CODE ASSIGMENT FOR FLIGHT RADAR

1. *Create a minimal API that contains the following two endpoints*

- *time - outputs current unix timestamp*
- *random - outputs a list of 10 random numbers (range: 0 to 5)*

For this task, a simple flask python application was developed. The python flask module (That I have used in the past, see https://github.com/rv0lt/Docker-K8s-WebApp) is straightforward to use. By using the time and random system libraries we can finish the logic of the api as follows

```python
from flask import Flask, jsonify
import time
import random

app = Flask(__name__)

@app.route('/time')
def get_current_time():
    current_time = int(time.time())
    return jsonify(timestamp=current_time)

@app.route('/random')
def get_random_numbers():
    random_numbers = [random.randint(0, 5) for _ in range(10)]
    return jsonify(numbers=random_numbers)

if __name__ == '__main__':
    app.run()
```

2. Create a Dockerfile that includes your API and make it runnable as a container.

We can create a minimal simple Dockerfile by using the Python base image, installing the flask library and listening on the ports used:

```
# Use the official Python image as the base image
FROM python:3.9-slim

# Install flask
RUN pip install --no-cache-dir flask

# Copy the API code into the container
COPY api.py .

# Expose the port on which the API will run
EXPOSE 5000

# Set the command to run the API when the container starts
CMD ["python", "api.py"]
```

However, to follow best practices, it is useful to define a non root user (by default containers usually run as root)

```
# Add a new user with user id 8877
RUN useradd -u 8877 noroot
# Change to non-root privilege
USER noroot

# Set the command to run the API when the container starts
CMD ["python", "api.py"]
```

NOTE: To further enhance security, we can create distroless images, however, due to time constraints for this assignments, that solution could not be adopted. But I have written blogs post about this on: https://rv0lt.github.io/2023/03/21/distroless-scratch-dockerifles.html

3. Create a minimal Terraform config that would deploy your container on AWS ECS/Fargate, you may assume the following:

- *Your docker image can be uploaded to a public docker registry*
- *You are given the values of the VPC*

I already have upload the image developed to a public registry at the endpoint rv0lt/flightradartest:v1 -- https://hub.docker.com/repository/docker/rv0lt/flightradartest/general

For the second part, I created an AWS account to do some testing, because the values of the VPC are assumed to be given, we can define the ecs fargate cluster as follows:

- First, we define the task definition, in which we specify the image used and the parameters required to run the container:

```
resource "aws_ecs_task_definition" "flight_api" {
  family = "flight_api"

  container_definitions = <<EOF
  [
    {
      "name": "flight_api",
      "image": "rv0lt/flightradartest:v1",
      "portMappings": [
        {
          "containerPort": 5000
        }
      ]
    }
  ]
  EOF

  execution_role_arn = aws_iam_role.api_task_execution_role.arn
  # These are the minimum values for Fargate containers.
  cpu = 256
  memory = 512
  requires_compatibilities = ["FARGATE"]

  # This is required for Fargate containers (more on this later).
  network_mode = "awsvpc"
}
```

- Then, with those values, we can create the cluster and the ecs service

As it is seen, apart from defining the required and the given values, we need one extra parameter the “desired count” is used to specify how many copies of our container will be run.

```
resource "aws_ecs_cluster" "app" {
   name = "app"
}

resource "aws_ecs_service" "flight_api" {
  name            = "flight_api"
  task_definition = "aws_ecs_task_definition.flight_api.arn"
  cluster         = aws_ecs_cluster.app.id
  launch_type     = "FARGATE"

 network_configuration {
   assign_public_ip = false

   security_groups = [
     aws_security_group.egress_all.id,
     aws_security_group.ingress_api.id,
   ]

   subnets = [
     aws_subnet.private_d.id,
     aws_subnet.private_e.id,
   ]
 }
 load_balancer {
   target_group_arn = aws_lb_target_group.flight_api.arn
   container_name   = "flight_api"
   container_port   = "5000"
 }
 desired_count = 1
}
```

Finally, don’t forget to setup the provider and the credentials:

```
provider "aws" {
    shared_credentials_file = "credentials"
    region = "us-east-1"  # Replace with your desired AWS region
}

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}
```

The credentials file contains the required credentials to connect to the instance (access key and secret access key)

Don’t forget that for this to run we need also the networks, the IAM permissions and a load balancing. Those values are supposed to be given for this report, but the code located in https://github.com/rv0lt/flightRadarTest Contains an example of them.