UNIT V – DEPLOYMENT AND CLOUD BASICS [9 hours]

How Software is Deployed in Real Life ,Cloud Platforms Overview (AWS/GCP/Azure), Hosting a Web App on Cloud (AWS EC2/Heroku), Docker Compose Basics, Introduction to Monitoring Tools (like Grafana)

DOCKER COMPOSE

Docker Compose is a tool used to define and manage **multi-container Docker applications**. In many real-world systems, applications are not made of a single container. Instead, they include several components such as a web server, database server, backend application, and caching system. Managing each container separately using individual commands can be difficult and time-consuming.

Docker Compose simplifies this process by allowing developers to define all required containers in a **single configuration file** called docker-compose.yml. Using this file, developers can start, stop, and manage the entire application with a single command. Docker Compose works together with Docker and is widely used in modern DevOps workflows.

Need for Docker Compose

In traditional Docker usage, developers must run each container manually using separate commands. For example, a web application might require:

- A **web server container**
- A **database container**
- A **backend application container**

Starting and linking these containers individually becomes complex as the application grows. Docker Compose solves this problem by allowing developers to define all services and their relationships in one file. This makes application setup easier, especially in development and testing environments.

Features of Docker Compose

1. **Multi-Container Management:** Docker Compose allows developers to manage multiple containers as a single application. All services can be started or stopped together.
2. **Single Configuration File:** All configurations are written in a docker-compose.yml file using YAML format. This file contains the definition of services, networks, and volumes.
3. **Easy Service Communication:** Containers created by Docker Compose can communicate with each other using service names, making networking simpler.
4. **Automated Environment Setup:** Developers can recreate the same environment on any machine by simply running Docker Compose.

5. **Simplified Deployment:** Instead of running multiple commands, developers can deploy the entire application with one command.

Components of Docker Compose

1. **Services:** Services represent the containers that make up the application. Each service defines which image to use, ports to expose, and environment variables.
2. **Networks:** Docker Compose automatically creates networks so that containers can communicate with each other.
3. **Volumes:** Volumes allow persistent storage so that data is not lost when containers stop or restart.

Structure of a Docker Compose File

Example docker-compose.yml file:

```
version: '3'
services:
 web:
  image: nginx
  ports:
   - "80:80"
 database:
  image: mysql
  environment:
   MYSQL_ROOT_PASSWORD: root
```

Explanation:

- **version** – Defines the Docker Compose version.
- **services** – Lists all containers required for the application.
- **web** – Defines a web server using the Nginx image.
- **database** – Defines a MySQL database container.
- **ports** – Maps container ports to host ports.
- **environment** – Defines environment variables.

Basic Docker Compose Commands

- Start all services: docker-compose up
- Run containers in background: docker-compose up -d
- Stop and remove containers: docker-compose down
- Check running containers: docker-compose ps
- Rebuild containers: docker-compose build

Working of Docker Compose

The working of Docker Compose involves several steps:

1. The developer creates a docker-compose.yml file describing the application services.
2. Each service specifies the Docker image and configuration details.
3. The command docker-compose up is executed.
4. Docker Compose automatically pulls the required images.
5. Containers are created and started in the correct order.
6. Containers communicate through Docker networks and work together as a complete application.

Advantages of Docker Compose

- Simplifies the management of multi-container applications
- Uses a single configuration file for easy setup
- Ensures consistent environments across different systems
- Reduces manual configuration and errors
- Improves productivity for developers and DevOps teams

INTRODUCTION TO MONITORING TOOLS

In modern software systems, applications run on servers, cloud platforms, and containers. To ensure that these systems run smoothly, organizations use monitoring tools. Monitoring tools help track system performance, detect errors, and analyze system behavior in real time.

Monitoring tools collect data such as CPU usage, memory usage, network traffic, response time, and application performance. This helps developers and system administrators identify problems quickly and maintain system stability. One popular monitoring tool widely used in DevOps environments is Grafana.

Monitoring tools are software applications used to **observe, measure, and analyze the performance of systems, servers, and applications**. They collect metrics and logs from different sources and present them in an understandable format such as dashboards, charts, and graphs.

These tools help organizations ensure that applications are running efficiently and detect issues before they affect users.

Need for Monitoring Tools

Monitoring tools are important in modern software systems for several reasons:

1. **Performance Monitoring:** They track system performance such as CPU usage, memory consumption, disk usage, and network traffic.
2. **Error Detection:** Monitoring tools help identify errors, failures, or crashes in applications.

3. **System Reliability:** Continuous monitoring ensures that systems are stable and available for users.
4. **Early Problem Detection:** Issues can be detected early before they affect a large number of users.
5. **Improved Decision Making:** Monitoring data helps developers analyze performance and improve system design.

Types of Monitoring

1. **Infrastructure Monitoring:** This focuses on monitoring servers, hardware resources, CPU usage, memory usage, and network performance.
2. **Application Monitoring:** This tracks application performance, response times, and error rates.
3. **Network Monitoring:** This monitors network traffic, connectivity, and data transfer performance.
4. **Log Monitoring:** This analyzes system logs to detect errors or unusual behavior.

GRAFANA

Grafana is an open-source monitoring and visualization tool used to display system metrics and data through interactive dashboards. Grafana collects data from different sources and displays them using graphs, charts, and tables. It helps developers and administrators understand system performance easily. Grafana is widely used in DevOps environments because it provides powerful visualization and monitoring capabilities.

Features of Grafana

1. **Data Visualization:** Grafana provides powerful dashboards that display system data using graphs, charts, and panels.
2. **Multiple Data Sources:** It can connect to many data sources such as databases and monitoring systems like Prometheus.
3. **Custom Dashboards:** Users can create customized dashboards to monitor specific system metrics.
4. **Real-Time Monitoring:** Grafana shows real-time system performance and updates data automatically.
5. **Alerting System:** Grafana can send alerts through email or messaging systems when system metrics exceed certain limits.
6. **User-Friendly Interface:** Grafana provides an easy-to-use interface that allows users to visualize and analyze monitoring data effectively.

Working of Monitoring Tools

The working process of monitoring tools generally includes the following steps:

1. System metrics are collected from servers, applications, or containers.

2. The data is stored in a database or monitoring system.
3. Monitoring tools analyze the collected data.
4. Dashboards display the metrics in graphical format.
5. Alerts are generated if any abnormal behavior is detected.

This process helps organizations maintain system performance and reliability.

Advantages of Monitoring Tools

Monitoring tools provide many benefits:

- Improve system performance and reliability
- Detect system failures quickly
- Provide real-time insights into system behavior
- Help in troubleshooting problems
- Support better resource management
- Improve user experience by reducing downtime

Grafana Works with Docker Compose

To manage multiple containers easily, developers use Docker Compose, which allows several services (containers) to run together using a single configuration file. By combining Grafana with Docker Compose, developers can quickly set up a complete monitoring system that collects and visualizes data from different services.

Why Use Grafana with Docker Compose

Using Grafana with Docker Compose provides several advantages:

- Easy setup of monitoring tools
- Multiple services can run together (Grafana + database + monitoring system)
- Consistent environment across development and testing systems
- Quick deployment using a single command
- Simplified configuration and management

Components in a Grafana Monitoring Setup

When Grafana is used with Docker Compose, the monitoring system usually includes the following components:

1. Grafana Container: The Grafana container provides dashboards and visualization of monitoring data.
2. Data Source: Grafana collects data from monitoring systems such as Prometheus.
3. Application Containers: These containers run the applications or services that generate performance data.
4. Docker Compose: Docker Compose manages and runs all these containers together.

Architecture of Grafana with Docker Compose

A typical architecture includes:

1. Application containers generate metrics.
2. Prometheus collects and stores these metrics.
3. Grafana retrieves the metrics from Prometheus.
4. Grafana dashboards display the data visually.

Docker Compose runs all these services together in separate containers.

Example Docker Compose Configuration

Example docker-compose.yml file:

```
version: '3'
services:
 prometheus:
   image: prom/prometheus
   ports:
     - "9090:9090"
 grafana:
   image: grafana/grafana
   ports:
     - "3000:3000"
```

Explanation

prometheus service

- Runs the Prometheus monitoring server
- Port **9090** is used for accessing Prometheus

grafana service

- Runs the Grafana dashboard
- Port **3000** allows users to access Grafana through a browser

Steps to Run Grafana with Docker Compose

1. Install **Docker and Docker Compose** on the system.
2. Create a project folder.
3. Create a file named **docker-compose.yml**.
4. Add services such as Grafana and Prometheus in the file.
5. Open terminal and run the command:

   docker-compose up -d
6. Docker Compose downloads the required images and starts the containers.
7. Open a browser and access Grafana using:

   http://localhost:3000
8. Configure Prometheus as the data source in Grafana.
9. Create dashboards to visualize monitoring metrics.

Working Process

The working of Grafana with Docker Compose involves the following steps:

1. Docker Compose starts with multiple containers.
2. Application containers generate performance metrics.
3. Prometheus collects and stores the metrics.
4. Grafana retrieves the data from Prometheus.
5. Grafana displays the metrics using dashboards, charts, and graphs.
6. System administrators monitor performance and detect issues.

Advantages of Using Grafana with Docker Compose

- Quick setup of monitoring environments
- Easy container management
- Better visualization of system metrics
- Real-time monitoring of applications
- Simplified DevOps workflow
