Agenda

The overall agenda of this document is to list down all the steps taken while doing POC's related to different stages of the software development life cycle. The POCs for which the documentation is done are

- Creating a spring boot based application which exposes few endpoints
- Pushing the application code to github
- Dockerizing the application
- Pushing the dockerized application to docker hub
- Deployment of application container directly using docker hub image
- Deployment of application container manually on AWS ECS instance
- Deployment of application container on AWS ECS instance using terraform

Prerequisites Knowledge

- Java
- SpringBoot
- Maven
- Git
- Docker
- Terraform
- AWS Cloud services like ECS, Security Groups

Prerequisites Softwares

- IntelliJ Idea 2023 or higher
- Maven 3.8.7 or higher
- Java 11.0.17 or higher
- Docker Desktop
- Terraform

Terminology

- Docker is a tool used to package application/software into units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.

- ECS Service is a fully managed container orchestration service that helps to easily deploy, manage, and scale containerized applications over AWS Cloud.

Github Link

All the code related to POC is available here
https://github.com/anshulsood2006/ecs-poc.git

POCs

Create Spring Boot Application

- Go to https://start.spring.io/ and create a new spring boot project zip file for a project based on maven, java 11 and spring boot version 2.7.15.
- Unzip the file and Import the project into IntelliJ Idea
- Add below dependencies
  - spring-boot-starter-web
  - lombok
- Create few endpoints in the project using spring framework annotations

Upload the application code to Git

- Login to github.com and create a new repository and copy the path
- Push the application code to git
  - Go to the project folder via terminal and run command "git init"
  - Add all the files using the command "git add ."
  - Create a commit using command "git commit -m "Initial Commit"
  - Connect your local git repo to remote git url using command "git remote add origin https://github.com/anshulsood2006/ecs-poc.git"
  - Push the code to remote repo using command "git push -u origin master

Dockerize the application

- Verify that docker desktop is installed and running on the system using command "docker ps". You should not be getting an error saying "docker daemon is not running".
- Create Dockerfile to package the application as docker image inside the application base folder.
- Update the pom.xml file of the application to add a new plugin that creates docker images and tag its locally

```
<plugin>
        <groupId>com.spotify</groupId>
        <artifactId>dockerfile-maven-plugin</artifactId>
        <version>1.4.13</version>
        <executions>
            <execution>
                <id>tag-latest</id>
                <goals>
                    <goal>build</goal>
                    <goal>tag</goal>
                    <goal>push</goal>
                </goals>
                <configuration>
                    <tag>latest</tag>
                </configuration>
            </execution>
        </executions>
        <configuration>
            <useMavenSettingsForAuth>true</useMavenSettingsForAuth>

    <repository>registry.hub.docker.com/anshulsood2006/${project.artifactId}</repository>
            <tag>latest</tag>
            <buildArgs>
                <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
            </buildArgs>
        </configuration>
    </plugin>
```

- Make sure that the tag name given in the execution configuration section is the same as that given in the plugin configuration section.
- Now on building the project using command "mvn clean install", docker image will also get created which can be verified using command "docker images"
- Run the dockerized poc application using command "docker run -p 8081:8080 poc"
- This will expose the application on host port 8081

Publish to Docker Hub

- Verify that docker desktop is installed and running on the system using command "docker ps". You should not be getting an error saying "docker daemon is not running".
- Make sure /m2/settings.xml file has the server configuration for docker hub

```
<servers>
    <server>
        <id>registry.hub.docker.com</id>
        <registry>docker.io</registry>
        <username>{DOCKERHUB_USER_NAME}</username>
```

```
                <password>{DOCKERHUB_PASSWORD}</password>
                <configuration>
                        <email>{DOCKERHUB_EMAIL}</email>
                </configuration>
        </server>
</servers>
```

- Update the pom.xml file of the application and make sure that the com.spotify plugin contains the configuration section where repo and tag are containing correct values
- The value of tag should be same as the value of tag inside execution section
- The value of repo should contain the id of the server as defined in settings.xml file viz. registry.hub.docker.com is this case

Deployment (manual) on standalone host using docker hub image

- Verify that docker desktop is installed and running on the system using command "docker ps". You should not be getting an error saying "docker daemon is not running".
- To run image directly from docker-hub repository image, on local run command
  docker run -d -p 8081:8081 anshulsood2006/poc

Deployment (manual) on ECS Service

- Verify that you have valid account credentials for the AWS account where you want to deploy your application manually.
- Log into AWS Console using above credentials.
- Select the region from drop down on upper right corner
- Search for Elastic Container Service in the search bar.
- Click on Get Started.
- Create a New Cluster with
  - name as say POC
  - Infrastructure as AWS Fargate
- Select the above cluster i.e POC.
- Go to task Definitions on the left panel
- Click on Create New Task Definition
- Specify
  - The family as say POC
  - Infrastructure as AWS Fargate
  - Container
    - Name as POC
    - Image name from dockerhub
- Click Create
- Click Deploy >> Create a Service
- Select the cluster POC
- Set service name as say POC

- Click Create
- Go to the task and select security group
- Edit inbound rule to allow Custom TCP traffic on port 8080 from all IP addresses 0.0.0.0/0
- The containerized application will now be accessible via public ip

Deployment (terraform) on ECS Service

- Verify that terraform is installed on the system using command “**terraform –version**”.
- Verify that aws cli is installed on the system using command “**aws –version**”.
- Verify that you have valid account credentials for the AWS account where you want to deploy your application using terraform.
- Verify that you have valid access key id and secret access key for the above account to access the account programmatically using terraform.
- Verify that system variable**AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY** are set for the operating system with correct values of access key id and secret access key repectively**.**
- Create a folder with the name terraform inside your project’s root directory.
- Go inside the terraform directory.
- Create a file named versions.tf
    - This file defines the terraform version and provider versions to be used.
- Run command “**terraform init**”.
- Create a file named variables.tf
    - This file defines all the variables to be used in terraform configuration
- Create a file named main.tf
    - This file defines all the aws resources needed for the application deployment in this file. In our case the below aws resources needed will be
        - 1 VPC to provide logical isolation of aws resources from each other.
        - 2 public subnets and 2 private subnets inside the above vpc. Public facing resources such as load balancers will be defined inside public subnet while the private resources like service will be defined inside private subnet
        - 1 internet gateway to allow resources in public subnet access the internet
        - 2 route table resources and 2 route table associations resources inside a route resource to contain routes (set of rules) which are used to determine where the network traffic from the subnet is directed to.
        - 2 eip’s (Elastic IP) to access the resource publicly using static IP address.
        - 2 NAT (Network Address Translation) gateways, to allow the resources in the private subnet to connect to external resources but prevent external resources from connecting to the resources in the private subnet.
        - 1 security group to allow or reject incoming or outgoing traffic from load balancers

        - 1 load balancer with target group resource and listener resource. The target group, along with the listener resource, tells the load balancer to forward all the incoming traffic on port 8080 to whatever is attached to it.
        - 1 ecs task definition for the required ecs container configuration.
        - 1 security group associated with the ecs task definition.
        - 1 ecs cluster resource.
        - 1 ecs service resource.
    - This file also defines all the data sources needed to create ECS instance
- To verify the terraform configuration above run command **terraform plan -out="tfplan" -var="access_key=%AWS_ACCESS_KEY_ID%" -var="secret_key=%AWS_SECRET_ACCESS_KEY%”**
- To create the terraform resources run command **terraform apply "tfplan"**.
- Once the plan is applied the application can be accessed using url http://poc-lb-17510045.us-east-2.elb.amazonaws.com/poc/products/103
  Where domain name poc-lb-17510045.us-east-2.elb.amazonaws.com is taken from the terminal via output variable
- To destroy the terraform configuration
    - run command **terraform plan -destroy -out=tfplan -var="access_key=%AWS_ACCESS_KEY_ID%" -var="secret_key=%AWS_SECRET_ACCESS_KEY%"**
    - run command **terraform apply "tfplan"**