Docker Commands Cheat Sheet

# Basic Commands

* docker --version # Show Docker version
* docker info # Display system-wide information
* docker help # Show help for Docker commands

# Image Management

* docker pull <image> # Pull an image from Docker Hub
* docker images # List downloaded images
* docker rmi <image\_id> # Remove an image
* docker tag <image\_id> <new\_name> # Rename/tag an image
* docker build -t <name> . # Build image from Dockerfile in current directory

# Container Management

* docker run <image> # Run a container from image
* docker run -d <image> # Run in detached mode (in background)
* docker run -p 8080:80 <image> # Run with port mapping (host:container)
* docker run --name <name> <image> # Assign name to container
* docker ps # List running containers
* docker ps -a # List all containers (including stopped)
* docker stop <container\_id/name> # Stop a running container
* docker start <container\_id/name> # Start a stopped container
* docker restart <container\_id/name> # Restart container
* docker rm <container\_id/name> # Remove container
* docker rename <old\_name> <new\_name> # Rename a container

# Volumes and Data

* docker volume create <volume\_name> # Create a volume
* docker volume ls # List volumes
* docker volume rm <volume\_name> # Remove a volume
* docker run -v <volume\_name>:/path <img> # Mount volume to container path

# Networking

* docker network ls # List networks
* docker network create <name> # Create a network
* docker network connect <net> <container> # Connect container to network
* docker network disconnect <net> <cont> # Disconnect container from network

# Docker Compose

* docker-compose up # Start services
* docker-compose up -d # Start in detached mode
* docker-compose down # Stop and remove containers/networks

# System Cleanup

* docker system prune # Remove unused data
* docker container prune # Remove stopped containers
* docker image prune # Remove unused images

Example: Integrating MongoDB with Docker

This example demonstrates how to run a MongoDB container and connect another application container (e.g., Node.js) to it using a custom Docker network.

# Step-by-Step Commands

* docker network create my-network # Create a custom Docker network
* docker run -d --name mongo-container --network my-network \
* -p 27017:27017 -e MONGO\_INITDB\_ROOT\_USERNAME=admin \
* -e MONGO\_INITDB\_ROOT\_PASSWORD=password mongo # Run MongoDB with credentials
* docker run -d --name app-container --network my-network \
* my-app-image # Run your app in the same network as MongoDB

## Access Mongo Shell

To connect to the MongoDB container via terminal:

* docker exec -it mongo-container mongosh -u admin -p password

## Verify Network

To ensure both containers are on the same network:

* docker network inspect my-network