DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An autonomous Institution affiliated by VTU, Belagavi)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore-560078

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Lab Manual

Of

Cloud Computing and Virtualization

Semester: VI
Course Code: 22CD62
Credit: 4

Faculty In charge: Dr.Aarthi

(Associate Professor)

SNO.	PROGRAM
1.	**Understand the components and layers of the NIST Cloud Computing Reference Architecture.**
2.	**Implement and configure virtual machines using a hypervisor.**
3.	**Implement storage virtualization and manage virtual clusters.**
4.	**Introduction to Docker and its components.**
5.	**Deploy applications using Kubernetes for orchestration.**
6.	**Deploy applications on public cloud platforms (AWS, Azure).**
7.	**Understand IAM challenges and implementation.**
8.	**Identify and mitigate virtualization-specific security threats.**
9.	**Design a hybrid cloud architecture combining on-premises and public cloud resources.**
10.	**Optimize cloud infrastructure performance and monitor resource utilization.**

Program 1: Understand the components and layers of the NIST Cloud Computing Reference Architecture.

Theory:

1. **NIST Cloud Model:** Defined five essential characteristics, three service models (IaaS, PaaS, SaaS), and four deployment models (Public, Private, Hybrid, Community).
2. **Key Components:** Included cloud consumer, provider, broker, auditor, and carrier, forming the cloud ecosystem.
3. **Cloud Service Interactions:** Described how different stakeholders interacted within cloud environments to ensure security, scalability, and service delivery.

Experiment:

1. **Studied** the official NIST Cloud Computing Reference Architecture document to understand its components and functions.
2. **Identified** the roles of cloud consumer, provider, broker, auditor, and carrier in a cloud environment.
3. **Explored** real-world cloud service providers (AWS, Azure, GCP) and mapped their architecture to NIST's framework.
4. **Examined** how different service models (IaaS, PaaS, SaaS) were implemented by cloud providers.
5. **Analyzed** the deployment models and observed how organizations selected public, private, hybrid, or community cloud solutions.
6. **Compared** security, governance, and compliance practices in cloud platforms with NIST's recommendations.
7. **Reviewed** case studies of companies implementing cloud solutions and aligned them with NIST's architectural principles.
8. **Documented** findings and created a summary report on how modern cloud providers align with the NIST model.

Program 2: Implement and configure virtual machines using a hypervisor.

Theory:

1. **QEMU-KVM Overview:** QEMU is a hardware emulator, and KVM (Kernel-based Virtual Machine) enables efficient virtualization on Linux.
2. **Libvirt and User Permissions:** Libvirt provides tools for managing VMs, requiring user permissions for access.
3. **Running a Virtual Machine:** After installing QEMU-KVM, an Alpine Linux ISO can be booted in a virtualized environment.

Experiment:

```
# Update the package list to ensure the latest versions are installed sudo apt
update

# Install QEMU, KVM, and necessary virtualization tools
sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virt-
manager

# Add the current user to the libvirt group for VM management permissions   sudo
usermod -aG libvirt $(whoami)

# Enable and start the libvirt service to manage virtual machines   sudo
systemctl enable --now libvirtd

# Download the Alpine Linux ISO image to the Downloads directory
wget https://dl-cdn.alpinelinux.org/alpine/v3.21/releases/x86_64/alpine-standard3.21.3-
x86_64.iso -P ~/Downloads

# Start a QEMU virtual machine with the Alpine Linux ISO, booting from the CD-ROM and
allocating 512MB RAM
qemu-system-x86_64 -boot d -cdrom ~/Downloads/alpine-standard-3.21.3-
x86_64.iso -m 512
```

Program 3: Implement storage virtualization and manage virtual clusters.

Theory:

1. **Storage Virtualization Overview:** QEMU-IMG is a disk image utility that allows creating, converting, and managing virtual disks for QEMU-based virtual machines.
2. **Disk Formats:** QEMU supports various disk formats, including **qcow2** (efficient, supports snapshots) and **raw** (faster but larger in size).
3. **Virtual Machine Storage Management:** A virtual disk is created using qemu-img, attached to a virtual machine, and managed using Linux utilities.

Experiment:

Storage Virtualization tool: qemu-img

Create a storage image

$ qemu-img create -f qcow2 virtual_disk.qcow2 40G

Boot the VM with the disk

$ qemu-system-x86_64 -boot d -cdrom ~/Downloads/alpine-standard-3.21.3x86_64.iso -m 256 -smp 2 -drive file=virtual_disk.qcow2,format=qcow2

Check the disc in the VM

fdisk -l

Network Virtualization

ip link add name br0 type bridge

ip link set br0 up

ip addr add 192.168.1.100/24 dev br0 # Adjust the IP as necessary

mkdir /etc/qemu

vim.timy /etc/qemu/bridge.conf

-- add the contents allow
br0

- Run the VM with the network

qemu-system-x86_64 -boot d -cdrom ~/Downloads/alpine-standard-3.21.3x86_64.iso -m 256 -smp 2 -drive file=virtual_disk.qcow2,format=qcow2 -netdev bridge,id=net0,br=br0 -device e1000,netdev=net0

Program 4: Introduction to Docker and its components.

Theory:

1. Docker is a containerization platform that allows developers to package applications with all their dependencies into a standardized unit called a container.
2. **Components of Docker:**
   Docker Engine: Runs and manages containers.
   Docker Images: Pre-configured environments for running applications.
   Docker Containers: Lightweight, standalone, and executable packages of software. **Docker Hub:** A repository for sharing and downloading container images.
3. Eliminates compatibility issues, enables rapid deployment and scalability.

Experiment:

Steps to Install Docker

To install Docker on Ubuntu, follow these steps:

Step 1: Update Your System

Before installing Docker, it's good practice to update the package index to make sure you have the latest versions of the repositories:

```
sudo apt update
```

Step 2: Install Required Dependencies

Next, install some dependencies that allow apt to use repositories over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Step 3: Add Docker’s Official GPG Key

Now, you need to add Docker's official GPG key to ensure that the Docker packages are authentic:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

Step 4: Add Docker’s Official APT Repository

Add Docker’s official repository to your system’s software sources:

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
```

/etc/apt/sources.list.d/docker.list > /dev/null

Step 5: Update the Package Index Again

Now, update the package index again to include Docker's official repository:

sudo apt update

Step 6: Install Docker

Now that everything is set up, you can install Docker:

sudo apt install docker-ce

To run a simple "Hello World" Python application on Docker, you'll need to create a basic Python script and then package it into a Docker container. Here's how you can do it step by step:

Steps:

Create a directory for your Python project: Start by creating a new directory where you will store your Python script and Docker configuration files. mkdir python-hello-world

cd python-hello-world

Create the Python script: Inside the python-hello-world directory, create a Python script named app.py with the following content:

app.py

print("Hello, World!")

Create a Dockerfile: In the same directory, create a file named Dockerfile (without an extension) to define the Docker image for your application. The Dockerfile should look like this:

Use the official Python base image

FROM python:3.9-slim

Set the working directory inside the container

WORKDIR /app

Copy the Python script into the container COPY
app.py /app/

Run the Python script when the container starts CMD
["python", "app.py"]

Explanation of the Dockerfile:

FROM python:3.9-slim: This tells Docker to use the official Python 3.9 slim image as the base.

WORKDIR /app: Sets /app as the working directory inside the container.

COPY app.py /app/: Copies the app.py file from your local machine into the container.
CMD ["python", "app.py"]: Specifies the command to run the Python script when the container starts.

Build the Docker image: Now, you need to build the Docker image from the Dockerfile. In the terminal, run the following command from within the python-hello-world directory:

docker build -t python-hello-world . This will build the Docker image
with the name python-hello-world.

Run the Docker container: After building the image, you can run the container with the following command:

docker run python-hello-world You
should see the output:

Hello, World!

Stop the container: If the container doesn't stop automatically after execution (it depends on the Docker settings), you can stop it manually by running:

docker ps # Get the container ID docker stop
<container_id> # Stop the container

Program 5: Deploy applications using Kubernetes for orchestration.

Theory:

1. **Docker Desktop with Kubernetes on Windows**:
   Docker Desktop provides an integrated Kubernetes environment that allows developers to deploy containerized applications locally. Enabling Kubernetes via Docker Desktop offers a single-node cluster for testing and development.
2. **Namespaces in Kubernetes**:
   Namespaces in Kubernetes allow the division of cluster resources between multiple users or projects. This helps in organizing and isolating resources logically.
3. **WordPress and MySQL Deployment**:
   WordPress is a PHP-based CMS that typically uses MySQL as its backend database. Using Kubernetes YAML configuration files, you can deploy both services within a namespace.

Experiment:

#Download Required Files

- Download wordpress.yaml and mysql.yaml and save them in a folder (e.g., C:\Users\<YourName>\Desktop\CCV Lab\).

#Install Docker Desktop

- Download from: https://www.docker.com/products/docker-desktop/
- Install and restart your system if required.

#Enable Kubernetes in Docker Desktop

- Open Docker Desktop.
- Go to Settings > Kubernetes.
- Check Enable Kubernetes.
- Click Apply & Restart.
- Wait for the Kubernetes cluster to be running (check in the Docker UI).

#Open Terminal (inside Docker Desktop)

Create a Kubernetes Namespace

kubectl create namespace wordpress

```
namespace/wordpress created
```

#Apply the YAML files

kubectl apply -n wordpress -f "C:\Users\<YourName>\Desktop\CCV Lab\mysql.yaml"

```
secret/mysql-pass created
deployment.apps/mysql created
service/mysql created
```

kubectl apply -n wordpress -f "C:\Users\<YourName>\Desktop\CCV Lab\wordpress.yaml"

```
persistentvolumeclaim/wordpress-pvc created
deployment.apps/wordpress created
service/wordpress created
```

Check Services

kubectl get svc -n wordpress

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
mysql	ClusterIP	10.103.46.218	<none>	3306/TCP	85s
wordpress	LoadBalancer	10.110.96.54	localhost	8015:30879/TCP	110s

Access WordPress

- Open your browser and go to:
  http://localhost:<NodePort>
  (Replace <NodePort> with the port shown next to WordPress service, e.g., 32750)

Program 6: Deploy applications on public cloud platforms (AWS, Azure).

Theory:

- **Virtualization**: Virtualization allows running multiple virtual systems on a single physical machine. In this program, tools like Node.js and Strapi are installed in virtual environments managed by the system or cloud provider.
- **Cloud Deployment with Strapi**: Strapi is an open-source headless CMS built on Node.js. It enables users to create and manage content APIs quickly. The application is deployed on Strapi Cloud, which provides a public hosting environment for global accessibility.
- **Networking**: Deploying to the Strapi Cloud enables external users to interact with the application over the internet using HTTP/HTTPS protocols, demonstrating basic cloud networking concepts.
- **Storage**: The application data (like content types, media, and configurations) is stored on cloud-managed persistent storage, ensuring data availability and reliability.

Experiment:

#Install Node.js, Download and install Node.js from the official link:

https://nodejs.org/dist/v22.16.0/node-v22.16.0-x64.msi

#Verify Node.js and NPX installation

#Open Command Prompt and run the following command to verify installation:

npx -v

#Create a new Strapi project

#Run the following command to install and create a new Strapi project:

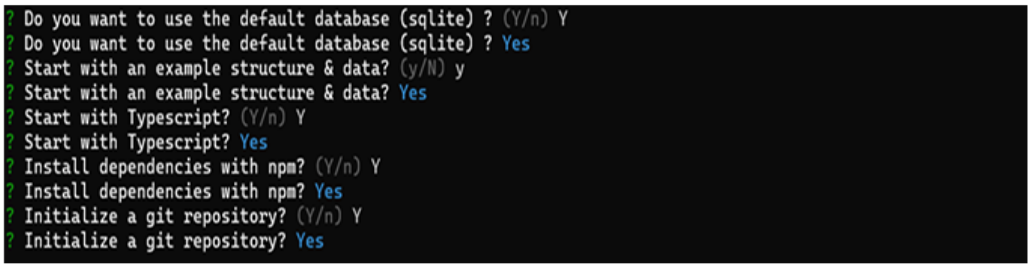
npx create-strapi@latest my-strapi-project

Follow the command prompts and select the default options when asked.

#Create the application locally

Choose Login / Sign Up option

Select the following command defaults



#Build the Strapi Admin Panel

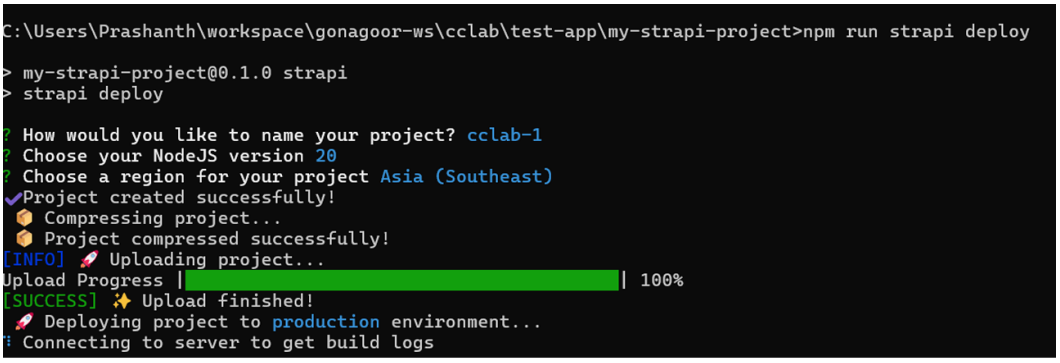
#Navigate to your project folder and run the following command:

npm run build

#Deploy the application to Strapi Cloud

#Use the following command to start the deployment process:

npm run strapi deploy



#During deployment, the CLI will prompt you to choose certain options:

- Project name
- Your appropriate version of nodejs, or follow recommended option in cmd.
- Region (choose a region closest to your users)-ASIA
- Environment (e.g., production)
- Accept defaults where applicable unless otherwise instructed.

#Copy the Cloud URL after successful deployment

- After deployment, a public URL will be provided.
- Save this URL — it will be used for access and in the load balancer script (in the hybrid cloud experiment).

Program 7: Understand Users and IAM in a cloud setup

Theory:

1. **Need for Cloud Access Management:**
Organizations with multiple users accessing shared cloud infrastructure require a centralized system to manage access and prevent unauthorized usage.

2. **IAM System Overview:**
Identity and Access Management (IAM) in the cloud defines **resources** (e.g., VMs, GPUs, services) and governs how users interact with them.

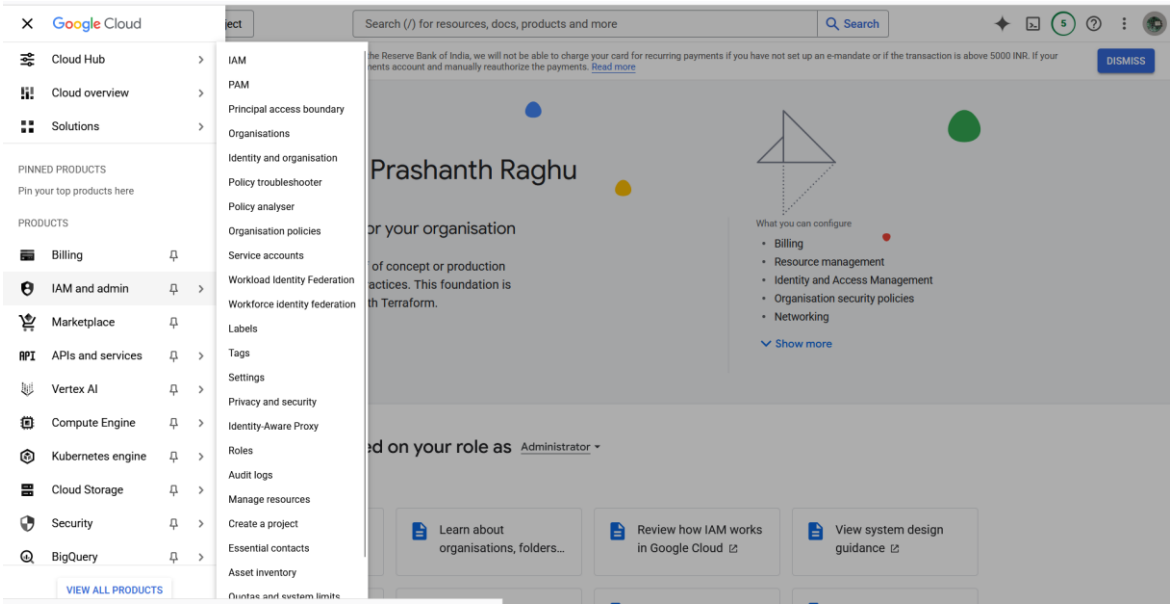
3. **Permissions and Operations:**
Operations like create, edit, or delete on resources are mapped to specific **permissions** (e.g., vm.create, gpu.use).

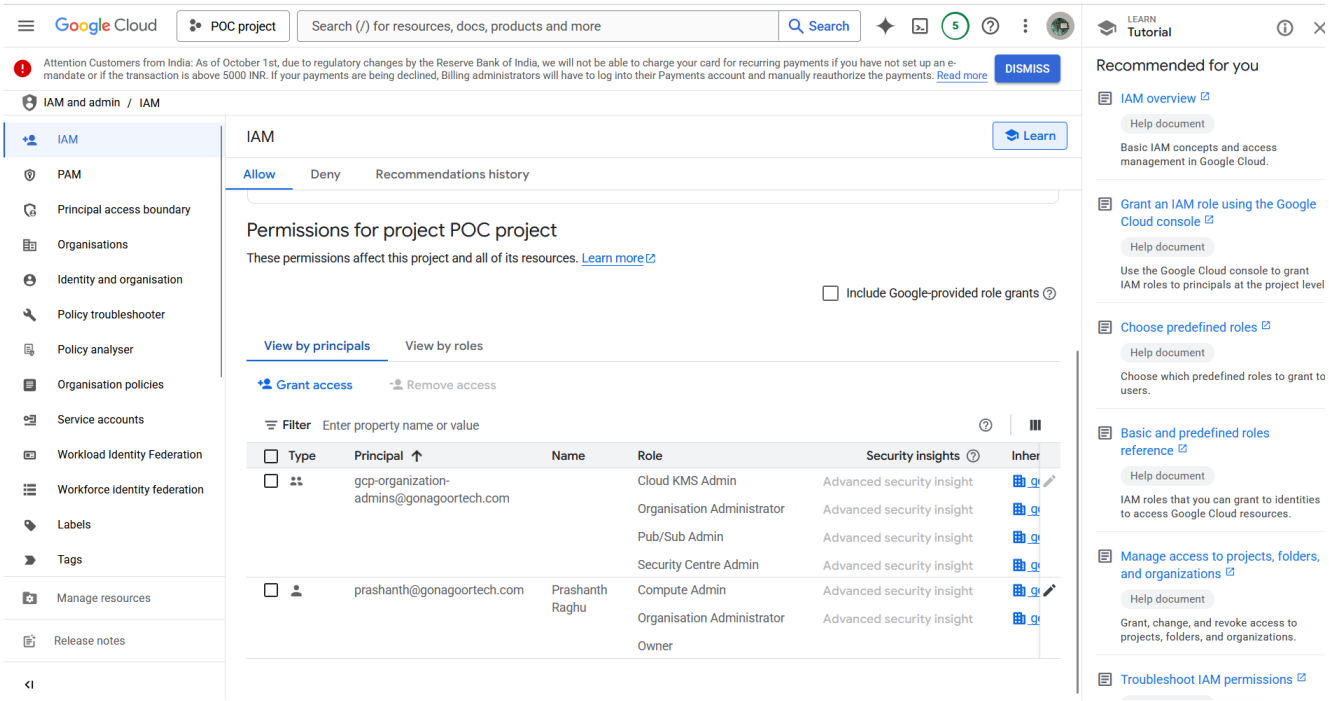
4. **Role-Based Access Control (RBAC):**
Permissions are grouped into **roles**, which act as wrappers for a set of actions. Roles simplify management by bundling access rules.

5. **User Role Assignment:**
Users are assigned roles based on their needs, ensuring they get the right access at the right time without overprovisioning.

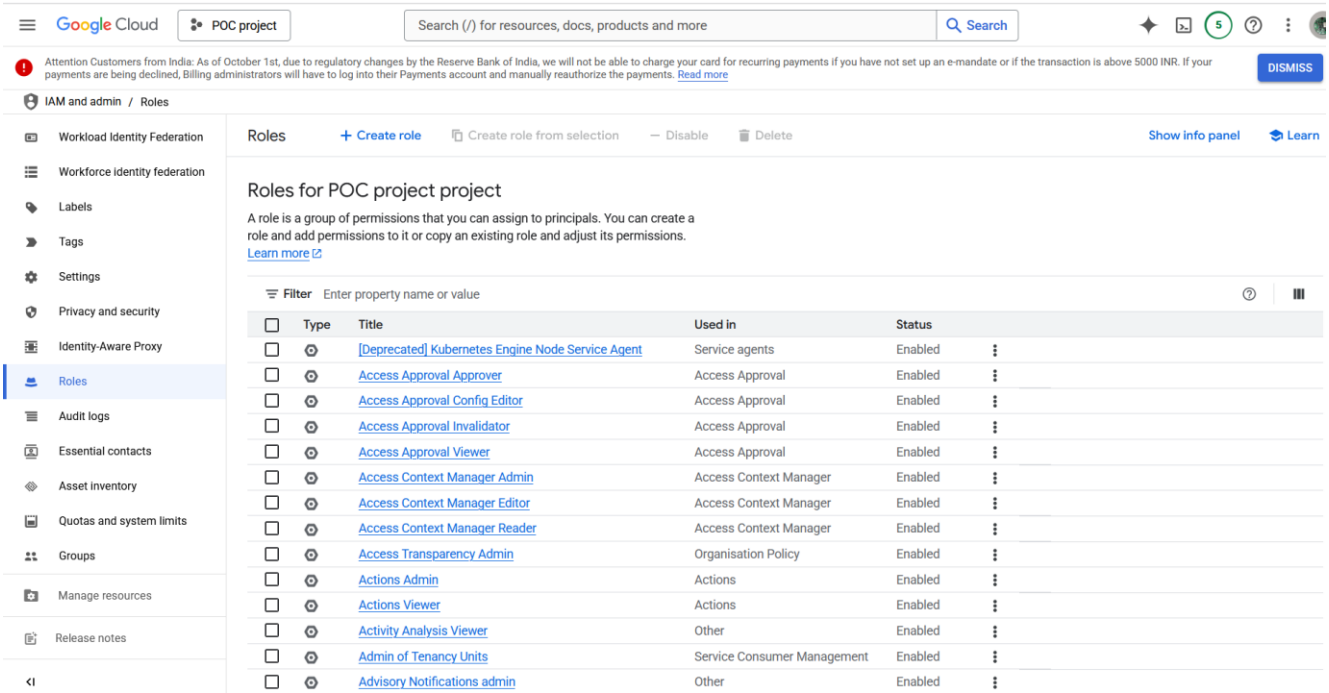
Experiment:

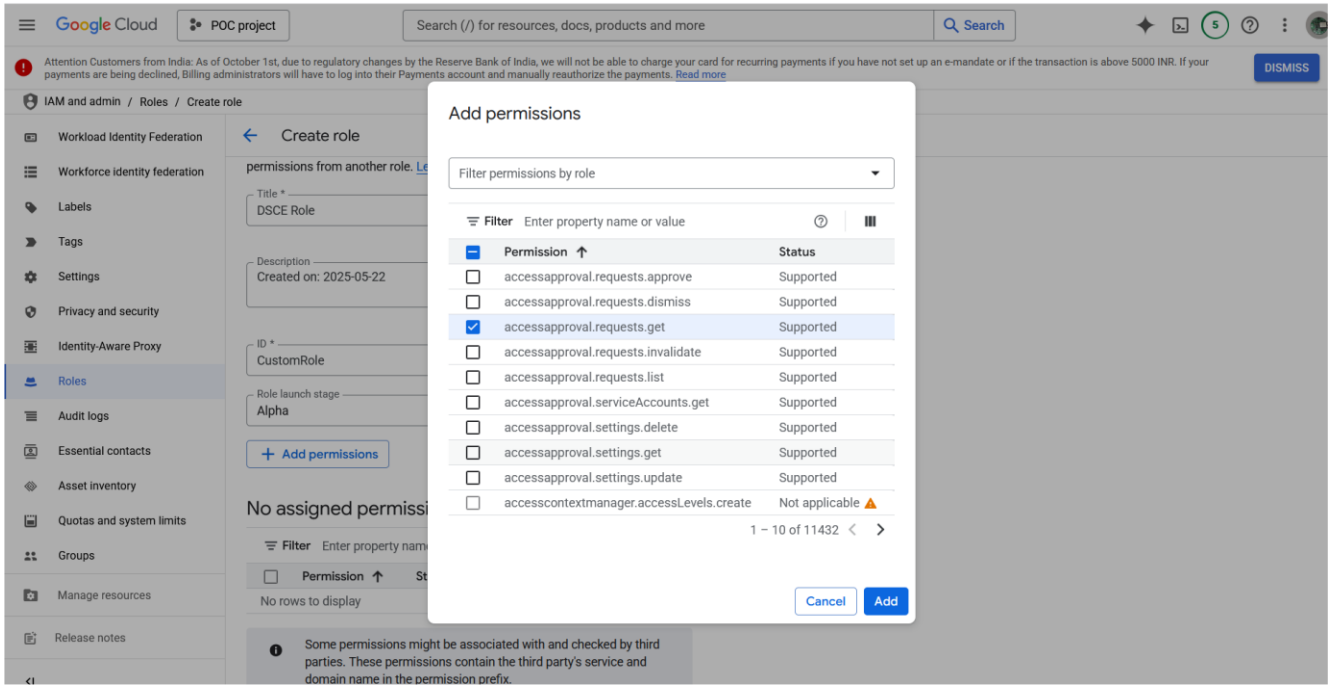
Login to google cloud console and open the IAM Console.



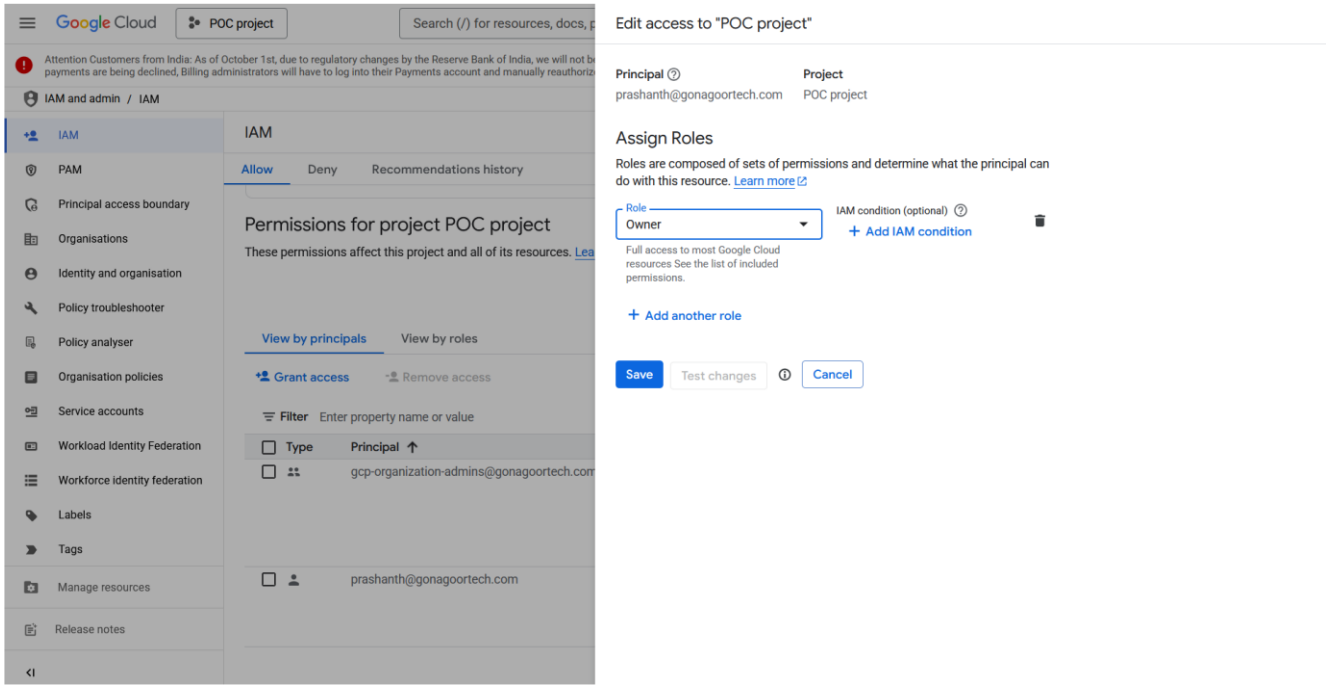


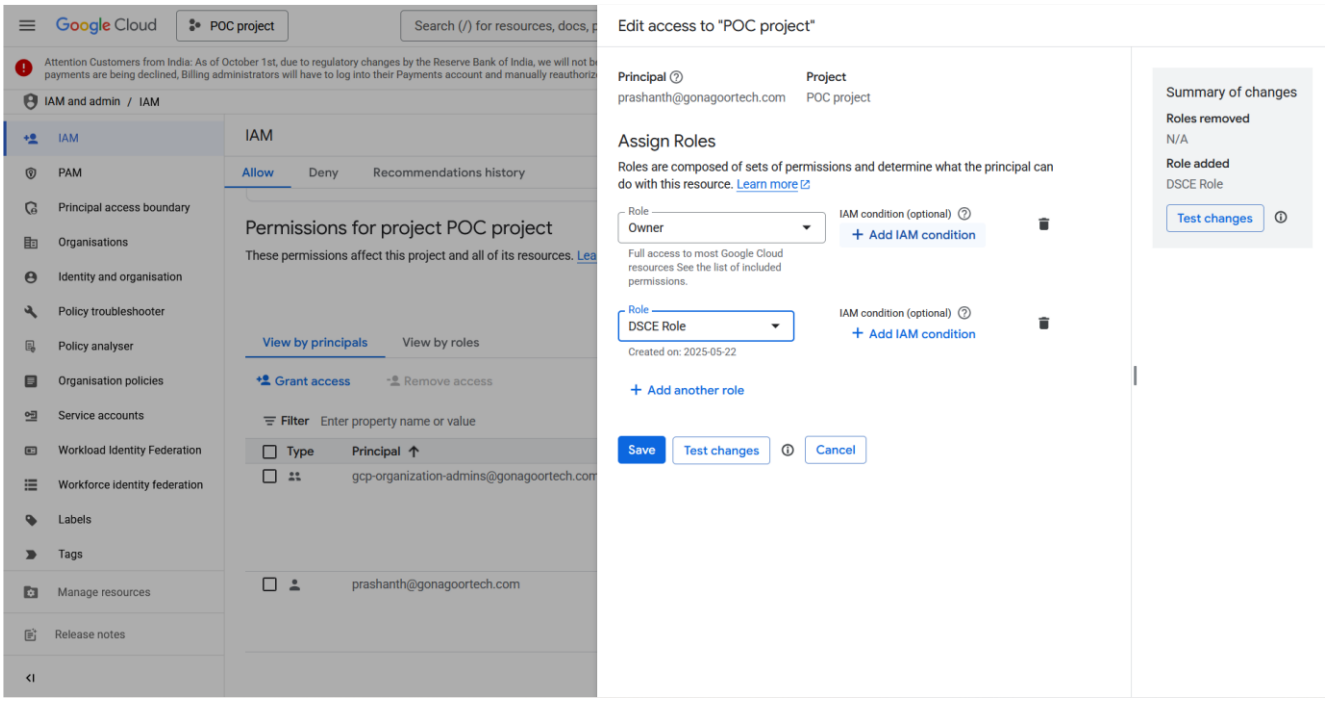
Check the available roles in the organization





Add roles to a user




Google Cloud
POC project
Search (/) for resources, docs, p
Attention Customers from India: As of October 1st, due to regulatory changes by the Reserve Bank of India, we will not b
payments are being declined, Billing administrators will have to log into their Payments account and manually reauthoriz
IAM and admin / IAM
IAM
PAM
Principal access boundary
Organisations
Identity and organisation
Policy troubleshooter
Policy analyser
Organisation policies
Service accounts
Workload Identity Federation
Workforce identity federation
Labels
Tags
Manage resources
Release notes
IAM
Allow
Deny
Recommendations history
Permissions for project POC project
These permissions affect this project and all of its resources. Lea
View by principals
View by roles
Grant access
Remove access
Filter
Enter property name or value
Type
Principal
gcp-organization-admins@gonagoortech.com
prashanth@gonagoortech.com
Edit access to "POC project"
Principal
prashanth@gonagoortech.com
Project
POC project
Assign Roles
Roles are composed of sets of permissions and determine what the principal can do with this resource. Learn more
Role
Owner
Full access to most Google Cloud resources See the list of included permissions.
IAM condition (optional)
Add IAM condition
Role
DSCE Role
Created on: 2025-05-22
IAM condition (optional)
Add IAM condition
Add another role
Save
Test changes
Cancel
Summary of changes
Roles removed
N/A
Role added
DSCE Role
Test changes

Program 8: Identify and mitigate virtualization-specific security threats.

Theory:

1. Container Vulnerabilities:
Containers can have security flaws that, if ignored, may compromise both the container and host system. Regular scanning is essential before production use.

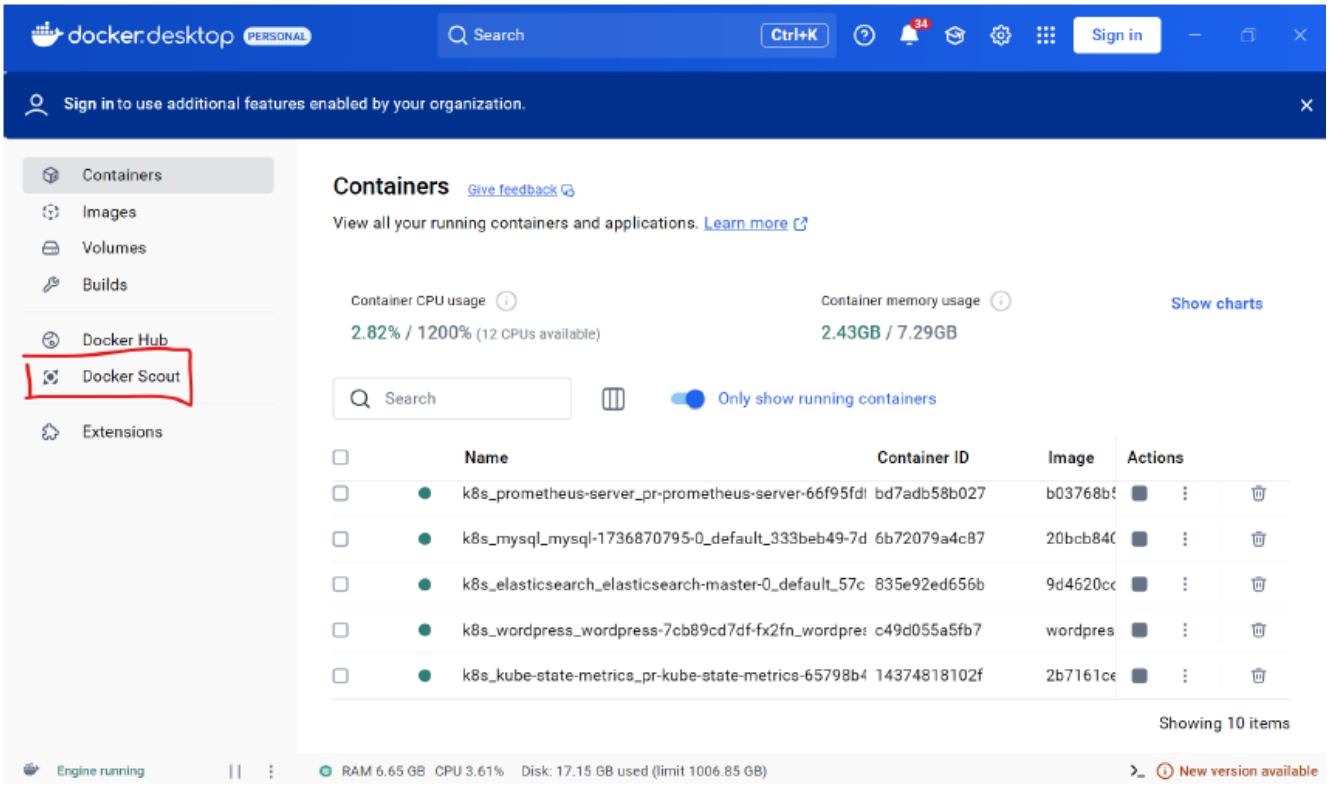
2. External Threats:
Risks can also come from outside the container, such as vulnerable runtimes or hypervisors, and must be addressed proactively.

3. Similarities to VMs and Languages:
Container vulnerabilities are comparable to those in virtual machines and programming languages, requiring similar security practices.

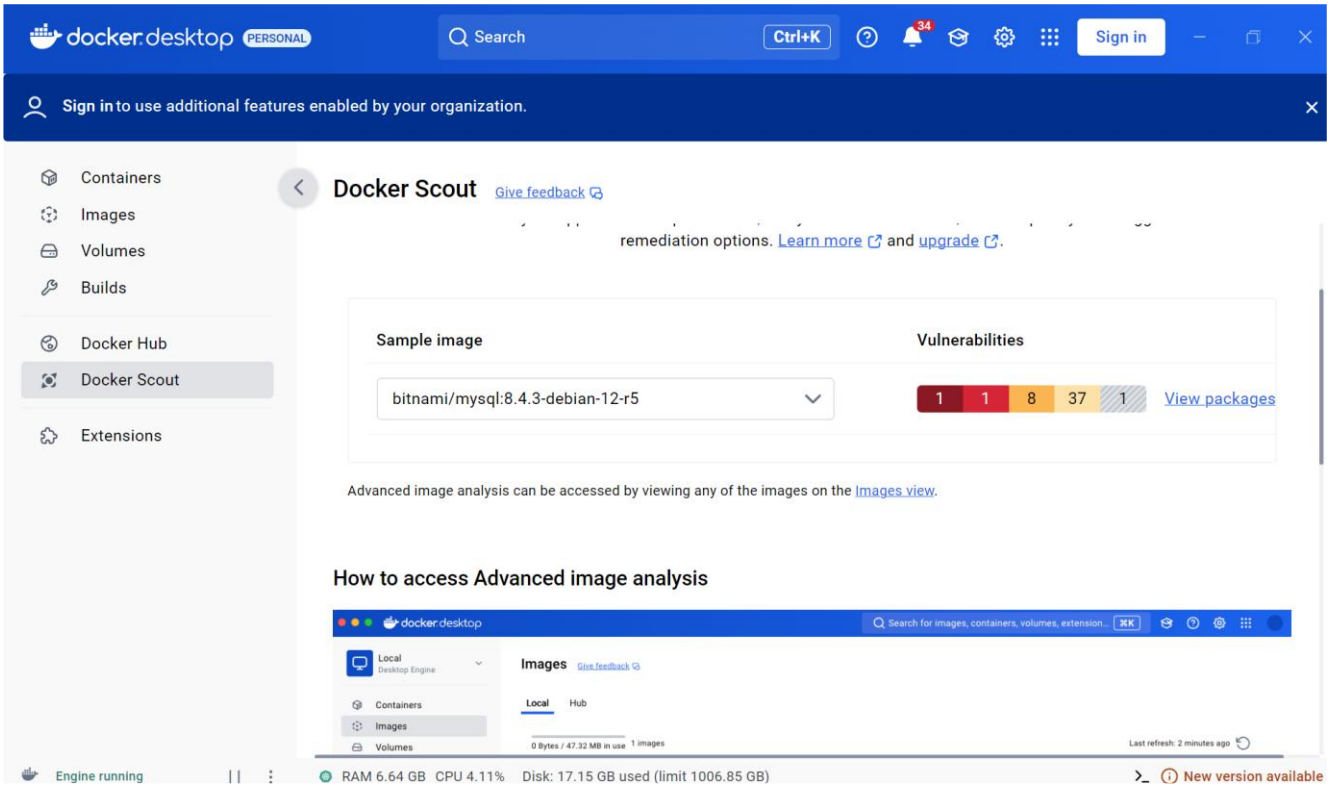
4. Using Docker Scout:
Open Docker Desktop and go to Docker Scout to scan images for known vulnerabilities before deployment.

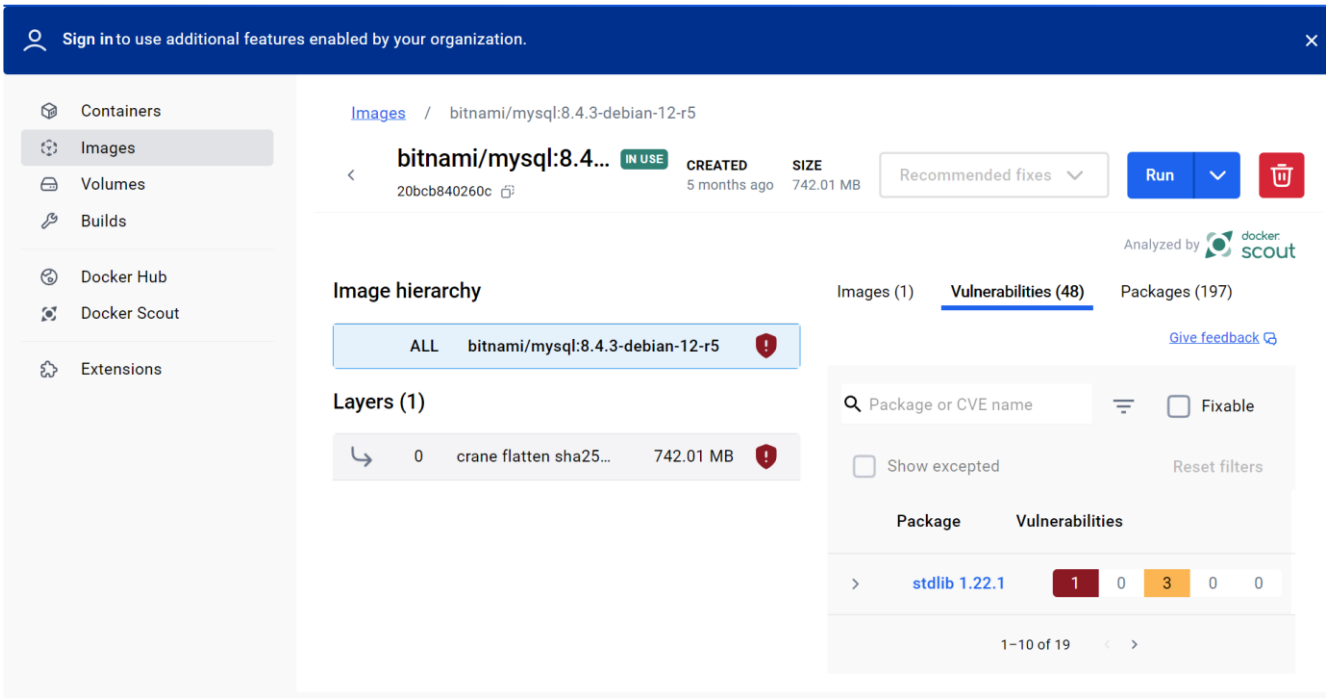
Experiment:

Open the docker desktop console and navigate to docker scout.



Select the MySQL image and click “Check CVC” (Container Vulnerabilities Check).





Review the list of vulnerabilities.

Explore details of each vulnerability. Focus on “Critical” and “High” severity issues.

Note down Severity level Affected package/version

Program 9: Design a hybrid cloud architecture combining on-premises and public cloud resources.

Theory:

1. **Hybrid Cloud Architecture**: A hybrid cloud setup integrates on-premises (local) resources with public cloud services to allow flexibility, scalability, and redundancy.
2. **Use of Load Balancer**: A simple Python-based load balancer using Flask alternates traffic between a local Strapi CMS instance and a cloud-hosted Strapi application.
3. **Strapi Cloud Deployment**: The local Strapi application is built and deployed both locally and to Strapi Cloud. The cloud URL serves as the public resource, while localhost acts as the private resource.
4. **Redirection Logic**: A round-robin redirection approach is used to alternate user traffic between the local and cloud URLs using simple counter logic.

Follow the instructions as per program 6 initially and copy the application URL provided by the strapi cloud. Copy the following load balancing script as load_balancer.py

Experiment:

```
# Importing Flask web framework and redirect function
from flask import Flask, redirect
# Creating a Flask app instance
app = Flask(__name__)
# List of backend targets - local and cloud instances of Strapi CMS
TARGETS = [
   "http://localhost:1337/admin",  # Localhost instance of Strapi
   "https://peaceful-confidence-a351f8d487.strapiapp.com/admin"
 # Strapi Cloud public URL
]
# Counter dictionary to keep track of requests for load balancing
request_counter = {"count": 0}
# Define route for root URL
@app.route("/")
def load_balancer():
```

```
    # Calculate which URL to redirect to using modulo for round-robin logic     index = request_counter["count"] % len(TARGETS)
      target_url = TARGETS[index]
      request_counter["count"] += 1
    # Redirect the user to selected backend target URL (302 by default)
      return redirect(target_url)
    # Entry point of the script
    if __name__ == "__main__":
            app.run(port=5000)   # Run the Flask app on localhost port 5000
```

Execution Steps:

```
#Run your local Strapi CMS
npm run build
npm run start
#Install flask and requests libraries
pip install flask requests
#Run the Python load balancer
python load_balancer.py
```

```
#Observe the following output on the terminal
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
* Running on http://127.0.0.1:5000
```

#Access the application in browser

Open: http://localhost:5000/

- First request → redirects to http://localhost:1337/admin
- Second request will point to public instance → redirects to https://peaceful-confidence-a351f8d487.strapiapp.com/admin
- Alternates with each refresh

Program 10: Optimize cloud infrastructure performance and monitor resource utilization.

Theory:

In cloud computing, optimizing infrastructure performance and monitoring resource utilization are essential to ensure efficient service delivery, cost-effectiveness, and scalability. Cloud infrastructure includes compute, storage, and network resources that support applications and services. As demand grows, proper resource allocation and monitoring become crucial to maintain system performance and availability.

1. Performance Optimization:
Cloud performance optimization involves tuning the virtualized environment to deliver high throughput, low latency, and balanced workloads. Techniques include:

- **Auto-scaling**: Automatically adding or removing resources based on demand.
- **Load Balancing**: Distributing traffic across multiple servers to prevent overload.
- **Caching**: Storing frequently accessed data in memory to reduce access time.
- **Resource Allocation**: Assigning appropriate virtual CPU, memory, and disk I/O based on workload requirements.

2. Resource Utilization Monitoring:
Monitoring involves tracking metrics such as CPU usage, memory consumption, disk I/O, and network traffic to ensure optimal operation and to detect anomalies or bottlenecks. Tools like **Prometheus**, **Grafana**, **Amazon CloudWatch**, or **Azure Monitor** provide real-time dashboards and alerts.

3. Benefits of Optimization and Monitoring:

- Improved application performance and user experience.
- Reduced operational costs through efficient resource use.
- Early detection of faults and performance degradation.
- Better planning for capacity and future scaling.

4. Key Metrics to Monitor:

- **CPU and Memory Usage**: Indicates system load and resource sufficiency.
- **Network Bandwidth**: Helps analyze data transfer efficiency.
- **Disk I/O**: Essential for applications with heavy read/write operations.
- **Latency and Response Time**: Measures service responsiveness.

By combining optimization strategies with continuous monitoring, organizations can ensure that their cloud-based systems remain resilient, responsive, and cost-effective.