Aurora Infrastructure Technical Test

Introduction

Welcome to the Aurora infrastructure technical test!
Everyone is unique in their own way, and we are looking for people from all walks of life to become part of our inclusive culture. We're committed to making Aurora a diverse and inclusive environment for everyone. Diversity for us means respect for and appreciation of differences in gender, age, sexual orientation, disability, race and ethnic origin, religion and faith, marital status, social, educational background and way of thinking. Our dedication is to ensure you get the chance to be the best that you can be, showcase your full potential and make the right choice for you.
We hope that you find this exercise fun and interesting. There are no trick questions; we want to see your solution to a simple problem with well thought out and structured code.
If you have any questions about any part of this exercise, please do not hesitate to reach out to us.
There is no time limit on how much time to commit to the test; however, we suggest you spend no more than 1 day on it.

The Brief

Our product team has developed a new API to manage Members and would like to deploy it to AWS. Unfortunately, the product team does not have experience in running production systems in the cloud and thus have come to you to help them create the environment for it.

Members API

The Members API is a simple Golang application wrapped in a Distroless Docker container. The container exposes a simple API on port 8080.
This API has 2 methods - POST to create a Member:

```
curl --request POST \
  --url http://localhost:8080/ \
  --header 'Content-Type: application/json' \
  --data '{
  "name": "Tester",
  "age": 35
}'
```

And GET to retrieve a list of all Members:

```
curl --request GET \
  --url http://localhost:8080/
```

The API uses a Postgres database to store the Member data. Connection to the database is configured through environment variables that are read during application start.

```
variable - description - example
DB_HOST - database server host address - localhost
PSQL_USER - database username - postgres
PSQL_PASS - password for the user - changeme
PSQL_DBNAME - schema name - members
PSQL_PORT - database server port - 5432
```

The API application will connect to the Database on start in order to run Schema migration.
If it can't connect, it will fail to start and will write error into logs.
If it successfully connects to the database, you will see `msg=SchemaUpToDate` in logs.

Running Members API Locally

You can run the Members API locally. In order to do so, follow these steps:

1. Clone this repository and navigate to the root folder
2. `docker build -t infrastructure-test .`
3. `docker compose up`

The development team has also published the Members API Docker image to Docker Hub,

```
docker run eldertech/infrastructure-test:latest -p 8080:8080
```

The Deliverable

The product team needs you to create an environment that will expose the Member API endpoints to be publicly accessible from AWS.
The Members API also needs to be able to save new Members into a database, and query

Members from the database.
The product team will run this in production, and you should thus consider HA and observability best practices.
You can use the image on Docker Hub, or you can build the image yourself. For the purposes of this task, you do not have to bother with a CI/CD pipeline and can take the image as-is.

We realise that this is a vague requirement - and that is intentional, there is no single right solution and we want to leave you free hand at implementing this requirement using whichever architecture approach you feel is the most appropriate.

You can use whichever set of tools and technologies that you think are the most appropriate but we ask you to use Infrastructure as Code (we use mostly Terraform). You can also use existing code and modules, as long as you have the rights to use it - we are not fans of reinventing the wheel as well and thus tend to re-use opensource code where appropriate.

You do not have to deploy your IaC to an actual environment that would incur costs, simply defining the environment in IaC will be sufficient. We can deploy it to an Aurora account together.

We do not expect you to come with the entire environment following the AWS Well-Architected Framework - you can assume that things like IAM, monitoring, security scanning and so on are taken care of at an Organisation level. We also do not want you to spend too much time on this task - focus on implementing what is most important and document any improvements that you would make in the README file.

Please ensure you include the following in your README.md:
1.A covering note explaining the architectural choices you have made, along with a high-level architecture diagram
2.Any assumptions you've made and why
3.Instructions on how to deploy your solution to a fresh AWS account
4.Are there any other considerations/future enhancements you would make given more time?

Please, treat this project like you're working as part of a team, and make your git etiquette reflect this. Email us the solution as an attachment or include a link to the git bundled repository showing your commit history with all your commits on the master branch:

```
git bundle create tech–test.bundle –all –branches
```