

**CLOUD COMPUTING LAB (22ECAC202)**

**Activity Report**

BATCH: **F1**

ROLL NO: 131

INSTRUCTOR: Ms. **: Sadaf Anjum Savanur**

COURSE DETAILS:

|  |  |
| --- | --- |
| **Course Name** | Cloud Computing |
| **Course Code** | 24ECAE317 |
| **Semester** | VI |
| **Division** | F |

**Team Members:**

|  |  |  |
| --- | --- | --- |
| **Name** | **USN** | **ROLL NO** |
| **Bhoomika Marigoudar** | **01FE22BCI035** | **131** |

|  |  |
| --- | --- |
| **TOPICS** | **PAGE NO.** |
| INTRODUCTION AND COURSE OUTCOMES | **3** |
| IMPLEMENTATION OF CLOUD SERVICE MODELS (IaaS, PaaS, SaaS) | **4** |
| VIRTUALIZATION AND CONTAINERIZATION TECHNIQUES FOR RESOURCE MANAGEMENT IN A CLOUD ENVIRONMENT | **6** |
| IMPLEMENT DIFFERENT LEVELS OF AUTOMATION AND CONTAINER ORCHESTRATION USING KUBERNETES | **7** |
| MICROSERVICES | **9** |
| ANSIBLE | **12** |

**Introduction and About Cloud Computing:**

Cloud computing is a revolutionary technology that enables on-demand access to a shared pool of configurable computing resources, such as servers, storage, databases, networking, software, and more, over the internet. It allows organizations and developers to scale applications efficiently, reduce infrastructure costs, and accelerate deployment cycles. Through various models like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), cloud computing provides flexibility and agility in managing IT services. This course introduces core concepts of cloud computing, emphasizing hands-on experience with virtualization, containerization, Kubernetes orchestration, serverless computing, microservices, and DevOps tools, making students industry-ready for modern cloud-based development and operations.

**Course Outcomes Overview:**

The projects in this course comprehensively address the key aspects of cloud computing. We begin by understanding and applying various cloud services and deployment models to build cloud-based applications. We then explore virtualization and containerization, managing resources efficiently in cloud environments. With Kubernetes, we implement automation and orchestration, streamlining deployment processes. Projects further involve developing scalable applications using serverless architecture and microservices, ensuring efficient event processing and communication. Finally, through tools like Ansible, Puppet, and Chef, we implement DevOps practices to automate configuration management and optimize infrastructure provisioning. Together, these projects align with the course outcomes, providing a holistic learning experience in cloud technologies and practices.

**Lab Evaluation 1: Introduction to Cloud Services and Deployment Models**

**Objective:**

* Understand cloud computing concepts, service models (IaaS, PaaS, SaaS).
* Deploy a simple cloud-based application using one of the service models.

**Tasks Performed:**

* Explored cloud platforms (AWS, Vercel, GCP) and their service offerings.
* Deployed a basic web application on AWS (PaaS).
* Configured environment variables and deployment parameters.

**Tools/Technologies Used:**

* GCP: Google Cloud ; Cloud Computing Services
* AWS Management Console
* Simple Web App (Node.js,HTML,CSS, React)

**Outcome:**

* Demonstrated ability to select appropriate cloud service models.
* Successfully deployed a web app on a cloud platform.
* Gained familiarity with cloud deployment processes.

**Different Cloud Services Available:**

**1. IaaS (Infrastructure as a Service)**

**Definition:**  
IaaS provides virtualized computing resources over the internet. This includes servers, storage, networking, and operating systems. Users can install their own applications and manage the entire software stack.

**Example Providers:**  
AWS EC2, Google Compute Engine, Microsoft Azure VM

**Use Case:**  
If a developer wants full control over their virtual machines and network, IaaS is the best choice.

**User Responsibility:**  
You handle the OS, applications, runtime, data, etc. The cloud provider only manages the infrastructure.

**2. PaaS (Platform as a Service)**

**Definition:**  
PaaS provides a ready-to-use platform with an environment for developing, testing, deploying, and maintaining applications. You don’t have to manage infrastructure.

**Example Providers:**  
AWS Elastic Beanstalk, Google App Engine, **Vercel**, Heroku

**Use Case:**  
If you're a developer focusing only on app development and not server setup or OS management, PaaS is ideal.

**User Responsibility:**  
You write and deploy your code. The provider manages everything else (OS, middleware, servers).

**3. SaaS (Software as a Service)**

**Definition:**  
SaaS delivers software applications over the internet on a subscription or free basis. Everything is managed by the service provider.

**Example Providers:**  
Gmail, Google Docs, Microsoft 365, Dropbox

**Use Case:**  
End-users who want ready-to-use software without worrying about installation or updates.

**User Responsibility:**  
Only to use the application.

**Vercel Deployment – Step-by-Step Implementation**

In our project, we used **Vercel**, which is a **PaaS** platform specialized in deploying **frontend frameworks and static sites**.

**Pre-requisites:**

* A GitHub account (to store code)
* A Vercel account (linked with GitHub)
* A web app built using **HTML, CSS, Node.js, and React**

**Step 1: Push Project to GitHub**

1. Create a GitHub repository.
2. Add your local project files (React + Node.js code) to the repo.
3. Commit and push the code:

git init

git add .

git commit -m "initial commit"

git remote add origin <repo-link>

git push -u origin main

**Step 2: Login to Vercel and Import Project**

1. Go to <https://vercel.com> and log in with your GitHub account.
2. Click on **“Add New Project”**.
3. Select the GitHub repository that contains your project.

**Step 3: Configure Project Settings**

1. Vercel will auto-detect the framework (e.g., React).
2. Set the **build command** (e.g., npm run build) and **output directory** (usually build or dist).
3. Set **Environment Variables** (if any) like API URLs, tokens, etc.

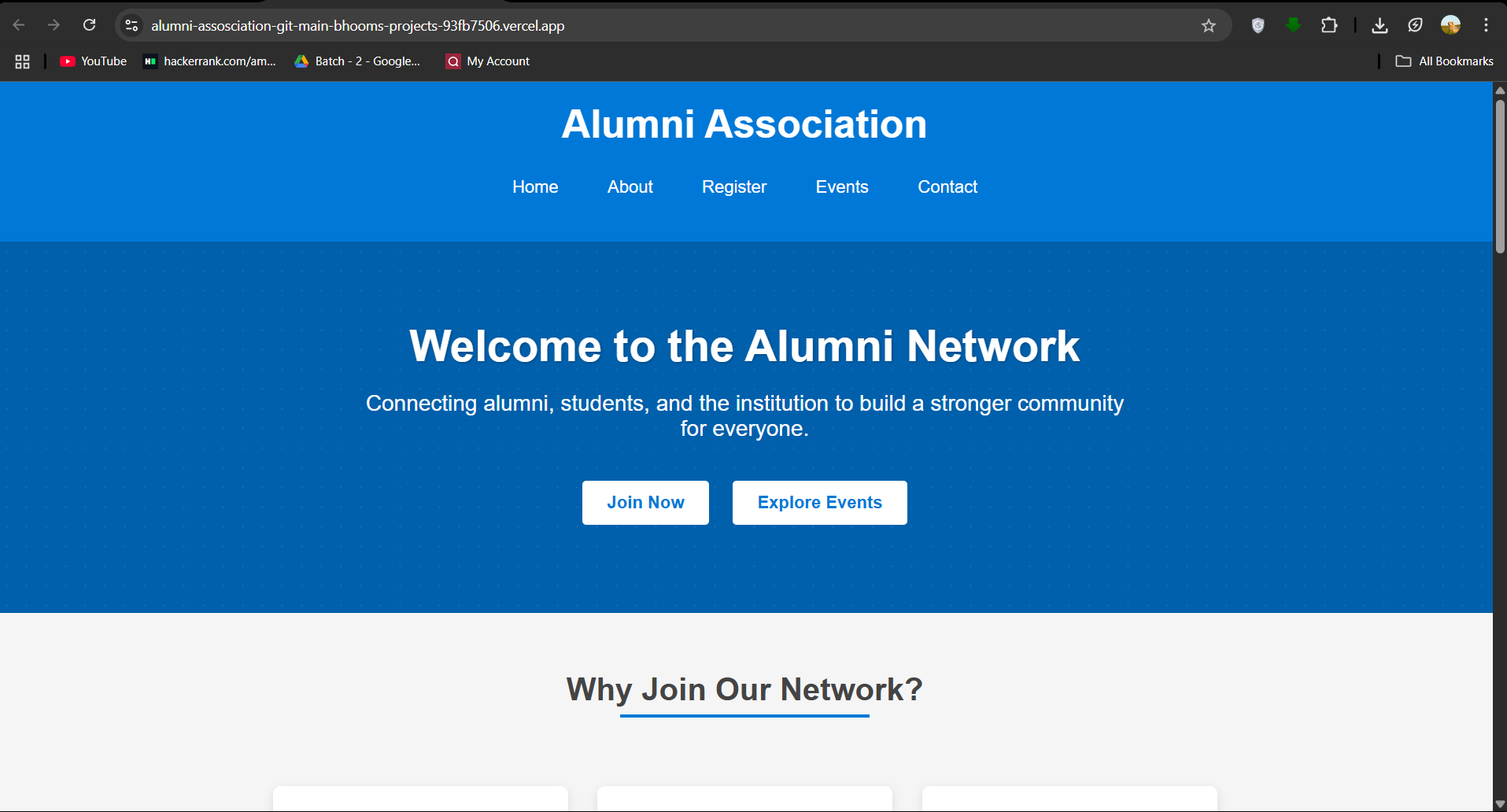
**Step 4: Deploy the Application**

1. Click on **“Deploy”**.
2. Vercel will install dependencies, build the app, and host it.
3. After a successful deployment, Vercel will provide a live URL (e.g., https://myapp.vercel.app).

**Step 5: Post-Deployment Configuration**

* Monitor logs in the Vercel dashboard.
* Set up custom domains (if needed).
* Configure automatic redeploys on GitHub pushes.

Link for Website: <https://alumni-assosciation-git-main-bhooms-projects-93fb7506.vercel.app/>



**Lab Evaluation 2: Virtualization and Containerization Techniques**

**Objective:**

* Learn virtualization basics and containerization using Docker.
* Create and manage containers for resource management.

**Tasks Performed:**

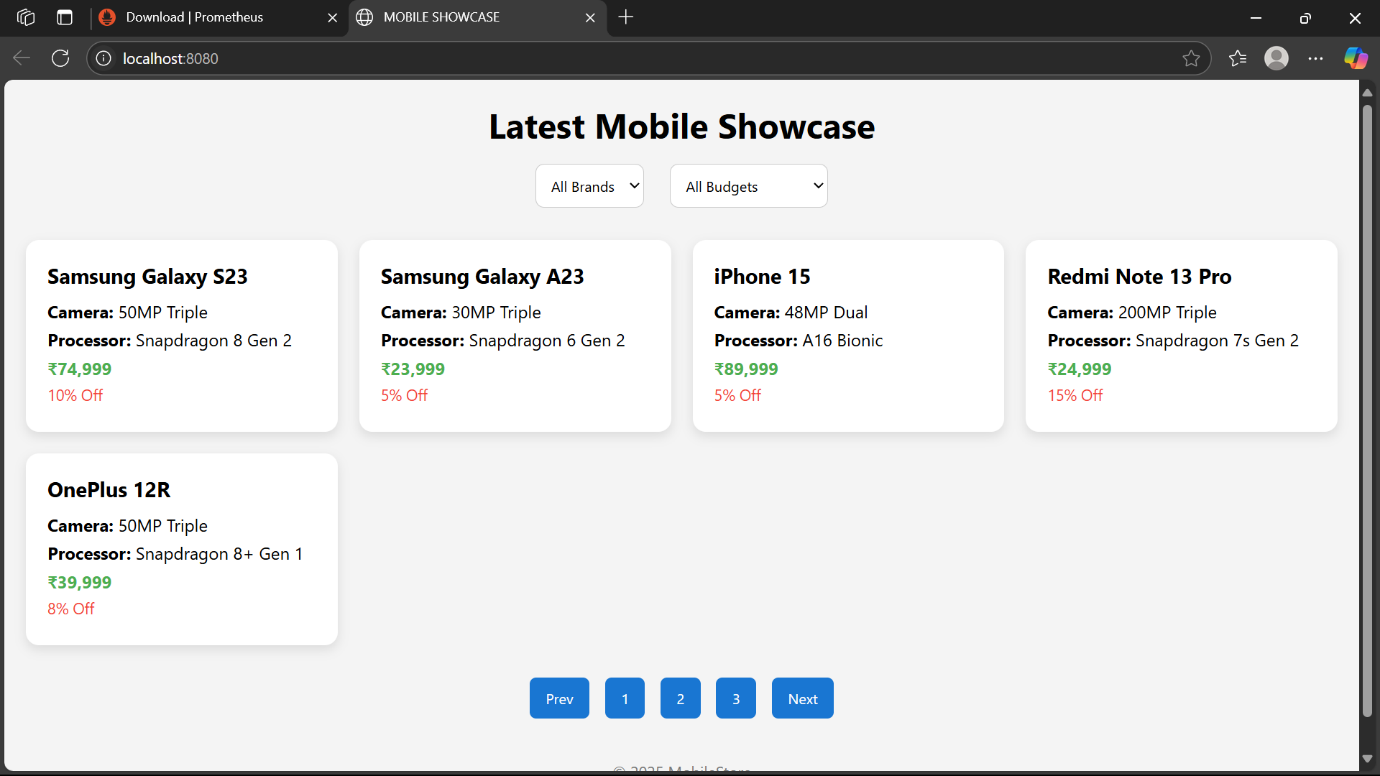
* Installed Docker on local machine.
* Created Docker images and containers using Dockerfiles.
* Managed containers lifecycle (start, stop, restart).
* Explored isolation and resource sharing in containers.

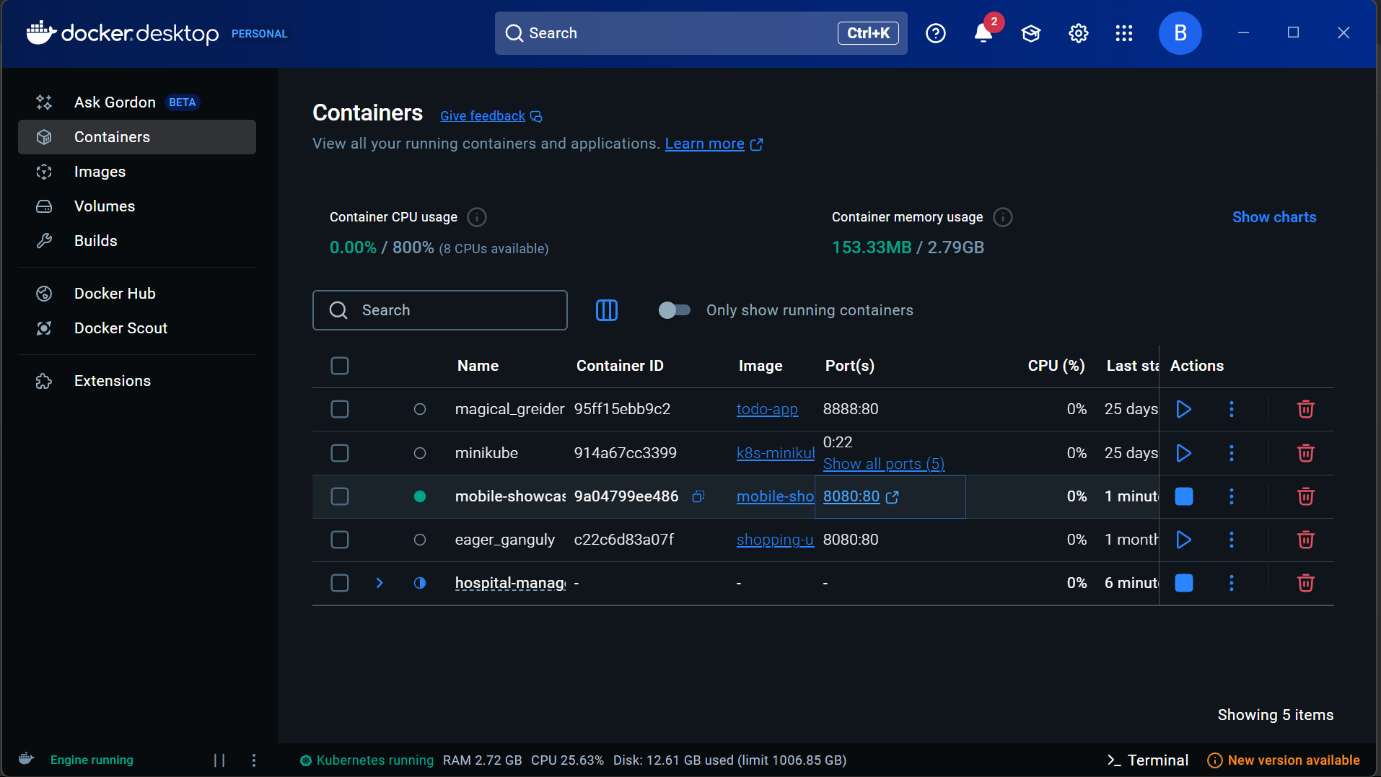
**Tools/Technologies Used:**

* Docker Engine and CLI
* Docker Hub for images

**Outcome:**

* Understood virtualization vs containerization differences.
* Created and deployed containers effectively.
* Recognized benefits of containers for cloud environments.





**Lab Evaluation 3: Kubernetes Orchestration and Automation**

**Objective:**

* Understand container orchestration and automation using Kubernetes.
* Deploy multi-container applications and **manage scaling**.

**Tasks Performed:**

* Set up a Kubernetes cluster using Minikube.
* Deployed sample multi-container apps with Kubernetes manifests (YAML).
* Explored pods, services, deployments, and scaling.
* Implemented basic automation for rolling updates.

**Tools/Technologies Used:**

* Minikube (local Kubernetes)
* kubectl CLI
* Sample microservices app (multi-container)

**Outcome:**

* Learned Kubernetes architecture and components.
* Successfully deployed and managed containerized apps with orchestration.
* Gained skills in automating deployment and scaling processes.

**Lab Evaluation 4: Development of Microservices-Based Applications**

**Hospital Management System – Monitoring with Prometheus and Grafana**

**Overview**

**In the Hospital Management System, we employed microservice architecture to break down functionality into separate services:**

* **patient-service**
* **doctor-service**
* **billing-service**
* **frontend**

**To monitor the health and performance of these services, we used:**

* **Prometheus (for collecting metrics)**
* **Grafana (for visualizing those metrics)**

**Microservices Used**

**Each of the microservices was containerized using Docker and run using Docker Compose. They included:**

| **Service Name** | **Port** | **Technology** |
| --- | --- | --- |
| **patient-service** | **5004** | **Spring Boot** |
| **doctor-service** | **5002** | **Node.js** |
| **billing-service** | **5003** | **Spring Boot** |
| **frontend** | **80/3000** | **React + Nginx** |

**Prometheus Integration**

**🔹 What is Prometheus?**

**Prometheus is an open-source monitoring tool that collects metrics from configured targets at given intervals, evaluates rule expressions, displays results, and can trigger alerts.**

**🔹 Prometheus Configuration (prometheus.yml)**

**yaml**

**scrape\_configs:**

**- job\_name: 'patient-service'**

**metrics\_path: /actuator/prometheus**

**static\_configs:**

**- targets: ['patient-service:5004']**

**- job\_name: 'doctor-service'**

**metrics\_path: /metrics**

**static\_configs:**

**- targets: ['doctor-service:5002']**

**- job\_name: 'billing-service'**

**metrics\_path: /actuator/prometheus**

**static\_configs:**

**- targets: ['billing-service:5003']**

**This configuration:**

* **Scrapes metrics from each service every 15 seconds (default).**
* **Uses the appropriate metrics endpoint:**
  + **/actuator/prometheus for Spring Boot (via Micrometer)**
  + **/metrics for Node.js (via prom-client)**

**With this, the services expose a Prometheus-compatible endpoint at:**

**bash**

**CopyEdit**

**http://localhost:5004/actuator/prometheus**

**Configuring Node.js Services**

**For doctor-service (Node.js), we used the prom-client library.**

**Installation:**

**npm install prom-client express**

**const express = require('express');**

**const client = require('prom-client');**

**const app = express();**

**client.collectDefaultMetrics(); // Enables basic system metrics**

**app.get('/metrics', async (req, res) => {**

**res.set('Content-Type', client.register.contentType);**

**res.end(await client.register.metrics());**

**});**

**app.listen(5002, () => console.log("Doctor Service running on port 5002"));**

**This exposes:**

**bash**

**CopyEdit**

**http://localhost:5002/metrics**

**📉 Common Prometheus Queries Used**

| **Query** | **Description** |
| --- | --- |
| **up** | **Displays the status of all targets (1 = UP, 0 = DOWN)** |
| **http\_server\_requests\_seconds\_count** | **Shows number of HTTP requests handled by Spring Boot services** |
| **http\_server\_requests\_seconds\_count{status=~"5.."}** | **Tracks how many 5xx server errors occurred** |
| **rate(http\_server\_requests\_seconds\_count[1m])** | **Rate of HTTP requests per second in the last minute** |

**📊 Grafana Integration**

**🔹 What is Grafana?**

**Grafana is a powerful visualization tool that allows users to create interactive dashboards from various data sources, including Prometheus.**

**🔹 Grafana Setup in Docker Compose**

**yaml**

**grafana:**

**image: grafana/grafana:latest**

**container\_name: grafana**

**ports:**

**- "3005:3000"**

**environment:**

**- GF\_SECURITY\_ADMIN\_USER=admin**

**- GF\_SECURITY\_ADMIN\_PASSWORD=admin**

**depends\_on:**

**- prometheus**

**networks:**

**- monitoring-net**

**🔹 Steps to Visualize Metrics**

1. **Visit Grafana at http://localhost:3005**
2. **Log in with:**
   * **Username: admin**
   * **Password: admin**
3. **Add Prometheus as a data source (URL: http://prometheus:9090)**
4. **Create a dashboard with panels for:**
   * **CPU usage**
   * **Memory usage**
   * **HTTP error rates**
   * **Service uptime**

**🔍 Debugging: Prometheus Not Seeing Services**

**If up = 0 for any service:**

* **Check Docker container logs (docker logs <container-name>)**
* **Ensure metrics endpoint is reachable from Prometheus container**
* **Confirm ports and paths match in prometheus.yml**
* **Visit Prometheus /targets UI: http://localhost:9090/targets**

**📦 Final Docker Compose Overview**

**yaml**

**services:**

**patient-service: ...**

**doctor-service: ...**

**billing-service: ...**

**frontend: ...**

**prometheus:**

**image: prom/prometheus**

**...**

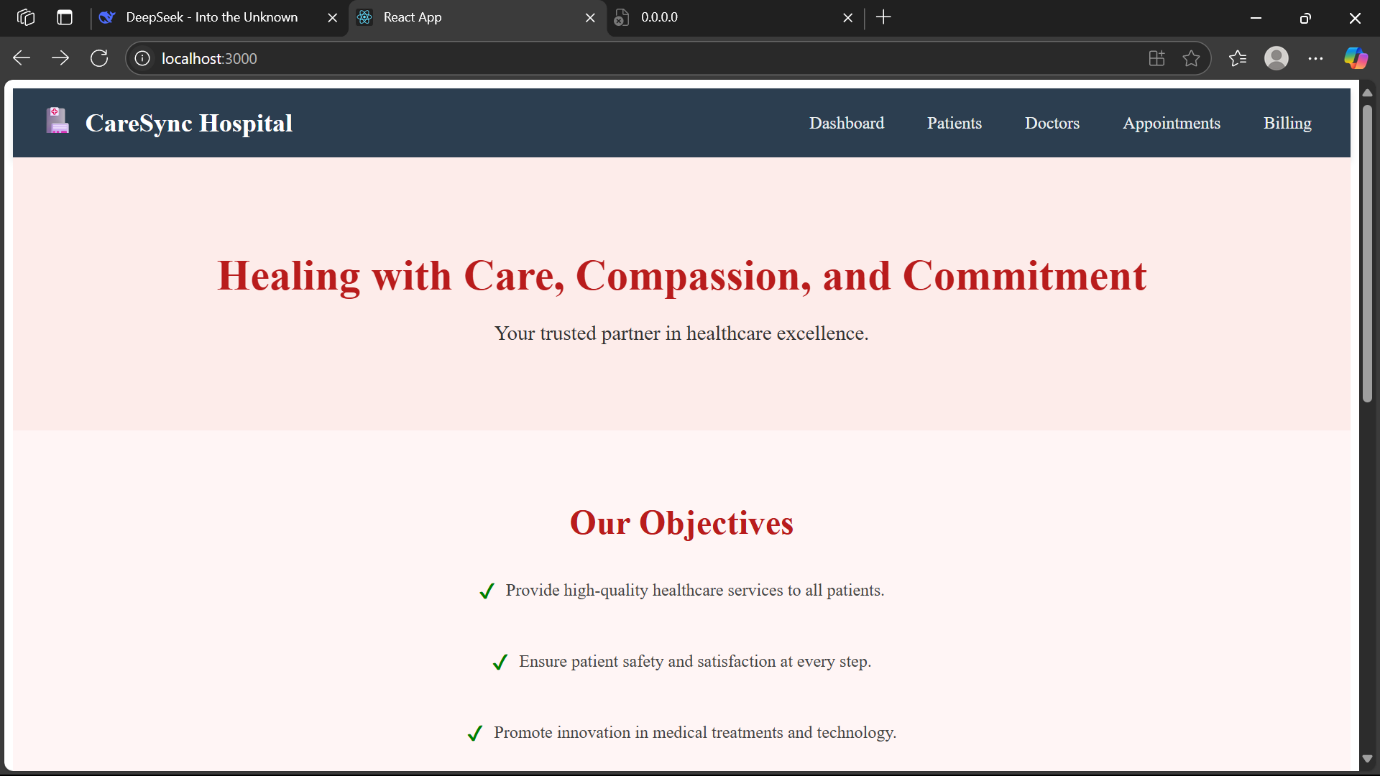
**grafana:**

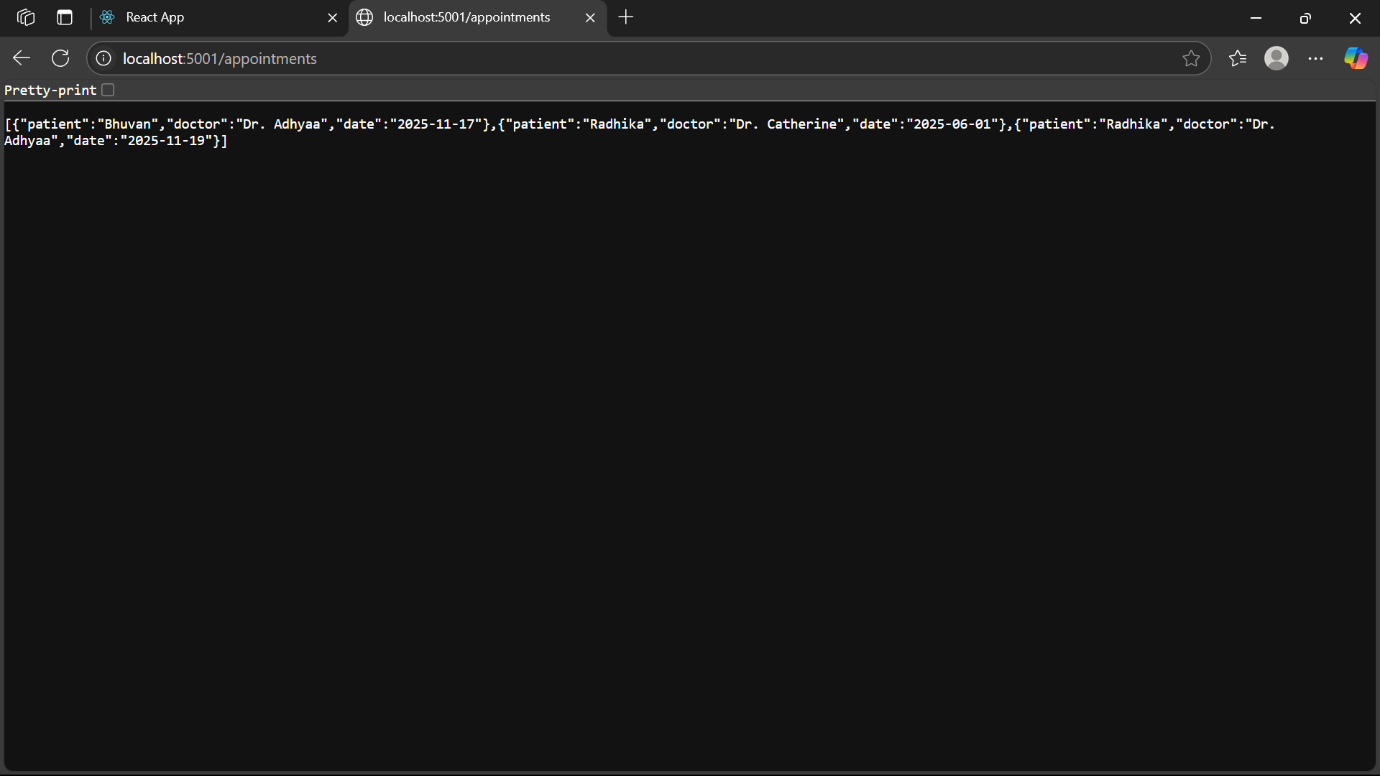
**image: grafana/grafana**

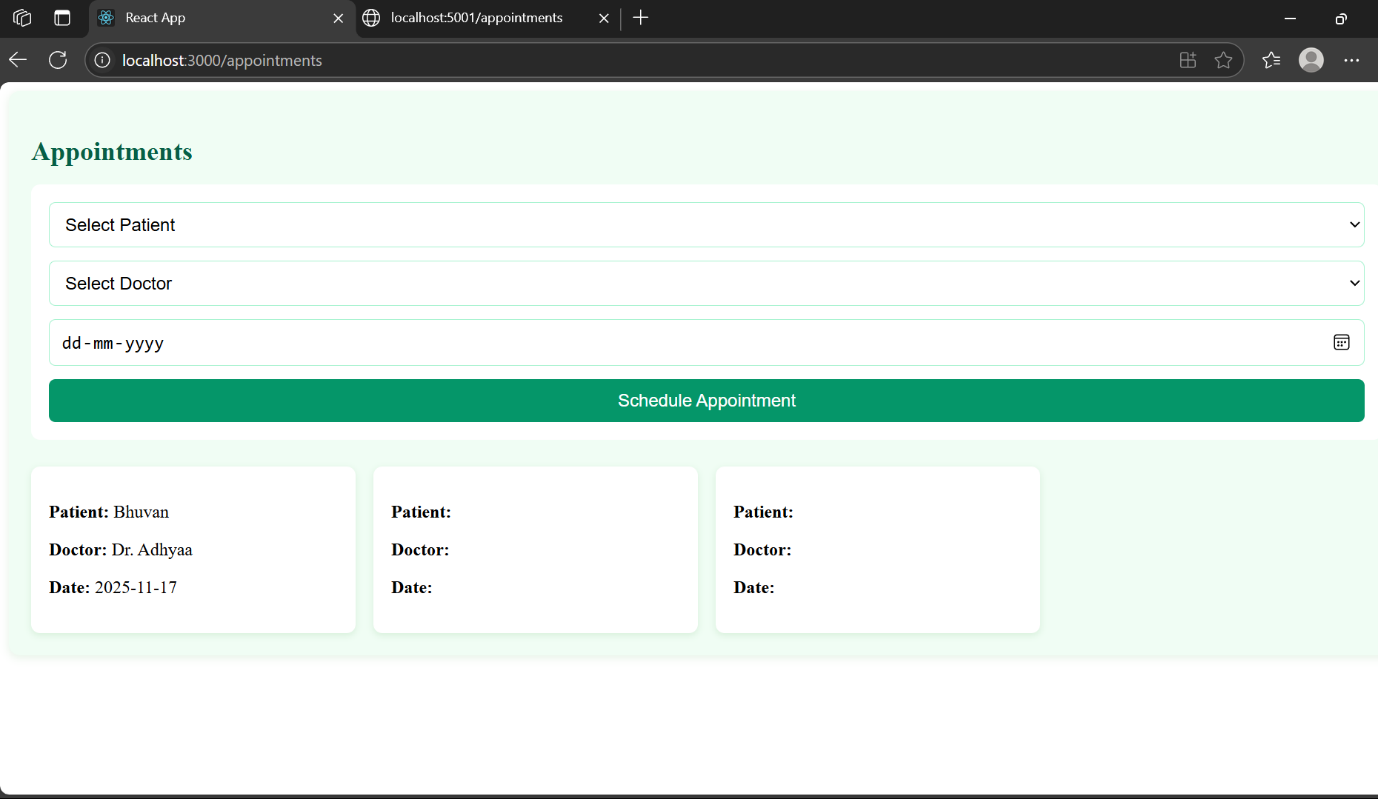
**...**

**All services run inside a shared Docker network (monitoring-net) to ensure proper communication.**

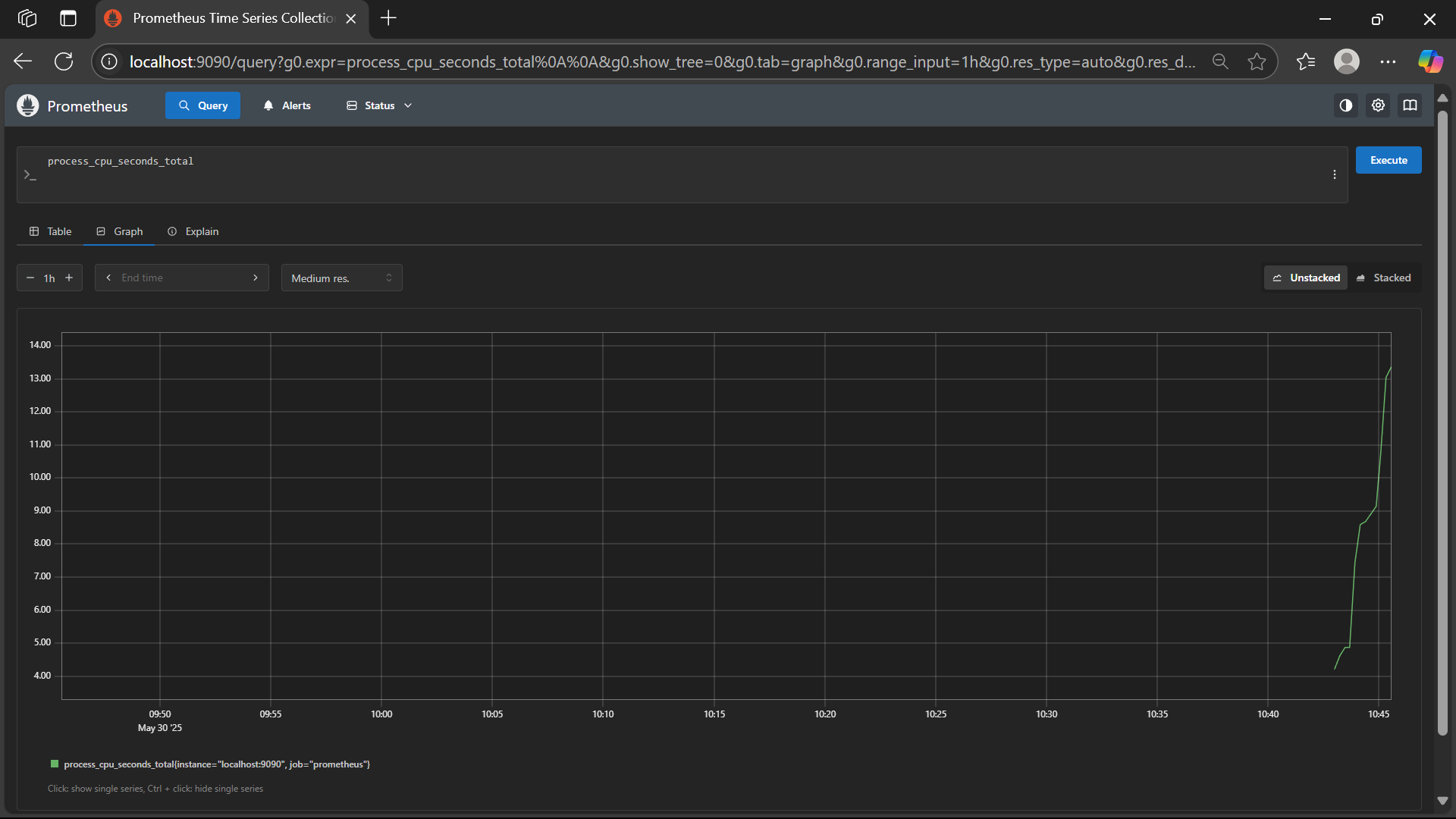
****

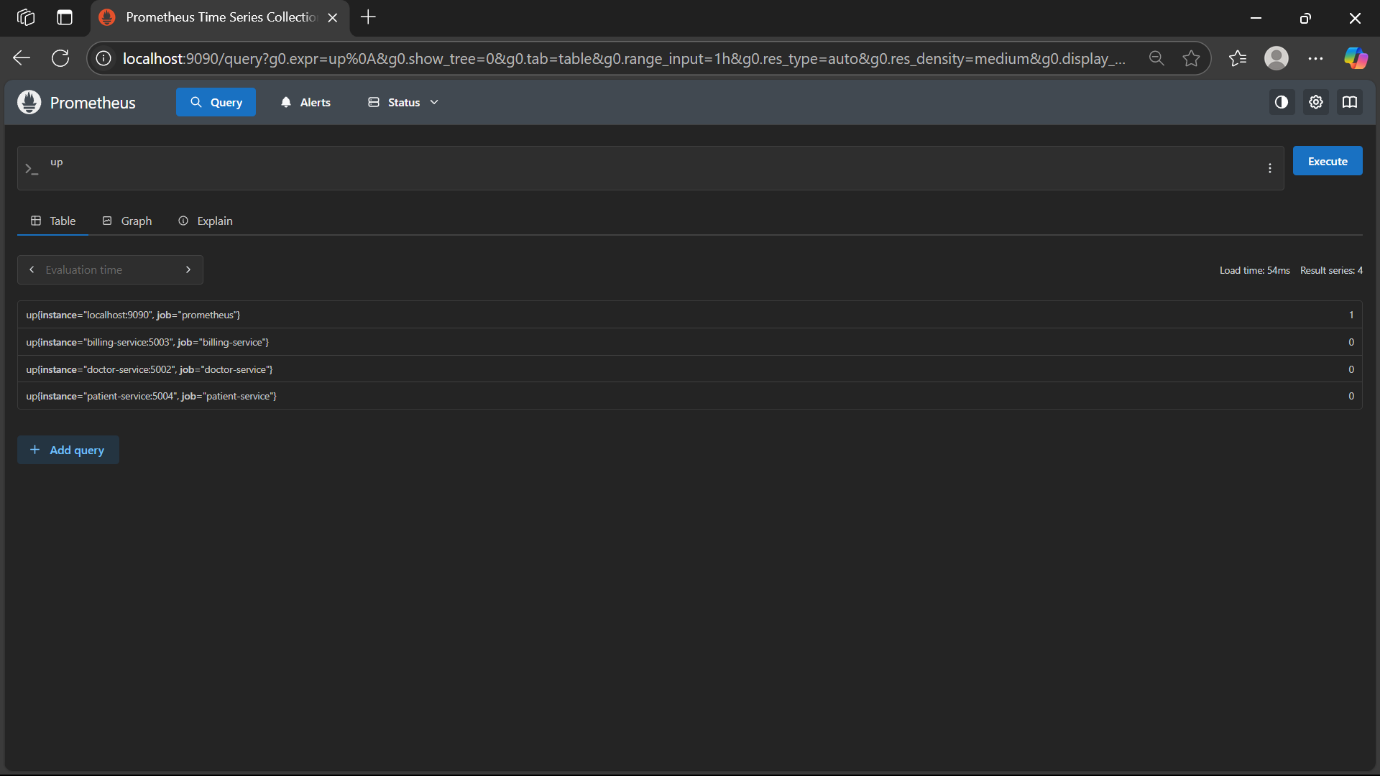
****

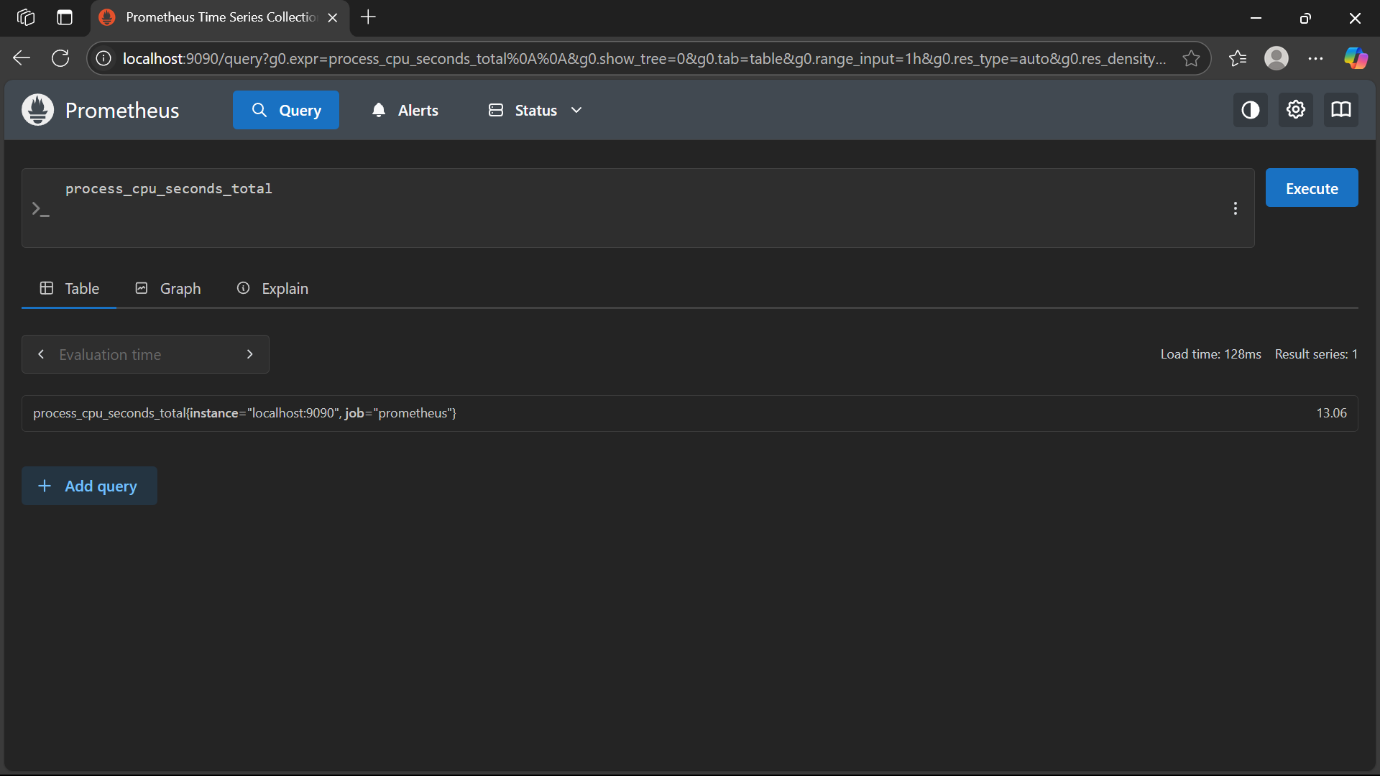
****

****

**PROMETHUS VISUALIZATION ALONG WITH LOAD BALANCER:**

****

****

****

**Lab Evaluation 5: Configuration Management Using Ansible**

**What is Ansible?**

**Ansible** is an **open-source automation tool** used for:

* **Provisioning** servers (setting them up)
* **Configuring** software/services
* **Deploying applications**
* **Managing multiple machines** at once

It follows the **Infrastructure as Code (IaC)** approach and is **agentless**, meaning it does not require any software to be installed on the target machines.

**Ansible Architecture Overview**

* **Control Node**: The machine where Ansible is installed (usually your local system or admin server).
* **Managed Nodes**: The servers you want to configure using Ansible (e.g., Ubuntu servers).
* **Inventory File**: A file listing the IP addresses or hostnames of all managed nodes.
* **Playbook**: A YAML file that defines the automation tasks to be run on the nodes.

**Steps to Configure and Use Ansible**

**Step 1: Install Ansible (on Control Node)**

**For Ubuntu/Debian:**

bash

sudo apt update

sudo apt install ansible -y

**To verify:**

bash

ansible --version

**Step 2: Set Up SSH Access to Managed Nodes**

* Ensure the control node can connect to managed nodes via **SSH**.

bash

ssh-copy-id user@<managed-node-ip>

This command adds your control node’s SSH key to the authorized keys of the managed node so that it can connect without a password.

**Step 3: Create an Inventory File**

The **inventory file** tells Ansible which servers to manage.

**Example (hosts file):**

ini

[webservers]

192.168.1.101

192.168.1.102

[dbservers]

192.168.1.201

Save this file as hosts or inventory.txt.

**Step 4: Ping Test to Check Connectivity**

Use Ansible’s built-in ping module to test the connection:

bash

ansible all -i hosts -m ping

Expected output:

ruby

192.168.1.101 | SUCCESS => {

"changed": false,

"ping": "pong"

}

**Step 5: Write Your First Ansible Playbook**

A **playbook** is a YAML file that defines what tasks to perform.

**Example (install\_apache.yaml)**:

yaml

---

- name: Install Apache on webservers

hosts: webservers

become: true # Runs tasks as root (sudo)

tasks:

- name: Install Apache package

apt:

name: apache2

state: present

update\_cache: yes

- name: Start Apache service

service:

name: apache2

state: started

enabled: yes

**Step 6: Run the Playbook**

Use the following command:

bash

ansible-playbook -i hosts install\_apache.yaml

This will:

* Install Apache on all servers under [webservers]
* Ensure the service is started and enabled at boot

**Step 7: Use Roles for Structured Playbooks (Optional but Recommended)**

Roles allow us to organize our playbooks in a reusable way.

**Folder structure:**

css

roles/

apache/

tasks/

main.yaml

handlers/

main.yaml

templates/

files/

Then your main playbook includes:

yaml

- hosts: webservers

roles:

- apache

**Step 8: Verify and Debug**

* Use ansible-playbook with -v or -vvv for verbose output.
* Check logs and execution results directly from terminal.

**Objective:**

* Automate infrastructure provisioning and system configuration using Ansible.
* Apply Infrastructure as Code (IaC) concepts for cloud environments.

**Tasks Performed:**

* Installed and configured Ansible on a control node.
* Defined inventory files to manage multiple target hosts.
* Wrote Ansible playbooks to automate tasks like installing packages, starting services, and configuring files.
* Tested idempotency and error handling of playbooks.
* Simulated deployment of a web server stack using Ansible roles.

**Tools/Technologies Used:**

* Ansible CLI
* YAML (for writing playbooks)
* Ubuntu/Linux servers (as managed nodes)
* SSH (for remote execution)

**Outcome:**

* Developed automation scripts to streamline infrastructure management.
* Demonstrated knowledge of Ansible architecture, inventory management, and playbooks.
* Enhanced deployment speed, consistency, and reduced human error in system configuration.
* Laid the foundation for implementing full DevOps pipelines in cloud environments.

