Docker is a containerization platform that allows to create, deploy, and run applications inside lightweight, portable containers.

- Consistency - Runs the same way on any environment (Dev, Test, Prod).
- Portability - Can run anywhere (cloud, server, Kubernetes).
- Efficiency - Uses fewer resources than Virtual Machines (VMs).
- Faster Deployment - No need to install dependencies separately.

Commands

1. Docker Basics

- [ ] Below commands provide fundamental information about the Docker installation, version, and help documentation.

- docker --version ---- Check Docker version
- docker info ---- Display system-wide information about Docker
- docker help ---- Show help for Docker commands

2. Working with Images

- [ ] Docker images are templates used to create containers. These commands help in searching, pulling, listing, tagging, and deleting images.

- docker images ---- List all downloaded images
- docker pull <image> ---- Download an image from Docker Hub
- docker search <image-name> ---- Search for images in Docker Hub
- docker rmi <image-id> ---- Remove an image
- docker tag <image-name>:<tag> <new-repository>:<tag> ---- Tag an image

3. Working with Containers

- [ ] A container is a running instance of a Docker image. These commands allow to start, stop, restart, remove, and interact with running containers.

- docker ps ---- List running containers
- docker ps -a ---- List all containers (including stopped ones)
- docker run <image> ---- Run a container from an image
- docker run -d <image> ---- Run a container in detached mode (background)
- docker run -it <image> /bin/bash ---- Run a container interactively
- docker start <container-id> ---- Start a stopped container
- docker stop <container-id> ---- Stop a running container
- docker restart <container-id> ---- Restart a container
- docker kill <container-id> ---- Forcefully stop a container
- docker rm <container-id> ---- Remove a container
- docker exec -it <container-id> /bin/bash ---- Access a running container's shell
- docker logs <container-id> or docker logs -f <container_id> ---- View container logs
- docker inspect <container-id> ---- Get details of a container

4. Docker Networking

- [ ] Docker allows containers to communicate with each other using networks. These commands help create, inspect, and manage Docker networks.

- docker network ls ---- List all networks
- docker network create <network-name> ---- Create a custom network
- docker network inspect <network-name> ---- Inspect network details
- docker network connect <network> <container> ---- Connect a container to a network
- docker network disconnect <network> <container> ---- Disconnect a container from a network
- docker network rm <network-name> ---- Remove a network

5. Docker Volumes (Persistent Storage)

- [ ] Containers are ephemeral, meaning data is lost when they stop. Volumes allow you to store data persistently, even after a container is removed.

- docker volume ls ---- List all volumes
- docker volume create <volume-name> ---- Create a volume
- docker volume inspect <volume-name> ---- Inspect a volume
- docker volume rm <volume-name> ---- Remove a volume
- docker run -d --name <container-name> -v <volume-name>:/data <image> ---- Mount Vol inside Container

6. Docker Compose

- [ ] Docker Compose is a tool for defining and running multi-container applications using a docker-compose.yml file. These commands help manage services in a Compose setup.

- docker compose up ---- Start all services in docker-compose.yml
- docker compose up -d ---- Start in detached mode
- docker compose down ---- Stop and remove containers
- docker compose ps ---- List running containers in the compose setup
- docker-compose logs ---- View logs for all services

7. Docker Build

- [ ] These commands are used to build custom Docker images from a Dockerfile, which contains instructions to create an image.

- docker build -t <image-name> . ---- Build an image from a Dockerfile
- docker build -f <Dockerfile> -t <image-name> . ---- Build an image using a specific Dockerfile

8. Docker Save and Load

- [ ] These commands allow you to export and import images as .tar files, useful for transferring images between systems.

- docker save -o myimage.tar <image-name> ---- Save an image as a .tar file
- docker load -i myimage.tar ---- Load an image from a .tar file

9. Docker Export and Import Containers

- [ ] Instead of saving an image, you can export a running container and later import it as an image.

- docker export -o mycontainer.tar <container-id> ---- Export a container
- docker import mycontainer.tar <new-image-name> ---- Import it as an image

10. Docker System Cleanup

- [ ] Docker can accumulate a lot of unused images, containers, networks, and volumes over time. These commands help clean up unnecessary resources.

- docker system prune -a ---- Remove (unused stopped) containers and data (images, networks)
- docker image prune -a ---- Remove unused images
- docker container prune ---- Remove stopped containers
- docker volume prune ---- Remove unused volumes

Docker Hub is a public container registry provided by Docker to store, manage, and distribute Docker images.

- Stores public and private Docker images
- Allows developers to share images
- Supports automated builds

Pull a Docker Image From DockerHub

- To pull an image from Docker Hub : docker pull <image_name>:<tag>

Push a Docker Image To DockerHub

- Step 1 : Log in to Docker Hub

  docker login ---- Enter your Docker Hub username and password.

- Step 2 : Tag the Image

  docker tag <local-image>:<tag> <docker-hub-username>/<repo-name>:<tag> (
  (docker tag nginx:latest SHIVAMNAIK****/nginx:latest)

- Step 3 : Push the Image

  docker push <docker-hub-username>/<repo-name>:<tag>
  (docker push SHIVAMNAIK****/nginx:latest)

AWS Elastic Container Registry (ECR) is Amazon's private container registry used to store, manage, and deploy Docker images securely in AWS.

- Secure - Stores images privately by default
- Integration - Works seamlessly with ECS, EKS, Lambda, and EC2
- Scalable - Handles large container deployments
- Fast - Uses Amazon CloudFront for image distribution

Pushing and Pulling Docker Images with AWS ECR

- Step 1 : AWS CLI Installation

  Install and configure AWS CLI (aws configure)

  Install Docker on your EC2 instance or local machine

  Provide required AWS IAM permissions to access ECR

- Step 2 : create a repository in AWS ECR to store your Docker images

  aws ecr create-repository --repository-name my-repo --region <aws-region>

  Repository URL ---- <aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/my-repo

  list all repositories ---- aws ecr describe-repositories

- Step 3 : Authenticate Docker with AWS ECR

  Authenticate Docker to AWS ECR ---- aws ecr get-login-password --region <aws-region> | docker login --username AWS --password-stdin <aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com

- Step 4 : Build and Tag the Docker Image

  Build the image using Dockerfile ---- docker build -t my-app .

  Tag the Image for ECR ---- docker tag my-app:latest <aws-account-id>.dkr.ecr.<aws-

region>.amazonaws.com/my-repo:latest

- **Step 5 : Push the Docker Image to ECR**

  push the image to AWS ECR ---- docker push <aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/my-repo:latest

- **Step 6 : To Pull the Image from ECR**

  Pull the Image ---- docker pull <aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/my-repo:latest

  Run the Container ---- docker run -d -p 80:80 <aws-account-id>.dkr.ecr.<aws-region>.amazonaws.com/my-repo:latest

- **Step 7 : Clean Up**

  To delete an image ---- aws ecr batch-delete-image --repository-name my-repo --image-ids imageTag=latest

  To delete the entire repository ---- aws ecr delete-repository --repository-name my-repo --force