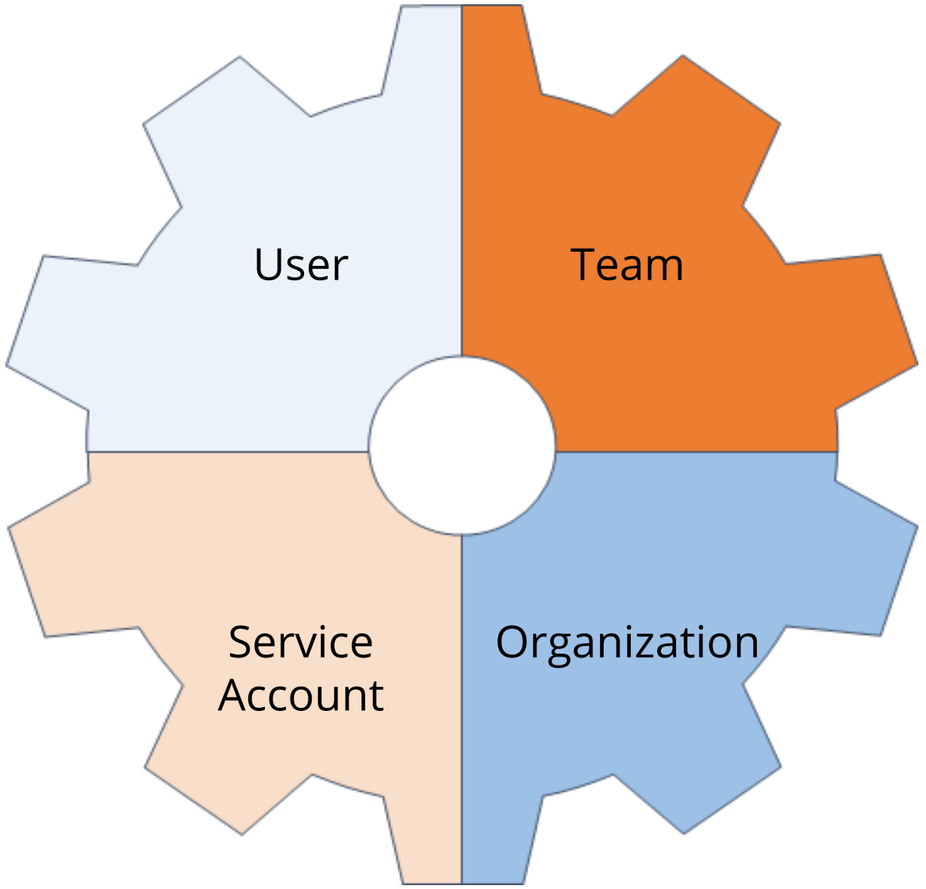


Grant: Subject

Subject

A subject represents a user, team, organization, or a service account. A subject can be granted a role that defines permitted operations against one or more resource sets.

Subject Types:



Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/authorize/access-control-model.html

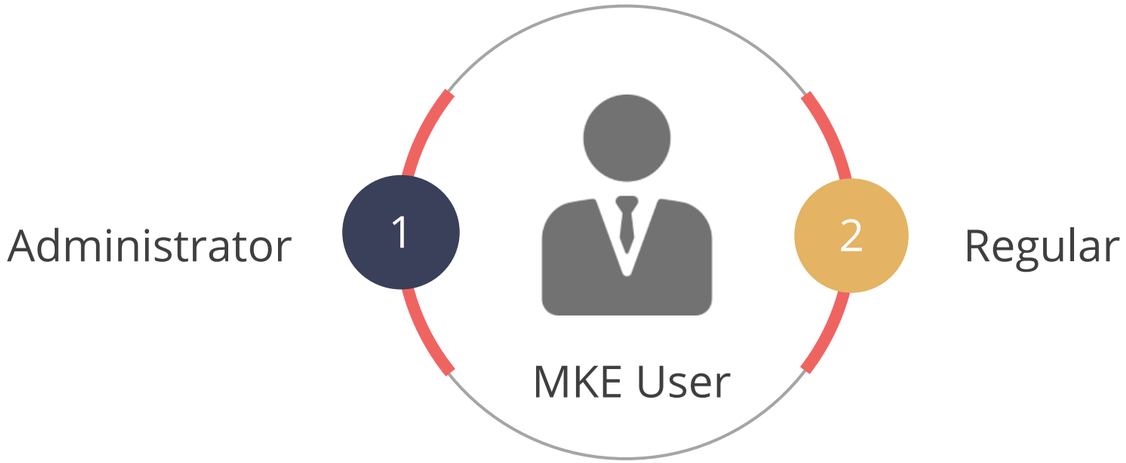
Subject

Subject Types:

- **User:** A person authenticated by the authentication backend. Users can belong to one or more teams and one or more organizations.
- **Team:** A group of users that share permissions defined at the team level. A team can be in one organization only.
- **Organization:** A group of teams that share a specific set of permissions, defined by the roles of the organization.
- **Service account:** A Kubernetes object that enables a workload to access cluster resources which are assigned to a namespace.

Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/authorize/access-control-model.html

Subject



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

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

Subject

Designing an organization in MKE:

1. Create an organization
2. Add users or enable LDAP (for syncing users)
3. Create teams under the organization
4. Add users to teams manually or sync with LDAP

Subject

Create an organization in MKE:

1. Go to the MKE web user interface

1. Click **Organization & Teams** under **User Management**

1. Click **Create Organization**

1. Enter the organization name

1. Click **Create** to create a new organization

Subject

Create a team in an organization:

1. Click on the **organization** name

1. Click on **Create Team**

1. Enter the team name. Description is *optional*

1. Click **Create** to create a team in the organization

1. Add existing users to the team

    - Click the team name and select **Actions > Add Users**

    - Check the users to include and click **Add Users**

Subject

Manually create users in MKE:

1. Go to the MKE web user interface

1. Click **Users** under **User Management**

1. Click **Create User**

1. Enter username, password, and full name

1. Click **Create** to create a new user

1. Optionally, check "**Is a Docker EE Admin**" to give the user administrator privileges

Assisted Practice
Create and Manage Teams and Users

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

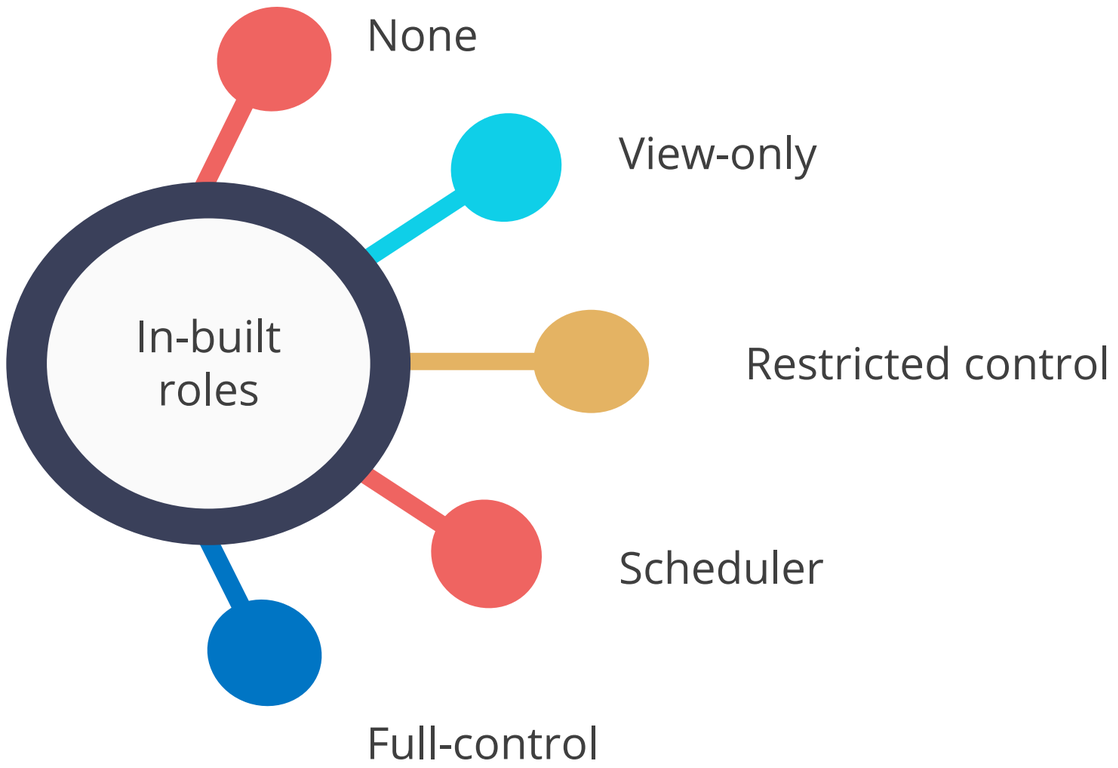
Steps to Perform:

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

Grant: Role

Role

A role is a set of permitted API operations on a resource set that you can assign to a specific user, team, or organization by creating a grant.



Role

Important rules for roles:

1. Roles are always enabled
2. Roles can’t be edited. Delete and recreate a role to modify it
3. Roles used within a grant can be deleted only if the grant is deleted
4. Only administrators can create and delete roles

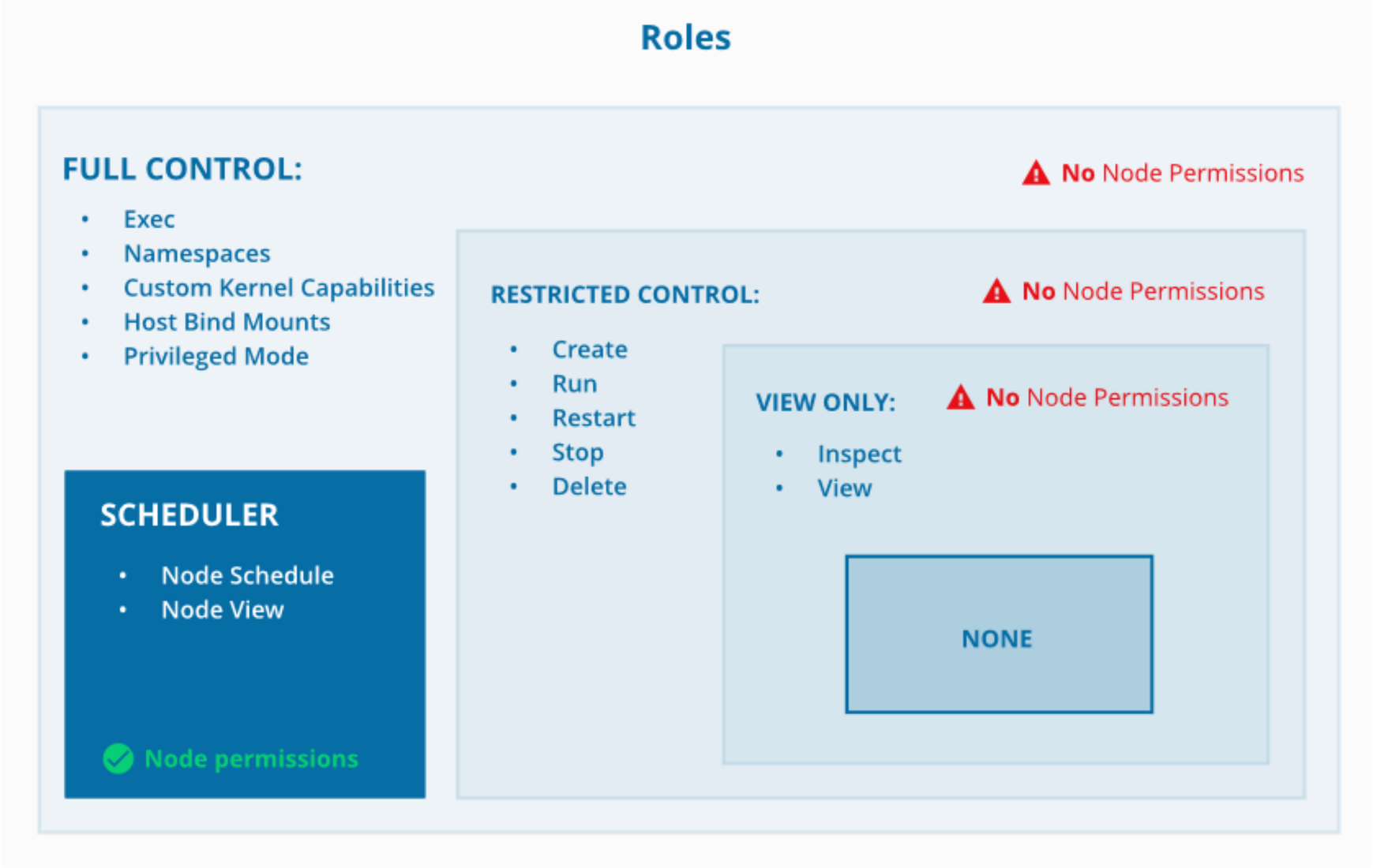
Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/mke/authorize/access-control-model.html

Built-in Roles

Built-in Roles:

- **None:** User has no access to Swarm or Kubernetes resources. This maps to the **No Access** role in UCP 2.1.x
- **View Only:** User can view resources but cannot create them
- **Restricted Control:** Users can view and modify resources but cannot run a service or container in a way that affects the node where it's running
- **Scheduler:** Users can view nodes and schedule workloads on these nodes. But to view workloads, users need permissions such as **Container View**
- **Full Control:** Users can view and edit all granted resources. They can create containers without any restriction, but can't see the containers of other users

Role



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

Create a Custom Role

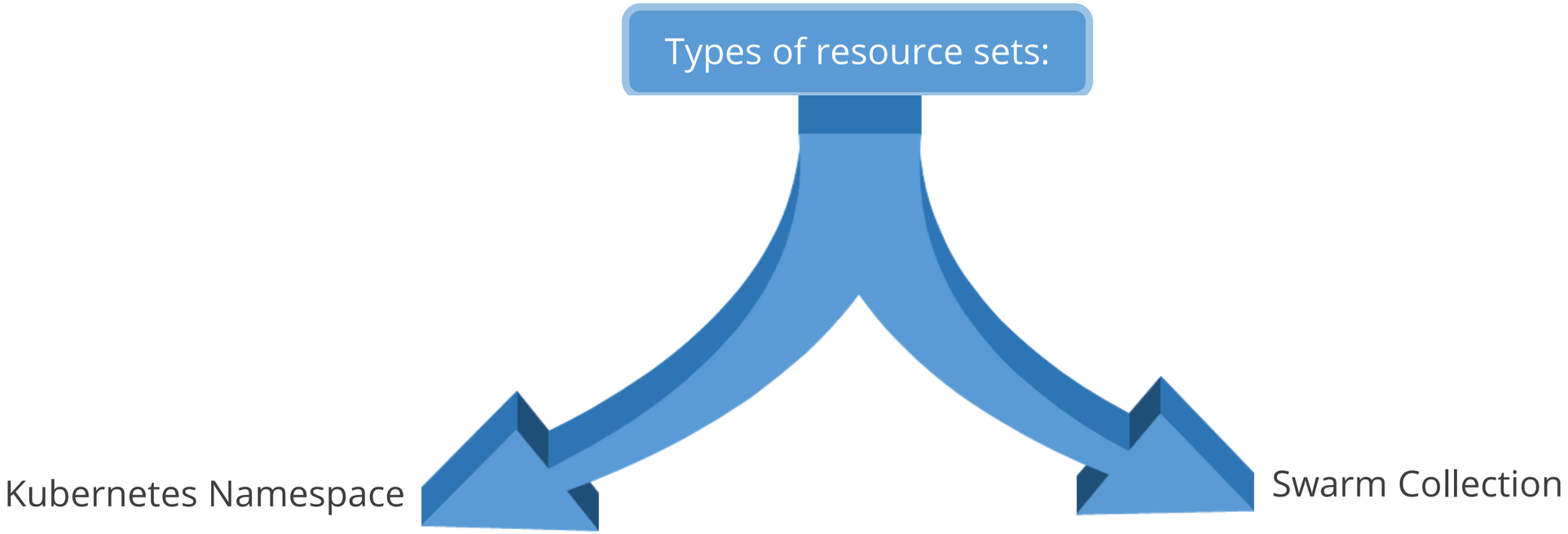
Create a custom role:

1. Go to the MKE web user interface
2. Click **Roles** under **Access Control**
3. Click **Create role**
4. Input the role name on the **Details** page
5. Click **Operations** to see all available API operations
6. Select the permitted operations as per the resource type
7. Click **Create** to create the custom role

Grant: Resource Collection

Resource Collection

In MKE, cluster resources can be accessed by grouping resources into resource sets. Resource sets can be combined with grants to give users permission to access specific cluster resources.



Resource Collection

Kubernetes namespaces:

A Kubernetes namespace is a group of Kubernetes-specific resources like pods, deployments, or services. RBAC policies and resource quotas can be enforced for a namespace. A Kubernetes resource can only be in one namespace, and namespaces cannot be nested inside one another.

Swarm collection:

A Swarm collection is a group of Swarm-specific resources like nodes, services, or volumes. A Swarm resource can only be in one collection at a time, but collections can be nested inside one another, to create hierarchies.

Granting Permissions

Create a Grant

Workflow for creating a grant:

1. Create and configure subjects
2. Define custom roles (or use defaults)
3. Group resources into a Swarm collection or Kubernetes namespace
4. Create grants by combining subject + role + resource set

Create a Kubernetes Grant

Creating a Kubernetes grant:

1. Log in to the MKE web UI and click **Access Control**
1. Click on **Grants**
1. In the Grants window, select **Kubernetes**
1. Click **Create Role Binding**
1. Under **Subject**, select Users, Organizations, or Service Account

Create a Kubernetes Grant

Creating a Kubernetes grant:

6. Click **Next** to save the Subject

6. Under Resource Set, enable the **Apply Role Binding to all namespaces** switch

6. Click **Next** to save the Resource set

6. Under **Role**, select a cluster role

6. Click **Create** to create the Kubernetes grant

Create a Swarm Grant

Creating a Swarm grant:

1. Log in to the MKE web UI and click **Access Control**
1. Click on **Grants**
1. In the Grants window, select **Swarm**
1. Click **Create Grant**
1. Under **Subject**, select Users or Organizations

Create a Swarm Grant

Creating a Swarm grant:

6. Click **Next** to save the Subject

6. Under Resource Set, click **View Children** until you get to the desired collection

6. Click **Select Collection** and then click **Next**

6. Under **Role**, select a role from the pull-down menu

6. Click **Create** to create the Kubernetes grant

Deploying a Service

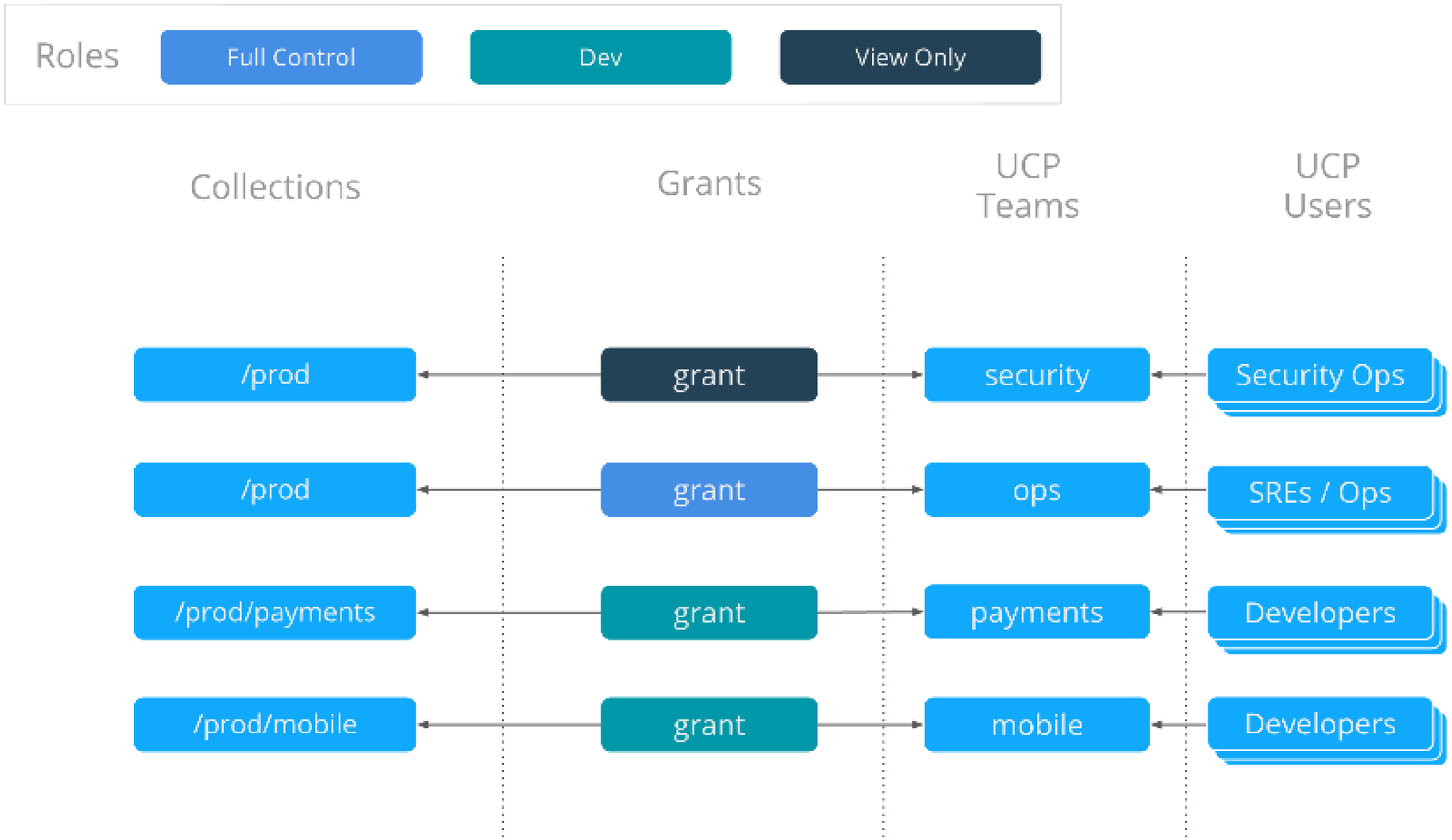
Service Deployment

Steps to deploy a service with view-only access:

1. Create an organization with a team and users
2. Define roles with allowable operations for resource type
3. Create collections or namespaces for accessing resources
4. Create grants that join team + role + resource set

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

Grant Composition



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

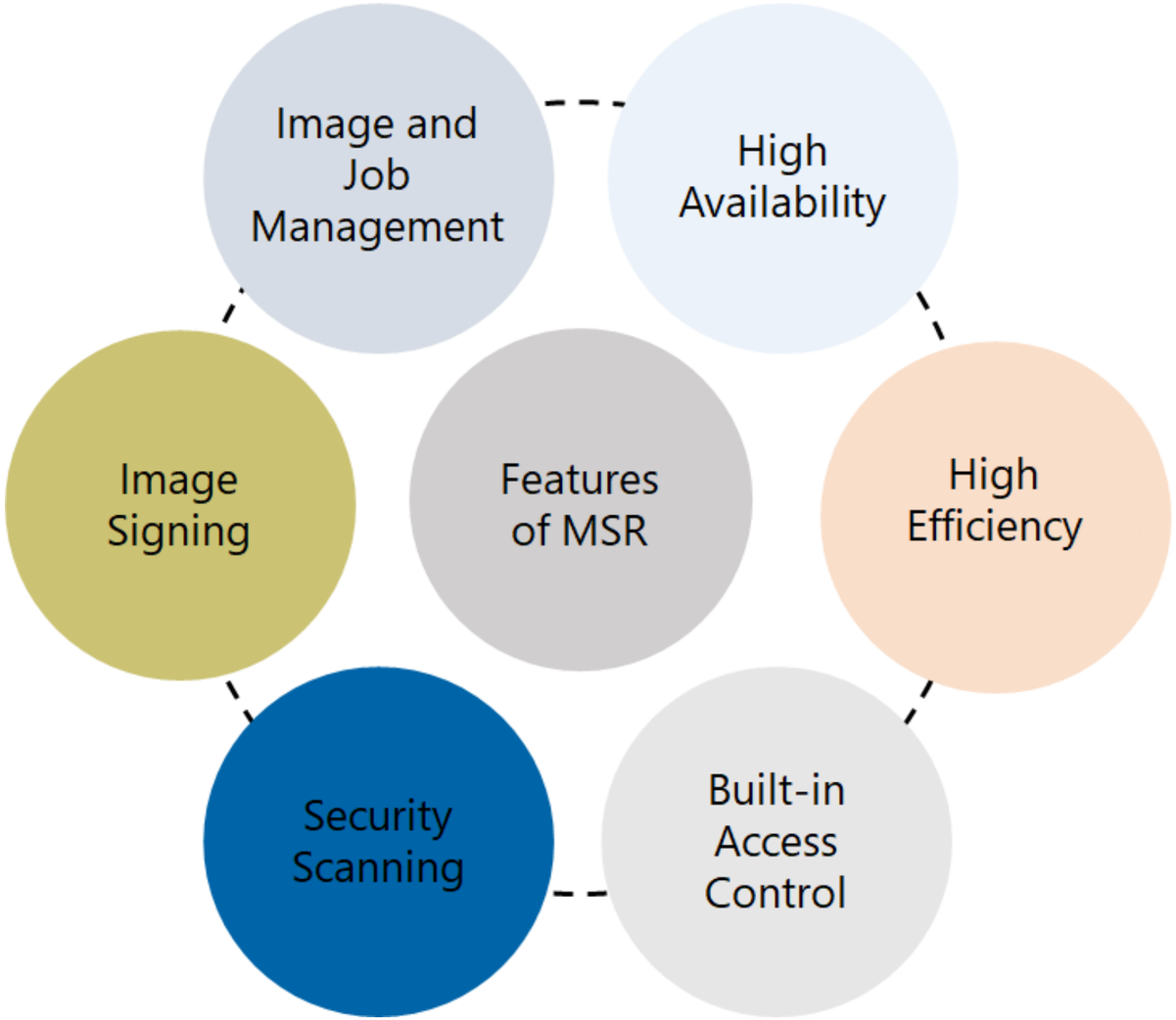
Mirantis Secure Registry

Introduction to Mirantis Secure Registry

Mirantis Secure Registry (MSR) is an enterprise-level image storage solution provided by the Mirantis. It is installed behind the firewall, either on-premises or on a virtual private cloud, to securely store and manage the Docker images.

Mirantis Secure Registry (MSR)

Features of MSR



Features of MSR

Image and job management:

MSR can serve as a Continuous Integration and Continuous Delivery (CI/CD) component, in the building, shipping, and running of applications. Web-based UI of MSR allows users to:

- Browse images and audit repository events
- Check Dockerfile for image production code
- Enable or disable security scanning
- Audit jobs

High availability:

- MSR is highly available with multiple replicas of all containers and metadata, so it will continue to work in the event of a machine failure, thus allowing for repair.

Features of MSR

High efficiency:

- MSR is highly efficient as it reduces the bandwidth used while pulling Docker images by caching images closer to users.
- It can clean up unreferenced manifests and layers.

Built-in access control:

- MSR uses Role Based Access Control (RBAC) to manage image access, either manually, with LDAP, or with Active Directory.

Features of MSR

Security scanning:

- A built-in security scanner can be used to discover the software versions in an image.
- It scans each layer and aggregates the results, offering a complete picture of what is being shipped as a part of your stack.
- It is able to provide unprecedented insight about exposure to known security threats.

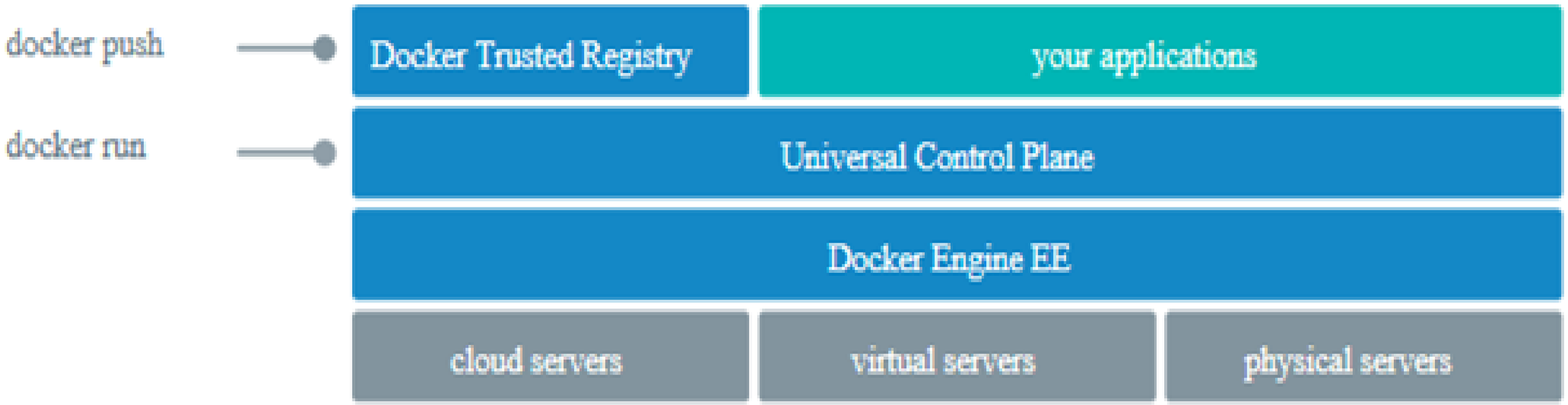
Image signing:

- MSR allows users to sign and verify images using Docker Content Trust.

MSR Architecture

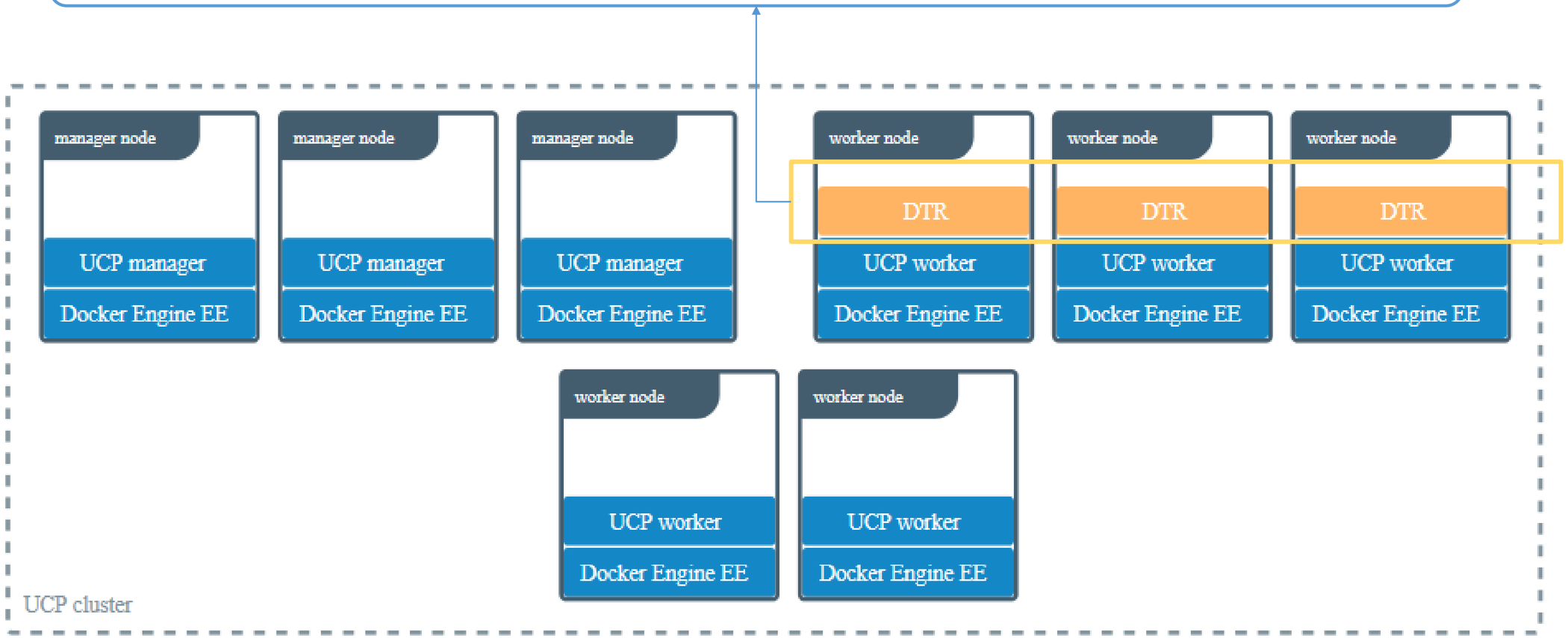
MSR Architecture

MSR is a containerized application that runs on a Mirantis Kubernetes Engine cluster. Once deployed, Docker CLI client can be used to log in, push, and pull images.



MSR Architecture

Multiple MSR replicas are deployed on the MKE worker nodes to achieve high availability.



MSR Components

Following containers are started after the installation of MSR:

Name	Description
dtr-api-<replica_id>	Executes the MSR business logic
dtr-garant-<replica_id>	Manages the MSR 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 MSR 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 MSR. This makes the containers capable of communication.

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

Volumes

Volumes used:

Volume name	Description
dtr-ca-<replica_id>	Root key material for the MSR 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 MSR is configured to store images on the local filesystem)
dtr-rethink-<replica_id>	Repository metadata
dtr-nfs-registry-<replica_id>	Docker images data (when MSR is configured to store images on NFS)

Image Storage

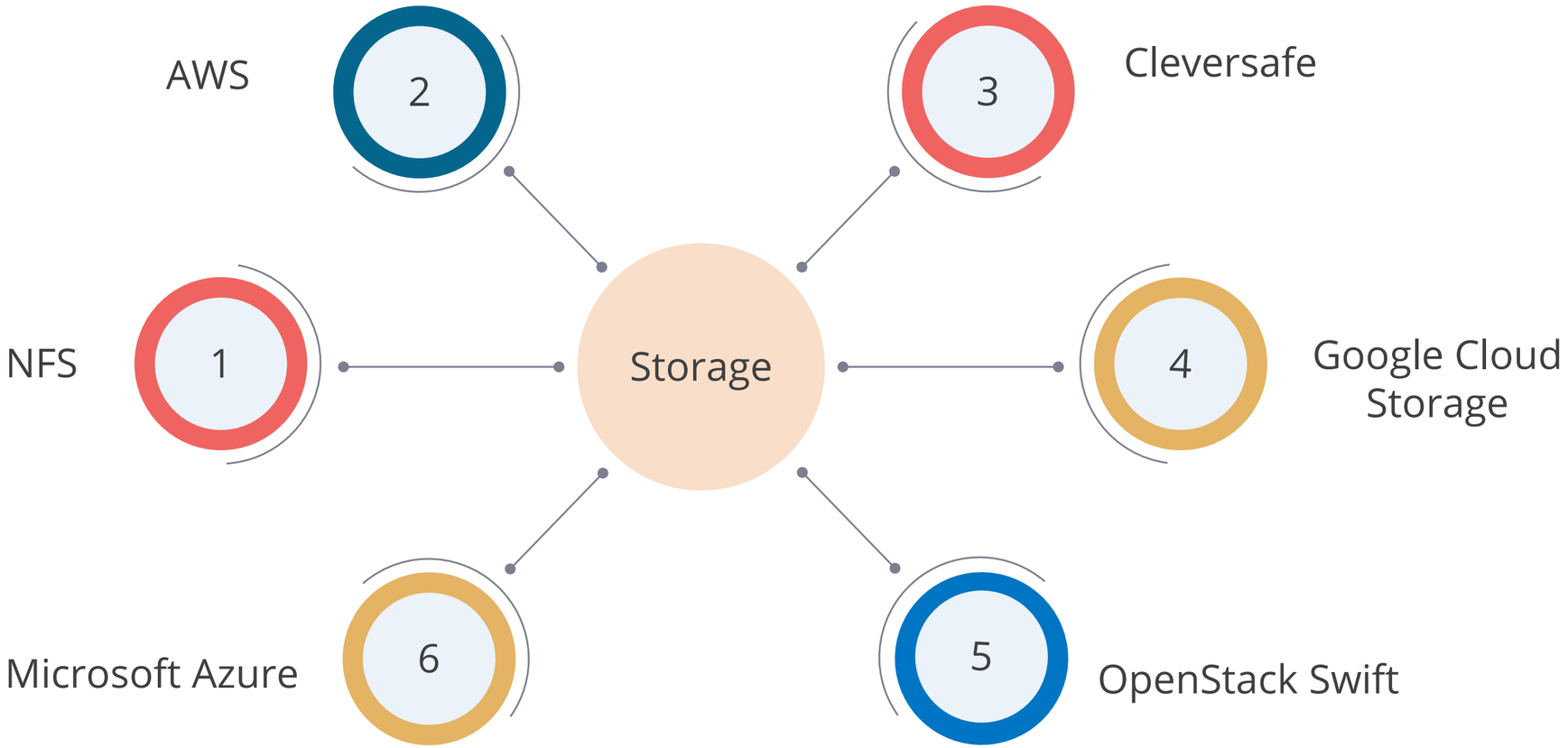
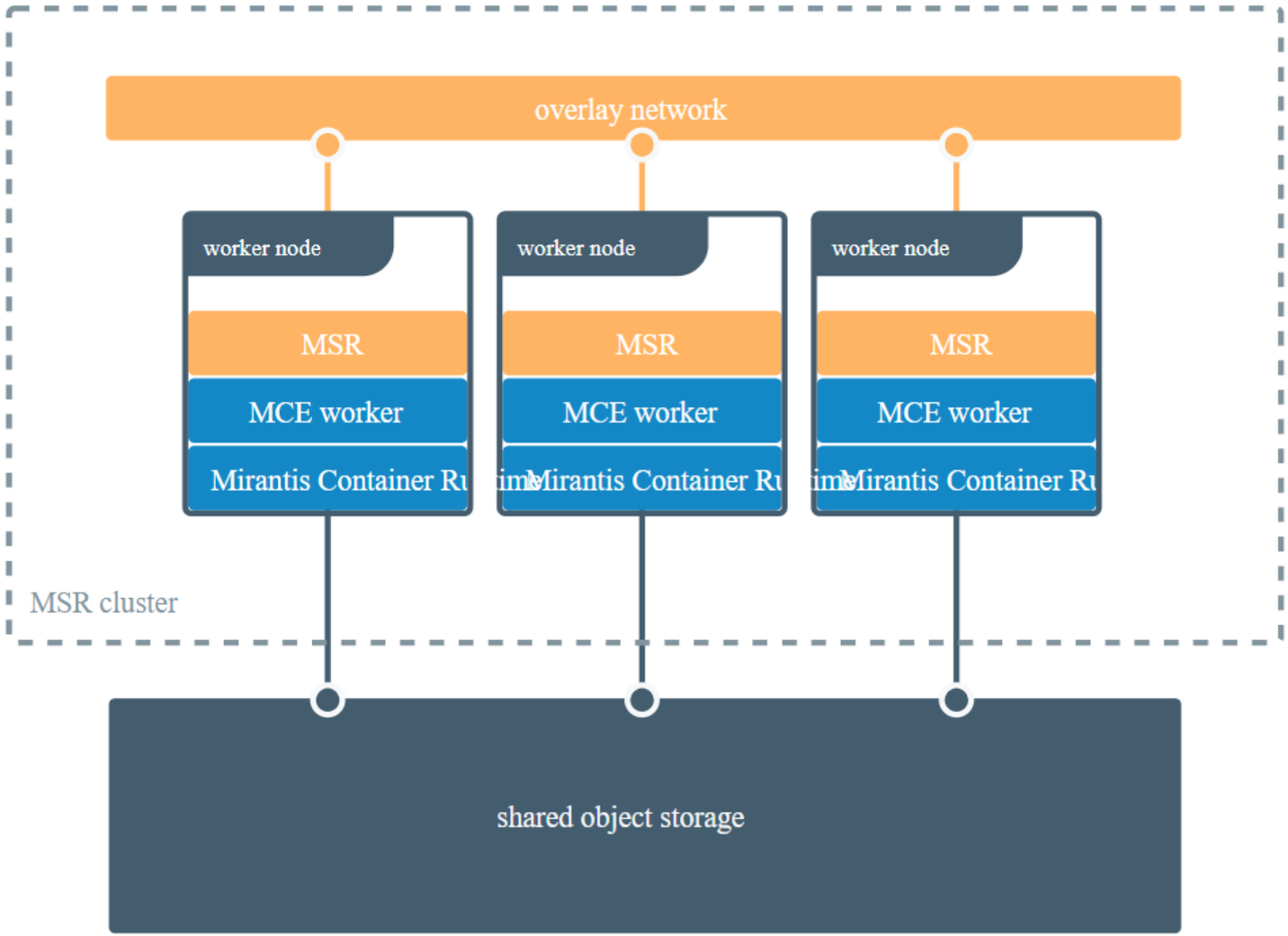


Image Storage



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

MSR Installation

Installation Requirements

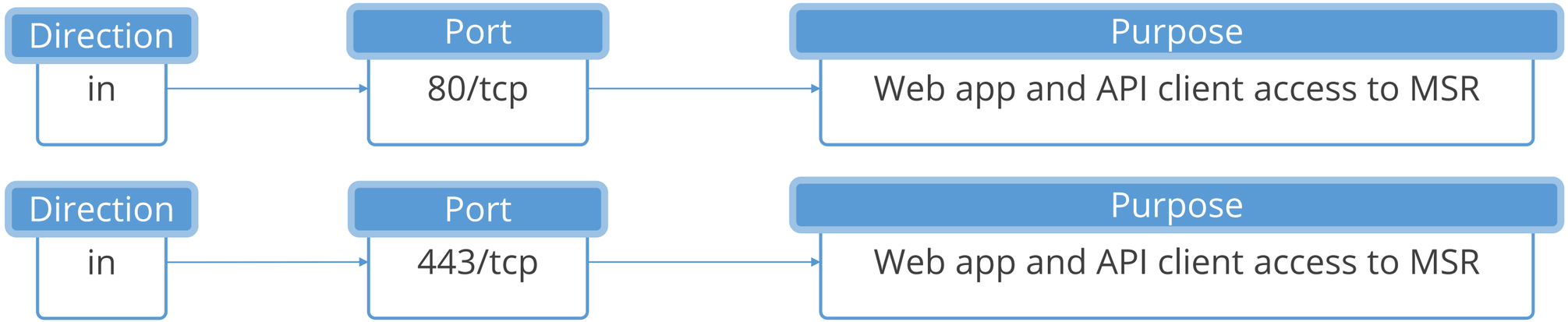
Hardware and software requirements:

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

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

Installation Requirements

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



Installation Requirements

MKE configuration:

- During the installation or backing up of MSR on a MKE cluster, the administrator must deploy containers on **MKE manager nodes or nodes running the MSR**.

- If administrators are not deployed on the **MKE manager nodes or the nodes running MSR**, then the MSR 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

MKE setting: Restricting users from deploying to manager nodes:

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

Assisted Practice
Install Mirantis Secure Registry

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

Steps to Perform:

1. Get MSR install command from Admin Settings > Mirantis Secure Registry tab in MKE
2. Run the command on Worker 1 node to install MSR
3. Log in to MSR with admin credentials

Post-Installation

Post-installation steps:

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

1. Access MSR from browser to configure:
   - The TLS certificates by updating them from **System > General**
   - The storage backend by navigating to **System > Storage**

Post-Installation

Post-installation steps:

3. Testing whether the images can be pushed or pulled:
    a. Configure the local Mirantis Container Runtime to trust the certificate:
        i. Download the **MSR CA certificate**

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

        i. Refresh the list of certificates to trust

        sudo update-ca-certificates

        i. Restart the Docker daemon

        sudo service docker restart

Post-Installation

Post-installation steps:

3. Testing whether the images can be pushed or pulled:
    b. Create an image repository:
        i. Log in to https://<dtr-url with MKE 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:

3. Testing whether the images can be pushed or pulled:
    c. Pull and Push an image:
        i. Tag the image:

        # Pull the latest wordpress image from Docker Hub

        docker pull wordpress:latest

        # Tag the wordpress:latest image with the full repository name of MSR repository

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

        i. Push the image to MSR:

        docker login msr-example.com

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

Post-Installation

Post-installation steps:

4. Join the replicas to the cluster for high availability:
    a. Load the MKE user bundle
    b. Run command:

    *docker run -it --rm *

    *mirantis/dtr:2.8.2 join *

    *--ucp-node <mke-node-name> *

    --ucp-insecure-tls

    a. Check all the running replicas:
        i. On the MKE web UI, navigate to **Shared Resources > Stacks**. The replicas will be displayed on the console

Uninstall MSR

Uninstallation step:

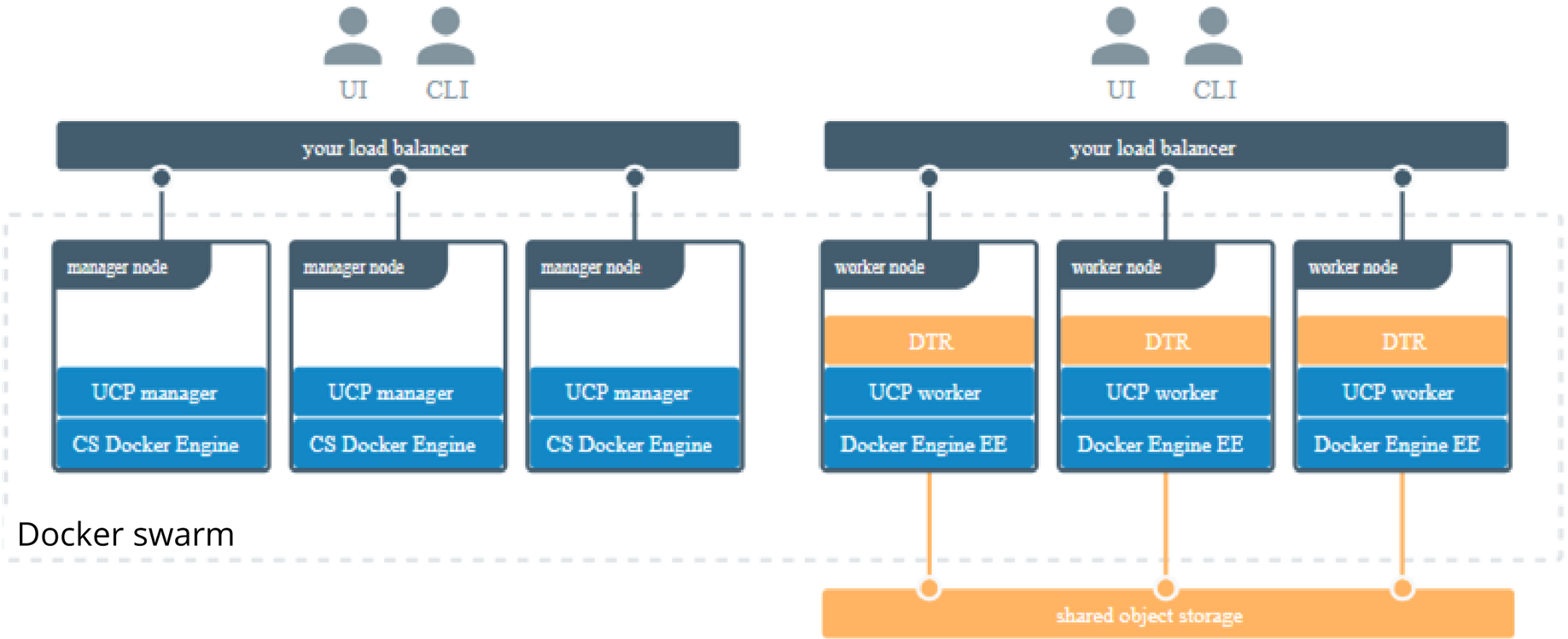
Destroy command must be run for every replica:

```
docker run -it --rm \
  mirantis/dtr:2.8.2 destroy \
  --ucp-insecure-tls
```

MSR: High Availability

High Availability

Mirantis Secure Registry (MSR) is an enterprise-level image storage solution that is capable of scaling horizontally. As the usage increases, the replicas can be added to make the DTR scale for high availability.



MSR Failure Tolerance

Additional replicas must be added to the MSR cluster to make MSR tolerant to failures.

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

Sizing MSR Installation

Thumb rules for sizing MSR installation with high availability:

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

High availability on MKE and MSR:

To have high availability on MKE and MSR, you need a minimum of:

- 3 dedicated nodes to install MKE with high availability
- 3 dedicated nodes to install MSR with high availability
- As many nodes as you want for running your containers and applications

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

Add Replicas

Adding replicas to an existing MSR deployment:

- Use *ssh* to log in to a node that is already a part of MKE
- Run the following MSR join command:

```
docker run -it --rm \
    mirantis/dtr:2.8.2 join \
  --ucp-node <mke-node-name> \
        --ucp-insecure-tls
```

- Add this MSR replica to the load balancing pool if the load balancer is available

It is the hostname of the MKE node where the MSR replica is deployed.

It confirms the certificates used by MKE

Remove Replicas

Removing replicas from MSR deployment:

1. Use *ssh* to log in to any node that is part of MKE

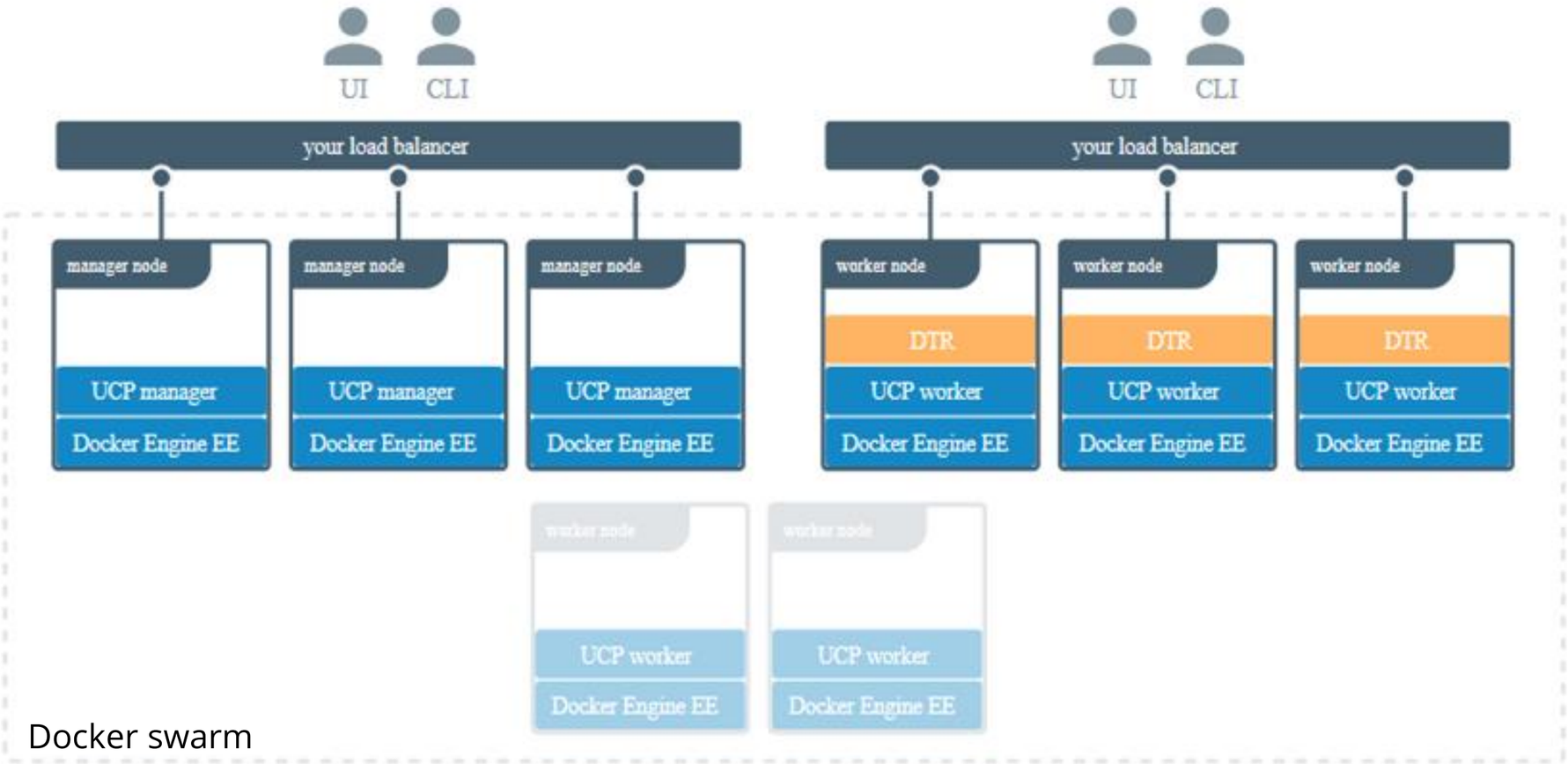
1. Run the following MSR remove command:
   *docker run -it --rm *
   *mirantis/dtr:2.8.2 remove *
   --ucp-insecure-tls

1. You will be prompted for:
   - Existing replica id: the id of any healthy MSR replica of that cluster
   - Replica id: the id of the MSR replica you want to remove. It can be the id of an unhealthy replica
   - MKE username and password: the administrator credentials for MKE

MSR: Load Balancer

Load Balancer

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



Load Balancer

Endpoints exposed by MSR:

- */_ping*: Checks the health of MSR replica and is useful for load balancing or other automated health check tasks
- */nginx_status*: Returns the number of connections being handled by the NGINX front-end used by MSR
- */api/v0/meta/cluster_status*: Returns extensive information about all MSR 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:

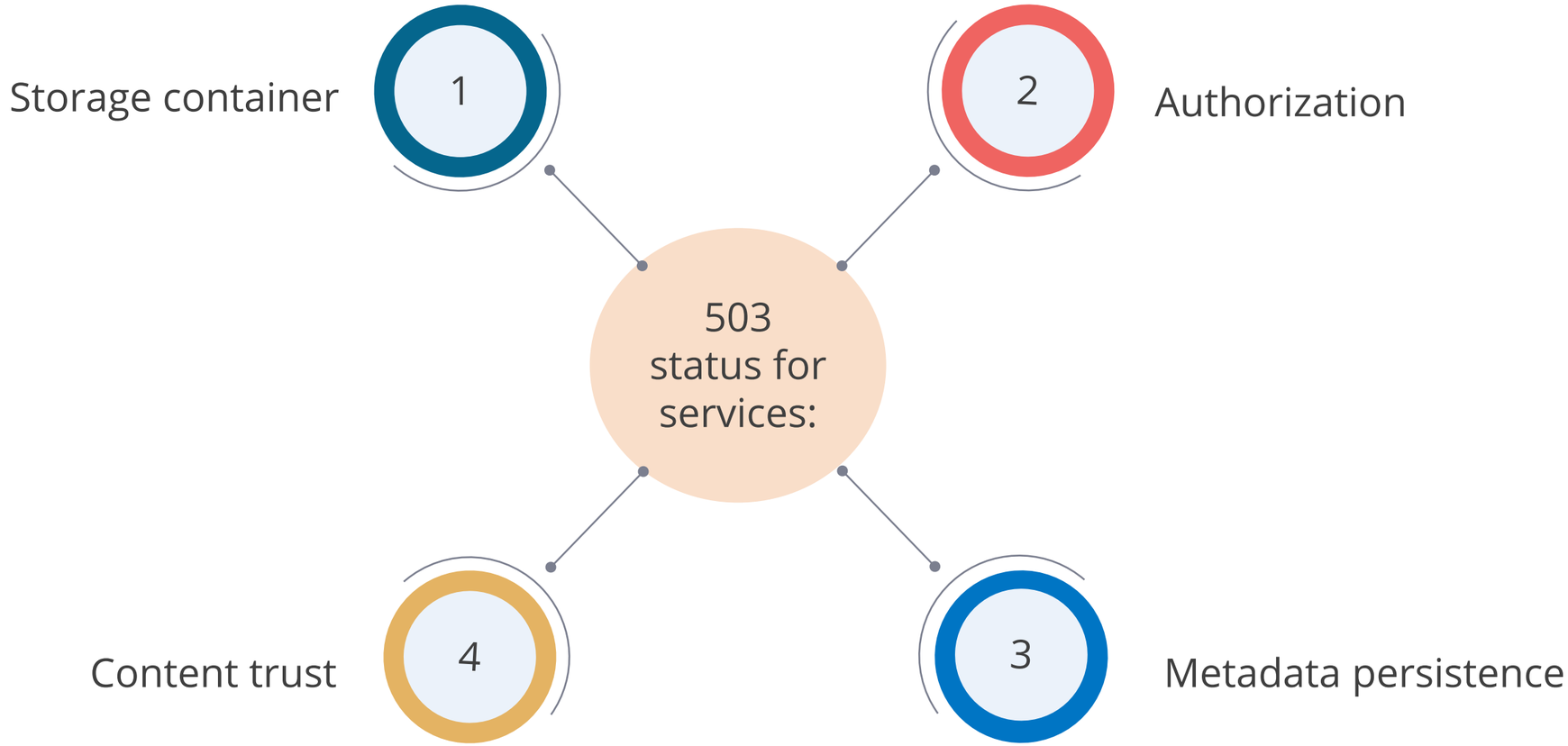
/_ping

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 MSR 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 <MSR_REPLICA_1_IP>:80  max_fails=2 fail_timeout=30s;
      server <MSR_REPLICA_2_IP>:80  max_fails=2 fail_timeout=30s;
      server <MSR_REPLICA_N_IP>:80   max_fails=2 fail_timeout=30s;
    }
```

Load Balancer: Configuration and Deployment

Configure load balancer for MSR using NGINX:

```
upstream dtr_443 {
    server <MSR_REPLICA_1_IP>:443 max_fails=2 fail_timeout=30s;
    server <MSR_REPLICA_2_IP>:443 max_fails=2 fail_timeout=30s;
    server <MSR_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:

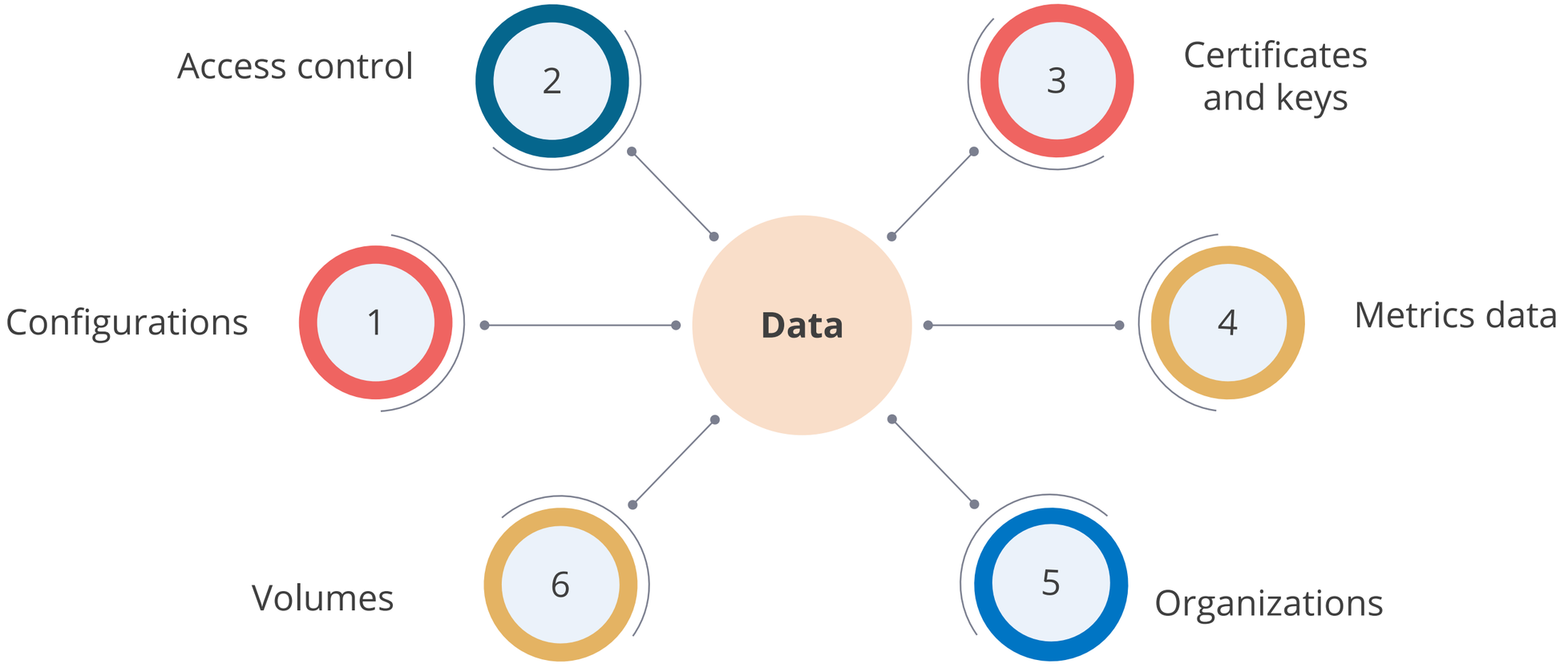
It is a two-step process:

1. Creating file *nginx.conf*
2. Deploying the load balancer using the following 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
```

MKE: Backup and Restore

MKE: Backup



MKE: Backup

Create an MKE backup using CLI:

1. Run the **mirantis/ucp:3.3.2** backup command on a single MKE manager and include the **--file** and **--include-logs** options

   *docker container run --rm *

   *--log-driver none --name ucp *

   *--volume /var/run/docker.sock:/var/run/docker.sock *

   *--volume /tmp:/backup mirantis/ucp:3.3.2 backup *

   *--file mybackup.tar --passphrase "secret12chars" *

   --include-logs=false

2. A **.tar** file will be created with the contents of all volumes used by MKE archived in it

Note: Replace 3.3.2 with the version you are currently running.

MKE: Backup

Create an MKE backup using MKE Web UI:

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

MKE: Restore

To restore MKE from a existing backup:

- Uninstall MKE from the swarm by using the **uninstall-ucp** command
- Restore operations must run using the same major/minor MKE version (and mirantis/ucp image version) as the backed up cluster
- MKE will start using new TLS certificates if MKE is restored using a different swarm cluster. New client bundles need to be downloaded as the existing ones won't work anymore.
- Example to show how to restore MKE from an existing backup file, presumed to be located at **/tmp/backup.tar**:

  *docker container run --rm --interactive --name ucp *

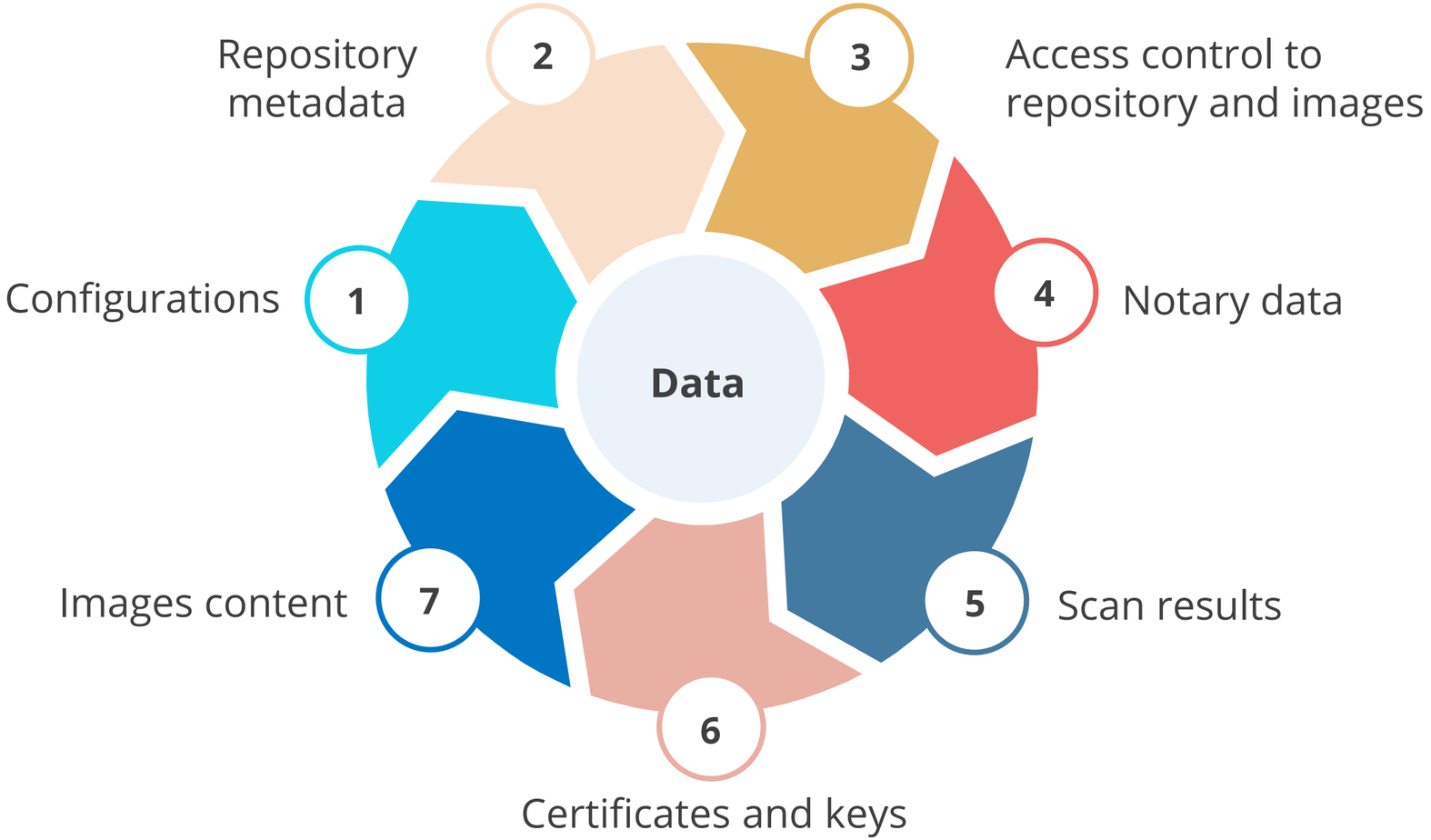
  *--volume /var/run/docker.sock:/var/run/docker.sock *

  mirantis/ucp:3.3.2 restore < /tmp/backup.tar

MSR: Backup and Restore

MSR: Backup

Data managed by MSR:



MSR: Backup

Procedure for MSR backup:

1. Find the Replica Id
2. Backup image content
3. Backup MSR metadata
4. Verify your backup

MSR: Restore

Steps to restore MSR:

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

Disaster Recovery

MKE Disaster Recovery

Recover swarm from losing the quorum:

- Uninstall MKE using the **uninstall-ucp** command, if MKE is still installed on the swarm
- Perform restore from an existing backup on any node:
  - Perform restore operation on a manager node if there is an existing swarm
  - Restore operation will create a manager node if no swarm exists

MSR Disaster Recovery

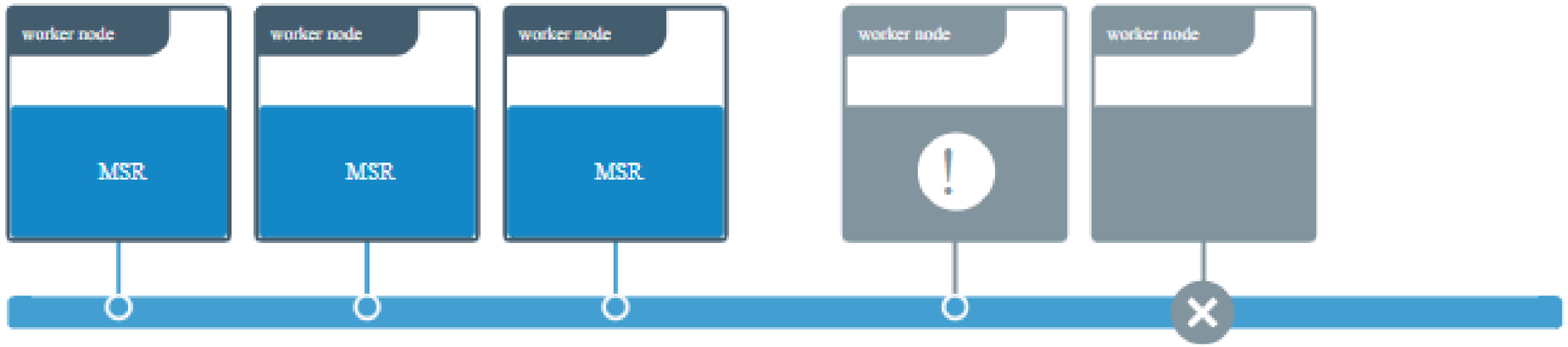
Disaster recovery in MSR:

- MSR is a clustered application, so users can join multiple replicas for high availability.
- To make an MSR 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

MSR Disaster Recovery

Scenario 1: 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.

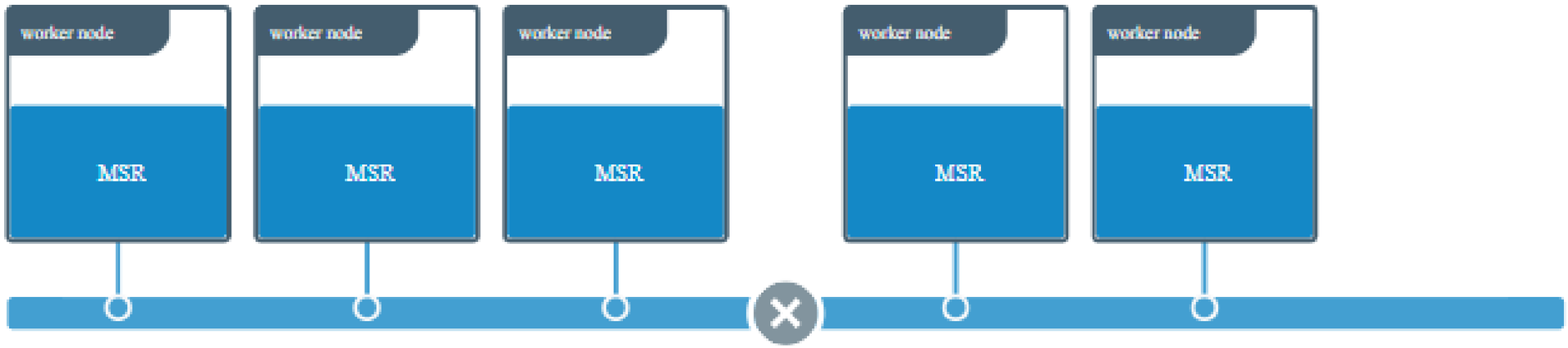


Source:: https://docs.mirantis.com/containers/v3.1/dockeree-products/msr/disaster-recovery/index.html

MSR Disaster Recovery

Split-brain scenario

- There is a five-replicas MSR deployment, and something goes wrong with the overlay network connecting the replicas, causing them to be separated in two groups.

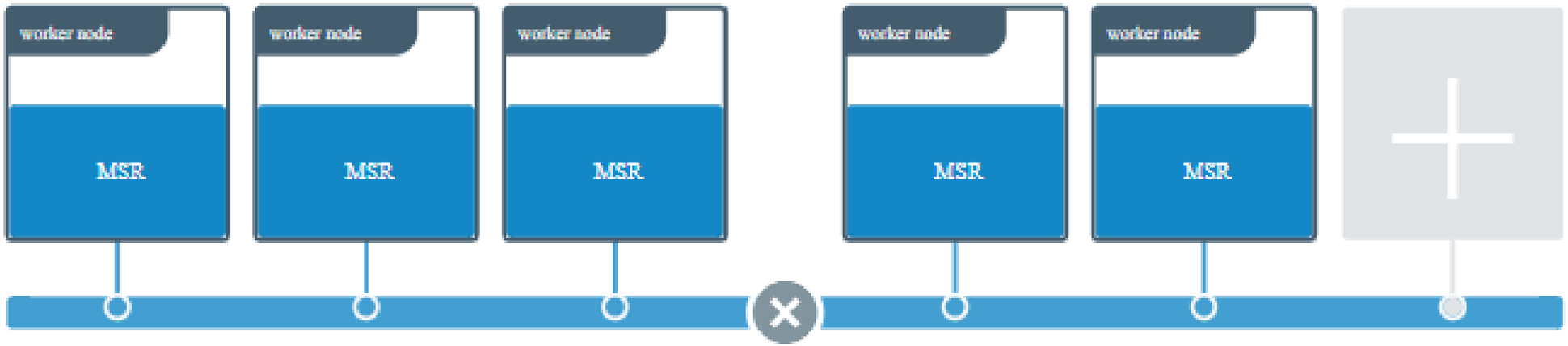


Source:: https://docs.mirantis.com/containers/v3.1/dockeree-products/msr/disaster-recovery/repair-a-single-replica.html

MSR Disaster Recovery

Split-brain scenario

- If 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 ends up in the side of the network partition that 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 as both groups can now accept writes and their histories start to diverge, making the two groups effectively two different clusters.

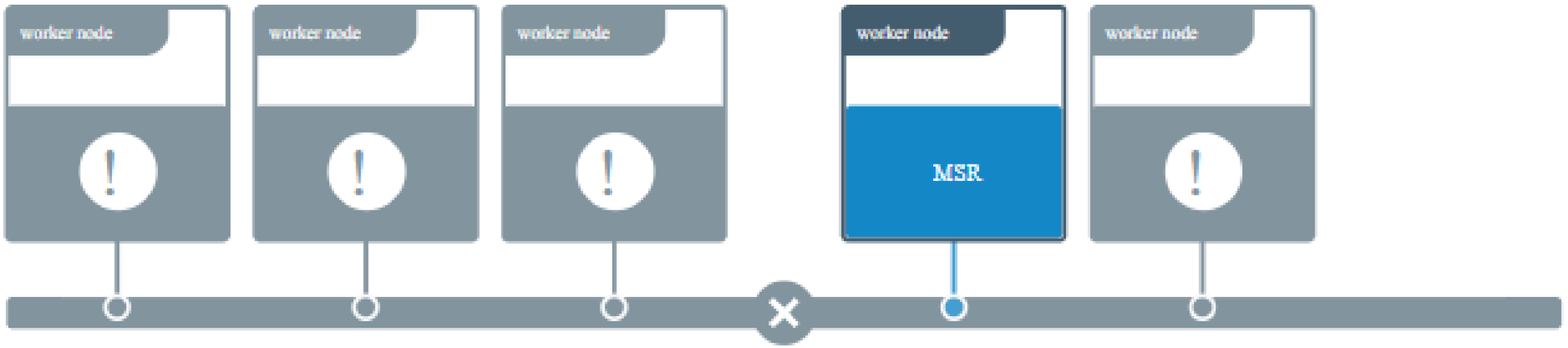


Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/msr/disaster-recovery/repair-a-single-replica.html

MSR Disaster Recovery

Scenario 2: 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 MSR 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.

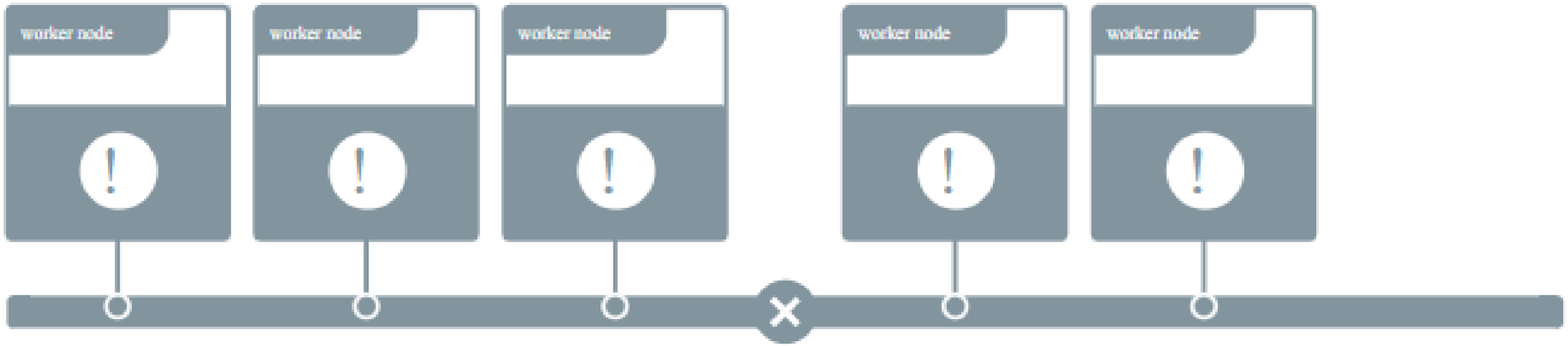


Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/msr/disaster-recovery/index.html

MSR Disaster Recovery

Scenario 3: All replicas are unhealthy

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



Source: https://docs.mirantis.com/containers/v3.1/dockeree-products/msr/disaster-recovery/index.html

Key Takeaways

- Mirantis Kubernetes Engine 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 MKE. MKE provides the capability of joining multiple manager nodes to the cluster to counter the failure of the manager node.
- Mirantis Secure Registry (MSR) provides image storing ability to MKE. On increased usage, the replicas are added to make the MSR scale for high availability.
- Caching the images closer to the user helps in reducing the bandwidth required for pulling Docker images.

Lesson-End Project

Create a Grant with Write-Access Role and Shared Resources

Problem Statement:

Your team lead has asked you to create an organization and a team on MKE 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

Thank You