DOCKER CONTAINERIZATION ESSENTIALS – GUIDE FOR LEARNERS

Table of Contents

**Module 1. Introduction to Containerization and Docker Architecture ..................................................... 3**

**Module 2. Managing Docker Containers and Images ............................................................................... 8**

**Module 3. Docker Volumes, Networking & Image Optimization ............................................................. 13**

**Module 4. Docker Compose and Multi-Container Applications .............................................................. 18**

**Module 5. Docker Security, Registries, and Best Practices ...................................................................... 23**

**Module 1. Introduction to Containerization and Docker Architecture**

Docker is a powerful platform designed to help developers build, deploy, and run applications in containers. This module introduces the foundational concepts of containerization and dives into the architecture and installation of Docker.

**1. Containerization Basics**

**Evolution from VMs to Containers**  
Traditional Virtual Machines virtualize entire operating systems, consuming more resources. Containers virtualize at the OS level, sharing the host kernel and thus being lighter and faster.

**Benefits of Containers over VMs**

* Lightweight and faster startup time
* Resource efficiency
* Portability across environments
* Consistent environments for dev/test/prod

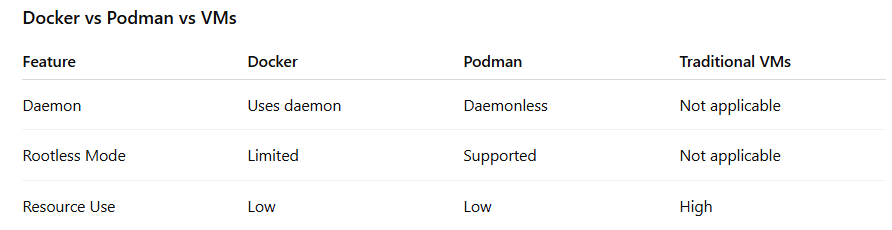
**Introduction to Microservices**  
Microservices architecture involves breaking applications into smaller, independently deployable services. Docker containers provide the perfect environment for microservice deployment due to their isolation and scalability.

**2. Introduction to Docker**

**What is Docker?**  
Docker is an open-source containerization platform that enables developers to package applications with all their dependencies into a standardized unit—**a container**.

**Key Docker Terminology**

* **Image**: A lightweight, standalone, and executable software package
* **Container**: A running instance of an image
* **Dockerfile**: A script defining how to build an image
* **Docker Hub**: Public registry for sharing Docker images



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**3. Docker Architecture Deep Dive**

* **Docker Engine**: Core client-server app that manages containers
* **Docker Daemon**: Background service managing containers
* **Docker CLI**: Command-line interface for interacting with Docker
* **Docker Hub/Registry**: Central repository to store/share images

**4. Installing Docker**

**A. Requirements**

* Linux/macOS/Windows system
* 64-bit CPU architecture
* Internet access for package download

**B. Installation Steps**

* **Linux**:

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

**Windows/macOS**: Download Docker Desktop from <https://www.docker.com/products/docker-desktop>

### ****5. Your First Container****

Run a test container:

docker run hello-world

**Basic Commands**:

* docker ps – list running containers
* docker stop <container\_id> – stop a container
* docker rm <container\_id> – remove a container

### ****Hands-on Labs (Module 1)****

* **Install Docker** on your preferred OS
* **Run your first container** (hello-world)
* **Explore CLI**: docker run, docker ps, docker stop, docker rm
* **Inspect container metadata** using docker inspect

## ****Module 2. Managing Docker Containers and Images****

Once Docker is installed and a basic container is running, it's essential to learn how to manage containers and images more effectively.

### ****1. Managing Containers****

**Basic Container Lifecycle Commands**:

docker run -d --name web nginx # Run container in detached mode

docker ps # List running containers

docker ps -a # List all containers

docker stop web # Stop container

docker start web # Start stopped container

docker restart web # Restart container

docker rm web # Remove container

**Container Logs**:

docker logs web # View logs of a container

**Container Shell Access**:

docker exec -it web bash # Execute bash shell inside a running container

### ****2. Managing Docker Images****

**Pulling and Listing Images**:

docker pull ubuntu # Download image from Docker Hub

docker images # List available images

**Removing Images**:

docker rmi ubuntu # Remove an image

**Building Custom Images with Dockerfile**:

# Dockerfile

FROM ubuntu

RUN apt update && apt install -y nginx

CMD ["nginx", "-g", "daemon off;"]

docker build -t custom-nginx . # Build image from Dockerfile

### ****3. Image Tagging and Versioning****

docker tag custom-nginx myrepo/nginx:1.0

### ****Hands-on Labs (Module 2)****

* Run and manage containers using run, stop, start, rm
* Create and execute a simple Dockerfile
* Tag and remove images
* Connect to a running container and explore logs

### ****Hands-on Labs (Module 2)****

* Run and manage containers using run, stop, start, rm
* Create and execute a simple Dockerfile
* Tag and remove images
* Connect to a running container and explore logs

## ****Module 3. Docker Volumes, Networking & Image Optimization****

### ****1. Working with Docker Volumes****

**Volume Commands**:

docker volume create myvol

docker run -v myvol:/data ubuntu

docker volume ls

docker volume inspect myvol

docker volume rm myvol

**Bind Mounts**:

docker run -v $(pwd)/html:/usr/share/nginx/html nginx

### ****2. Docker Networking****

**Network Types**:

* **Bridge** (default)
* **Host**
* **None**
* **User-defined bridge**

**Commands**:

docker network ls

docker network create mynet

docker network inspect mynet

docker network rm mynet

**Connect containers over user-defined networks**:

docker run -dit --name app1 --network mynet alpine

docker run -dit --name app2 --network mynet alpine

docker exec -it app1 ping app2

### ****3. Image Optimization Best Practices****

* Use minimal base images (e.g., alpine)
* Combine RUN instructions to reduce image layers
* Clean cache in Dockerfile (apt clean, rm -rf /var/lib/apt/lists/\*)
* Use .dockerignore to avoid unnecessary files

### ****Hands-on Labs (Module 3)****

* Create and use named volumes and bind mounts
* Create a user-defined network and ping between containers
* Optimize a Dockerfile to reduce image size

## ****Module 4. Docker Compose and Multi-Container Applications****

Docker Compose simplifies managing multi-container applications using a YAML configuration file.

### ****1. Introduction to Docker Compose****

**Install Docker Compose (Linux)**:

bash

CopyEdit

sudo apt install docker-compose

### ****2. Creating a Compose File****

# docker-compose.yml

version: "3"

services:

web:

image: nginx

ports:

- "80:80"

db:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD: example

**Run Compose**:

docker-compose up -d

docker-compose ps

docker-compose down

### ****3. Compose File Features****

* build: Build from a Dockerfile
* volumes: Persist data
* networks: Isolated application networks
* depends\_on: Control service startup order

### ****Hands-on Labs (Module 4)****

* Create a simple multi-container app (e.g., nginx + MySQL)
* Use docker-compose up, ps, and down
* Add volumes, environment, and depends\_on fields

## ****Module 5. Docker Security, Registries, and Best Practices****

Security and best practices are critical when running Docker containers in production.

### ****1. Docker Security Best Practices****

* Use official images and scan for vulnerabilities
* Run containers with non-root user
* Limit container capabilities using --cap-drop
* Use read-only root filesystem: --read-only

### ****2. Working with Docker Registries****

**Push/Pull from Docker Hub**:

docker login

docker tag myapp username/myapp:latest

docker push username/myapp:latest

docker pull username/myapp:latest

**Self-Hosted Registry**:

docker run -d -p 5000:5000 --name registry registry:2

docker tag myapp localhost:5000/myapp

docker push localhost:5000/myapp

### ****3. Best Practices Summary****

* Use .dockerignore to reduce context size
* Multi-stage builds for clean production images
* Keep containers stateless
* Periodic vulnerability scans

### ****Hands-on Labs (Module 5)****

* Create Docker Hub account and push/pull your image
* Set up a local Docker registry and push an image
* Practice running containers in secure mode (non-root, read-only)