My Terraform GCP Infrastructure Assignment

What I Built:

I set up a GCP infrastructure using Terraform with two VMs - one public VM running NGINX and one private VM. The private VM doesn't have an external IP and can only be accessed through IAP. It gets internet access through Cloud NAT which I also set up.

How I Organized the Code:

I used modules to keep things organized. Made three modules - networking, ssh-key, and compute. Then the root folder calls these modules and ties everything together.

Steps I Followed:

1. GCP Project Setup

First thing I did was create a GCP project in the console. Then I enabled the APIs I needed using gcloud commands.

```
export PROJECT_ID="my-project-123"
gcloud config set project $PROJECT_ID
gcloud services enable compute.googleapis.com iam.googleapis.com secretmanager.googleapis.com
```

Then I created a GCS bucket to store the terraform state file. This was important because it lets terraform track what resources exist.

```
export BUCKET_NAME="${PROJECT_ID}-terraform-state"
gsutil mb -p $PROJECT_ID gs://$BUCKET_NAME
gsutil versioning set on gs://$BUCKET_NAME
```

2. Setting up Service Account

I needed a service account for GitHub Actions to deploy automatically. Created it like this:

```
gcloud iam service-accounts create github-actions-sa --display-name="GitHub Actions SA"
export SA_EMAIL="github-actions-sa@${PROJECT_ID}.iam.gserviceaccount.com"
```

Then gave it permissions to create compute resources, manage storage, and handle secrets:

```
gcloud projects add-iam-policy-binding $PROJECT_ID --member="serviceAccount:${SA_EMAIL}" --
role="roles/compute.admin"
gcloud projects add-iam-policy-binding $PROJECT_ID --member="serviceAccount:${SA_EMAIL}" --
```

role="roles/storage.admin"
gcloud projects add-iam-policy-binding $PROJECT_ID --member="serviceAccount:${SA_EMAIL}" --role="roles/secretmanager.admin"

3. Workload Identity Setup

This part was tricky but basically lets GitHub Actions authenticate to GCP without needing to store JSON keys which is more secure.

export GITHUB_REPO="myusername/my-repo"
gcloud iam workload-identity-pools create "github-pool" --location="global" --display-name="GitHub Actions Pool"
gcloud iam workload-identity-pools providers create-oidc "github-provider" --location="global" --workload-identity-pool="github-pool" --display-name="GitHub Provider" --attribute-mapping="google.subject=assertion.sub,attribute.repository=assertion.repository" --issuer-uri="https://token.actions.githubusercontent.com"

Then I had to bind the service account to the workload identity pool:

gcloud iam service-accounts add-iam-policy-binding "${SA_EMAIL}" --role="roles/iam.workloadIdentityUser" --member="principalSet://iam.googleapis.com/${POOL_ID}/attribute.repository/${GITHUB_REPO}"

4. GitHub Repository

Created a new repo on GitHub and cloned it locally. Made the folder structure:

mkdir -p .github/workflows
mkdir -p modules/networking modules/ssh-key modules/compute
mkdir -p root

5. Writing the Terraform Code

Networking Module:
This module creates the VPC, two subnets (public and private), Cloud NAT for the private subnet, and firewall rules. The public subnet gets 10.0.1.0/24 and private gets 10.0.2.0/24.

SSH Key Module:
Generates an SSH key pair using the tls_private_key resource and stores them in Secret Manager. I did this because hardcoding SSH keys is bad practice.

Compute Module:
Creates two e2-micro VMs. The public VM gets an external IP and has NGINX installed via startup script. The private VM only has an internal IP.

Root Module:

This is where I call all the other modules and pass variables between them. Also configured the GCS backend here for state storage.

6. GitHub Secrets

Had to add these secrets in the GitHub repo settings:
- GCP_PROJECT_ID
- GCS_BACKEND_BUCKET
- GCP_WORKLOAD_IDENTITY_PROVIDER
- GCP_SERVICE_ACCOUNT

7. GitHub Actions Workflow

Created a workflow file that runs on push to main. It authenticates using workload identity, runs terraform init, plan, and apply. Also set it up to show deployment summary at the end.

8. Testing

Pushed everything to GitHub and the workflow ran automatically. Took a few minutes but it worked. After deployment I got the outputs showing the external IP of the public VM.

To test NGINX I just opened http://EXTERNAL-IP in browser and saw the default NGINX page.

To SSH into the public VM:
gcloud compute ssh assignment-public-vm --zone=us-central1-a

To SSH into the private VM I had to use IAP:
gcloud compute ssh assignment-private-vm --zone=us-central1-a --tunnel-through-iap

9. Verification

Checked that the private VM could reach internet through NAT:
gcloud compute ssh assignment-private-vm --zone=us-central1-a --tunnel-through-iap --command="curl -s http://www.google.com"

It worked which meant Cloud NAT was functioning properly.

Problems I Ran Into:

- Initially forgot to enable the Secret Manager API which caused terraform to fail
- Had to make sure the firewall rules allowed HTTP traffic on port 80 for the public VM
- The IAP tunnel took me a while to figure out, needed to make sure the IAP firewall rule was in place
- First time the GitHub Actions workflow failed because I didn't set the secrets correctly

What the Infrastructure Does:

Public VM: Has external IP, runs NGINX, accessible from internet
Private VM: No external IP, only internal network access, uses Cloud NAT for outbound internet
VPC: Custom VPC with two subnets in us-central1
Cloud NAT: Allows private VM to access internet without exposing it
Firewall Rules: Allow HTTP (80), SSH (22), and internal communication

Overall it was a good learning experience setting up infrastructure as code and automating deployment with GitHub Actions.