Get the best out of Live Sessions

HOW?

e!

Check your Internet Connection

Log in 10 mins before, and check your internet connection to avoid any network issues during the LIVE session.

Speak with the Instructor

By default, you will be on mute to avoid any background noise. However, if required you will be **unmuted by instructor**.

Clear Your Doubts

Feel free to clear your doubts. Use the "**Questions**" tab on your webinar tool to interact with the instructor at any point during the class.

Let us know if you liked our content

Please share feedback after each class. It will help us to enhance your learning experience.

edureka!

edureka!

DevOps Certification Training

COURSE OUTLINE
MODULE 07

1. Introduction to DevOps
2. Version Control with Git
3. Git and Jenkins
4. Continuous Integration with Jenkins
5. Configuration Management using Ansible
6. Containerization using Docker Part - I
7. Containerization using Docker Part - II
8. Container Orchestration Using Kubernetes Part-I
9. Container Orchestration Using Kubernetes Part-II
10. Monitoring Using Prometheus and Grafana
11. Provisioning Infrastructure using Terraform Part - I
12. Provisioning Infrastructure using Terraform Part - II

edureka!

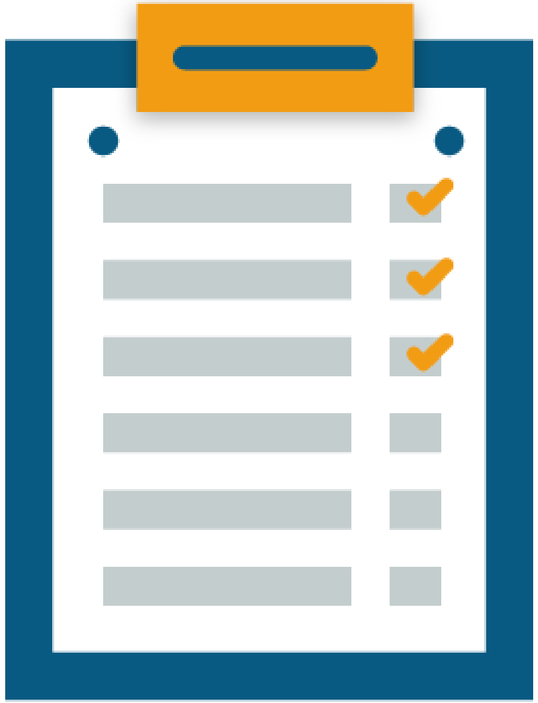
Module 7 – Containerization using Docker Part - II

edureka!

Topics

Following are the topics covered in this module:

- Docker Registry
- Container Storage
- Volumes
- Docker Compose
- Docker Swarm



Objectives

After completing this module, you should be able to:

- Use Docker Hub to store custom Images
- Store data in Container Volumes for persistent storage
- Setup Docker Compose
- Deploy a multi-container application using Docker Compose
- Deploy a Swarm Cluster

edureka!

Managing Docker Images

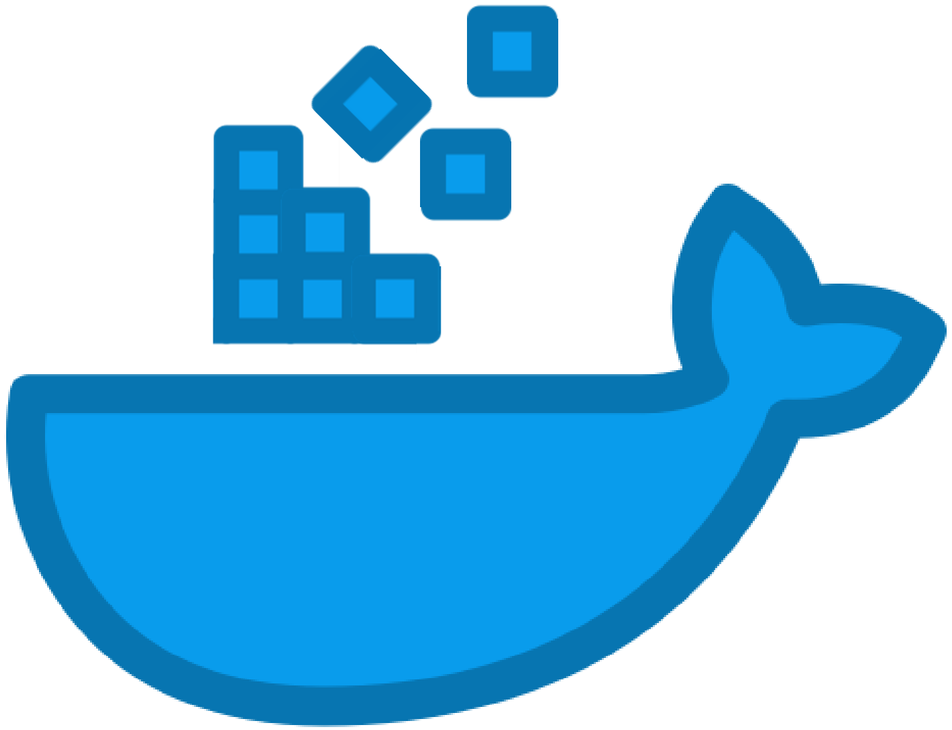
edureka!

Managing Docker Images

- Docker images are portable and can be distributed amongst the entire organization making it very accessible
- The easiest way to make these images available to others is by using a Docker registry

Docker Registry

- Docker Registry is a part of the Docker ecosystem
- A registry holds named Docker images for content delivery and storage
- The registry can be configured by creating a new configuration in YAML (Yet Another Markup Language) format



Managing Docker Images

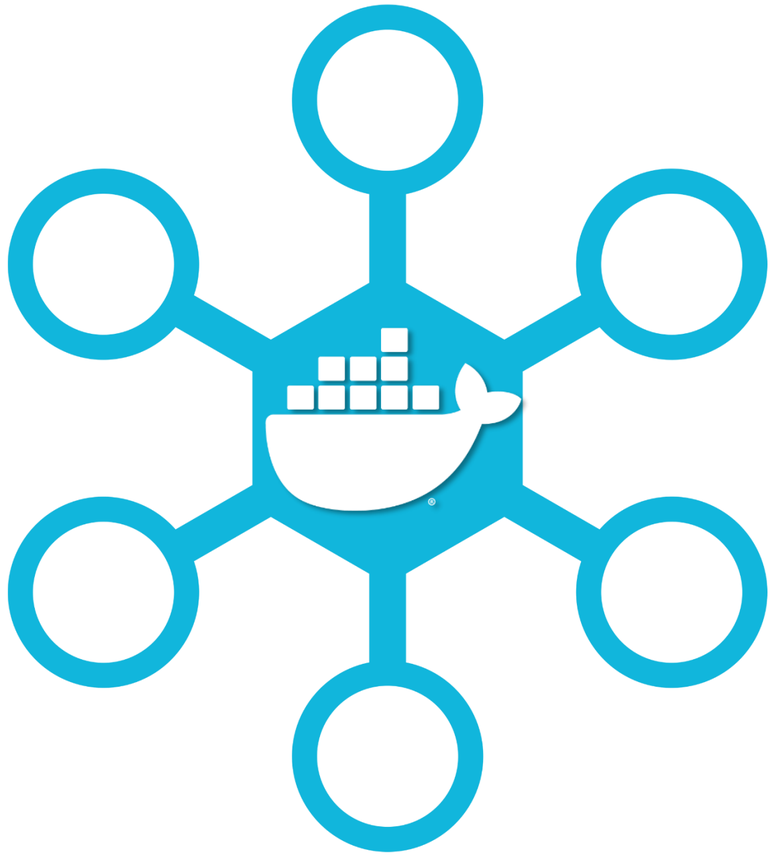
The images can be distributed using either of the following:

- Docker Hub (configured as default registry when docker is installed)
- Private registry
  - Quay.io
  - Google Container Registry
  - Artifactory
  - Amazon ECR Registry
  - Sonatype Nexus

Note: Docker Trusted Registry is taught in a separate module.

Docker Hub

- Docker Hub is a cloud-based docker registry
- It is a public repository for hosting, building and testing docker images
- It also provides a paid version which lets the user host private and team registries



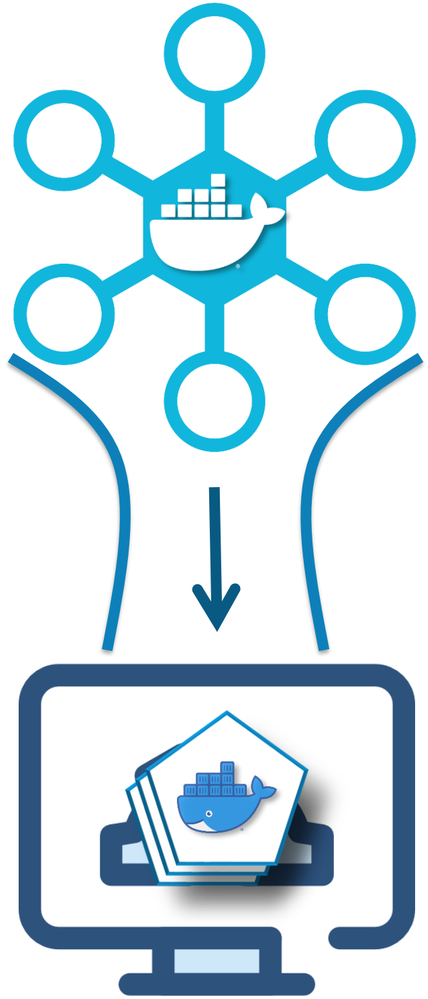
Pulling and Pushing Images

edureka!

Pulling Images from Docker Hub

When the `docker run` command is executed:

- Docker searches for the corresponding image on the local system
- If not found, Docker automatically pulls the image from the Docker Hub registry to create the container
- Pulling an image from a private repository requires authentication



```
# In case, the user wants to explicitly pull an image but
# not run it, the docker pull command can be used
            docker pull registryName/imageName
```

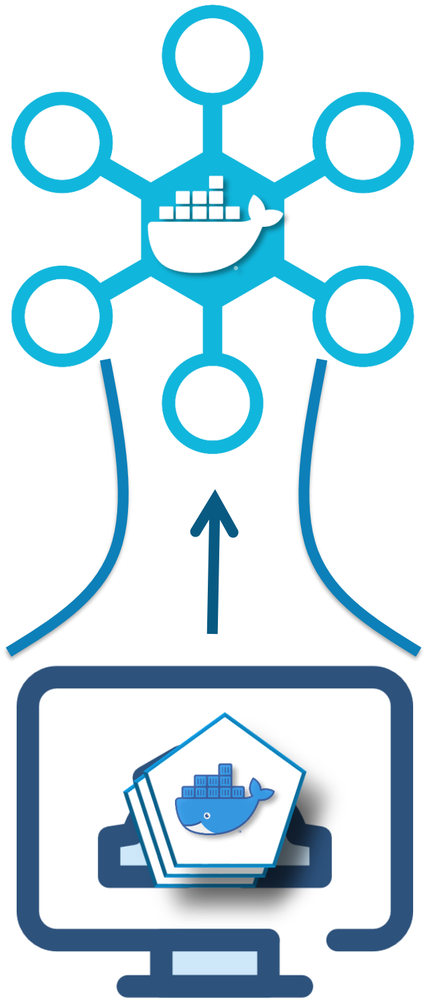
edureka!

Pushing Images to Docker Hub

In order to push images to Docker Hub:

- Create a Docker Hub account
- Create a new repository
- Login to the repo from Docker CLI and push the image

```
# Log into Docker Hub from CLI using:
  docker login --username=yourUsername --email=yourEmail
# This will prompt a password response. Enter it and your
# credentials will be saved
# To push a custom image to Docker Hub use:
                docker push username/imageName
```



Demo: Setting up Docker Hub

edureka!

Storage and Volumes in Docker

edureka!

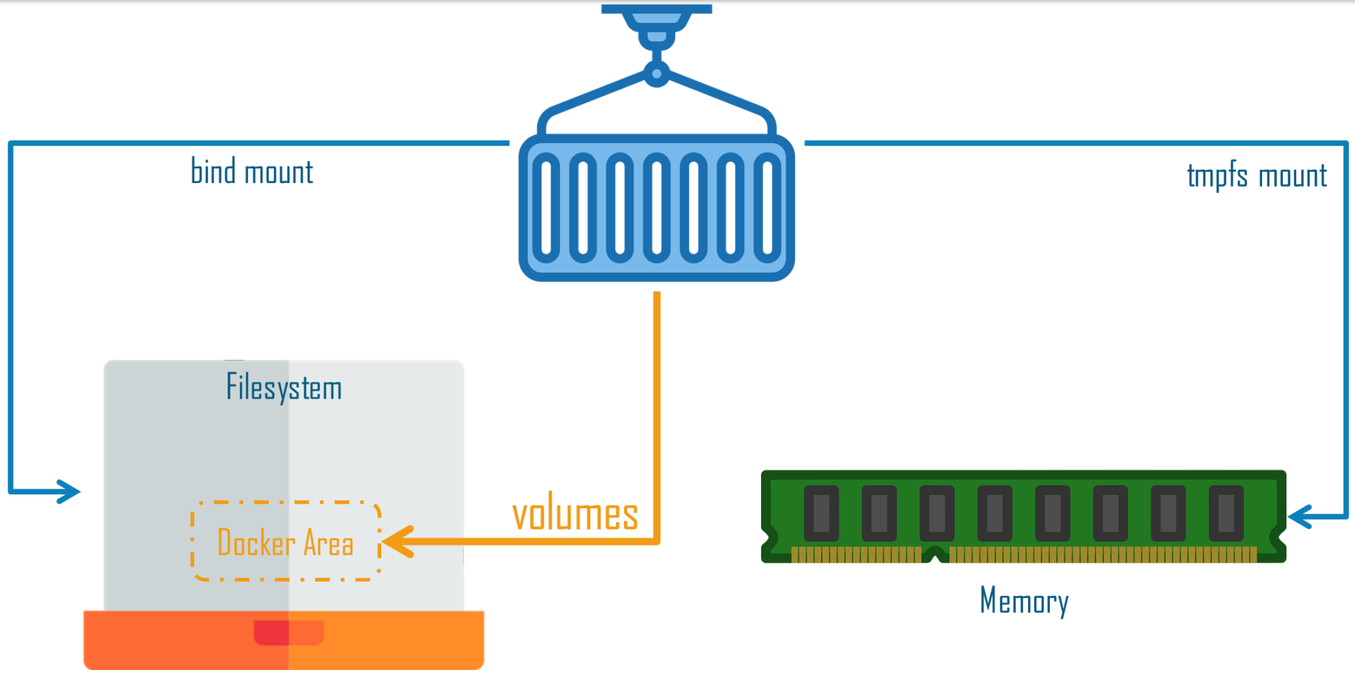
Storage in Docker

- Docker by default stores all its data on a writable layer inside the container itself
- This data is not persistent and is lost if the container no longer exists
- It is very difficult to extract data out of the container if an external application requires it
- Writing directly into containers requires a storage driver which reduces performance
- Storing data inside the container causes issues while replicating the application

edureka!

Volumes

- Volumes are the most reliable way to store persistent data for Docker containers
- Volumes can be easily managed using the Docker CLI Commands
- They can be easily shared amongst multiple containers



Creating and Manging Volumes

- Volumes can be created outside the scope of containers
- New volumes can also be pre-populated by existing containers

```
# To create a new volume
        docker volume create <volumeName>
# To list volumes
                docker volume ls
# To inspect the volume specifications
       docker volume inspect <volumeName>
# To remove a volume
          docker volume rm <volumeName>
```

Flag Behaviour: -v and --mount

-v Flag	--mount Flag
It consists of three separate fields separated by a colon ‘:’	Mount consists of multiple key-value pairs separated by a comma ‘,’
The first field denotes the volume name. It can be omitted in case of anonymous volumes	The order of the keys is not consequential.
The second field sets the path where the directory is mounted inside the container	Following keys can be defined: type, source, destination, readonly, volume-opt
The final field is optional and can be used to add options separated by a comma ‘,’	The volume-opt option can be specified multiple times

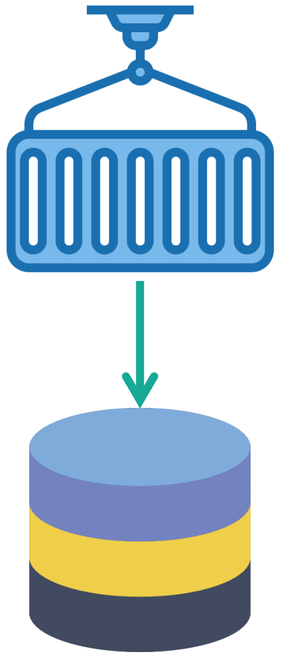
Note: All the options for volumes are available in both -v and --mount flags.

Starting Container with a Volume

- Containers can be assigned a volume when they are created
- In case a volume does not exist, Docker creates a new volume from scratch

```
# To start a container with a volume using --mount flag
docker run -d \
--name <containerName> \
--mount source=<volumeName>, dst=<containerPath> \
nginx:latest
```

```
# To start a container with a volume using -v flag
docker run -d \
--name <containerName> \
-v <volumeName>:<containerPath> \
nginx:latest
```



edureka!

Demo: Docker Volumes

edureka!

Docker Compose

edureka!

Docker Compose

- Compose is a Docker tool for running multi-container applications
- It uses a YAML format compose file to configure the application's services
- Using a command, the user can then create and start the services from the configurations

Docker Compose: Working

Compose works in three steps:

- Creating a `Dockerfile` for the application
- Define the services that make the application in `docker-compose.yml`
- Run `docker-compose up` to start the applications

Demo: Installing Docker Compose

edureka!

Multi-container Deployment with Compose

edureka!

Multi-container Deployment with Compose

- Docker Compose uses project names to isolate environments
- The data for all the containers in a service is preserved in volumes
- If a service is restarted but nothing has changed, Compose re-uses the existing containers
- Variables can be used in Compose files to customise the service for different environments

Compose Common Commands

Command	Description
`docker-compose build`	Builds or rebuilds a service from the given Dockerfile
`docker-compose run`	Allows user to run a one-off command in the service
`docker-compose up`	Creates and runs service containers
`Docker-compose down`	Removes the containers, networks, images, and volumes related to the service

Environment Variables in Compose

- Compose can utilize environment variables inside the compose file from the shell
- Environment variables can also be set in the containers by using the `docker run` command with `-e` flag
- Similarly, this can be accomplished using the `docker-compose run` command
- Default values for environment variables can also be set by creating a new `.env` file

Demo: Deploying a Multi-container Application using Compose

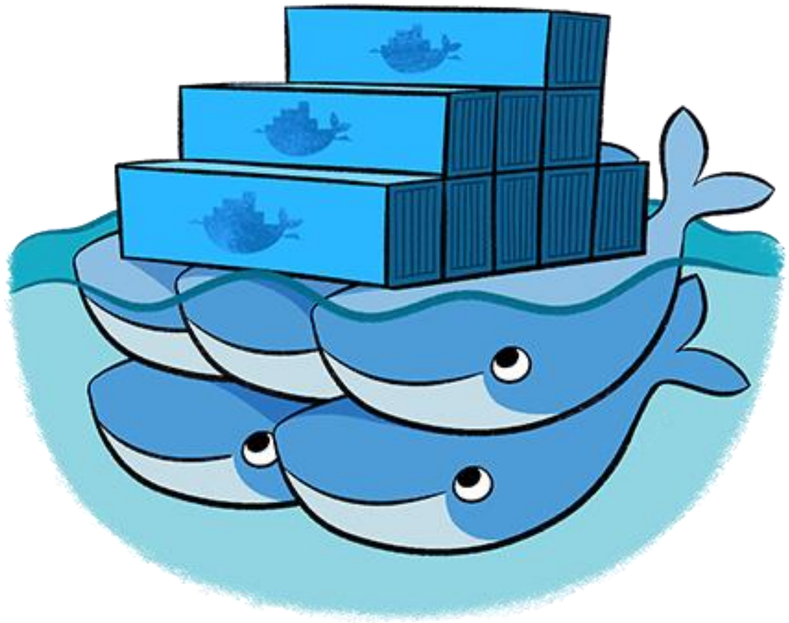
edureka!

Docker Swarm

edureka!

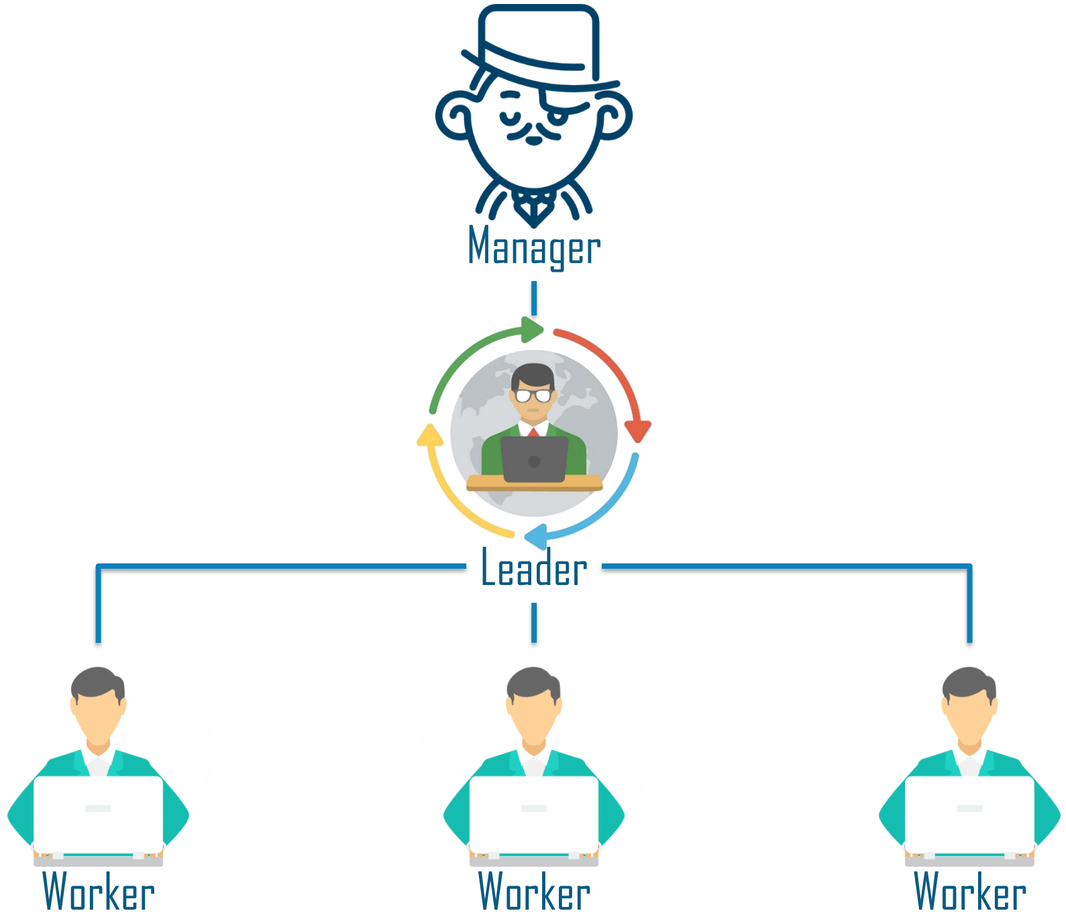
Docker Swarm

Swarm is a group of machines running Docker and configured to work together in a cluster. Docker swarm is a Container Orchestration tool that controls and operates on this cluster



Docker Swarm Nodes

Docker swarm comprises of three different types of Nodes:



Docker Swarm Nodes

Manager Node

The manager node is responsible for assigning tasks to the worker nodes. It also takes care of some of the management related tasks in a swarm

Leader Node

After a cluster is created, Docker assigns a leader node using the Raft consensus algorithm. Leader node does the management tasks and takes the orchestration related decisions

Worker Node

Worker nodes receive and executes the tasks allocated by the manager node

Docker Swarm Features

Integrated Orchestrator

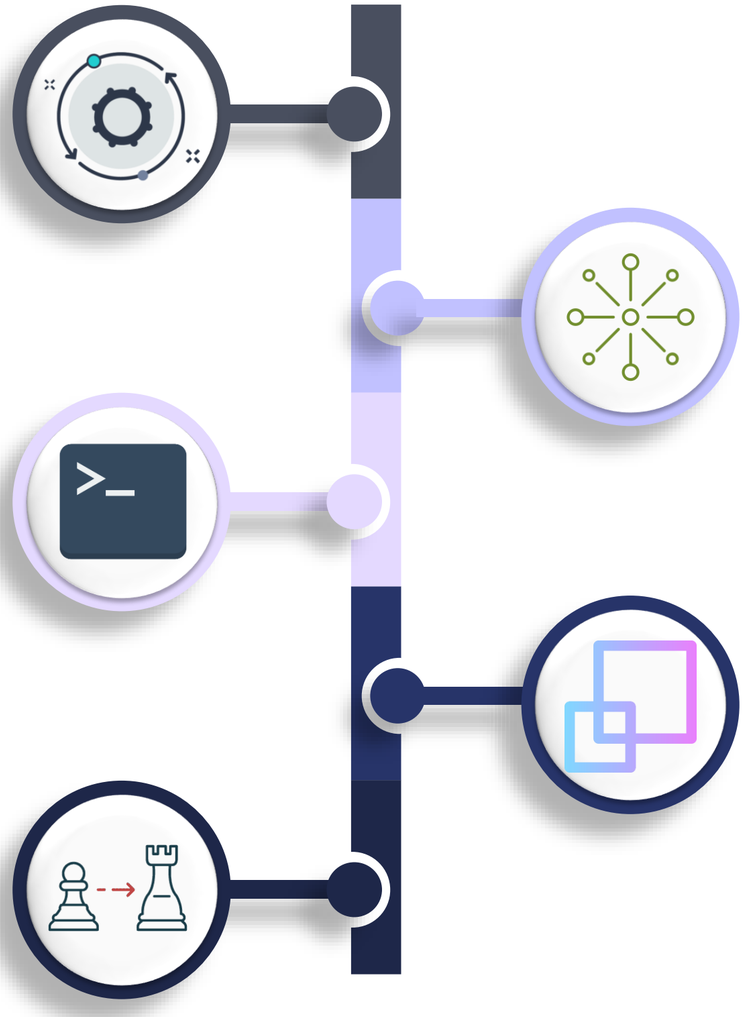
- Integrated within the Docker Engine
- New cluster can be spun up with few commands

Declarative Service Model

- The desired state of the services is defined using a declarative approach

State Reconciliation

- The state of the cluster is constantly monitored by the swarm manager
- It is responsible for reconciling the current state to the desired state



Decentralized

- The Docker Engine handles all specializations during runtime
- The entire swarm of nodes can be built from a single image

Scaling

- Number of tasks can be defined for each service
- Swarm manager automatically adapts when scaling the services

edureka!

Docker Swarm Features (Contd.)

Service Discovery

- Each service in the swarm is assigned a unique DNS
- Services can be queried inside the swarm cluster using the embedded DNS server

Security

- Swarm enforces nodes to authenticate communication through TLS certificates
- Users can also use self-signed root certificates

Network

- Overlay network can be assigned to the services in the cluster
- Swarm manager auto-assigns addresses to the containers during updates

Load Balancing

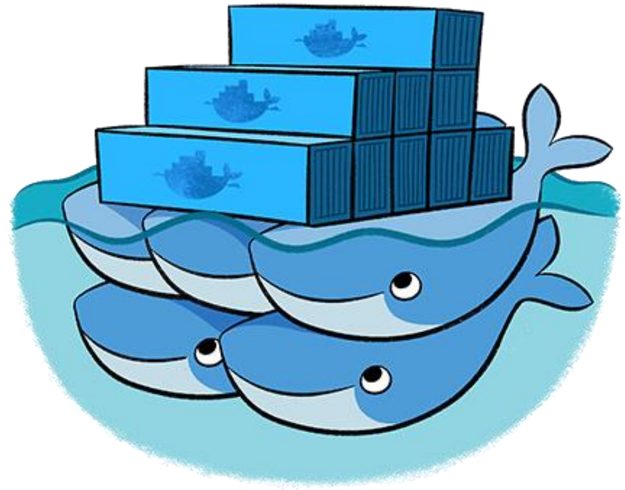
- An external load balancer can be used to expose the services
- Swarm also enables users to specify the distribution of services between nodes

Rolling Updates

- To ensure zero downtime swarm gives users control of deployment delays
- If something goes wrong, roll-back is always an option

Limitations of Docker Swarm

- No “Infrastructure as Code” philosophy:
  - Example: A user may execute a command for everything and the other team members will have no visibility or accountability
- No fail-safe implementation:
  - Example: In case, a service is scaled up to 500 replicas and the infrastrucutre only has 3 servers, it will actually crash the entire cluster



Demo: Running Docker in Swarm Mode

edureka!

Summary

Docker Registry

- Docker Registry is a part of the Docker ecosystem
- A registry holds named Docker images for content delivery and storage
- The registry can be configured by creating a new configuration in YAML (Yet Another Markup Language) format

edureka!

Docker Hub

- Docker Hub is a cloud-based docker registry
- It is a public repository for hosting, building and testing docker images
- It also provides a paid version which lets the user host private and team registries

edureka!

Storage in Docker

- Docker by default stores all its data on a writable layer inside the container itself
- This data is not persistent and is lost if the container no longer exists
- It is very difficult to extract data out of the container if an external application requires it
- Writing directly into containers requires a storage driver which reduces performance
- Storing data inside the container causes issues while replicating the application

edureka!

Docker Compose

- Compose is a Docker tool for running multi-container applications
- It uses a YAML format compose file to configure the application's services
- Using a command, the user can then create and start the services from the configurations

edureka!

Multi-container Deployment with Compose

- Docker Compose uses project names to isolate environments
- The data for all the containers in a service is preserved in volumes
- If a service is restarted but nothing has changed, Compose re-uses the existing containers
- Variables can be used in Compose files to customise the service for different environments

edureka!

Docker Swarm

Swarm is a group of machines running Docker and configured to work together in a cluster. Docker swarm is a Container Orchestration tool that controls and operates on this cluster

edureka!

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co