**ID.me Platform Engineering Tech Exercise Notes:**

As per the tech exercise, the solution includes, Python, Terraform, Database and monitoring dashboard.

**Python server application:**

This is implemented with python flask, a web application framework. The app uses multiple python modules as required, has Web Server Gateway Interface (WSGI).

It runs on default port 5000. It has the following paths:

/ - root

The root will fetch the index string from the MongoDB database. There are few other routes which were used for testing.

/index.html

The index.html access will fetch the index string from the templates index.html file

/get\_data – This path will fetch the index string from the MongoDB database plus the current UTC time.

/get\_ip & /client– This path will find and display the user IP address plus the current UTC time.

/post\_data?text=<string> – This path will update the index string with the given <string> in the MongoDB database, also displays the posted string plus the current UTC time.

/echo?text=<string> – This path will displays the string plus the location and current UTC time.

**Docker:**

Docker file is used to dockerize the application so it can run anywhere in a x86\_64 environment.

**Terraform:**

As part of IaC, the TF modules are used to create a GKE (Google Kubernetes Engine) cluster with two node groups, one for the control plan and another pool with spot instances for workloads. The TF will create the related VPC, subnets (public and private), Internet gateways (IGW), NATs gateways, route tables etc. It also creates relevant service accounts as required.

The gce folder will create a gce instance via TF in GCP.

**Database:** Since unstructured data is used, MongoDB cluster is used to write and read the data as required. MongoDB Kubernetes operator is used to deploy the MongoDB and replicas. MongoDB Kubernetes operator, controls over your MongoDB deployment from a single Kubernetes control plane, to give a consistent experience across different deployment environments.

**Monitoring:**

Prometheus and Grafana is used to collect the metrics from various sources, Kubernetes metrics, Database, as well as Prometheus and Grafana apps also. Prometheus is used for data collection, export it to Grafana for dashboard views. Both these apps were used since Google also uses and markets Prometheus as a managed service.

MongoDB dashboard JSON is downloaded from the Grafana DB website and used here.

**Application Architecture:**

Once deployed, the application runs as a one container in one pod in the GKE cluster with private IPs. One needs cloud credentials and “k” commands to directly connect to the pods. Two replicas are provisioned so the app is always available during a patch or deployments. A service is provisioned with private IPs to access the container ports. A Load balancer is provisioned to connect to the service ports internally and for connecting it externally. Since it is not part of the requirement, SSL certs are not implemented with this effort.

The python app will take a client request and routes the traffic as required. For requests to the web root, the app will pull the dictionary object from the MongoDB database, iterates and get the relevant string which is returned to the client.

**GitHub Actions workflow:**

The GA config file is tested and working fine .github/workflows/eb-build.yaml-working. The service account impersonation w/o a permanent SA key is working fine but SA impersonation (using temp key is giving errors as mentioned cloudAdvocate-GA-error-comments.txt)

**Challenges faced:**

1. Getting the right instance size to deploy all the application pods with various CPU, memory and disk requirements.
2. Getting services to use google service accounts, impersonate and access the cloud resources.
3. Google free tier have limitations.
4. GA with WKIF temp keys did not work for that repo!

From Anton Putra <https://github.com/antonputra/tutorials/tree/main/lessons/108>