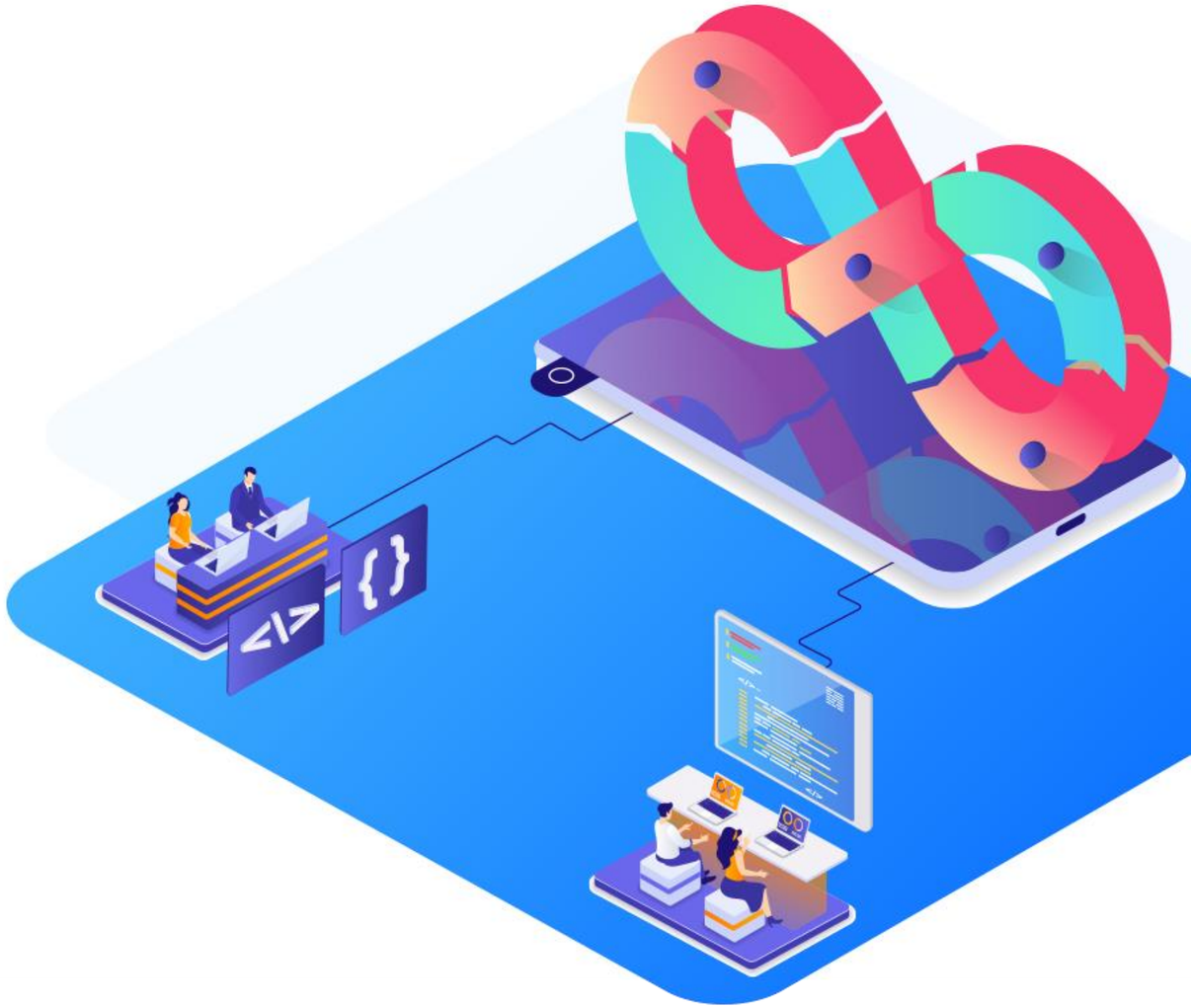
Containerization with Docker



Docker Container Orchestration

Learning Objectives

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

- Set up a Swarm cluster to work with manager and worker nodes
- Run replicated and global services inside Swarm to facilitate isolated service executive
- Deploy application stack on Swarm to scale multi-container across nodes
- Build template-based service creation to automate repetitive tasks
- Mount volumes on Swarm to provide persistent storage for container for data accessibility

Introduction to Orchestration

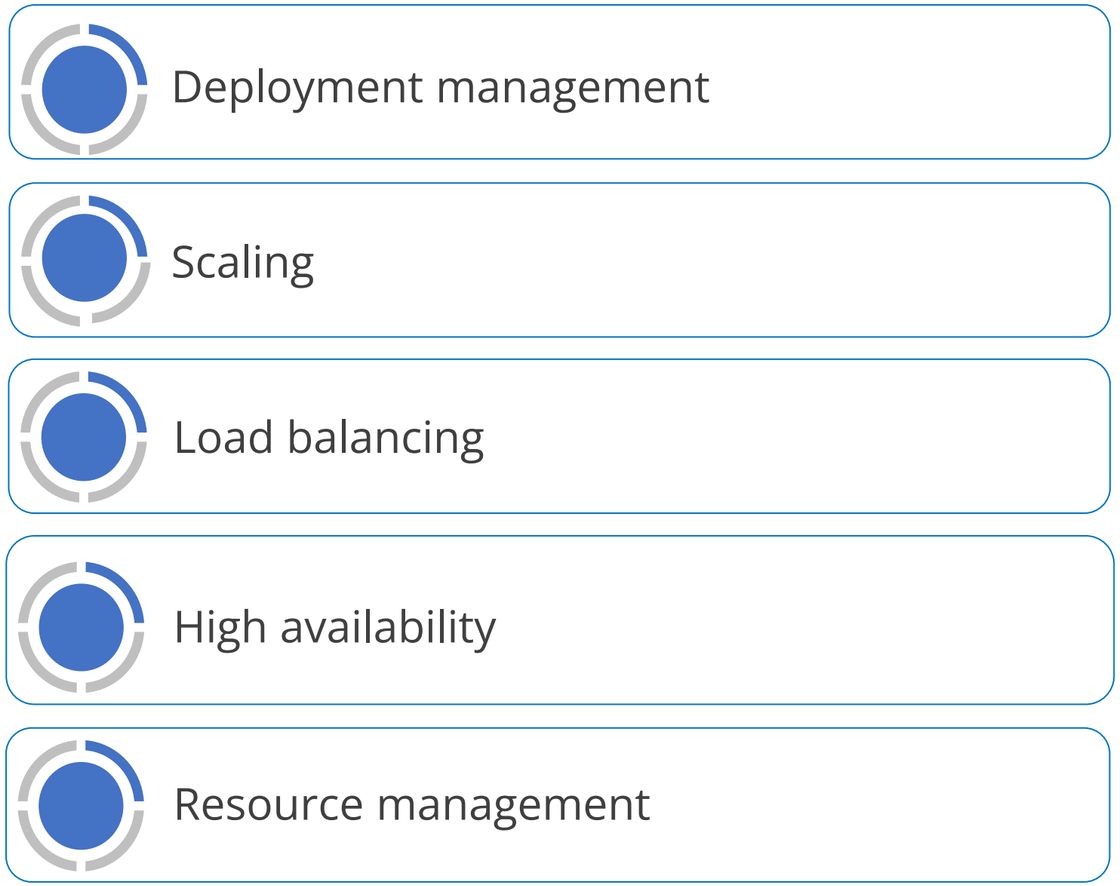
What Is Container Orchestration?

It refers to the management and coordination of containerized applications to ensure they run efficiently and reliably across various environments.

It involves automating deployment, scaling, load balancing, and container resource allocation tasks. It uses multiple tools, such as Swarm and Kubernetes.

Container Orchestration: Key Aspects

Here are the key aspects of container orchestration:



What Problems Do Orchestration Solves?

1. Increased Complexity with Multi-Service Applications
2. Automation Needs
3. Service Discovery Challenges
4. Sensitive Data Management
5. Health Monitoring and Self-Healing

How Industries Use Container Orchestration?

Healthcare

Docker orchestration can be used to deploy and manage containerized applications that process patient data in a secure, isolated environment.

Finance

It can help manage the deployment of trading algorithms across multiple nodes, ensuring that they are always running and can scale up during high-traffic periods.

Ecommerce

It can automatically scale microservices such as user authentication, product catalog, and payment processing.

How Industries Use Container Orchestration?

Telecommunication

It can manage the deployment of VNFs across multiple data centers, enabling the company to dynamically allocate resources based on network demand.

Edtech

Docker orchestration can be used to manage and scale the platform’s backend services, such as user authentication, content delivery, and video streaming.

Manufacturing

It manages the deployment of containerized applications that analyze data from IoT devices, enabling real-time monitoring and automation of production processes.

Business Use Cases

Scalable Web Applications

- **Scenario:** An online retailer experiences fluctuating traffic, particularly during sales events.
- **Use Case:** Docker container orchestration enables the automatic scaling of web applications to handle traffic spikes, ensuring a seamless user experience and preventing downtime during high-demand periods.

Microservices Management

- **Scenario:** A software company develops a complex application composed of multiple microservices that need to interact with each other reliably.
- **Use Case:** Container orchestration simplifies the management of these microservices by handling service discovery, load balancing, and communication, allowing developers to focus on building features rather than infrastructure management.

Business Use Cases

Disaster Recovery and High Availability

- **Scenario:** A financial institution needs to ensure that its critical applications are always available, even during hardware failures or disasters.
- **Use Case:** Container orchestration provides automated failover and redundancy, ensuring that applications continue running smoothly across different data centers or cloud regions, thereby enhancing business continuity and minimizing downtime.

Big Data and Machine Learning

- **Scenario:** A data analytics company processes large datasets and trains machine learning models that require significant computational resources.
- **Use Case:** Docker container orchestration allows efficient task distribution across multiple nodes, automatically scaling resources up or down based on the workload, ensuring that data processing and model training are completed efficiently.

Quick Check

A small startup wants to deploy a simple web application across multiple servers to ensure it's always available, even if one server goes down, and also make the solution scalable based on the workload. Which Docker orchestration tool should they choose?

A. Docker Compose

B. Docker Swarm

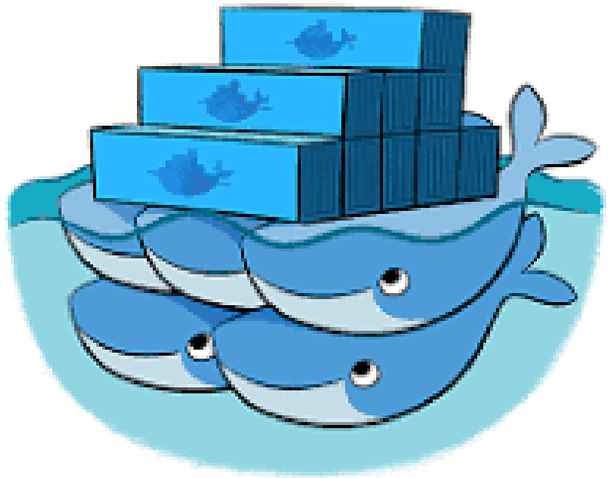
C. Git

D. Docker CLI

Docker Swarm

Docker Swarm: Overview

Docker Swarm is a container orchestration tool designed to manage a cluster of Docker hosts and containers effectively.



It allows users to create a cluster of Docker hosts, enabling high availability, scalability, and load balancing for containerized applications.

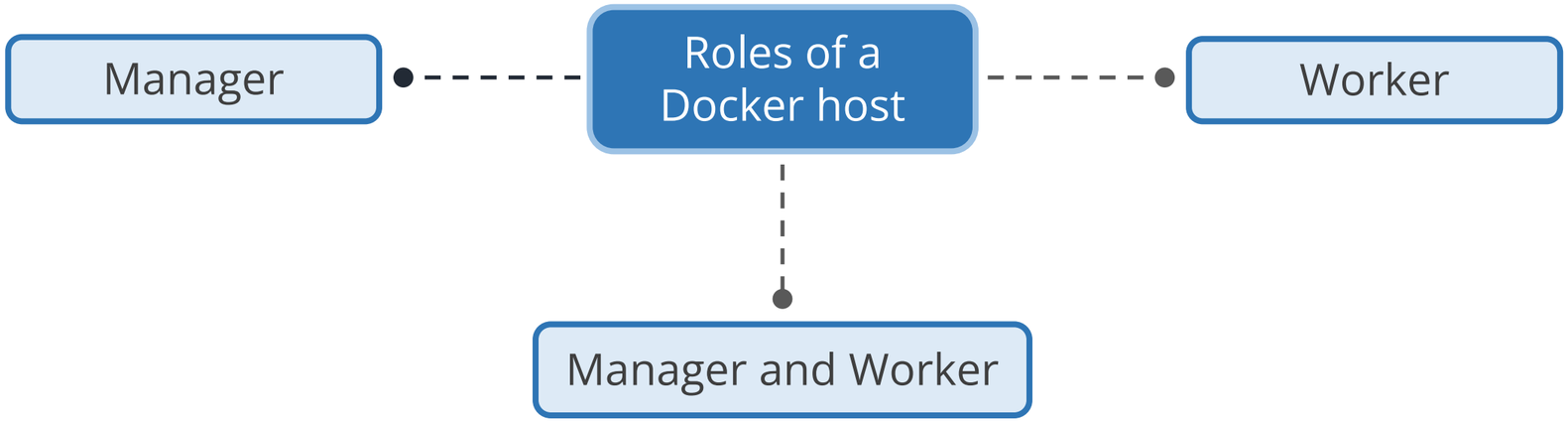
Docker Swarm: Features

- The Swarm environment is transformed into a highly scalable infrastructure through load balancing.
- Any communication between the Swarm's manager and client nodes is encrypted with high level security.
- Within your environment, there is auto load balancing, which you can script into how you write out and build the Swarm environment.
- Swarm makes accessing and managing the environment very simple.
- Swarm allows you to revert surroundings to a previous, safe state.

How Does Docker Swarm Work?

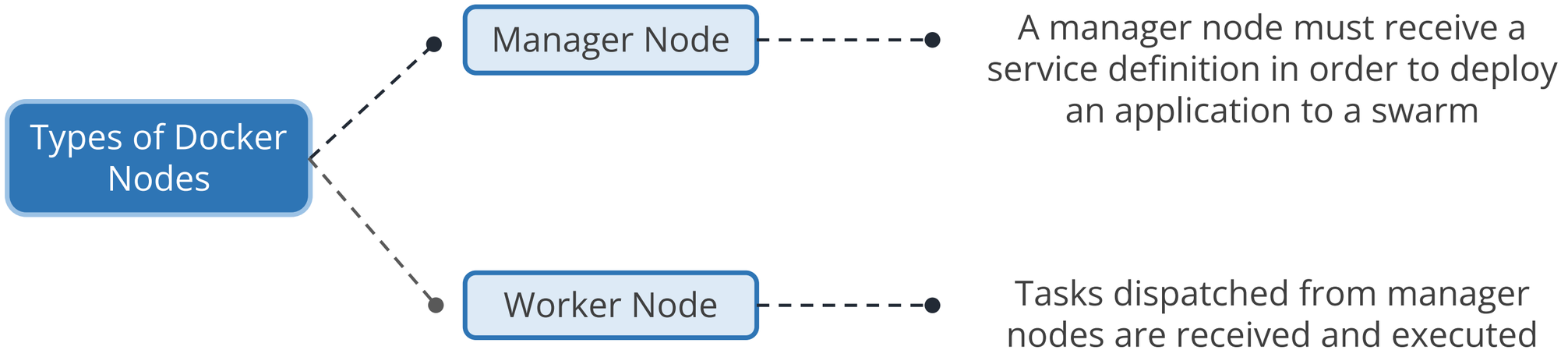
In Docker Swarm, services launch containers, enabling scalable applications. A service comprises containers sharing the same image.

Before deploying a service, Docker Swarm must have at least one node installed:



Nodes: Overview

In swarm, the instances of Docker Engine are distributed across multiple physical or virtual machines. These instances of Docker engine are known as Nodes.



Manager Node: Tasks Handled

Cluster management tasks handled by Manager Node:

- Maintaining cluster state
- Scheduling services
- Serving HTTP API endpoints (swarm mode)

Worker Node

It is a Docker Engine instance that executes the containers.

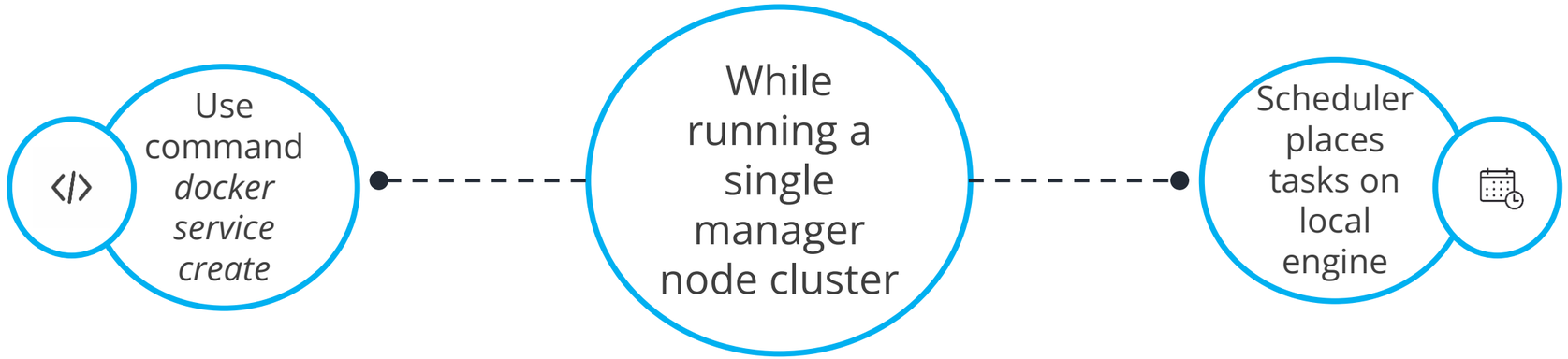
Worker nodes do not play a role in:

- Raft distributed state
- Scheduling decisions
- Serving the HTTP API (swarm mode)

Note:

- All manager nodes are worker nodes.
- Swarm can be created for one manager node.
- At least one manager node is necessary to have a worker node.

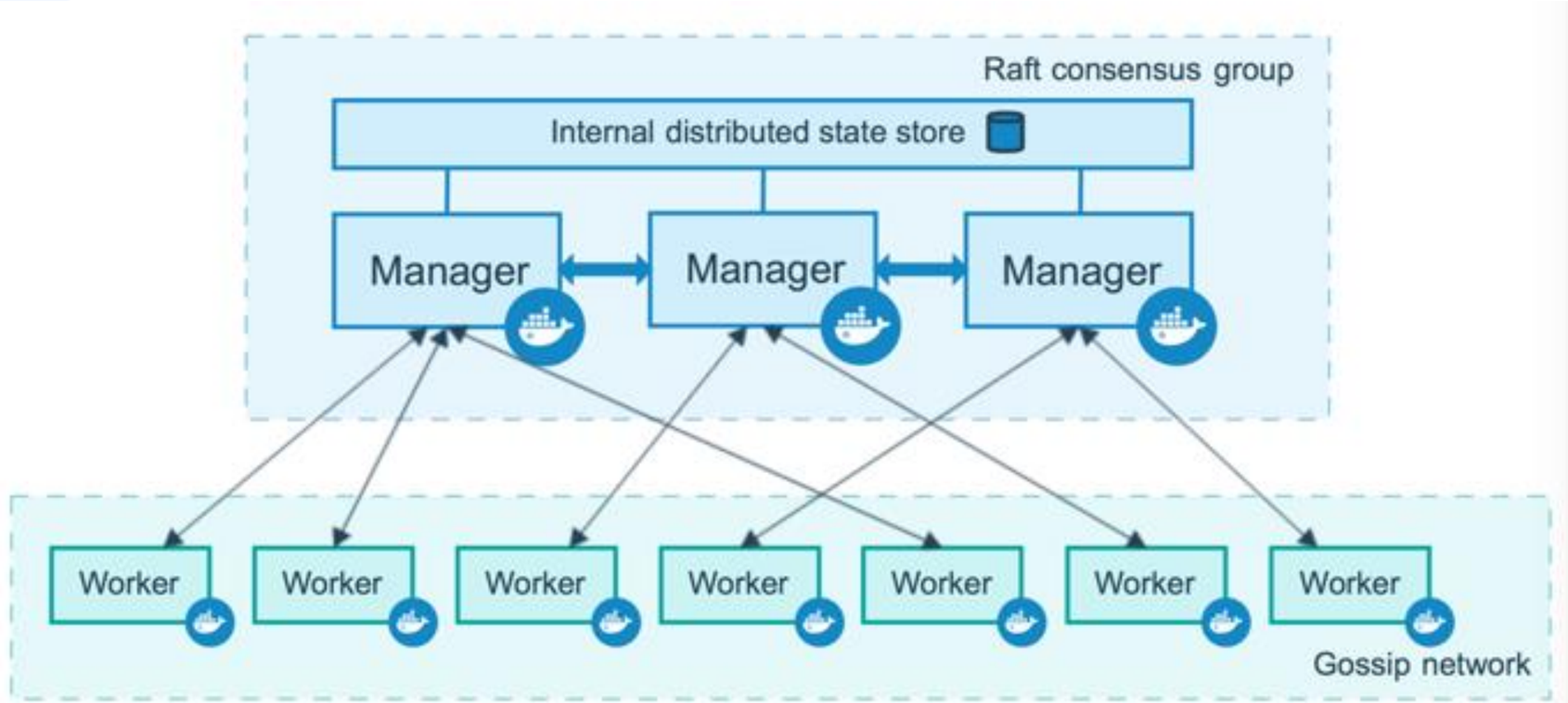
Worker Node: Scheduler



How to stop scheduler from placing tasks on manager node in a multi-node swarm:

- Manager node availability must be set to Drain
  - Result:
    - Tasks are stopped on nodes by the scheduler.
    - Tasks are scheduled on an Active node by the scheduler.
    - New tasks are not assigned to nodes by the scheduler.

How Does a Swarm Node Work?



- Swarm mode helps in creating one or more Docker Engine's cluster.
- These clusters of Docker Engine are known as the swarm.
- Swarm contains one or more than one nodes.

Source: https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

Docker Swarm: Set Up

Users can follow these general steps to install Docker swarm:

1. Pre-requisite: Install Docker
2. Initialize swarm
3. Add worker nodes
4. Check nodes

Docker Swarm: Set Up

Pre-requisite: Install Docker

You can install Docker using package managers or by downloading it directly from the Docker website.

Initialize Swarm

Run the **docker swarm init** command on the manager node to initialize the Swarm.

Add worker nodes

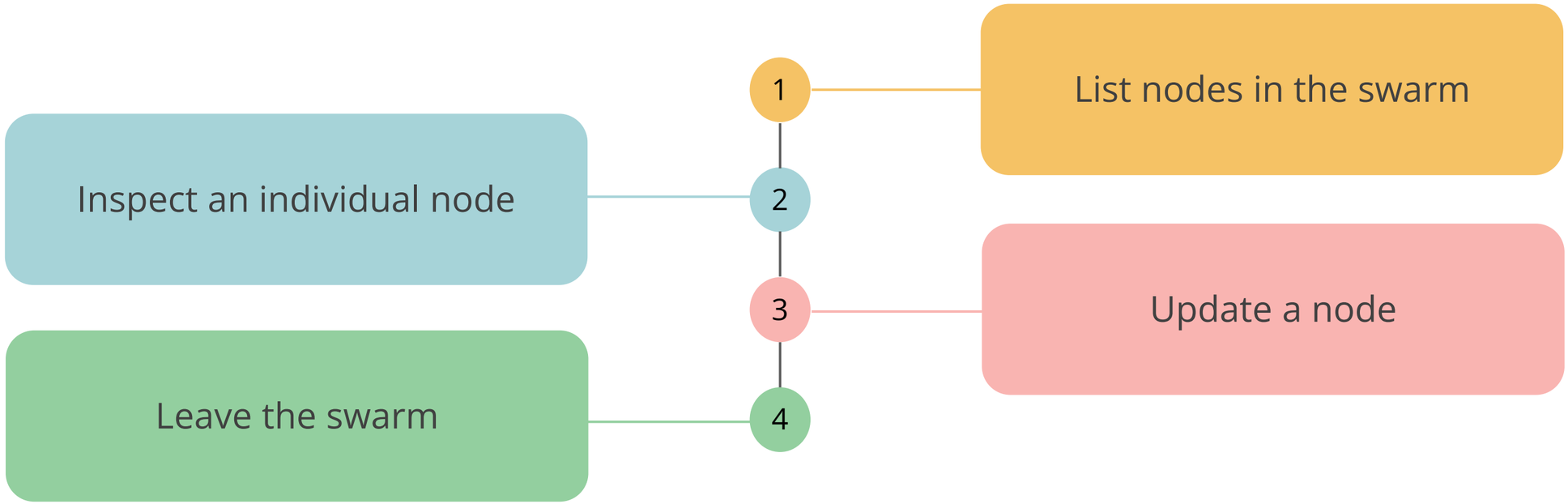
Run the **docker swarm join** command on each worker node to add worker nodes to the cluster.

Check nodes

Run the **docker node ls** command to verify that all nodes have successfully joined the Swarm cluster.

Managing Nodes in Docker Swarm

Users may need to do the following tasks as part of the swarm management life cycle:



Managing Nodes in Docker Swarm

List all Nodes in Swarm	docker node ls
Inspect an individual node	docker node inspect <NODE-ID>
Update a Node	docker node update --availability <availability> <NODE-ID>
Leave the Swarm	docker swarm leave

Assisted Practice

Setting up a Swarm Cluster with Managers and Worker Nodes **Duration: 10 Min.**

Problem statement:

You have been tasked with demonstrating the implementation of shared storage across Docker cluster nodes, which involves setting up a volume, replicating a service across nodes, and inspecting worker nodes for effective data management.

Outcome:

By completing this demo, you will configure a Docker Swarm cluster with a manager and worker nodes, showcasing distributed container management and node coordination.

Note: Refer to the demo document for detailed steps
01_Setting_Up_a_Swarm_Cluster_With_Managers_and_Worker_Nodes

Assisted Practice: Guidelines

Steps to be followed:

1. Initialize a swarm on a master node
2. Join the worker node in the swarm
3. List nodes in the swarm

ASSISTED PRACTICE

Quick Check

You are managing a Docker Swarm cluster and need to verify the status of all nodes, update the availability of one node to "drain," and finally remove a node from the swarm. Which commands should you use?

A. docker node ls, docker node update --availability drain <NODE-ID>, docker swarm leave

B. docker node inspect, docker node ls, docker swarm leave

C. docker swarm leave, docker node inspect, docker node update --availability active <NODE-ID>

D. docker node ls, docker node update --availability active <NODE-ID>, docker swarm leave

Quorum in a Swarm Cluster

Quorum: Overview

Quorum in a Docker Swarm cluster refers to the minimum number of manager nodes that must be available and in agreement for the cluster to remain operational.

- Quorum ensures cluster stability, fault tolerance, and effective decision-making processes.
- It is essential for tasks such as maintaining consensus, handling node additions or removals, and managing service deployments.
- Without quorum, the cluster may experience disruptions, loss of data integrity, and inability to perform critical operations.

Quorum of Managers

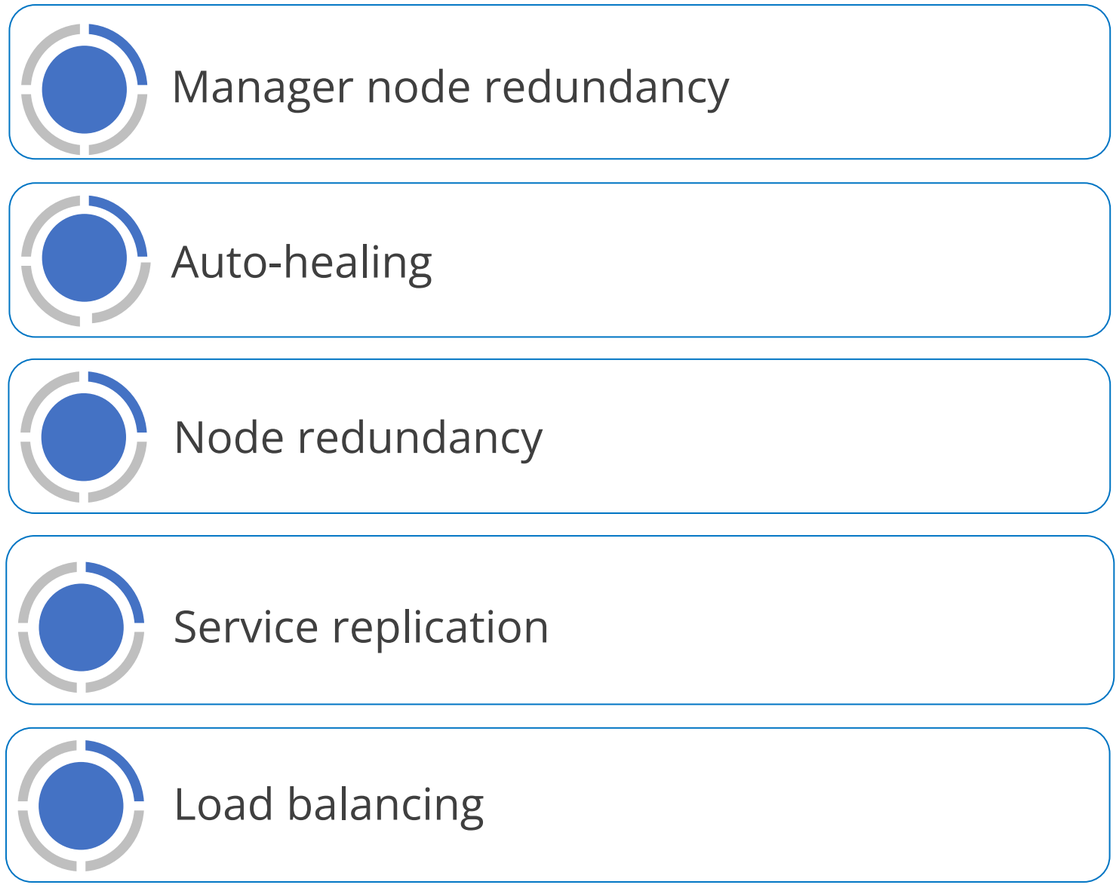
A quorum of managers in a Docker Swarm cluster refers to the minimum number of manager nodes required to maintain cluster stability and make critical decisions.

The specific number of managers required for a quorum depends on the total number of manager nodes (N) in the swarm. It follows the formula (N – 1)/2, which means:

- With 1 manager node, a quorum of 1 is always available (all managers participate).
- With 2 manager nodes, a quorum of 2 is required (both managers must be operational).
- With 3 or more manager nodes, a majority (greater than half) is needed for a quorum (e.g., 2 out of 3 managers, 3 out of 4 managers, and so on)

Fault Tolerance

Fault tolerance in Docker Swarm ensures the continued operation of containerized applications even in the face of failures. Here's how Docker Swarm achieves fault tolerance:



Fault Tolerance

Manager node redundancy

Docker Swarm recommends multiple manager nodes for fault tolerance; if one fails, others maintain the cluster's state.

Auto-healing

Docker Swarm's auto-healing feature restores the desired state if a container or node fails, ensuring fault tolerance.

Node redundancy

It is recommended to use an odd number of nodes in Docker Swarm for better fault tolerance and high availability.

Fault Tolerance

Service replication

Docker Swarm replicates services into tasks across the cluster, enhancing fault tolerance by distributing workload across multiple nodes.

Load balancing

Docker Swarm employs load balancing strategies to distribute incoming traffic evenly across nodes, improving fault tolerance by preventing overloading on any single node.

Loss of Quorum: Causes

Quorum loss occurs when the number of manager nodes drops below the threshold required for consensus, which typically means most of the manager nodes must be present..

Manual configuration errors

Node failures

Security incidents

Software bugs

Quorum Recovery

It refers to the process of restoring the required quorum of manager nodes in a Docker Swarm cluster after it has been compromised or lost.

- To recover from quorum loss, failed manager nodes need to be brought back online.
- If this is not possible, the **docker swarm init** command can be run with the **--force-new-cluster** flag to reinitialize the cluster with the available nodes.

Quick Check

Your Docker Swarm cluster needs to ensure high availability and continued operation during node failures. Which combination of features would help maintain service uptime in such a scenario?

A. Service replication and load balancing

B. Manager node redundancy and node redundancy

C. Auto-healing and load balancing

D. All of the above

Deploying Services in a Docker Swarm

Docker Swarm Services: Types

Replicated services

- These are defined with a specified number of replica tasks.
- These services provide fault tolerance and load distribution by distributing tasks across nodes.

Global services

- These ensure that a single task of the service runs on every available node in the Swarm.
- These are useful for deploying monitoring agents or any other system-level service on each node.

Creating Docker Swarm Services

Docker Swarm services creation involves deploying containerized applications within a Docker Swarm cluster. Below is an overview of how to create services creation:

Swarm initialization

Before creating services, initialize a Docker Swarm cluster using the **docker swarm init** command on a manager node.

Declarative model

Swarm services utilize a declarative model where you define the desired state of the service, such as the container image, replicas, network configuration, and more.

Service deployment

This can be done using the docker service create command where the service name, desired number of replicas, and container images are specified.

Scaling Services for High Availability

Scaling services for high availability involves designing systems that can maintain continuous operation even in the face of failures. Here is how services can be scaled:

Load balancing

Implement load balancers to distribute incoming traffic across multiple servers to prevent overload.

Horizontal scaling

Scale services horizontally by adding more instances or nodes to handle increased demand.

Fault tolerance

Design redundant systems with failover mechanisms for continuous operation despite component failures.

Automated scaling

Utilize auto-scaling solutions to dynamically adjust resources based on demand and performance.

Manipulating Services

Following commands are used to manipulate the running services in a stack:

Command	Description
docker stack deploy	Deploys a new stack or updates an existing stack
docker stack ls	Lists stacks
docker stack ps	Lists the tasks in the stack
docker stack rm	Removes one or more stacks
docker stack services	Lists the services in the stack

Scaling the Service

Command:

docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]

A parent command which manage service

Scales the replicated services to the desired number of replicas

Option	Description
--detach , -d	This does not wait for the service to converge and exit immediately

Assisted Practice

Running Replicated and Global Services **Duration: 10 Min.**

Problem statement:

You are tasked with demonstrating the use of shared storage across Docker cluster nodes by setting up a volume, replicating a service, and inspecting worker nodes for efficient data management.

Outcome:

By completing this demo, you will successfully demonstrate Docker's capacity to manage both replicated and global services within a swarm cluster. This exercise will illustrate the robust scalability and distribution capabilities of Docker, ensuring effective service management across multiple nodes.

Note: Refer to the demo document for detailed steps
02_Running_Replicated_and_Global_Services

Assisted Practice: Guidelines

Steps to be followed:

1. Create a replicated service
2. Create a global service
3. List all Docker services
4. Check the status of global services
5. Check the status of replicated services

ASSISTED PRACTICE

Assisted Practice

Running a Container into Services Running Under Swarm **Duration: 10 Min.**

Problem statement:

You are tasked with showcasing the implementation of shared storage across Docker cluster nodes. This involves setting up a shared volume, replicating a service using this volume across various nodes, and conducting inspections of the worker nodes to ensure effective data management.

Outcome:

By completing this demo, you will showcase Docker's ability to manage replicated and global services within a swarm cluster, highlighting its scalability and effective multi-node service management.

Note: Refer to the demo document for detailed steps
03_Running_a_Container_into_Services_Running_Under_Swarm

Assisted Practice: Guidelines

Steps to be followed:

1. Deploy the nginx service in global mode
2. Create nginx services with replicas
3. Check the container status

ASSISTED PRACTICE

Quick Check

Your web application experiences sudden traffic spikes and occasional server failures. Which combination of techniques would best ensure high availability and performance?

A. Horizontal scaling and fault tolerance

B. Load balancing and automated scaling

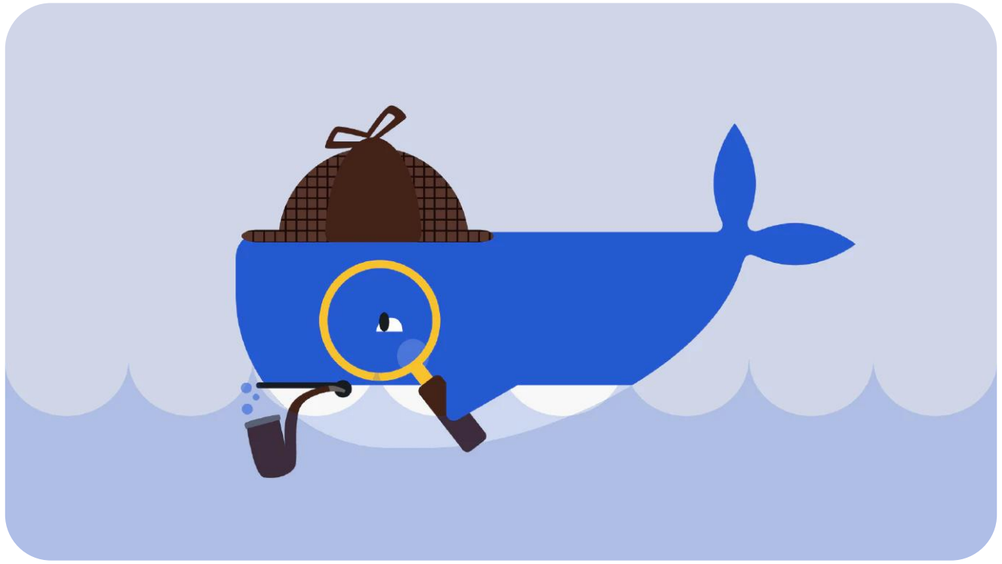
C. Fault tolerance and load balancing

D. Horizontal scaling and automated scaling

Docker Inspect

Docker Inspect: Overview

It is a powerful command-line tool that provides detailed, low-level information about Docker objects such as containers, images, networks, volumes, and more.



It offers insights into various aspects of Docker components, including configuration details, metadata, and runtime data.

Source: https://www.kosli.com/blog/docker-inspect-explained-the-essential-guide/

Docker Inspect: Features

01 Users can examine configuration settings like base image, volume mounts, and port mappings using Docker Inspect.

02 Docker Inspect offers experimental features for testing and feedback, although their functionality or design may change between releases.

03 It assists in analyzing and troubleshooting Docker environments by providing detailed information about containers, images, networks, and volumes.

Command: Docker Inspect

docker inspect → This command provides detailed information on various constructs controlled by the Docker. The result of this command reflects in a JSON array.

Command:

docker inspect [OPTIONS] NAME|ID [NAME|ID...]

It's the base command for Docker CLI

Options	Description
--format, -f	An output is formatted using GO template
--size, -s	File sizes are displayed if the type is a container
--type, -t	Return JSON for a specified type

Docker Inspect: Examples

Fetch IP address of an instance:

```
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $INSTANCE_ID
```

Fetch log path of an instance:

```
$ docker inspect --format='{{.LogPath}}' $INSTANCE_ID
```

Fetch image name of an instance:

```
$ docker inspect --format='{{.Config.Image}}' $INSTANCE_ID
```

Assisted Practice

Inspecting a Service on Swarm **Duration: 10 Min.**

Problem statement:

You are tasked with demonstrating the inspection process of a service within a Docker swarm, including deploying a replicated Redis service and monitoring task distribution.

Outcome:

By completing this demo, you will effectively inspect and monitor a Redis service in a Docker swarm, gaining detailed insights into task distribution and service configurations.

Note: Refer to the demo document for detailed steps
04_Inspecting_a_Service_on_Swarm

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Deploy a replicated Redis service
2. Inspect the service on the Docker swarm

ASSISTED PRACTICE

Docker Compose

Docker Compose: Overview

Docker Compose simplifies managing multi-container Docker applications by enabling developers to specify services, networks, and volumes in a single YAML configuration file.

With Docker Compose, users can easily deploy and manage complex applications consisting of multiple interconnected containers.

Docker Compose: Benefits

- Docker Compose configuration files are easy to share, facilitating collaboration among developers, operations teams, and other stakeholders.
- It supports variables in the Compose file, which can be used to customize your composition for different environments or different users.
- It simplifies the complex task of orchestrating and coordinating various services, making it easier to manage and replicate your application environment.

Docker Compose Configuration

Docker Compose simplifies managing multi-container applications through the following configuration capabilities:

Compose file

Configuration is defined in a YAML file, typically named docker-compose.yml, which specifies services, networks, and volumes required for the application stack.

Service configuration

Each service is configured with parameters such as image, ports, environment variables, and dependencies.

Networks

Custom networks are defined for communication between services, enabling isolation and security.

Docker Compose Configuration

Volumes

Volumes are specified to preserve data generated by containers, ensuring data integrity and persistence.

Environment variables

Variables are set for services, facilitating configuration and runtime behavior customization.

Extensibility

Compose allows extending of configurations with additional files, enabling modular and reusable configuration setups.

Compose: Overview

Compose is great for:

- Development environment
- Testing environment
- Staging environment
- CI workflow

The three-step process that is required to use the Compose:

Define the environment of the app with a Dockerfile in order to reproduce it anywhere.

↓

Define services that are part of an app in the *docker-compose.yml* in order to run them together in an isolated environment.

↓

Run the command *docker-compose up.* This command will start the Compose and run the entire app.

Compose: Overview

A *docker-compose.yml* looks like this:

```
version: '3’
services:
    web:
        build: .
        ports:
        - "5000:5000"
        volumes:
        - .:/code
        - logvolume01:/var/log
        links: - redis
    redis:
        image: redis
    volumes:
        logvolume01: {}
```

Docker Stack Deploy

It is a command used in Docker Swarm mode to deploy and manage application stacks defined in a compose file.

- The deploy command is used to deploy or update a stack from a Compose file on the Docker Swarm. It is executed on a Swarm manager node.
- Docker Stack Deploy simplifies the deployment process by automating the creation and management of application stacks, making it easier to deploy complex multi-container applications.

Docker Stack Deploy

Command *docker stack deploy [OPTIONS] STACK* deploys a new stack or updates an existing stack.

Option	Description
--bundle-file	Provides a path to a Distributed Application Bundle file
--compose-file , -c	Provides a path to a Compose file, or “-“ to read from stdin
--namespace	Allows to use Kubernetes namespace
--prune	Prunes services that are no longer referenced
--resolve-image	Queries the registry in order to resolve the image digest and supported platforms (“always”|”changed”|”never”)
--with-registry-auth	Sends registry authentication details to Swarm agents
--kubeconfig	Provides a Kubernetes config file
--orchestrator	Allows the Orchestrator to use (swarm|kubernetes|all)

Note: In order to run this command, the client and daemon API must be of version 1.25.

Example

Create services that form the stack using command *docker stack deploy:*

Command:

$ docker stack deploy --compose-file docker-compose.yml vossibility

It’s the name of the Compose file.

Output:

```
Creating network vossibility_vossibility
Creating network vossibility_default
Creating service vossibility_nsqd
Creating service vossibility_logstash
Creating service vossibility_elasticsearch
Creating service vossibility_kibana
Creating service vossibility_ghollector
Creating service vossibility_lookupd
```

An output like this tells that the service has been deployed

Assisted Practice

Converting an Application Deployment into Stack **Duration: 10 Min.**

Problem statement:

You are tasked with transitioning an application deployment into a Docker stack to enhance scalability and manageability within a Docker swarm environment.

Outcome:

By completing this demo, you will successfully convert an application deployment into a Docker stack, demonstrating scalable and manageable deployments using Docker Compose and swarm features.

Note: Refer to the demo document for detailed steps
05_Converting_an_Application_Deployment_into_Stack

Assisted Practice: Guidelines

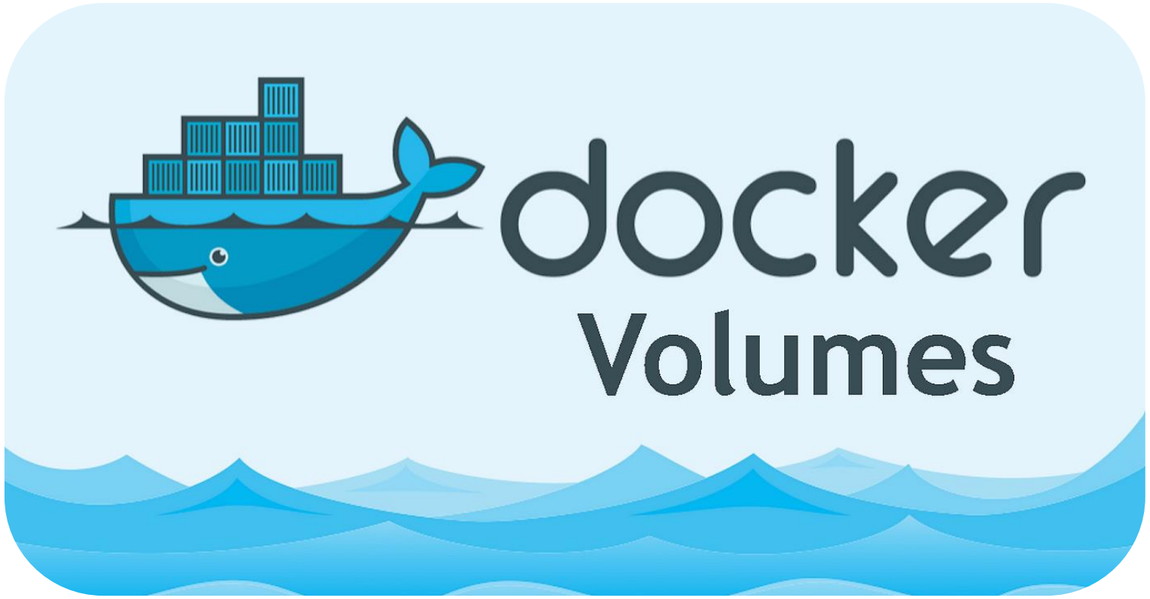
Steps to be followed:

1. Drain worker nodes
2. Start, check, and verify the registry service
3. Create a flask application
4. Create a requirement file
5. Create a Dockerfile and define the docker-compose configuration
6. Install docker-compose and start the application
7. Stop the application and push it to the registry
8. Deploy the stack and check its status
9. Test the application and remove the stack

Volumes and Mounts

Volumes in Orchestration

These refer to the mechanism of managing persistent data storage for containers within a distributed system orchestrated by a container orchestrator like Docker Swarm.



Orchestration services integrate with storage solutions to provide scalable and resilient storage volumes for containerized applications.

Bind Mounts

Bind mounts in orchestration provide a way to share directories from the host's file system into containers, allowing for flexible data management across a distributed system.

- Docker Swarm, a container orchestration tool, supports bind mounts for managing containerized applications.
- Each container can have its own bind mount, facilitating data access and configuration flexibility across the swarm.
- During development, bind mounts enable developers to dynamically modify code or configurations on the host machine, instantly reflecting changes within the container environment.

Advantages of Volumes over Bind Mounts

Following are the benefits of using Volumes over Bind mounts:

- Easier management
- Portability
- Backup and migration

Volumes are easier to manage than bind mounts, especially when dealing with multiple containers or complex data storage setups.

Advantages of Volumes over Bind Mounts

- Easier management
- Portability
- Backup and migration

Volumes are more portable, making it simpler to move containers between different environments without worrying about host file system specifics.

Advantages of Volumes over Bind Mounts

- Easier management
- Portability
- Backup and migration

Volumes are easier to back up or migrate compared to bind mounts, offering better data persistence and recovery options.

Assisted Practice

Mounting Volumes via Swarm Services **Duration: 10 Min.**

Problem statement:

You are tasked with demonstrating the mounting of volumes and data management within Docker containers to enable persistent and temporary storage solutions.

Outcome:

By completing this demo, you will effectively manage data storage across Docker containers, highlighting persistent and temporary storage capabilities.

Note: Refer to the demo document for detailed steps
06_Mounting_Volumes_via_Swarm_Services

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Mount volumes in Docker
2. Manage data and temporary file systems within Docker containers

ASSISTED PRACTICE

Quick Check

You need to ensure persistent data storage for a service running in a Docker Swarm, so that data remains intact even if the container is restarted. Which approach should you use to achieve this?

A. Use a bind mount to link the container to a directory on the host machine.

B. Store data directly within the container’s filesystem.

C. Mount a Docker volume to the service in the Swarm.

D. Save data to a temporary directory within the container.

Templates and Logs

Templates: Overview

Templates in Docker orchestration refer to predefined configurations or specifications used to deploy and manage containerized applications efficiently.

- Templates typically include information such as container images, resource requirements, networking configurations, environment variables, and dependencies.
- They allow for automation of deployment tasks, simplifying the management of complex containerized applications.
- Docker Swarm utilize templates to streamline the deployment process, ensuring consistency and scalability across distributed environments.

Usage of Templates

Here is how templates are used:

Automation

Templates enable automation of deployment tasks by specifying the desired state of the infrastructure, allow users to define configurations once.

Customization

Users can customize templates by using placeholders or variables to define values specific to their environment.

Scalability

Templates facilitate the scaling of applications by providing a standardized approach to deploying additional resources as needed.

Templates

Using the syntax provided by Go's template package, the templates are used for the following flags of *service create*:

- *--hostname*
- *--mount*
- *--env*

Placeholder	Description
.Service.ID	Service ID
.Service.Name	Service Name
.Service.Labels	Service Labels
.Node.ID	Node ID
.Node.Hostname	Node Hostname
.Task.ID	Task ID
.Task.Name	Task Name
.Task.Slot	Task Slot

The table above lists the description of different placeholders that are valid for Go template.

Assisted Practice

Using Templates with Docker Create Service **Duration: 10 Min.**

Problem statement:

You are tasked with demonstrating Docker's capability to use templates in service creation, configuring dynamic container attributes like hostname based on service name, node ID, and other parameters

Outcome:

By completing this demo, you will effectively showcase how to use templates in Docker services to dynamically configure and manage container attributes, enhancing deployment efficiency.

Note: Refer to the demo document for detailed steps
07_Using_Templates_with_Docker_Create_Service

Assisted Practice: Guidelines

Steps to be followed:

1. Set a container template
2. Check the service status
3. Inspect the service

Logs: Overview

Logs are essential in Docker orchestration for monitoring, troubleshooting, and maintaining containerized applications.

- Docker logs are valuable for debugging and troubleshooting, offering insights into application errors, warnings, and events.
- Logs are often streamed in real-time, providing operators and developers instant visibility into containerized applications.

Accessing Logs

To access logs in Docker, you can use the following methods:

Docker logs command

Docker's docker logs command retrieves logs from running containers. Use docker logs <container_name> to access logs from a specific container.

Logging mechanism

Docker stores container logs as JSON files but can be configured to forward them to external log management systems for centralized logging.

Standard output or error

Docker treats any output to standard streams in a container as logs, accessible via docker logs or by inspecting runtime information.

Logs Commands

Command	Description
docker logs	This command shows information logged by a running container.
docker service logs	This command shows information logged by all containers participating in a service.

Three streams of the Unix/Linux commands:

Stream	Description
STDIN	A file handle, which is read by the process to get information
STDOUT	A file handle on which information is written by the process
STDERR	A file handle on which the error information is written by the process

Logs Commands

The following are the cases when the log command does not show useful information:

- The Docker logs may not show useful information when logging drivers are used to send logs.
- The application may send its output to the log files instead of *STDOUT* and *STDERR* when the image runs a non-interactive process.

Accessing Logs

Log location on different operating systems:

Operating system	Location/Command
RHEL, Oracle Linux	/var/log/messages
Debian	/var/log/daemon.log
Ubuntu 16.04+, CentOS	Command: journalctl -u docker.service
Ubuntu 14.10-	/var/log/upstart/docker.log
macOS (Docker 18.01+)	~/Library/Containers/com.docker.docker/Data/vms/0/console-ring
macOS (Docker <18.01)	~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/console-ring
Windows	AppData\Local

Troubleshooting Services

Reasons for a service to go into a pending state:

- When all the nodes are drained
- When constraints are applied on the nodes
- When all the nodes in a swarm do not have enough memory to perform tasks

Steps for troubleshooting a service:

- Find all services using *docker service ls.*
- Check the service that is in pending state by using *docker service ps troub. troub* is the name of the service.
- Inspect the service tasks by using *docker inspect*.
- Find “Err” under “Status” to know the reason behind the pending state of the service.

Network Debugging

Network debugging:

Command:

docker network inspect [OPTIONS] NETWORK [NETWORK...]

netshoot container:

The *netshoot* container contains network troubleshooting tools to resolve Docker networking issues.

The *netshoot* container is able to:

- Get attached to any network
- Get placed in any network namespace of any container

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- Host’s Network Namespace
- Network’s Network Namespace

Problem: Networking issues with the application's container

Solution: Launch *netshoot* by using command

$ docker run -it --net container:<container_name> nicolaka/netshoot

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- **Host's Network Namespace**
- Network's Network Namespace

Problem: Networking issues on the host

Solution: Launch *netshoot* by using command

$ docker run -it --net host nicolaka/netshoot

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- Host’s Network Namespace
- Network’s Network Namespace

Problem: Networking issues on Docker network

Solution: Use *nsenter*

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- Host’s Network Namespace
- Network’s Network Namespace

Command for launching *netshoot* in privileged mode:

docker run -it --rm -v /var/run/docker/netns:/var/run/docker/netns --privileged=true nicolaka/netshoot

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- Host’s Network Namespace
- Network’s Network Namespace

Command for getting inside the network namespace:

nsenter –net /var/run/docker/netns/<networkid> sh

Network Troubleshooting

Troubleshooting using *netshoot*:

- Container's Network Namespace
- Host’s Network Namespace
- Network’s Network Namespace

Commands for assessing network after getting inside the network namespace:

ipconfig

brctl show

ip route show

ip –s neighbor show

ip –d link show

bridge fdb show

Assisted Practice

Troubleshooting an Undeployable Docker Service **Duration: 10 Min.**

Problem statement:

You are tasked with troubleshooting an undeployable Docker service, focusing on listing, inspecting, and diagnosing service tasks to identify and resolve issues affecting deployment.

Outcome:

By completing this demo, you will successfully diagnose and resolve deployment issues within a Docker service, ensuring smooth operation and optimal service availability.

Note: Refer to the demo document for detailed steps
08_Troubleshooting_an_Undeployable_Docker_Service

Assisted Practice: Guidelines

Steps to be followed:

1. List and inspect the Docker services
2. Inspect and diagnose service tasks

ASSISTED PRACTICE

Quick Check

Your service has suddenly gone to the pending state. As a DevOps engineer, you are troubleshooting the issue. Which of the following can be a reason for this?

A. Applied contraints

B. Image lost

C. Docker host is down

D. Storage removed

Key Takeaways

- Orchestrators like Kubernetes and Docker Swarm automate the deployment process, ensuring that containers are started, stopped, and replicated as needed.
- Docker Engine introduces swarm mode that enables you to create a cluster of one or more Docker Engines called a swarm.
- Incorrect or unintended changes to the swarm's configuration can reduce the number of active manager nodes, leading to quorum loss.
- Volumes enhance portability by ensuring that data can easily be moved or replicated across different environments without depending on the specific directory structure of the host machine.
- A service may go into a pending state if all nodes are drained, constraints are applied, or if nodes lack sufficient memory.

Creating a Docker Image and Replicated Service on a Swarm

Duration: 25 Min.

Project agenda: To create a Docker image, push it to Docker Hub, and deploy a replicated service on a Swarm cluster to ensure high availability and simplifying the management of application updates

Description: Your company is experiencing a need for scalable and reliable deployment of its applications. To address this, you are undertaking a project to implement Docker containerization and orchestration using Docker Swarm, facilitating efficient management and deployment of services across multiple nodes.

LESSON-END PROJECT

Creating a Docker Image and Replicated Service on a Swarm

Duration: 25 Min.

Perform the following:

1. Create a Dockerfile
2. Tag the image
3. Push the image to Docker Hub
4. Create a container
5. Check nodes in Swarm where the manager node is running
6. Create a service and check the default container

Expected deliverables: Dockerized application managed on a Swarm cluster

LESSON-END PROJECT

Thank You