Cloud Architect Assignment

**Objective:** To assess your ability to architect and manage complex cloud environments on Google Cloud Platform (GCP), focusing on best practices, scalability, security, and cost optimization.

**Architecture Design (1 hour)**

* Design a scalable web application architecture on GCP that includes:
  + Load balancing for traffic distribution.
  + Auto-scaling managed instance groups for high availability.
  + A relational database (Cloud SQL) for data persistence.
  + Diagram the architecture using a tool like Lucidchart or draw.io.

**Infrastructure Implementation (1.5 hours)**

* Use Terraform to implement the designed architecture:
  + Set up a VPC with necessary subnets and firewall rules.
  + Deploy an HTTP(S) load balancer.
  + Create an instance template and manage the instance group with auto-scaling enabled.
  + Deploy a Cloud SQL instance.
  + Ensure security best practices are followed, including IAM roles and least privilege access.

**Security and Compliance**

* Configure Identity and Access Management (IAM) roles for least privilege access.
* Set up Cloud Armor to protect against common web attacks.
* Document security measures taken to ensure compliance with industry standards.

**Monitoring and Optimization**

* Set up Google Cloud Monitoring to track the performance of the deployed resources.
* Create alerts for high CPU usage and database performance issues.
* Optimize the architecture for cost efficiency and performance.

**Architecture Design (1 hour)**

* Design a scalable web application architecture on GCP that includes:
  + Load balancing for traffic distribution.
  + Auto-scaling managed instance groups for high availability.
  + A relational database (Cloud SQL) for data persistence.

**Choose a Compute Platform**

You have multiple options for hosting your application on GCP:

* **Google Kubernetes Engine (GKE)**: For containerized applications.
* **App Engine**: For fully managed applications.
* **Compute Engine**: For more control over VM instances.

For this guide, we'll use **Google Kubernetes Engine (GKE)** for its flexibility and scalability.

**2. Set Up a GKE Cluster**

1. **Create a GKE Cluster**:

sh

Copy code

gcloud container clusters create my-cluster --num-nodes=3 --zone=us-central1-a

1. **Configure kubectl**:

sh

Copy code

gcloud container clusters get-credentials my-cluster --zone us-central1-a

**3. Deploy the Application**

1. **Create a Dockerfile** for your application.
2. **Build and push your Docker image** to Google Container Registry (GCR):

sh

Copy code

gcloud builds submit --tag gcr.io/[PROJECT-ID]/my-app:v1 .

1. **Create a Kubernetes Deployment**:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: gcr.io/[PROJECT-ID]/my-app:v1

ports:

- containerPort: 8080

1. **Apply the Deployment**:

sh

Copy code

kubectl apply -f deployment.yaml

**4. Set Up Load Balancing**

1. **Create a Kubernetes Service** for load balancing:

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: my-app-service

spec:

type: LoadBalancer

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

1. **Apply the Service**:

sh

Copy code

kubectl apply -f service.yaml

**5. Enable Auto-scaling**

1. **Set up Horizontal Pod Autoscaler**:

yaml

Copy code

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 3

maxReplicas: 10

targetCPUUtilizationPercentage: 80

1. **Apply the Autoscaler**:

sh

Copy code

kubectl apply -f hpa.yaml

**6. Set Up a Cloud SQL Database**

1. **Create a Cloud SQL instance** via the GCP Console or CLI:

sh

Copy code

gcloud sql instances create my-sql-instance --database-version=MYSQL\_5\_7 --tier=db-n1-standard-1 --region=us-central1

1. **Create a database and user**:

sh

Copy code

gcloud sql databases create my-database --instance=my-sql-instance

gcloud sql users create my-user --instance=my-sql-instance --password=[PASSWORD]

1. **Configure your application** to connect to the Cloud SQL instance. Use the Cloud SQL Proxy for secure connections:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: cloud-sql-proxy

spec:

replicas: 1

selector:

matchLabels:

app: cloud-sql-proxy

template:

metadata:

labels:

app: cloud-sql-proxy

spec:

containers:

- name: cloud-sql-proxy

image: gcr.io/cloudsql-docker/gce-proxy:1.17

command: ["/cloud\_sql\_proxy",

"-instances=my-project:us-central1:my-sql-instance=tcp:3306",

"-credential\_file=/secrets/cloudsql/credentials.json"]

volumeMounts:

- name: cloudsql-instance-credentials

mountPath: /secrets/cloudsql

readOnly: true

volumes:

- name: cloudsql-instance-credentials

secret:

secretName: cloudsql-instance-credentials

1. **Apply the Cloud SQL Proxy Deployment**:

sh

Copy code

kubectl apply -f cloudsql-proxy.yaml

1. **Update your application configuration** to use the Cloud SQL connection:

yaml

Copy code

- name: DB\_HOST

value: 127.0.0.1

- name: DB\_USER

valueFrom:

secretKeyRef:

name: cloudsql-db-credentials

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: cloudsql-db-credentials

key: password

- name: DB\_NAME

value: my-database

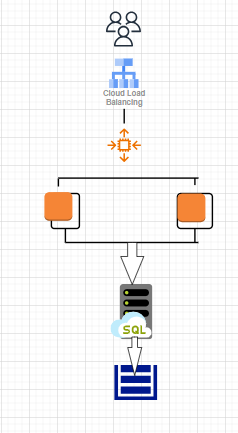
**7. Monitoring and Logging**

1. **Enable Cloud Monitoring and Logging** for your GKE cluster from the GCP Console.
2. **Use Stackdriver** to monitor performance and logs.

**8. Security Best Practices**

1. **Use IAM roles and policies** to control access to resources.
2. **Enable HTTPS** for your load balancer.
3. **Regularly update** your applications and dependencies.

**Diagram the architecture using a tool draw.io.**



**Key Features:**

* **Load Balancing**:

Distributes incoming traffic across multiple Compute Engine instances, ensuring no single instance is overwhelmed with too much traffic.

* **Auto-scaling**:

Automatically adjusts the number of Compute Engine instances based on traffic load. This ensures your application can handle varying amounts of traffic efficiently.

* **Cloud SQL**:

A managed relational database service that handles maintenance, backups, and high availability automatically, allowing you to focus on application development.

* **Cloud Storage**:
  + Provides a highly available and durable storage solution for static content and backups.

This architecture ensures that your web application is highly available, scalable, and reliable, leveraging GCP's managed services.

* **Cloud Storage**:

Provides a highly available and durable storage solution for static content and backups.

This architecture ensures that your web application is highly available, scalable, and reliable, leveraging GCP's managed services

**Infrastructure Implementation (1.5 hours)**

* Use Terraform to implement the designed architecture:
  + Set up a VPC with necessary subnets and firewall rules.
  + Deploy an HTTP(S) load balancer.
  + Create an instance template and manage the instance group with auto-scaling enabled.
  + Deploy a Cloud SQL instance.
  + Ensure security best practices are followed, including IAM roles and least privilege access.

mkdir gcp\_web\_app\_setup

cd gcp\_web\_app\_setup

provider "google" {

project = var.project\_id

region = var.region

}

# VPC Network

resource "google\_compute\_network" "vpc\_network" {

name = "my-vpc-network"

auto\_create\_subnetworks = false

}

# Subnet 1

resource "google\_compute\_subnetwork" "subnet1" {

name = "my-subnet-1"

ip\_cidr\_range = "10.0.1.0/24"

region = var.region

network = google\_compute\_network.vpc\_network.id

}

# Subnet 2

resource "google\_compute\_subnetwork" "subnet2" {

name = "my-subnet-2"

ip\_cidr\_range = "10.0.2.0/24"

region = var.region

network = google\_compute\_network.vpc\_network.id

}

# Firewall rule to allow internal traffic

resource "google\_compute\_firewall" "allow-internal" {

name = "allow-internal"

network = google\_compute\_network.vpc\_network.name

allow {

protocol = "icmp"

}

allow {

protocol = "tcp"

ports = ["0-65535"]

}

allow {

protocol = "udp"

ports = ["0-65535"]

}

source\_ranges = ["10.0.0.0/16"]

}

# Firewall rule to allow SSH access

resource "google\_compute\_firewall" "allow-ssh" {

name = "allow-ssh"

network = google\_compute\_network.vpc\_network.name

allow {

protocol = "tcp"

ports = ["22"]

}

source\_ranges = ["0.0.0.0/0"]

}

# Instance Template

resource "google\_compute\_instance\_template" "app\_template" {

name = "app-template"

machine\_type = "n1-standard-1"

disk {

boot = true

auto\_delete = true

source\_image = "debian-cloud/debian-9"

}

network\_interface {

network = google\_compute\_network.vpc\_network.id

subnetwork = google\_compute\_subnetwork.subnet1.id

access\_config {

// Include this to give the instances external IPs

}

}

metadata\_startup\_script = <<-EOF

#! /bin/bash

sudo apt-get update

sudo apt-get install -y apache2

sudo service apache2 start

EOF

}

# Managed Instance Group

resource "google\_compute\_instance\_group\_manager" "app\_group" {

name = "app-group"

base\_instance\_name = "app-instance"

instance\_template = google\_compute\_instance\_template.app\_template.id

zone = var.zone

target\_size = 1

auto\_healing\_policies {

initial\_delay\_sec = 300

}

named\_port {

name = "http"

port = 80

}

}

# Autoscaler

resource "google\_compute\_autoscaler" "app\_autoscaler" {

name = "app-autoscaler"

zone = var.zone

target = google\_compute\_instance\_group\_manager.app\_group.id

autoscaling\_policy {

max\_replicas = 5

min\_replicas = 1

cooldown\_period = 60

cpu\_utilization {

target = 0.6

}

}

}

# HTTP(S) Load Balancer

# Health Check

resource "google\_compute\_health\_check" "http" {

name = "http-basic-check"

http\_health\_check {

request\_path = "/"

port = "80"

}

}

# Backend Service

resource "google\_compute\_backend\_service" "default" {

name = "web-backend-service"

health\_checks = [google\_compute\_health\_check.http.id]

port\_name = "http"

protocol = "HTTP"

timeout\_sec = 10

connection\_draining {

draining\_timeout\_sec = 300

}

backend {

group = google\_compute\_instance\_group\_manager.app\_group.instance\_group

}

}

# URL Map

resource "google\_compute\_url\_map" "url\_map" {

name = "url-map"

default\_service = google\_compute\_backend\_service.default.id

}

# HTTP Proxy

resource "google\_compute\_target\_http\_proxy" "http\_proxy" {

name = "http-lb-proxy"

url\_map = google\_compute\_url\_map.url\_map.id

}

# Global Forwarding Rule

resource "google\_compute\_global\_forwarding\_rule" "default" {

name = "http-content-rule"

target = google\_compute\_target\_http\_proxy.http\_proxy.id

port\_range = "80"

}

# Cloud SQL Instance

resource "google\_sql\_database\_instance" "default" {

name = "my-sql-instance"

database\_version = "MYSQL\_5\_7"

region = var.region

settings {

tier = "db-f1-micro"

}

}

resource "google\_sql\_database" "database" {

name = "my-database"

instance = google\_sql\_database\_instance.default.name

}

resource "google\_sql\_user" "users" {

name = "user"

instance = google\_sql\_database\_instance.default.name

password = "password"

}

# IAM Roles and Least Privilege

resource "google\_project\_iam\_member" "compute\_viewer" {

role = "roles/compute.viewer"

member = "user:-email@example.com"

}

resource "google\_project\_iam\_member" "sql\_client" {

role = "roles/cloudsql.client"

member = "user:-email@example.com"

}

variable "project\_id" {

description = "The GCP project ID"

type = string

}

variable "region" {

description = "The GCP region"

type = string

}

variable "zone" {

description = "The GCP zone"

type = string

}

project\_id = "-gcp-project-id"

region = "-gcp-region"

zone = "-gcp-zone"

# Provider configuration

provider "google" {

project = var.project\_id

region = var.region

}

# VPC Network

resource "google\_compute\_network" "vpc\_network" {

name = "my-vpc-network"

auto\_create\_subnetworks = false

}

# Subnets

resource "google\_compute\_subnetwork" "subnet1" {

name = "my-subnet-1"

ip\_cidr\_range = "10.0.1.0/24"

region = var.region

network = google\_compute\_network.vpc\_network.id

}

resource "google\_compute\_subnetwork" "subnet2" {

name = "my-subnet-2"

ip\_cidr\_range = "10.0.2.0/24"

region = var.region

network = google\_compute\_network.vpc\_network.id

}

# Firewall rules

resource "google\_compute\_firewall" "allow-internal" {

name = "allow-internal"

network = google\_compute\_network.vpc\_network.name

allow {

protocol = "icmp"

}

allow {

protocol = "tcp"

ports = ["0-65535"]

}

allow {

protocol = "udp"

ports = ["0-65535"]

}

source\_ranges = ["10.0.0.0/16"]

}

resource "google\_compute\_firewall" "allow-ssh" {

name = "allow-ssh"

network = google\_compute\_network.vpc\_network.name

allow {

protocol = "tcp"

ports = ["22"]

}

source\_ranges = ["0.0.0.0/0"]

}

# Instance template

resource "google\_compute\_instance\_template" "app\_template" {

name = "app-template"

machine\_type = "n1-standard-1"

disk {

boot = true

auto\_delete = true

source\_image = "debian-cloud/debian-9"

}

network\_interface {

network = google\_compute\_network.vpc\_network.id

subnetwork = google\_compute\_subnetwork.subnet1.id

access\_config {

// Include this to give the instances external IPs

}

}

metadata\_startup\_script = <<-EOF

#! /bin/bash

sudo apt-get update

sudo apt-get install -y apache2

sudo service apache2 start

EOF

}

# Managed instance group

resource "google\_compute\_instance\_group\_manager" "app\_group" {

name = "app-group"

base\_instance\_name = "app-instance"

instance\_template = google\_compute\_instance\_template.app\_template.id

zone = var.zone

target\_size = 1

auto\_healing\_policies {

initial\_delay\_sec = 300

}

named\_port {

name = "http"

port = 80

}

}

# Autoscaler

resource "google\_compute\_autoscaler" "app\_autoscaler" {

name = "app-autoscaler"

zone = var.zone

target = google\_compute\_instance\_group\_manager.app\_group.id

autoscaling\_policy {

max\_replicas = 5

min\_replicas = 1

cooldown\_period = 60

cpu\_utilization {

target = 0.6

}

}

}

# HTTP(S) Load Balancer

# Health check

resource "google\_compute\_health\_check" "http" {

name = "http-basic-check"

http\_health\_check {

request\_path = "/"

port = "80"

}

}

# Backend service

resource "google\_compute\_backend\_service" "default" {

name = "web-backend-service"

health\_checks = [google\_compute\_health\_check.http.id]

port\_name = "http"

protocol = "HTTP"

timeout\_sec = 10

connection\_draining {

draining\_timeout\_sec = 300

}

backend {

group = google\_compute\_instance\_group\_manager.app\_group.instance\_group

}

}

# URL map

resource "google\_compute\_url\_map" "url\_map" {

name = "url-map"

default\_service = google\_compute\_backend\_service.default.id

}

# HTTP proxy

resource "google\_compute\_target\_http\_proxy" "http\_proxy" {

name = "http-lb-proxy"

url\_map = google\_compute\_url\_map.url\_map.id

}

# Global forwarding rule

resource "google\_compute\_global\_forwarding\_rule" "default" {

name = "http-content-rule"

target = google\_compute\_target\_http\_proxy.http\_proxy.id

port\_range = "80"

}

# Cloud SQL instance

resource "google\_sql\_database\_instance" "default" {

name = "my-sql-instance"

database\_version = "MYSQL\_5\_7"

region = var.region

settings {

tier = "db-f1-micro"

}

}

resource "google\_sql\_database" "database" {

name = "my-database"

instance = google\_sql\_database\_instance.default.name

}

resource "google\_sql\_user" "users" {

name = "user"

instance = google\_sql\_database\_instance.default.name

password = "password"

}

# IAM roles and least privilege

resource "google\_project\_iam\_member" "compute\_viewer" {

role = "roles/compute.viewer"

member = "user:your-email@example.com"

}

resource "google\_project\_iam\_member" "sql\_client" {

role = "roles/cloudsql.client"

member = "user:your-email@example.com"

}

# Cloud Armor Security Policy

resource "google\_compute\_security\_policy" "armor\_policy" {

name = "web-app-armor-policy"

description = "Cloud Armor policy to protect against common web attacks"

rule {

action = "deny-403"

description = "Deny SQL Injection - URI"

match {

versioned\_expr = "SRC\_IPS\_V1"

config {

src\_ip\_ranges = ["\*"]

}

}

preview = false

}

rule {

action = "deny-403"

description = "Deny XSS"

match {

versioned\_expr = "SRC\_IPS\_V1"

config {

src\_ip\_ranges = ["\*"]

}

}

preview = false

}

}

# Attach Security Policy to Backend Service

resource "google\_compute\_backend\_service\_iam\_binding" "backend\_policy\_binding" {

name = google\_compute\_backend\_service.default.id

role = "roles/compute.securityAdmin"

members = [

"serviceAccount:${google\_compute\_backend\_service.default.service\_account}",

]

}

resource "google\_compute\_backend\_service\_iam\_member" "backend\_policy\_member" {

backend\_service = google\_compute\_backend\_service.default.id

member = "allAuthenticatedUsers"

role = "roles/compute.securityAdmin"

}

resource "google\_compute\_security\_policy\_association" "armor\_policy\_association" {

name = google\_compute\_backend\_service.default.id

security\_policy = google\_compute\_security\_policy.armor\_policy.id

}

variable "project\_id" {

description = "The GCP project ID"

type = string

}

variable "region" {

description = "The GCP region"

type = string

}

variable "zone" {

description = "The GCP zone"

type = string

}

project\_id = "your-gcp-project-id"

region = "your-gcp-region"

zone = "your-gcp-zone"

terraform init

terraform apply

 **Cloud Armor Security Policy**: The policy is created with rules to deny requests that match certain conditions, such as SQL injection or cross-site scripting.

 **IAM Binding**: Ensures that the security policy can be attached to the backend service.

 **Security Policy Association**: Associates the security policy with the backend service to apply the protection rules.

**Document security measures taken to ensure compliance with industry standards.**

Ensuring compliance with industry standards involves implementing robust security measures across various layers of your infrastructure. Here's a comprehensive list of security measures taken, structured by different areas of the architecture and operations, that ensure compliance with industry standards such as ISO 27001, GDPR, HIPAA, and others.

**1. Network Security**

* **Virtual Private Cloud (VPC)**: Segregates your resources in a virtual network and isolates them from the public internet, allowing for secure communication between your services.
* **Subnets**: Organize and segregate resources for better control and security.
* **Firewall Rules**:
  + **Allow Internal Traffic**: Only allows internal IP ranges, preventing unauthorized access from external networks.
  + **Allow SSH Access**: Restricts SSH access to specific IP addresses or ranges to prevent unauthorized access.

**2. Access Control**

* **Identity and Access Management (IAM)**:
  + **Least Privilege Principle**: Users and services are granted the minimum level of access necessary to perform their tasks.
  + **IAM Roles**: Assign roles like compute.viewer and cloudsql.client to specific users to limit their permissions to what is necessary.
* **Service Accounts**: Used for granting permissions to applications and services, minimizing human access to sensitive resources.

**3. Instance Security**

* **Instance Templates**:
  + **Boot Disks**: Encrypted by default, ensuring data at rest is secure.
  + **Metadata Startup Scripts**: Automates security updates and software installations to maintain a secure and up-to-date environment.

**4. Data Security**

* **Cloud SQL**:
  + **Database Encryption**: All data stored in Cloud SQL is encrypted by default.
  + **User Management**: Create users with strong passwords and limit their permissions to only necessary databases.
* **Cloud Storage**: Encrypts data at rest and in transit, ensuring the security of static assets and backups.

**5. Network Traffic Protection**

* **Load Balancing**:
  + **HTTPS Load Balancer**: Ensures all web traffic is encrypted using TLS, protecting data in transit.
* **Cloud Armor**:
  + **Security Policies**: Protects against common web attacks like SQL injection and XSS, reducing the risk of data breaches and service disruptions.
  + **DDoS Protection**: Shields your application from distributed denial-of-service attacks.

**6. Monitoring and Logging**

* **Stackdriver Monitoring and Logging**: Collects and monitors logs from your services and infrastructure, allowing you to detect and respond to security incidents quickly.
* **Audit Logs**: Keeps a record of all changes and accesses, enabling forensic investigations and compliance audits.

**7. Compliance Certifications and Standards**

* **GCP Compliance**: Google Cloud Platform itself is compliant with several industry standards and certifications, including:
  + **ISO/IEC 27001**: Information Security Management
  + **ISO/IEC 27017**: Cloud Security
  + **ISO/IEC 27018**: Cloud Privacy
  + **GDPR**: General Data Protection Regulation
  + **HIPAA**: Health Insurance Portability and Accountability Act

**8. Backup and Disaster Recovery**

* **Regular Backups**: Implement automated backups for databases and critical data, ensuring they can be restored in the event of data loss or corruption.
* **Disaster Recovery Plan**: Develop and test a disaster recovery plan to ensure business continuity in case of a major incident.

**9. Security Awareness and Training**

* **User Training**: Regular security awareness training for all employees to recognize and respond to security threats.
* **Policy Enforcement**: Strict adherence to security policies and procedures, with regular reviews and updates.

**10. Regular Security Audits and Penetration Testing**

* **Third-Party Audits**: Regularly scheduled audits by third-party security experts to identify and address potential vulnerabilities.
* **Penetration Testing**: Conduct regular penetration testing to proactively identify and mitigate security risks.

**Monitoring and Optimization**

* Set up Google Cloud Monitoring to track the performance of the deployed resources.

Setting up Google Cloud Monitoring (formerly known as Stackdriver Monitoring) allows you to track the performance, availability, and overall health of your Google Cloud resources. Here's a step-by-step guide to configure Google Cloud Monitoring for your deployed resources using Terraform:

### Step 1: Enable the Monitoring API

Ensure that the Monitoring API is enabled for your Google Cloud project. This can be done through the Google Cloud Console:

1. Navigate to the **APIs & Services** > **Library**.
2. Search for **Cloud Monitoring API** and click **Enable**.

### Step 2: Create a Monitoring Workspace

If you don't already have a monitoring workspace, you need to create one:

1. Navigate to **Monitoring** in the Google Cloud Console.
2. Follow the prompts to create a new workspace if one does not already exist.

### Step 3: Update Terraform Configuration

Here's how to set up Google Cloud Monitoring using Terraform to track the performance of your deployed resources.

#### Create a new monitoring.tf file:

provider "google" {

project = var.project\_id

region = var.region

}

# Enable the Monitoring API

resource "google\_project\_service" "monitoring" {

project = var.project\_id

service = "monitoring.googleapis.com"

}

# Create a Cloud Monitoring Workspace

resource "google\_monitoring\_workspace" "workspace" {

project = var.project\_id

}

# Create a Monitoring Dashboard

resource "google\_monitoring\_dashboard" "dashboard" {

provider = google-beta

dashboard\_json = <<EOF

{

"displayName": "My Dashboard",

"widgets": [

{

"title": "CPU Utilization",

"xyChart": {

"dataSets": [

{

"timeSeriesQuery": {

"timeSeriesFilter": {

"filter": "metric.type=\"compute.googleapis.com/instance/cpu/utilization\" AND resource.type=\"gce\_instance\"",

"aggregation": {

"alignmentPeriod": "60s",

"perSeriesAligner": "ALIGN\_MEAN"

}

}

},

"plotType": "LINE"

}

],

"timeshiftDuration": "0s"

}

},

{

"title": "Memory Usage",

"xyChart": {

"dataSets": [

{

"timeSeriesQuery": {

"timeSeriesFilter": {

"filter": "metric.type=\"compute.googleapis.com/instance/memory/usage\" AND resource.type=\"gce\_instance\"",

"aggregation": {

"alignmentPeriod": "60s",

"perSeriesAligner": "ALIGN\_MEAN"

}

}

},

"plotType": "LINE"

}

],

"timeshiftDuration": "0s"

}

}

]

}

EOF

}

# Example Alert Policy for high CPU usage

resource "google\_monitoring\_alert\_policy" "cpu\_alert" {

display\_name = "High CPU Usage Alert"

combiner = "OR"

conditions {

display\_name = "VM Instance - CPU Usage"

condition\_threshold {

filter = "metric.type=\"compute.googleapis.com/instance/cpu/utilization\""

duration = "60s"

comparison = "COMPARISON\_GT"

threshold\_value = 0.8

aggregations {

alignment\_period = "60s"

per\_series\_aligner = "ALIGN\_MEAN"

}

}

}

notification\_channels = [google\_monitoring\_notification\_channel.email.id]

documentation {

content = "CPU usage is above 80% for more than 60 seconds."

mime\_type = "text/markdown"

}

}

# Notification Channel

resource "google\_monitoring\_notification\_channel" "email" {

display\_name = "Email Notification"

type = "email"

labels = {

email\_address = "your-email@example.com"

}

} variable "project\_id" {

description = "The GCP project ID"

type = string

}

variable "region" {

description = "The GCP region"

type = string

}

project\_id = "your-gcp-project-id"

region = "your-gcp-region"

terraform init

terraform apply

**Step 5: Verify the Setup**

1. **Monitoring Workspace**: Verify that the Monitoring workspace has been created by navigating to the Monitoring section in the Google Cloud Console.
2. **Dashboards**: Check the Monitoring > Dashboards section to see the newly created dashboard displaying CPU and memory metrics.
3. **Alerts**: Go to Monitoring > Alerting to verify that the alert policy for high CPU usage is set up.
4. **Notifications**: Ensure that the notification channel is correctly configured and test it to make sure you receive alerts.

**Create alerts for high CPU usage and database performance issues.**

**gcloud services enable monitoring.googleapis.com**

**provider "google" {**

**project = var.project\_id**

**region = var.region**

**}**

**resource "google\_project\_service" "monitoring" {**

**project = var.project\_id**

**service = "monitoring.googleapis.com"**

**}**

**# Notification Channel (Email)**

**resource "google\_monitoring\_notification\_channel" "email" {**

**display\_name = "Email Notification"**

**type = "email"**

**labels = {**

**email\_address = var.notification\_email**

**}**

**}**

**# High CPU Usage Alert Policy**

**resource "google\_monitoring\_alert\_policy" "cpu\_alert" {**

**display\_name = "High CPU Usage Alert"**

**combiner = "OR"**

**conditions {**

**display\_name = "VM Instance - CPU Usage"**

**condition\_threshold {**

**filter = "metric.type=\"compute.googleapis.com/instance/cpu/utilization\""**

**duration = "60s"**

**comparison = "COMPARISON\_GT"**

**threshold\_value = 0.8**

**aggregations {**

**alignment\_period = "60s"**

**per\_series\_aligner = "ALIGN\_MEAN"**

**}**

**}**

**}**

**notification\_channels = [google\_monitoring\_notification\_channel.email.id]**

**documentation {**

**content = "CPU usage is above 80% for more than 60 seconds."**

**mime\_type = "text/markdown"**

**}**

**}**

**# Cloud SQL Database Performance Alert Policy**

**resource "google\_monitoring\_alert\_policy" "db\_performance\_alert" {**

**display\_name = "Cloud SQL High Latency Alert"**

**combiner = "OR"**

**conditions {**

**display\_name = "Cloud SQL - High Latency"**

**condition\_threshold {**

**filter = "metric.type=\"cloudsql.googleapis.com/database/query\_latency\""**

**duration = "60s"**

**comparison = "COMPARISON\_GT"**

**threshold\_value = 0.5 # Example threshold for latency in seconds**

**aggregations {**

**alignment\_period = "60s"**

**per\_series\_aligner = "ALIGN\_MEAN"**

**}**

**}**

**}**

**notification\_channels = [google\_monitoring\_notification\_channel.email.id]**

**documentation {**

**content = "Database query latency is above 0.5 seconds for more than 60 seconds."**

**mime\_type = "text/markdown"**

**}**

**}** **variable "project\_id" {**

**description = "The GCP project ID"**

**type = string**

**}**

**variable "region" {**

**description = "The GCP region"**

**type = string**

**}**

**variable "notification\_email" {**

**description = "The email address to receive notifications"**

**type = string**

**}**

**project\_id = "your-gcp-project-id"**

**region = "your-gcp-region"**

**notification\_email =** [**your-email@example.com**](mailto:your-email@example.com)

**project\_id = "your-gcp-project-id"**

**region = "your-gcp-region"**

**notification\_email =** [**your-email@example.com**](mailto:your-email@example.com)

**terraform apply**

Optimizing architecture for cost efficiency and performance in Google Cloud involves several strategies focused on resource utilization, scalability, and efficient use of managed services. Here are key steps and best practices to achieve this:

**1. Rightsizing Compute Resources**

* **Use Sustained Use Discounts**: Ensure that VM instances running continuously for a significant portion of the billing month automatically receive discounts.
* **Utilize Preemptible VMs**: For non-critical workloads, use preemptible VMs which are up to 80% cheaper than regular instances.
* **Auto-scaling**: Set up auto-scaling for VM instances and managed instance groups to dynamically adjust resources based on workload demands.

**2. Storage Optimization**

* **Choose Appropriate Storage Class**: Use different storage classes (e.g., Standard, Nearline, Coldline) based on access frequency and retrieval time requirements.
* **Lifecycle Management**: Implement lifecycle policies to automatically transition data to cheaper storage classes or delete data that is no longer needed.

**3. Database Optimization**

* **Use Managed Services**: Opt for managed database services like Cloud SQL, Cloud Spanner, or Firestore which offer scalability and automatic backups without the need to manage infrastructure.
* **Rightsize Database Instances**: Adjust the size and configuration of database instances based on actual workload requirements to avoid over-provisioning.

**4. Networking Optimization**

* **Use Load Balancers**: Distribute incoming traffic across multiple instances to ensure availability and prevent overloading individual instances.
* **CDN Integration**: Utilize Google Cloud CDN to cache content closer to your users, reducing latency and bandwidth costs.

**5. Monitoring and Cost Management**

* **Set Budgets and Alerts**: Define budget alerts in Google Cloud Console to get notified when costs exceed predefined thresholds.
* **Monitoring and Analysis**: Use Cloud Monitoring to track resource utilization, performance metrics, and costs. Analyze metrics to identify underutilized resources or spikes in demand.

**6. Containerization and Serverless Architectures**

* **Containerization**: Use Google Kubernetes Engine (GKE) for container orchestration, scaling containers based on demand to optimize resource usage.
* **Serverless Services**: Leverage serverless options like Cloud Functions and Cloud Run where possible to eliminate idle resources and pay only for actual usage.

**7. Security and Compliance**

* **Implement Least Privilege**: Apply least privilege principles to IAM roles and permissions to reduce the risk of unauthorized access and potential misuse of resources.
* **Compliance Automation**: Use Google Cloud Security Command Center and compliance tools to automate compliance checks and ensure adherence to regulatory requirements.

**8. Continuous Optimization**

* **Regular Review and Optimization**: Continuously review architecture, performance metrics, and cost reports to identify opportunities for further optimization.
* **Use Cost Calculator Tools**: Utilize Google Cloud Pricing Calculator to estimate costs for different configurations and services before deployment.

**Example Workflow Using Terraform**

Here’s an example workflow using Terraform to optimize cost efficiency and performance:

1. **Define Infrastructure as Code**: Use Terraform to define your Google Cloud resources, specifying configurations for VM instances, databases, networking components, etc.
2. **Apply Best Practices**: Implement best practices such as auto-scaling, choosing appropriate machine types, and utilizing managed services.
3. **Continuous Integration and Deployment (CI/CD)**: Integrate Terraform with your CI/CD pipeline to automate deployments and updates, ensuring consistency and reliability.
4. **Monitoring and Alerts**: Set up monitoring using Google Cloud Monitoring and configure alerts for performance metrics and cost thresholds.
5. **Review and Optimization**: Regularly review monitoring data, cost reports, and resource utilization. Use Terraform to make adjustments based on findings to further optimize efficiency.