**The Goals:**

* **deploy isolated web applications**
* **write infrastructure as code explain**
* **design decisions and chosen technologies handover the project to new colleagues**

**Requirements**

* **The applicant can choose any technology stack, and should meet the following requirements:**
* **The entire stack should be able to run locally on a developer's machine**
* **The country- and airport-service run isolated from each other**
* **No inter-communication between the two services is possible**
* **No direct communication from the "outside world" is possible with the two services A reverse-proxy and/or load-balancer exposes the services on port 8000 Initially, airports version 1.0.1 is deployed An update to the airports service from version 1.0.1 to version 1.1.0 can be triggered at the code review without causing a service interruption. Bonus points for increased use of automation, e.g.: Automation of the deployment using a CI/CD tool. Single click/command startup of the initial stack.**

**The Solution:**

**The first scenario is designing, implementing a H/A ad robust solution stack using AWS services:**

**As you know, there are multiple technologies to dockerixze and deploy the mentioned services. I preferred to use the latest technology to deploy and serve JAR microservices on AWS platform using ECS (Fargate).**

* **The used technologies and services:**
  + **GitHub: Manage code versioning**
  + **Docker engine: to dockerize the airports and countries microservices**
  + **AWS ECR (Image repository)**
  + **ECS Cluster (FARGATE)**
  + **AWS CodePipeline**
  + **Application loadbancer (Our APIs endpoint)**
  + **Jenkins: CI tool**

**Herein you can find the steps to convert a plain jar file to ready dockerized API services.**

1. **Install Jenkins as CI tool that integrates with our Github and AWS accounts.**
2. **Install docker build and publish Jenkins plugin that is responsible for building the docker image and publishing it to ECR**
3. **Create AWS ECR repository**
4. **Create AWS ECS (Fargate) cluster**
5. **Create ECS task definition for each API service**
6. **Create ECS service for each API based on its own task definition**
7. **Associate each ECS service with the same ALB but with different Target group to be differentiated when routing the incoming traffic and point to the proper target groups using ALB rules.**
8. **Jenkins pulls the latest Jar version or a specific tag and build the required docker image. Then it pushes the images to AWS ECR repository to be ready for the deployment stage on ECS cluster.**
9. **The deployment will be triggered based on the change on ECR or can be done manually.**

**You can check APIs endpoint after the deployment:**

* **Airport API Test URL :**
  + [**http://airportsatck-472127968.eu-west-1.elb.amazonaws.com/airports/AE**](http://airportsatck-472127968.eu-west-1.elb.amazonaws.com/airports/AE)
* **Countries API Test URL** 
  + [**http://airportsatck-472127968.eu-west-1.elb.amazonaws.com/countries/AE**](http://airportsatck-472127968.eu-west-1.elb.amazonaws.com/countries/AE)

**The second scenario is running those jars micro services ion stand-alone server (Local machine)**

* **Using a docker orchestration and management platform like SWARM or Rancher to create and manage local docker cluster that serves the created images from JARs**
* **Exposing ports for each running images and reverse-proxy using Nginx to route the traffic for each service.**
* **This solution is ready also if you need to check in my technical interview.**

**Finally, It is just a quick review for descripting how to design and implement the required stacks.**

**I look forward to discussing in details in my technical interview.**

**I will attach the used Dockerfiles and CloudFormation template as well.**