## **Challenge 4: Dockerize the Application**

Table Of Contents

**1**

[Dockerize the application](#_bwfskl4nxwzu) 1

[Steps](#_4i2h63954f34) 1

[Push the docker image to docker hub](#_xu8votbonsxq) 2

[Steps](#_l1tvyxr0vbwz) 2

[Deploy the container](#_35up9daq62ls) 2

[Breakdown: providers.tf](#_9b6tddz7z2v) 2

[Breakdown: variables.tf](#_5d3olnrxvtap) 3

[Breakdown: main.tf](#_i936f99ii3fj) 3

[Breakdown: deployments.tf](#_l99d2vp7evde) 3

[Breakdown: services.tf](#_jgpwog4rdq36) 3

[Breakdown: output.tf](#_qn6q13l79k5w) 3

### Dockerize the application

To Dockerize the application follow the below steps,

###### Steps

1. Create a Dockerfile in the project folder (i.e. in the /Challenge1\_ DevOps Challenge\_2&3/OrclDevopsChallenge folder) and update the contents as,

FROM java:8

EXPOSE 8080

ADD /target/OrclDevopsChallenge-0.0.1-SNAPSHOT.jar OrclDevopsChallenge-0.0.1-SNAPSHOT.jar

ENTRYPOINT ["java","-jar","OrclDevopsChallenge-0.0.1-SNAPSHOT.jar","server"]

1. Navigate to the project folder and type following command to create image and run that image,

$ mvn clean

$ mvn install

$ docker build -f Dockerfile -t orcl-devops-challenge

$ docker run -p 8080:8080 -t orcl-devops-challenge

### Push the docker image to docker hub

To push the docker image to docker hub follow the below steps,

###### Steps

1. Sign Up for Docker Hub (if not already) to get the credentials.
2. After logging in to Docker Hub, create a repository from Docker Hub welcome page. For our example I created a repository named ***example-image***.
3. Log into the Docker public registry from your local machine terminal using Docker CLI,

*$ docker login*

1. Tag the image. According to that naming convention, the unofficial image name should be named as follows: *<username>/<image\_name>:<tag\_name>.* In this case, I need to rename it as *user1330/example\_image:latest*

*$ docker tag example\_image:latest user1330/example\_image:latest*

1. Publish the image

*$ docker push user1330/example\_image:latest*

1. If you want to test out your image, use the below command and launch a container from it:

*$ docker pull user1330/example\_image:latest*

*$ docker run -it user1330/example\_image:latest*

### Deploy the container

To deploy the container into AppServer

##### Breakdown: providers.tf

The provider.tf file is where we define the Terraform provider we’ll be using which is the Terraform Kubernetes provider. The provider.tf is shown below,

##### Breakdown: variables.tf

This file should seem familiar and is similar to the part 1 variables.tf file, but this particular file only specifies the input variables that this Terraform Kubernetes project uses.

##### Breakdown: main.tf

It specifies the type of Terraform Backend. A “backend” in Terraform determines how state is loaded and how an operation such as apply is executed. This abstraction enables non-local file state storage, remote execution, etc. In this code block, we’re using the remote backend.

##### Breakdown: deployments.tf

This file uses the Terraform Kubernetes Deployment resource to define, configure, and create all the Kubernetes resources required to release our application to the GKE cluster

##### Breakdown: services.tf

The services.tf file is where we define a Terraform Kubernetes service. It will wire up the Kubernetes elements to provide ingress access to our application running on pods in the cluster.

##### Breakdown: output.tf

The output.tf file we’re using output values to readout values like Cluster name and the ingress IP address of our newly created LoadBalancer service. This address is where we can access our application hosted in Pods on the GKE cluster.