Multi-Cloud Architecture: Designing Enterprise Architecture for Multi-Cloud

Session 1: Multi-Cloud Setup and Access

Objectives:

- Understand multi-cloud principles
- Create cloud accounts (AWS, Azure, GCP)
- Install CLI tools
- Launch and access VMs
- Deploy web servers and verify access

🧩 What is Multi-Cloud?

Multi-cloud architecture is a strategy where an organization uses **two or more cloud computing platforms** (like **Amazon Web Services (AWS)**, **Microsoft Azure**, **Google Cloud Platform (GCP)**, **IBM Cloud**, etc.) simultaneously to distribute workloads and applications.

Unlike **hybrid cloud**, which combines **public and private** clouds, **multi-cloud** refers strictly to the **use of multiple public cloud providers**.

📦 Why Use Multi-Cloud?

Organizations adopt multi-cloud for reasons such as:

1. 🏗️ **Avoid Vendor Lock-in**
    - Reduces dependency on a single provider's infrastructure or tools.
    - Makes it easier to switch if one provider increases costs or changes policies.
2. 🔁 **Redundancy & High Availability**
    - Running applications across multiple clouds ensures business continuity even if one provider experiences an outage.

3. 🌍 **Geographic Reach**
    - Different cloud providers have data centers in different regions.
    - Applications can be deployed closer to users for lower latency.
4. 💸 **Cost Optimization**
    - Organizations can choose the most cost-effective cloud provider for each service (compute, storage, database, etc.).
5. ⚙️ **Best-of-Breed Services**
    - Allows using the most powerful or specialized services from each provider.
        - Example: GCP's BigQuery for analytics, AWS Lambda for serverless compute.

📊 Example of a Multi-Cloud Setup

Component	Cloud Provider
Web Frontend	Google Cloud
Backend Services	AWS Lambda
Database	Azure SQL Database
CI/CD Pipeline	GitHub Actions + AWS ECS
Monitoring	Prometheus + Grafana (self-hosted on any cloud)

🚧 Challenges of Multi-Cloud

1. **Complex Management** – Different providers have different interfaces, APIs, billing systems, and security models.
2. **Data Transfer Costs** – Moving data between clouds can be expensive.
3. **Skill Requirements** – Teams need knowledge of multiple cloud platforms.
4. **Security** – Enforcing consistent security and compliance across providers can be hard.

✅ Multi-Cloud Use Cases

- **Disaster Recovery**: Data replicated across providers.

- **Global SaaS Deployment**: Serving users closer to their region via multiple clouds.
- **Mergers/Acquisitions**: Companies using different cloud providers can continue operating without immediate migration.

🧠 Key Concepts to Understand in Multi-Cloud:

- **Inter-cloud networking**
- **Identity and Access Management (IAM) across clouds**
- **Observability and unified monitoring**
- **Multi-cloud DevOps workflows**
- **Data synchronization and governance**

✅ II. Account Creation (Multi-Cloud Setup)

Setting up cloud accounts on **AWS**, **Azure**, and **Google Cloud Platform (GCP)** is the first step in any hands-on multi-cloud architecture project. Each provider offers a **free tier** or **trial credits**, allowing you to explore and deploy services with minimal or no cost initially.

🌩️ Amazon Web Services (AWS)

🔗 URL: https://aws.amazon.com

🪜 Steps to Create an AWS Account:

1. **Go to AWS Home Page**
   - Click **"Create an AWS Account"** on the top right.
2. **Enter Credentials**
   - Provide a **valid email address** (used as root account), **secure password**, and **account name**.
3. **Billing Information**
   - Enter **credit/debit card** details.
   - AWS may make a small refundable transaction to verify the card.
4. **Identity Verification**

- Enter a phone number.
- You'll receive an **OTP (One-Time Password)** via SMS or call.

5. **Support Plan**
   - Choose the **Basic Support Plan** (free).
6. **Account Activation**
   - Once verified, account setup may take a few minutes.
   - You'll get access to the **AWS Management Console**.
7. **Free Tier Access**
   - AWS Free Tier includes:
     - 750 hours/month of EC2 (t2.micro/t3.micro)
     - 5GB of S3 storage
     - 25GB of DynamoDB
     - 1 million Lambda requests/month

☁️ Microsoft Azure

🔗 URL: https://portal.azure.com

🪜 Steps to Create an Azure Account:

1. **Visit Azure Portal**
   - Click **"Start Free"** to begin the setup.
2. **Sign in or Create a Microsoft Account**
   - Use an existing **Microsoft email** (like Outlook) or create a new one.
3. **Identity Verification**
   - Enter your **phone number** and verify using an OTP.
4. **Billing Information**
   - Provide **card details**. A ₹2 or $1 refundable charge may be applied for verification.
5. **Activate Trial**
   - Get **₹13,300 / $200 free credit** for the first 30 days.

6. **Free Tier Access**
    - Includes:
        - B1S VM: 750 hours/month
        - 5GB Blob storage
        - 250GB SQL DB/month
        - 1 million Azure Functions executions/month

🌐 Google Cloud Platform (GCP)

🔗 URL: https://console.cloud.google.com

🪜 Steps to Create a GCP Account:

1. **Visit GCP Console**
    - Sign in with your **Google Account** or create a new one.
2. **Billing Profile Setup**
    - Select **country**, **currency**, and provide **credit/debit card** info.
3. **Free Trial Activation**
    - Accept terms to get **$300 credit** for 90 days.
4. **Complete Onboarding**
    - Optionally, select preferences like organization type, usage plans (e.g., learning, development).
5. **Free Tier Access**
    - Always-free GCP products include:
        - 1 f1-micro VM/month (US regions)
        - 5GB Cloud Storage
        - 1GB Outbound Network egress/month
        - 2 million Cloud Functions invocations/month

🔐 ⚠️ Important Notes for All Cloud Accounts:

- You **must provide valid billing info** even if you plan to use free services.

- Set **budgets and alerts** to prevent accidental charges.
- Always **shut down resources** after use (especially VMs).
- Multi-cloud setups require **IAM policies** for each cloud (e.g., AWS IAM, Azure RBAC, GCP IAM).

🧰 III. CLI Tool Installation for Multi-Cloud Setup

Command-Line Interface (CLI) tools allow you to **interact with cloud services** programmatically and automate infrastructure tasks. Installing and configuring CLI tools for **AWS**, **Azure**, and **Google Cloud Platform (GCP)** is essential for a seamless multi-cloud experience.

✅ 1. AWS CLI (Amazon Web Services Command Line Interface)

The AWS CLI lets you manage AWS services (like EC2, S3, IAM, etc.) from the terminal.

🔗 Installation Links:

- **Windows**:

  Download `.msi` installer:

  👉 https://awscli.amazonaws.com/AWSCLIV2.msi

- **Ubuntu/Linux**:

```
sudo apt update
sudo apt install unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "a
wscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- **macOS**:

```
brew install awscli
```

⚙️ Post-Installation Setup:

To configure AWS CLI:

```
aws configure
```

You'll be prompted to enter:

- **Access Key ID**
- **Secret Access Key**
- **Default region** (e.g., `us-east-1`)
- **Output format** (e.g., `json`, `table`, or `text`)

> ✅ Use the IAM user credentials generated from the AWS console.

✅ 2. Azure CLI

Azure CLI enables you to manage Azure resources (VMs, storage, web apps, etc.) from the command line.

🔗 Installation Links:

- **Windows**:

  Download from Microsoft:

  👉 https://aka.ms/installazurecliwindows

- **Ubuntu/Linux**:

  ```
  curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
  ```

- **macOS**:

  ```
  brew install azure-cli
  ```

⚙️ Login to Azure:

```
az login
```

This opens a browser window to log in to your Azure account securely. After login, you can use `az account show` to verify your default subscription.

✅ 3. Google Cloud CLI (gcloud) – Installation on All Platforms

The `gcloud` CLI lets you manage Google Cloud Platform (GCP) resources such as **Compute Engine**, **Cloud Storage**, **BigQuery**, **App Engine**, and more — all from the command line.

🚪 A. Installation on Windows

🔗 Download:

👉 https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe

🧭 Steps:

1. Run the downloaded `.exe` file.
2. In the installer, check:
   - ✅ *Install Bundled Python*
   - ✅ *Add gcloud to PATH*
   - ✅ *Install additional components* (optional)
3. Complete installation.

✅ Post-Install:

Open **Command Prompt** and run:

```
gcloud init
```

🐧 B. Installation on Ubuntu / Debian Linux

🧭 Steps:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates gnupg curl
```

```
# Add Google Cloud's GPG key
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-ke
y add -

# Add Cloud SDK repository
echo "deb http://packages.cloud.google.com/apt cloud-sdk main" | \
  sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

# Install the SDK
sudo apt update && sudo apt install google-cloud-sdk
```

✅ Post-Install:

```
gcloud init
```

🍎 C. Installation on macOS (via Homebrew)

🧭 Steps:

```
brew install --cask google-cloud-sdk
```

If the SDK does not auto-load into your terminal, add this to your shell config (`.zshrc`, `.bash_profile`, etc.):

```
source "$(brew --prefix)/Caskroom/google-cloud-sdk/latest/google-cloud
-sdk/path.bash.inc"
```

✅ Post-Install:

```
gcloud init
```

⚙️ gcloud init – What It Does

Running `gcloud init`:

- Opens a browser for **Google account authentication**

- Lets you choose or create a **GCP project**
- Sets default **region** and **zone**
- Configures the CLI for use

🔐 Useful gcloud Commands

Command	Description
`gcloud --version`	Show installed version
`gcloud auth list`	List authenticated accounts
`gcloud config list`	Show current config (project, region, zone)
`gcloud components update`	Update the SDK and components

✅ IV. VM Deployment and Web Server Setup (with Troubleshooting and Updates)

This guide walks you through creating virtual machines on **AWS**, **Azure**, and **GCP**, installing **Apache web server**, and ensuring proper **SSH access** and **firewall configuration**.

☁️ A. AWS EC2 – VM + Apache Web Server

🔷 Steps:

1. **Go to**: https://console.aws.amazon.com/ec2
2. Click **Launch Instance**
3. Configure:
   - **Name**: `aws-web-vm`
   - **Image (AMI)**: Amazon Linux 2023 (Free Tier eligible)
   - **Instance Type**: `t2.micro`
   - **Key Pair**: Create/download a `.pem` file
   - **Firewall (Security Group)**:
     - ✅ Allow **SSH (port 22)**

- ✅ Allow **HTTP (port 80)**

4. Click **Launch Instance**

🔐 Connect via SSH:

Convert `.ppk` to `.pem` if needed using PuTTYgen, then:

```
chmod 400 my_key.pem
ssh -i my_key.pem ec2-user@<public-ip>
```

🔧 Install Apache:

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
```

🔗 **Test in browser**:

👉 `http://<public-ip>`

☁️ B. Azure Virtual Machine – Ubuntu + Apache

🔷 Steps:

1. **Go to**: https://portal.azure.com
2. Create VM:
   - **Image**: Ubuntu 20.04 LTS
   - **Size**: B1s (Free Tier eligible)
   - **Authentication**: Password or SSH Public Key
   - **Inbound ports**:
     - ✅ SSH (22)
     - ✅ HTTP (80)
3. Go to **Networking Tab**:
   - Configure **Virtual Network** or create new

- Ensure **Public IP** is assigned
- NSG (Firewall): Allow SSH and HTTP

🔐 Connect via SSH:

```
ssh azureuser@<public-ip>
```

If you get `Permission denied (publickey)`, ensure:

- You're using the correct username
- You use `ssh -i privatekey.pem azureuser@<public-ip>` if SSH key was used

🔧 Install Apache:

```
sudo apt update
sudo apt install apache2 -y
sudo systemctl start apache2
sudo systemctl enable apache2
```

🔗 **Test in browser**:

👉 `http://<public-ip>`

☁️ C. GCP Compute Engine – Ubuntu + Apache

🔷 Steps:

1. Go to: https://console.cloud.google.com/compute/instances
2. Click **Create Instance**
3. Configure:
   - **Name**: e.g., `gcp-web-vm`
   - **Region/Zone**: e.g., `us-central1-f`
   - **Machine Type**: `e2-micro` (Free Tier)
   - **Boot Disk**:
     - Click **Change**

    - OS: Ubuntu → Version: **20.04 LTS**
- **Firewall**:
    - ✅ Allow HTTP
    - ✅ Allow HTTPS

4. Click **Create**

⚙️ Setup gcloud CLI (if using terminal):

```
gcloud config set project <your-project-id>
gcloud config set compute/zone us-central1-f
```

Find your instance's project by checking:

```
gcloud compute instances list --project <project-id>
```

🔐 SSH via gcloud CLI:

```
gcloud compute ssh <instance-name> --zone=us-central1-f
```

If prompted:

- `gcloud` will auto-generate SSH keys and upload them to the instance

---

🔧 Install Apache:

```
sudo apt update
sudo apt install apache2 -y
sudo systemctl start apache2
sudo systemctl enable apache2
```

🔗 **Test in browser**:

👉 `http://<external-ip>`

🧯 Common Troubleshooting Summary

Issue	Fix
`.ppk` used in ssh command	Convert to `.pem` using PuTTYgen
Permission denied (publickey)	Use `-i private_key.pem`, ensure key matches the instance
Can't change boot disk	Must create a **new VM** to use different OS
gcloud: project not set	Use `gcloud config set project <project-id>`
SSH keys missing	Let `gcloud` auto-generate, or use `ssh-keygen` and reset keys
HTTP not accessible	Ensure **port 80 is allowed** in firewall/NSG settings

☁️ AWS EC2 – Ubuntu + Apache Web Server

```
chmod 400 my_key.pem              # Secures the private key file by restrictin
g access
ssh -i my_key.pem ec2-user@<ip>    # Connects to the EC2 instance using
the private key
sudo yum update -y              # Updates all installed packages
sudo yum install httpd -y       # Installs Apache HTTP server
sudo systemctl start httpd       # Starts the Apache service
sudo systemctl enable httpd      # Enables Apache to start on system boot
```

☁️ Azure VM – Ubuntu + Apache Web Server

```
ssh azureuser@<ip>              # SSH into the Azure VM using username a
nd public IP
sudo apt update                 # Updates Ubuntu's package lists
sudo apt install apache2 -y      # Installs Apache HTTP server
sudo systemctl start apache2      # Starts the Apache service
sudo systemctl enable apache2     # Enables Apache to start on boot
```

☁️ GCP Compute Engine – Ubuntu + Apache Web Server

```
gcloud config set project <project-id>      # Sets your current GCP project f
or gcloud CLI
gcloud config set compute/zone us-central1-f # Sets your default zone for
```

```
resources
gcloud compute ssh <instance-name> --zone=us-central1-f   # SSH into G
CP VM using gcloud
sudo apt update                  # Updates package index on the VM
sudo apt install apache2 -y       # Installs Apache server
sudo systemctl start apache2       # Starts Apache
sudo systemctl enable apache2      # Enables Apache to auto-start at boot
```

📌 Command Summary Table

Command	Purpose
`chmod 400 file.pem`	Protects your SSH key by restricting access
`ssh -i key.pem user@ip`	SSH into a cloud VM securely using the private key
`gcloud compute ssh`	SSH into a GCP VM using Google-managed SSH key setup
`sudo yum update -y / apt update`	Updates the system's package list
`sudo yum install httpd -y / apt install apache2 -y`	Installs Apache web server
`sudo systemctl start apache2/httpd`	Starts Apache web server
`sudo systemctl enable apache2/httpd`	Enables Apache to start automatically on reboot
`gcloud config set project`	Configures the active GCP project
`gcloud config set compute/zone`	Sets the default compute zone for GCP resources

✅ Session 2: Multi-Cloud Web App Deployment & DNS Routing

🧰 Pre-requisites

✅ GitHub Repository Setup

Recommended structure:

```
multi-cloud-webapp/
├── frontend/
```

```
│   └── index.html
├── backend/
│   ├── app.py
│   ├── config.py
│   ├── models.py
│   ├── requirements.txt
│   └── runtime.txt (optional)
├── sql/
│   └── init.sql
├── .github/
│   └── workflows/
│       └── azure-deploy.yml
├── README.md
└── .gitignore
```

> Push this to GitHub and use it for deployment via GitHub Actions.

✅ Cloud SQL Setup (Pre-Azure)

Before setting up the backend:

1. **Go to:** https://console.cloud.google.com/sql
2. **Create MySQL Instance**
3. Create DB:

```
CREATE DATABASE multicloud;
USE multicloud;
CREATE TABLE messages (
  id INT AUTO_INCREMENT PRIMARY KEY,
  content VARCHAR(255)
);
INSERT INTO messages (content) VALUES ('Hello from GCP'), ('Welco
me to Multi-Cloud');
```

4. **Users:** Create or use an existing user (`DB_USER`) with a password (`DB_PASS`)
5. **Authorized Networks:** Add Azure's outbound IP to allow access

6. **Copy Values:**
    - `DB_HOST`: Public IP of SQL instance
    - `DB_NAME`: `multicloud`

🧱 I. Architecture Overview

A decoupled application:

Layer	Platform	Service
Frontend	AWS	S3 Static Website
Backend	Azure	App Service (Flask API)
Database	GCP	Cloud SQL (MySQL)
DNS/CDN	Cloudflare	Domain routing + SSL

🌐 II. Frontend (AWS S3 Static Website)

frontend/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Multi-Cloud Demo</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstra
p.min.css" rel="stylesheet">
</head>
<body class="bg-light">
<div class="container mt-5">
  <h2 class="text-primary text-center">Multi-Cloud Architecture</h2>
  <ul id="messageList" class="list-group mt-4"></ul>
</div>
<script>
fetch('https://<azure-backend-url>/messages')
  .then(response => response.json())
  .then(data => {
    const list = document.getElementById('messageList');
    data.forEach(msg => {
      const li = document.createElement('li');
```

```
      li.className = 'list-group-item';
      li.innerText = msg.content;
      list.appendChild(li);
    });
  });
</script>
</body>
</html>
```

🚀 Hosting Steps

1. **Create S3 bucket** (name: `multicloud-frontend`)
2. **Disable Block Public Access** → Permissions → Edit
3. **Enable Static Website Hosting** → Set `index.html`
4. **Add Bucket Policy**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::multicloud-frontend/*"
    }
  ]
}
```

🔧 Troubleshooting

- 403 Access Denied? Check:
  - Bucket policy
  - File permissions (public read access)
  - Static hosting enabled

⬅️ BACK III. Backend (Azure App Service – Flask API)

🔧 Folder: `backend/`

`app.py`

```
from flask import Flask, jsonify
from config import Config
from models import db, Message

app = Flask(__name__)
app.config.from_object(Config)
db.init_app(app)

@app.route("/")
def home():
    return "<h3>Azure Backend Running</h3>"

@app.route("/messages")
def get_messages():
    messages = Message.query.all()
    return jsonify([{"id": m.id, "content": m.content} for m in messages])

@app.route("/health")
def health():
    return "OK", 200

if __name__ == '__main__':
    app.run()
```

`config.py`

```
import os
class Config:
    DB_USER = os.getenv("DB_USER")
    DB_PASS = os.getenv("DB_PASS")
    DB_HOST = os.getenv("DB_HOST")
    DB_NAME = os.getenv("DB_NAME")
```

```
    SQLALCHEMY_DATABASE_URI = f"mysql+pymysql://{DB_USER}:{DB_PA
SS}@{DB_HOST}/{DB_NAME}"
```

models.py

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.String(255), nullable=False)
```

requirements.txt

```
Flask==2.3.2
Flask-SQLAlchemy==3.0.3
pymysql==1.0.3
```

🛠️ Deployment on Azure

1. Go to **Azure Portal → App Services → Create**
2. Select:
   - Runtime: Python 3.10
   - OS: Linux
3. In **Configuration → Application Settings**, add:
   - `DB_USER`, `DB_PASS`, `DB_HOST`, `DB_NAME`
4. In **General Settings**:
   - Startup command:

     ```
     python app.py
     ```

5. Deploy using:
   - GitHub Actions (`azure-deploy.yml`)
   - FTP or ZIP deploy

🛢️ IV. GCP Cloud SQL (MySQL)

1. Instance created in **Session 1 / Pre-reqs**
2. Ensure **Public IP** is used for Azure to connect
3. Add Azure's **App Service outbound IP** to **Authorized networks**
4. Add SQL user and initialize database using `sql/init.sql`

🌍 V. DNS Routing with Cloudflare

🔧 Steps:

1. Go to https://cloudflare.com → Create account
2. Add your domain (e.g., `example.com`)
3. Change DNS at your registrar to Cloudflare nameservers
4. Add DNS records:

Type	Name	Value
A	app	AWS S3 endpoint IP (or CNAME to bucket)
A	api	Azure App Service IP or hostname

1. Go to **SSL/TLS → Full or Full (Strict)** for secure HTTPS

🧪 Testing & Validation

- Access frontend: `http://app.example.com`
- Access backend: `http://api.example.com/messages`
- Ensure:
  - CORS is handled if needed
  - APIs return data
  - SSL certs issued by Cloudflare

🧯 Troubleshooting Checklist

Issue	Fix
S3 Access Denied	Check public access, policy, permissions
Azure API not responding	Validate DB_HOST, DB_USER, DB_PASS
GCP SQL connection refused	Add Azure IP to Authorized Networks
Cloudflare not routing	Wait for DNS propagation, check SSL mode

Would you like this content exported as a **PDF or PowerPoint presentation**? I can also include an architecture diagram if needed.

✅ Session 3: CI/CD, Monitoring, and Cost Management

🔍 Objectives

By the end of this session, participants will:

- Set up a **CI/CD pipeline** using **GitHub Actions** to automate app deployment.
- Use cloud-native and custom tools to **monitor health and performance**.
- Configure **cost monitoring and budget alerts** to prevent unexpected cloud expenses.

I. CI/CD with GitHub Actions (Azure Web App Deployment)

This section demonstrates automated deployment of a Python web app (e.g., Flask) to **Azure Web App** using **GitHub Actions**.

✅ Steps:

1. Create a **Python Flask App** and commit it to GitHub.
2. In Azure Portal:
    - Create an **App Service (Web App)**.
    - Go to **Deployment Center → Get Publish Profile**.
3. In GitHub:
    - Go to **Repo → Settings → Secrets → Actions → New Repository Secret**.

- Name: `AZURE_WEBAPP_PUBLISH_PROFILE`, Value: paste content from `.PublishSettings` file.

4. Create the GitHub Actions workflow:

✅ **`.github/workflows/deploy.yml`**

```
yaml
CopyEdit
name: CI/CD Azure Deploy

on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
        with:
          app-name: <your-app-name>
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
          package: .
```

1. On every push to `main`, the app will be rebuilt and deployed to Azure.

II. Monitoring Your Application

A. Cloud-Native Monitoring Tools

Cloud Provider	Monitoring Tool	Key Features
AWS	CloudWatch	Metrics, logs, dashboards, alarms
Azure	Application Insights	Live metrics, usage tracking, failures
GCP	Operations Suite (Stackdriver)	Logs, metrics, tracing

B. Custom Monitoring with Prometheus + Grafana

Install Prometheus and Grafana on a Linux VM or EC2 instance:

```
bash
CopyEdit
sudo apt update
sudo apt install prometheus grafana -y
sudo systemctl enable --now prometheus
sudo systemctl enable --now grafana-server
```

- **Grafana UI:** Accessible at `http://<your-vm-ip>:3000` (default user: `admin` / password: `admin`)
- Add Prometheus as a data source.
- Import dashboards for CPU, memory, requests per second, etc.

III. Application Health Checks

Health checks are crucial for ensuring the application is running correctly.

Example for Flask:

```
python
CopyEdit
@app.route("/health")
def health():
```

```
    return "OK", 200
```

🔁 Uptime Monitoring Tools:

Use UptimeRobot:

- Add your app URL (`https://yourapp.azurewebsites.net/health`)
- Set check interval: Every 5 minutes
- Notification via: Email, Slack, Webhooks, etc.

IV. Cloud Cost Monitoring & Budget Alerts

A. AWS – Cost Explorer & Budgets

1. Go to **AWS Billing Console**
2. Navigate to **Budgets** → **Create Budget**
3. Define type (Cost/Usage), set thresholds
4. Configure notifications (email or SNS)

B. Azure – Cost Management & Budgets

1. Azure Portal → **Cost Management + Billing**
2. Create **Budget** based on subscription/resource group
3. Set limits (e.g., ₹1000/month) and alert rules

C. GCP – Budgets and Alerts

1. Open **Billing → Budgets & Alerts**
2. Choose project and set budget amount
3. Add email recipients or Pub/Sub notifications