DevOps Shack

Docker Comprehensive Guide

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]

Introduction to Docker

Docker is a powerful platform for developing, shipping, and running applications inside containers. Containers allow developers to package an application with all of its dependencies into a standardized unit of software. This ensures that the application runs consistently across different environments. In this guide, we will cover everything you need to know to get started with Docker, from basic concepts to advanced usage.

Chapter 1: Understanding Docker

1.1 What is Docker?

Docker is an open-source platform that automates the deployment of applications inside lightweight, portable containers. Containers are isolated environments that share the host system's kernel but operate independently, providing a consistent and reproducible runtime environment.

1.2 Why Use Docker?

- **Portability**: Run your application on any system that supports Docker without changes.
- **Isolation**: Containers encapsulate your application, isolating it from the host system and other containers.
- **Scalability**: Easily scale applications up or down by adding or removing containers.
- **Efficiency**: Containers use fewer resources compared to virtual machines because they share the host OS kernel.

1.3 Docker vs. Virtual Machines

Unlike virtual machines, which require a full OS for each instance, Docker containers share the host OS kernel and run as isolated processes, making them lightweight and faster to start.

Chapter 2: Installing Docker

2.1 System Requirements

Docker supports various operating systems including Linux, Windows, and macOS. Ensure your system meets the minimum requirements.

2.2 Installing Docker on Linux

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list
> /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-
plugin
```

2.3 Installing Docker on Windows and macOS

Download the Docker Desktop installer from the Docker website and follow the installation instructions.

Chapter 3: Docker Basics

3.1 Docker Architecture

Docker uses a client-server architecture:

- **Docker Client**: Issues commands to the Docker daemon (server).
- **Docker Daemon**: Builds, runs, and manages Docker containers.
- **Docker Images**: Read-only templates used to create containers.
- **Docker Containers**: Instances of Docker images.

3.2 Docker Commands

- **docker version**: Check Docker version.
- **docker info**: Display system-wide information.
- **docker pull**: Download an image from a registry.
- **docker run**: Create and start a container.
- **docker ps**: List running containers.
- **docker stop**: Stop a running container.
- **docker rm**: Remove a container.
- **docker rmi**: Remove an image.

Chapter 4: Working with Docker Images

4.1 Understanding Docker Images

Docker images are the building blocks of Docker containers. They are created from a series of layers, each representing a step in the build process.

4.2 Building Docker Images

Create a `Dockerfile` to define your image:

```
# Use an existing image as a base
FROM ubuntu:20.04

# Install dependencies
RUN apt-get update && apt-get install -y \
    python3 \
    python3-pip

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Run the application
CMD ["python3", "app.py"]
```

Build the image:

```
docker build -t my-python-app .
```

4.3 Managing Docker Images

- **docker images**: List all images on your system.
- **docker tag**: Tag an image.
- **docker push**: Push an image to a registry.
- **docker inspect**: Display detailed information about an image.

Chapter 5: Working with Docker Containers

5.1 Running Containers

Start a container from an image:

```
docker run -d -p 80:80 my-python-app
```

This command runs the container in detached mode (`-d`) and maps port 80 of the container to port 80 on the host (`-p 80:80`).

5.2 Interacting with Containers

- **docker exec**: Run a command in a running container.
- **docker logs**: Fetch the logs of a container.
- **docker inspect**: Display detailed information about a container.
- **docker stats**: Display resource usage statistics for running containers.

5.3 Managing Containers

- **docker stop**: Stop a running container.
- **docker start**: Start a stopped container.
- **docker restart**: Restart a container.
- **docker rm**: Remove a container.
- **docker prune**: Remove all stopped containers, networks not used by at least one container, images without at least one container associated with them, and build cache.

Chapter 6: Docker Networking

6.1 Understanding Docker Networking

Docker provides several networking modes:

- **Bridge**: The default network driver, suitable for standalone containers.
- **Host**: Containers share the host's networking namespace.
- **Overlay**: For multi-host networking, typically used with Docker Swarm.
- **Macvlan**: Assigns a MAC address to each container, making it appear as a physical device on the network.

6.2 Configuring Docker Networking

Create a user-defined bridge network:

```
docker network create my-bridge-network
```

Run a container on this network:

```
docker run -d --name webapp --network my-bridge-network my-web-app
```

6.3 Docker Compose Networking

Docker Compose allows you to define and manage multi-container applications. Define networks in the `docker-compose.yml` file:

```
version: '3'
services:
  web:
    image: nginx
    networks:
      - frontend
  app:
    image: my-python-app
    networks:
      - frontend
      - backend
  db:
    image: postgres
    networks:
      - backend
networks:
  frontend:
  backend:
```

Chapter 7: Docker Volumes and Data Management

7.1 Understanding Docker Volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. They are managed by Docker and can be more efficient and offer better performance than using bind mounts.

7.2 Managing Docker Volumes

- **docker volume create**: Create a volume.
- **docker volume ls**: List all volumes.
- **docker volume inspect**: Display detailed information about a volume.
- **docker volume rm**: Remove a volume.

7.3 Using Volumes in Containers

Mount a volume when running a container:

```
docker run -d -v my-volume:/data my-python-app
```

7.4 Docker Compose and Volumes

Define volumes in the `docker-compose.yml` file:

```
version: '3'
services:
  web:
    image: nginx
    volumes:
      - web-data:/var/www/html
  db:
    image: postgres
    volumes:
      - db-data:/var/lib/postgresql/data
volumes:
  web-data:
  db-data:
```

Chapter 8: Docker Compose

8.1 Introduction to Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. Use a YAML file to configure your application's services, networks, and volumes.

8.2 Installing Docker Compose

Docker Desktop includes Docker Compose. For Linux, install it separately:

```
sudo curl -L "https://github.com/docker/compose/releases/download/$(curl -s
https://api.github.com/repos/docker/compose/releases/latest | grep -oP
'(?<="tag_name": ")[^"]*')/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

8.3 Creating a `docker-compose.yml` File

Example `docker-compose.yml` for a web application:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - web-data:/usr/share/nginx/html
  app:
    image: my-python-app
    ports:
      - "5000:5000"
    volumes:
      - app-data:/app
  db:
    image: postgres
```

```
    environment:
      POSTGRES_PASSWORD: example
    volumes:
      - db-data:/var/lib/postgresql/data
volumes:
  web-data:
  app-data:
  db-data:
```

8.4 Managing Multi-Container Applications

- **docker-compose up**: Create and start containers.
- **docker-compose down**: Stop and remove containers, networks, images, and volumes.
- **docker-compose ps**: List containers.
- **docker-compose logs**: View output from containers.

Chapter 9: Advanced Docker Usage

9.1 Docker Swarm

Docker Swarm is Docker's

native clustering and orchestration tool. It turns a pool of Docker hosts into a single, virtual Docker host.

- **docker swarm init**: Initialize a swarm.
- **docker node ls**: List nodes in the swarm.
- **docker service create**: Create a new service.

9.2 Docker Secrets and Configs

Securely manage sensitive data such as passwords and certificates with Docker secrets.

- **docker secret create**: Create a secret.
- **docker secret ls**: List secrets.
- **docker service update**: Update a service to use a secret.

9.3 Docker and Kubernetes

Kubernetes is a powerful orchestration tool for managing containerized applications at scale. Docker can work seamlessly with Kubernetes.

- **kubectl create**: Create Kubernetes resources.
- **kubectl apply**: Apply configuration to Kubernetes resources.

- **kubectl get**: List resources.

Chapter 10: Docker Security Best Practices

10.1 Running Containers as Non-Root Users

Avoid running containers as the root user to minimize security risks.

```
# Use a non-root user
USER nonroot
```

10.2 Limiting Container Resources

Limit the resources a container can use to prevent it from affecting the host system.

```
docker run -d --cpus="1.0" --memory="512m" my-python-app
```

10.3 Regularly Updating Images

Keep your images up to date with the latest security patches.

```
docker pull my-python-app
```

10.4 Using Docker Bench for Security

Docker Bench is a script that checks for common best practices around deploying Docker containers.

```
docker run -it --net host --pid host --cap-add audit_control \
    -v /var/lib:/var/lib \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /usr/lib/systemd:/usr/lib/systemd \
    -v /etc:/etc \
    --label docker_bench_security \
    docker/docker-bench-security
```

Conclusion

Docker revolutionizes the way applications are developed, deployed, and managed. By understanding and mastering the basics, advanced features, and best practices, you can harness the full power of Docker to create robust, scalable, and secure applications. This comprehensive guide provides a solid foundation for your Docker journey, but continuous learning and practice are key to becoming proficient. Happy Dockerizing!