Task 1: Provision a GCP Kubernetes Engine (GKE) cluster using Terraform

Here's an example of how you can create a GKE cluster using Terraform:

|  |
| --- |
| # main.tf  provider "google" {  credentials = file("path/to/service-account-key.json")  project = "your-gcp-project-id"  region = "your-preferred-region"  }  resource "google\_container\_cluster" "my\_cluster" {  name = "my-cluster"  **location** = "your-preferred-region"  initial\_node\_count = **3**  node\_config {  machine\_type = "n1-standard-2"  disk\_size\_gb = **100**  }  }  **output** "kubeconfig" {  value = google\_container\_cluster.my\_cluster.master\_auth[**0**].cluster\_ca\_certificate  } |

Make sure to replace the path/to/service-account-key.json with the actual path to your GCP service account key JSON file. You can also adjust the cluster name, region, node count, machine type, and disk size according to your requirements.

Task 2: Provision a GCP Persistent Disk and attach it to the cluster

You can use the following Terraform code to create a GCP Persistent Disk and attach it to the GKE cluster:

|  |
| --- |
| # main.tf  resource "google\_compute\_disk" "my\_disk" {  name = "my-persistent-disk"  type = "pd-standard"  size = **100**  zone = "your-preferred-zone"  }  resource "google\_container\_node\_pool" "my\_node\_pool" {  name = "my-node-pool"  cluster = google\_container\_cluster.my\_cluster.name  node\_count = **3**  autoscaling {  min\_node\_count = **3**  max\_node\_count = **5**  }  node\_config {  machine\_type = "n1-standard-2"  disk\_size\_gb = **100**  metadata = {  "gce-container-declaration" = <<EOF  apiVersion: v1  kind: Pod  metadata:  name: your-pod-name  spec:  volumes:  - name: your-volume-name  gcePersistentDisk:  pdName: ${google\_compute\_disk.my\_disk.name}  fsType: ext4  containers:  - name: your-container-name  image: your-container-image  volumeMounts:  - name: your-volume-name  mountPath: /mnt/data  EOF  }  }  } |

Replace your-preferred-zone, your-pod-name, your-volume-name, your-container-name, and your-container-image with the appropriate values for your setup.

Task 3: Provision a GCP Cloud SQL instance and create a database

Use the following Terraform code to provision a GCP Cloud SQL instance and create a database:

|  |
| --- |
| # main.tf  resource "google\_sql\_database\_instance" "my\_instance" {  name = "my-cloud-sql-instance"  database\_version = "POSTGRES\_13"  settings {  tier = "db-f1-micro"  backup\_configuration {  enabled = **true**  }  database\_flags {  name = "log\_statement"  value = "all"  }  }  }  resource "google\_sql\_database" "my\_database" {  name = "my-database"  instance = google\_sql\_database\_instance.my\_instance.name  } |

Adjust the instance name, database version, tier, and other settings as per your requirements.

Task 4: Provision a GCP Cloud Storage bucket and upload a file

To create a GCP Cloud Storage bucket and upload a file to it using Terraform, use the following code:

|  |
| --- |
| # main.tf  resource "google\_storage\_bucket" "my\_bucket" {  name = "my-storage-bucket"  location = "us-central1"  force\_destroy = **true**  }  resource "google\_storage\_bucket\_object" "my\_object" {  name = "my-file.txt"  bucket = google\_storage\_bucket.my\_bucket.name  source = "path/to/local/file.txt"  } |

Replace my-storage-bucket with your desired bucket name and path/to/local/file.txt with the actual path to the file you want to upload.

Task 6: Create a Kubernetes deployment and service configuration

Here's an example of a Kubernetes deployment and service configuration for a simple web application:

|  |
| --- |
| # deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: my-webapp  spec:  replicas: 3  selector:  matchLabels:  app: my-webapp  template:  metadata:  labels:  app: my-webapp  spec:  containers:  - name: webapp  image: my-webapp-image  ports:  - containerPort: 80  ---  # service.yaml  apiVersion: v1  kind: Service  metadata:  name: my-webapp-service  spec:  selector:  app: my-webapp  ports:  - protocol: TCP  port: 80  targetPort: 80  type: LoadBalancer |

Replace my-webapp-image with the actual image name of your web application.

Task 9: Automate the Terraform configuration and testing process

To automate the process of applying the Terraform configuration and running tests, you can create a script using a scripting language like Bash or PowerShell. Here's an example of a Bash script that performs these tasks:

|  |
| --- |
| #!/bin/bash  # Apply Terraform configuration  terraform init  terraform apply -auto-approve  # Run tests  # Add your test commands here |

Save this script in a file (e.g., terraform\_deploy.sh), make it executable (chmod +x terraform\_deploy.sh), and execute it to apply the Terraform configuration and run your tests.