



Building and Deploying a Scalable Java Microservice on GCP

Intermediate Assessment

Contents

Title: Building and Deploying a Scalable Java Microservice on GCP	3
Difficulty Level	3
Duration	3
What you will learn	3
What you will be provided	3
What you need to know	3
Skill Tags	3
What you will do.....	3
Activities	4
1. Building the Java Application	6
2. Creating the Terraform Backend Bucket	7
3. Configure kubectl and Cluster Access	9
4. Enabling Required GCP APIs.....	6
5. Install and Configure Terraform	11
6. Building and pushing the Docker Image.....	15
7. Deploy to Google Kubernetes Engine (GKE)	18
8. Deploying the Application to Cloud Run	21
9. Testing the Cloud Run Deployment	14
Testcases	22

Title: Building and Deploying a Scalable Java Microservice on GCP

Difficulty Level

Intermediate

Duration

90 minutes

What you will learn

By the end of this, you will be able to:

- Build a Java application using Maven.
- Provisioning infrastructure with Terraform.
- Build and push Docker images to Artifact Registry.
- Deploy applications to Google Kubernetes Engine (GKE) and Cloud Run.
- Validating deployments via curl and kubectl.

What you will be provided

- A Linux Virtual Machine with the necessary software, including Visual Studio Code, Docker, Maven, Jenkins, and Java libraries, is available in the lab.
- The project folder, containing the required files, is located at Desktop > Project.
- Access to a Google Cloud Platform (GCP) project.

What you need to know

- Fundamentals of Docker and containerization.
- Familiarity with Google Cloud SDK.
- Terraform basics.
- Kubernetes resource management with kubectl.

Skill Tags

- Maven
- Docker
- Terraform
- GKE
- Cloud Run
- Artifact Registry
- Kubernetes

Scenario

A technology company is modernizing its application deployment by shifting to a cloud-native architecture on Google Cloud Platform (GCP). The company is focused on improving scalability, reliability, and speed of delivery by containerizing its Java microservices and automating infrastructure provisioning.

To achieve this, they require a DevOps engineer to lead the setup and deployment of a Java application using modern tools like Maven, Docker, Terraform, and GCP services such as Google Kubernetes Engine (GKE) and Cloud Run. The goal is to establish a robust, automated deployment pipeline that supports both Kubernetes and serverless environments.

What you will do

- Build the Java application with Maven
- Use Terraform to provision GCP infrastructure
- Store Terraform state in a GCS backend bucket
- Enable required GCP APIs
- Build and push Docker image to Artifact Registry
- Deploy the service to GKE and Cloud Run
- Verify deployments using kubectl and curl

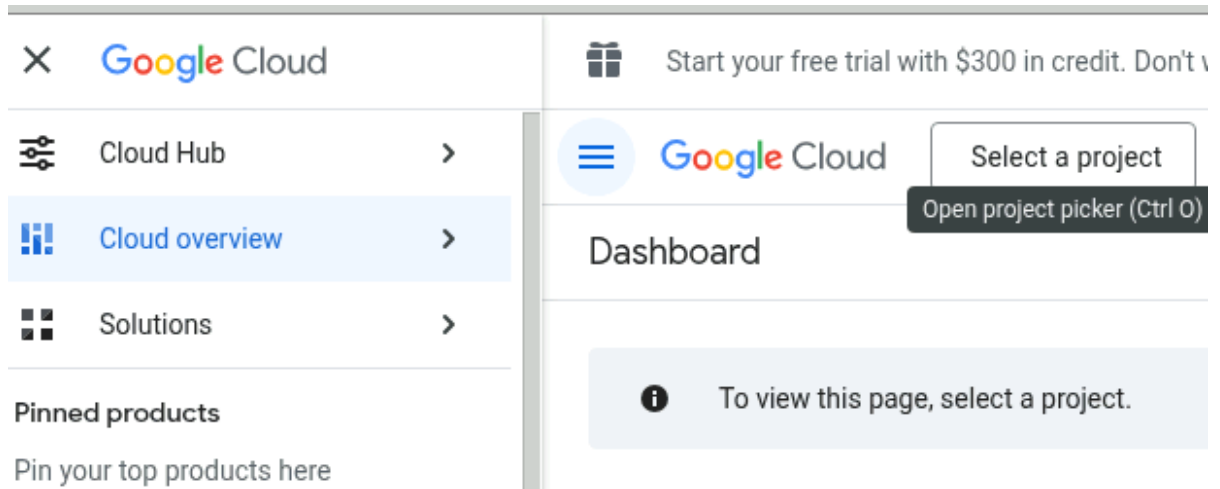
Activities

Note:

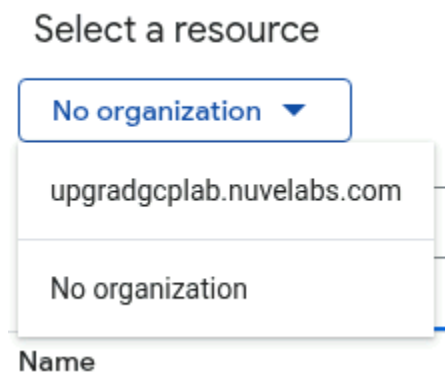
1. In the GCP Console, click the project drop-down and select the exact Project ID as specified in the "**Access**" tab on the "**Lab Access**" page. Using the wrong project may lead to validation failures.
2. If you encounter errors such as "**Retry budget exhausted**" or permission/role-related issues while creating the cluster, wait a few minutes or restart the required APIs. If the issue persists, try disabling and re-enabling both the Compute Engine API and the Kubernetes Engine API, then attempt the operation again.
3. To ensure automated grading functions are correct, make sure to use the exact resource names and configurations provided in the instructions.

Set Up GCP Environment:

- In the Cloud Overview dashboard page, click on Select a project.



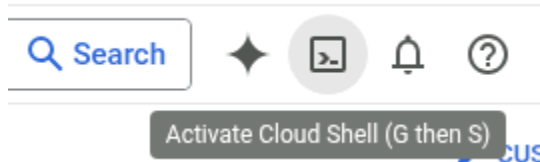
- For organisation, select upgradgcplab.nuvelabs.com




- Select the project under this organisation.

Name	Type	ID
▼ upgradgcplab.nuvelabs.com	Organization	282910303075
upgradlabs-1748929025392	Project	upgradlabs-1748929025392

- Click on the Activate cloud shell. Click on continue. Click on Authorise.



- In the Cloud Overview dashboard page, copy the Project ID and save it somewhere which will be used in later steps.



Project info

Project name
upgradlabs-1748929025392

Project number
413246426591

Project ID
upgradlabs-1748929025392

Task 1. Building the Java Application

Goal: Compile and package the Java application

What you create: my-web-app-1.0-SNAPSHOT.jar

Navigate to the Project folder on the Desktop. Use Maven to build the Java application.

- Open the terminal and navigate to the Desktop/Project folder where your Java source code is located.

cd ~/Desktop/Project

- Run `mvn clean install` to compile the application and generate the JAR which cleans any previous builds and compile the Java application to create a fresh JAR file.

mvn clean install

```
[INFO] Replacing main artifact /home/labuser/Desktop/Project/target/my-web-app-1.0-SNAPSHOT.jar with repackaged archive, adding nested
dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to /home/labuser/Desktop/Project/target/my-web-app-1.0-SNAPSHOT.jar.original
[INFO] --- install:3.1.1:install (default-install) @ my-web-app ---
[INFO] Installing /home/labuser/Desktop/Project/pom.xml to /home/labuser/.m2/repository/com/example/my-web-app/1.0-SNAPSHOT/my-web-app-
1.0-SNAPSHOT.pom
[INFO] Installing /home/labuser/Desktop/Project/target/my-web-app-1.0-SNAPSHOT.jar to /home/labuser/.m2/repository/com/example/my-web-
app/1.0-SNAPSHOT/my-web-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.074 s
[INFO] Finished at: 2025-06-04T18:04:51Z
[INFO] -----
labuser@ip-172-31-20-74:~/Desktop/Project$
```

- Navigate to the target/ directory.

cd target

- Run the JAR file to test it locally to confirm the build was successful and the application works as expected.

java -jar my-web-app-1.0-SNAPSHOT.jar

```
labuser@ip-172-31-20-74:~/Desktop/Project$ cd target/
labuser@ip-172-31-20-74:~/Desktop/Project/target$ java -jar my-web-app-1.0-SNAPSHOT.jar
```



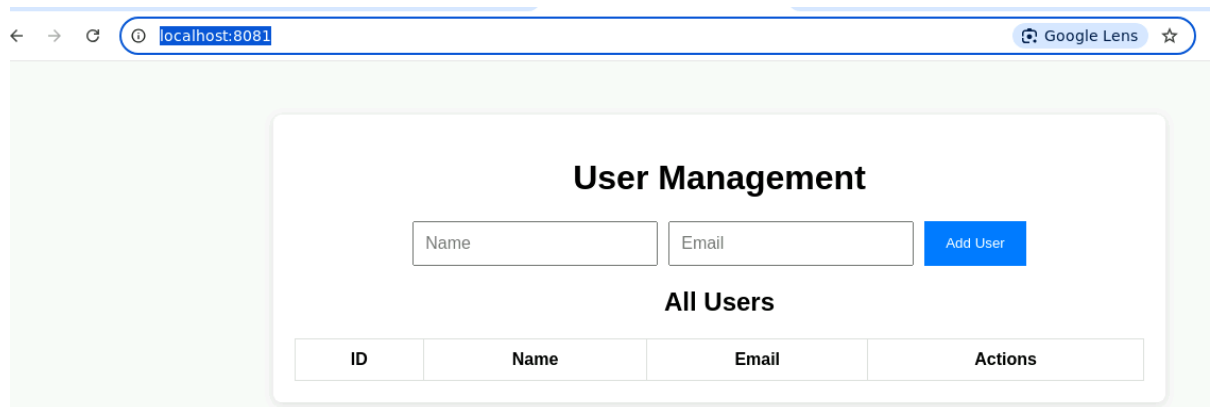
```

=====
:: Spring Boot ::
=====
2025-06-04T18:09:56.401Z INFO 4448 --- [main] com.example.WebAppApplication : Starting WebAppApplication v1.0-SNAPSHOT using Java 17.0.13 with PID 4448 (/home/labuser/Desktop/Project/target/my-web-app-1.0-SNAPSHOT.jar started by labuser in /home/labuser/Desktop/Project/target)

```

- To test it locally:

<http://localhost:8081/>



ID	Name	Email	Actions
----	------	-------	---------

- Enter Ctrl+c

Task 2. Creating the Terraform Backend Bucket

Goal: Store Terraform state remotely in GCS

What you create: Versioned GCS bucket

Use `gsutil` to create a versioned Terraform backend bucket in the gcp cloud shell.

- Set your GCP project ID as an environment variable for reuse in upcoming commands.

export PROJECT_ID=<your-gcp-project-id>

- Create a Cloud Storage bucket named logix-terraform-state in the US Central region with STANDARD storage.

gsutil mb -p \$PROJECT_ID -c STANDARD -l us-central1 gs://logix-terraform-state<any-identifier>/

- Enable versioning on the bucket to retain Terraform state history.

gsutil versioning set on gs://logix-terraform-state<any-identifier>/

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to guidedlabs-1686748562261.
Use 'gcloud config set project [PROJECT_ID]' to change to a different project.
badiya_ce1b9b27_7a0f_457e_adcd_7@cloudshell:~ (guidedlabs-1686748562261)$ export PROJECT_ID=guidedlabs-1686748562261
badiya_ce1b9b27_7a0f_457e_adcd_7@cloudshell:~ (guidedlabs-1686748562261)$ gsutil mb -p $PROJECT_ID -c STANDARD -l us-central1 gs://logix-terraform-state04/
Creating gs://logix-terraform-state04/...
badiya_ce1b9b27_7a0f_457e_adcd_7@cloudshell:~ (guidedlabs-1686748562261)$ gsutil versioning set on gs://logix-terraform-state04/
Enabling versioning for gs://logix-terraform-state04/...
badiya_ce1b9b27_7a0f_457e_adcd_7@cloudshell:~ (guidedlabs-1686748562261)$
```

Task 3. Enabling Required GCP APIs

Goal: Activate necessary services for deployment

What you create: Enabled services for Terraform, GKE, Cloud Run, IAM, and more

Enable all the required services using the gcloud CLI.

- Enable APIs: Compute Engine, GKE, Cloud Run, Artifact Registry, IAM, and Cloud Resource Manager.
- Use the below command with gcloud services to activate all APIs.

gcloud services enable compute.googleapis.com container.googleapis.com artifactregistry.googleapis.com run.googleapis.com iam.googleapis.com cloudresourcemanager.googleapis.com

```
badiya_ce1b9b27_7a0f_457e_adcd_7@cloudshell:~ (guidedlabs-1686748562261)$ gcloud services enable compute.googleapis.com container.googleapis.com artifactregistry.googleapis.com run.googleapis.com iam.googleapis.com cloudresourcemanager.googleapis.com
Operation "operations/acf.p2-597098670-22788ef7-5a14-47fe-80a9-ad04d47b029b" finished successfully.
```


Task 4. Configure kubectl and Cluster Access

Goal: Set up kubectl and authenticate with GKE

What you create: Cluster access for Kubernetes CLI

Set up kubectl CLI to access GKE.

- In the terminal, go to the home directory

cd ~

- Check kubectl version.

kubectl version --client

```
labuser@ip-172-31-20-74:~/Desktop/Project/target$ cd ~
labuser@ip-172-31-20-74:~$ kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
```

- Authenticate using `gcloud auth login` which opens browser to log into your **Google account** and Set the active GCP project.

gcloud auth login


gcloud config set project <your-project-id>


```
labuser@ip-172-31-20-74:~$ gcloud auth login # Open browser to log into your Google account

gcloud config set project guidedlabs-1686748562261
Your browser has been opened to visit:


https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fsqlservice.login+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=SjX3s1CGKIzU1l1L2NVGGYb0ZxuhJC&access_type=offline&code_challenge=xY_xjshbxf80bl_3B_WN78c-ep0CSsdKj7zKvkkTpc&code_challenge_method=S256

You are now logged in as [badiya_celb9b27-7a0f-457e-adcd-74e03b69f6c5@guidedlabsgcp.cloudloka.com].
Your current project is [None]. You can change this setting by running:
$ gcloud config set project PROJECT_ID
Updated property [core/project].
```

 Sign in with Google



Sign in to Google Cloud SDK

 badiya_ce1b9b27-7a0f-457e-adcd-74e03b69f6c5...

By continuing, Google will share your name, email address and profile picture with Google Cloud SDK. See Google Cloud SDK's privacy policy and Terms of Service.


You can manage Sign in with Google in your [Google Account](#).

Cancel Continue


NOTE: Click Continue. Click Allow.

- To set Application Default Credentials. After logging in, run this command to set the application default credentials:

gcloud auth application-default login

 Sign in with Google

Google Auth Library wants access to your Google Account

 chavan_8b776d88-872c-41da-9d99-5d8dfe305e9a@gui...

Select what [Google Auth Library](#) can access

☒ Select all

- ☒ See, edit, configure and delete your Google Cloud data and see the email address for your Google Account.. [Learn more](#)
- ☒ View and sign in to your Google Cloud SQL instances. [Learn more](#)

Note: Enable "Select all" and click Continue.

- Install the `gke-gcloud-auth-plugin` to authenticate kubectl with GKE.

export

GOOGLE_APPLICATION_CREDENTIALS="\$HOME/.config/gcloud/application_default_credentials.json"

ls -l ~/.config/gcloud/application_default_credentials.json

sudo apt-get install -y google-cloud-sdk-gke-gcloud-auth-plugin

5. Install and Configure Terraform

Goal: Install Terraform and initialize config

What you create: Terraform CLI and backend config

Install Terraform and prepare configuration files.

- Add the HashiCorp repository and install Terraform.

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-  
common curl
```

```
# Install dependencies for HashiCorp packages
```

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
```

```
# Add HashiCorp's official GPG key
```

```
sudo apt-add-repository "deb https://apt.releases.hashicorp.com  
$(lsb_release -cs) main"
```

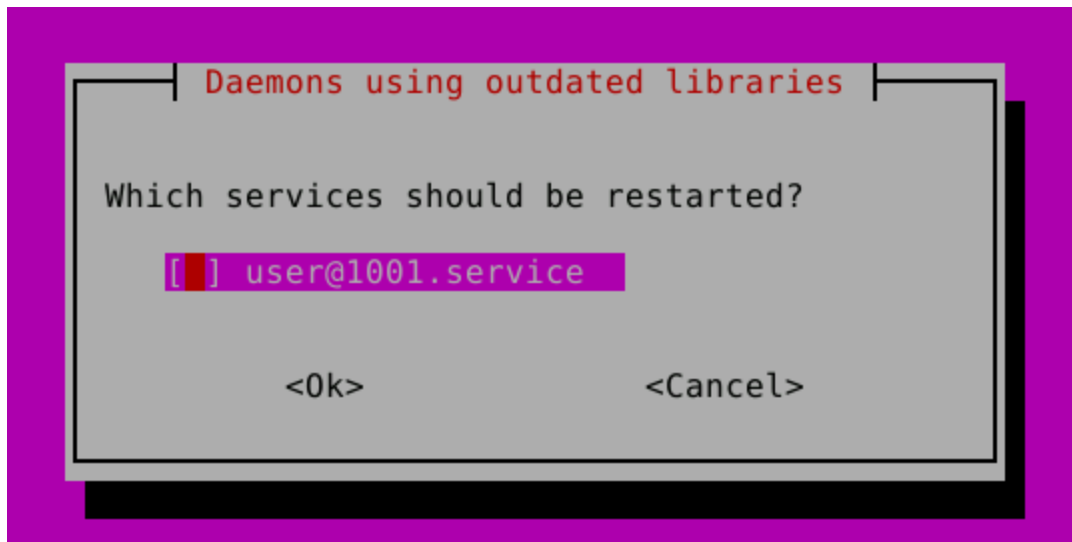
```
# Add HashiCorp's repository to apt sources
```

```
sudo apt-get update && sudo apt-get install terraform
```

```
# Install Terraform CLI
```

```
terraform -version
```

```
# Confirm installation
```



If you get something like above, click tab and enter.

Press Enter.

```
Running kernel seems to be up-to-date.
Restarting services...
Service restarts being deferred:
  systemctl restart user@1001.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
Terraform v1.12.1
on linux_amd64
_
```

- Navigate to the Project > Terraform folder and review the files: main.tf, variables.tf, backend.tf.

main.tf:

```
provider "google" { project = var.project_id region = var.region }

resource "google_compute_network" "vpc_network" { name = "logix-vpc" }

resource "google_compute_subnetwork" "subnet" { name = "logix-subnet"
ip_cidr_range = "10.0.0.0/16" region = var.region network =
google_compute_network.vpc_network.name }
```

```

resource "google_container_cluster" "primary" { name = "logix-gke" location =
var.region remove_default_node_pool = true initial_node_count = 1 network =
google_compute_network.vpc_network.name subnetwork =
google_compute_subnetwork.subnet.name }

resource "google_container_node_pool" "primary_nodes" { cluster =
google_container_cluster.primary.name location = var.region name = "default-
node-pool" node_count = 2

node_config { machine_type = "e2-medium" } }

resource "google_artifact_registry_repository" "app_repo" { provider = google
location = var.region repository_id = "logix-repo" format = "DOCKER" }

resource "google_cloud_run_service" "stub" { name = "logix-cloudrun" location =
var.region

template { spec { containers { image = "gcr.io/cloudrun/hello" } } }

traffic { percent = 100 latest_revision = true } }

```

variables.tf:

```

variable "project_id" {}
variable "region" {
  default = "us-central1"
}

```

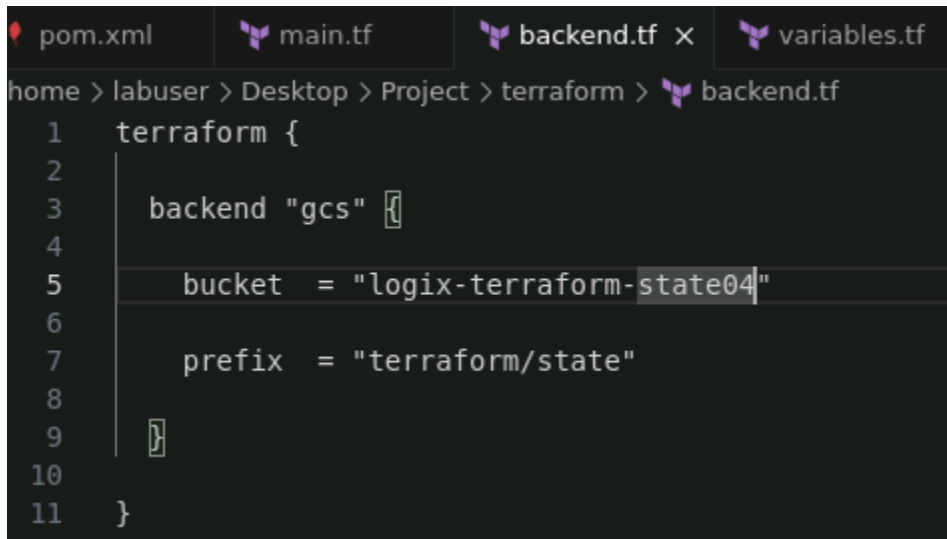
backend.tf:

```

terraform {
  backend "gcs" {
    bucket  = "logix-terraform-state01"
    prefix  = "terraform/state"
  }
}

```

- Set up `backend.tf` to use the previously created GCS bucket.



The screenshot shows a code editor with four tabs: `pom.xml`, `main.tf`, `backend.tf` (active), and `variables.tf`. The active tab displays the following Terraform configuration for the backend:

```
1 terraform {
2
3   backend "gcs" {
4
5     bucket = "logix-terraform-state04"
6
7     prefix = "terraform/state"
8   }
9 }
10
11 }
```

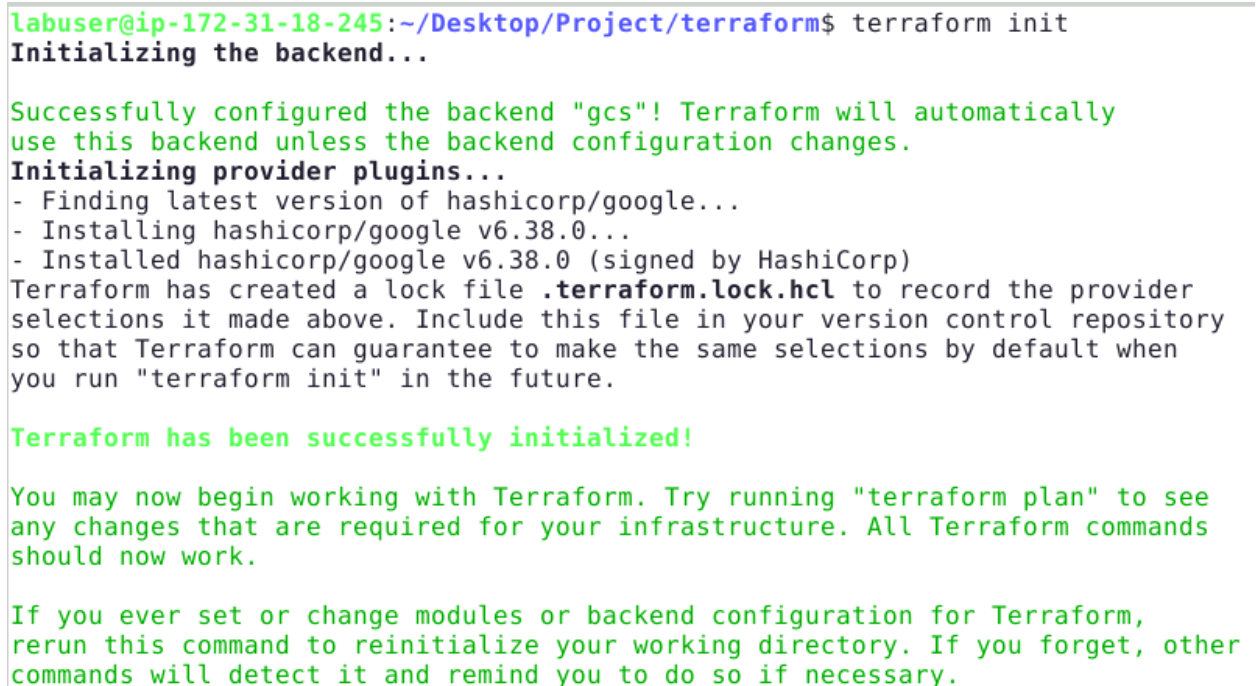
- Initialize and apply the configuration using Terraform commands.

`cd ~/Desktop/Project/terraform`

- To initialize the Terraform working directory and connect to the backend:

`terraform init`

Initialize backend



The screenshot shows the terminal output of the `terraform init` command. The output is as follows:

```
labuser@ip-172-31-18-245:~/Desktop/Project/terraform$ terraform init
Initializing the backend...

Successfully configured the backend "gcs"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing provider plugins...
- Finding latest version of hashicorp/google...
- Installing hashicorp/google v6.38.0...
- Installed hashicorp/google v6.38.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- To run the configuration and create the defined infrastructure in GCP:

terraform apply -var="project_id=<your-project-id>"

Enter yes

(This may take some time around 10minutes)

```
google_container_node_pool.primary_nodes: Still creating... [01m40s elapsed]
google_container_node_pool.primary_nodes: Still creating... [01m50s elapsed]
google_container_node_pool.primary_nodes: Still creating... [02m00s elapsed]
google_container_node_pool.primary_nodes: Creation complete after 2m1s [id=projects/guidedlabs-1686748563544/locations/us-central1/clusters/logix-gke/nodePools/default-node-pool]
Releasing state lock. This may take a few moments...

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
```

6. Building and pushing the Docker Image

Goal: Create and upload Docker image to Artifact Registry

What you create: Docker image in your registry

Navigate to the Project folder and create a Dockerfile.

1. Navigate to the Project folder on the Desktop.

```
cd ..
```

2. Open the Dockerfile using Visual Studio Code in the VM lab.
3. Build the Docker image using the configurations below:

Stage 1: Build the project using “maven:3.9.9-eclipse-temurin-17.”

- Set the working directory inside the container.
- Copy the *pom.xml* and source code.
- Build the Maven project.

Stage 2: Run the project using openjdk:17-slim

- Copy the JAR file from the build stage.
- Expose the application port on *port 8081*.
- Run the JAR file.

Dockerfile:

Step 1: Build the Spring Boot application

Use an official Maven image to build the application

FROM maven:3.9.9-eclipse-temurin-17 AS build

Set the working directory

WORKDIR /app

Copy the pom.xml and source code into the container

COPY pom.xml . COPY src ./src

Run Maven to build the application (creates the .jar file)

RUN mvn clean package

Step 2: Prepare the runtime environment

Use a slim OpenJDK image to run the application

FROM openjdk:17-jdk-slim

Set the working directory for the runtime container

WORKDIR /app

Copy the compiled jar file from the build stage

COPY --from=build /app/target/my-web-app-1.0-SNAPSHOT.jar app.jar

Expose the port the application will run on

EXPOSE 8081

Command to run the application

ENTRYPOINT ["java", "-jar", "app.jar"]

4. Authenticate Docker with Artifact Registry which set up Docker to authenticate with Google Artifact Registry in the US Central region.

```
gcloud auth configure-docker us-central1-docker.pkg.dev
```



```

labuser@ip-172-31-18-245:~/Desktop/Project$ gcloud auth configure-docker us-cent
rall-docker.pkg.dev
Adding credentials for: us-central1-docker.pkg.dev
After update, the following will be written to your Docker config file located
at [/home/labuser/.docker/config.json]:
{
  "credHelpers": {
    "us-central1-docker.pkg.dev": "gcloud"
  }
}

Do you want to continue (Y/n)? y

Docker configuration file updated.

```

5. Build the Docker image with the name backend:latest. and tag it properly with the registry path:

```
cd ~/Desktop/Project
```

```
docker build -t us-central1-docker.pkg.dev/<project-id>/logix-repo/backend:latest .
```

NOTE: Update your project-id

```

=> [stage-1 1/3] FROM docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b 4.6s
=> => resolve docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b3cb27e3e 0.0s
=> => sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc12 547B / 547B 0.0s
=> => sha256:779635c0c3d23cc8dbab2d8c1ee4cf2a9202e198dfc8f4c 953B / 953B 0.0s
=> => sha256:37cb44321d0423bc57266a3bff658daf00478e4cdf2 4.80kB / 4.80kB 0.0s
=> => sha256:1fe172e4850f03bb45d41a20174112bc119fbfec4 31.38MB / 31.38MB 0.5s
=> => sha256:44d3aa8d076675d49d85180b0ced9daef210fe4fdff 1.58MB / 1.58MB 0.7s
=> => sha256:6ce99fdf16e86bd02f6ad66a0e1334878528b5a 187.90MB / 187.90MB 2.3s
=> => extracting sha256:1fe172e4850f03bb45d41a20174112bc119fbfec42a650ed 1.6s
=> => extracting sha256:44d3aa8d076675d49d85180b0ced9daef210fe4fdff4bdbb 0.1s
=> => extracting sha256:6ce99fdf16e86bd02f6ad66a0e1334878528b5a4b5487850 1.7s
=> [stage-1 2/3] WORKDIR /app 5.8s
=> [build 2/5] WORKDIR /app 0.4s
=> [build 3/5] COPY pom.xml . 0.0s
=> [build 4/5] COPY src ./src 0.0s
=> [build 5/5] RUN mvn clean package 6.4s
=> [stage-1 3/3] COPY --from=build /app/target/my-web-app-1.0-SNAPSHOT.j 0.1s
=> exporting to image 0.1s
=> => exporting layers 0.0s
=> => writing image sha256:8dd69ea49c591689bd223fc5781cb2b0db9f53990dc1b 0.0s
=> => naming to us-central1-docker.pkg.dev/guidedlabs-1686748563544/logi 0.0s

```

6. Push the Docker image to Artifact Registry in the format:
us-central1-docker.pkg.dev/<project-id>/logix-
repo/backend:latest

```
docker push us-central1-docker.pkg.dev/<project-id>/logix-repo/backend:latest
```

```
labuser@ip-172-31-18-245:~/Desktop/Project$ docker push us-central1-docker.pkg.dev/guidedlabs-1686748563544/logix-repo/backend:latest
The push refers to repository [us-central1-docker.pkg.dev/guidedlabs-1686748563544/logix-repo/backend]
63ff6cadba8d: Pushed
60825f7e6f7c: Pushed
6be690267e47: Pushed
13a34b6fff78: Pushed
9c1b6dd6c1e6: Pushed
latest: digest: sha256:bb09864419f17000ba4da0c46a89db7a3815e635b781717b3b9445481d4143f7 size: 1371
```

7. Deploy to Google Kubernetes Engine (GKE)

Goal: Deploy application to a managed Kubernetes cluster

What you create: Kubernetes deployment and service

Create a folder k8s and add deployment and service YAML files.

mkdir k8s

cd k8s

Note: Update project-id in the deployment.yaml file

deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: logix-app

spec:

replicas: 2

selector:

matchLabels:

app: logix-app

template:

metadata:

labels:

app: logix-app

spec:

containers:

- name: backend

image: us-central1-docker.pkg.dev/<project-id>/logix-repo/backend:latest

ports:

- containerPort: 8081

service.yaml:

apiVersion: v1

kind: Service

metadata:

name: logix-service

spec:

selector:

app: logix-app

type: LoadBalancer

ports:

- port: 80

targetPort: 8081

- Retrieve cluster credentials using `gcloud container clusters get-credentials` which fetches credentials and configure context for your GKE cluster.

gcloud container clusters get-credentials logix-gke --region us-central1

```
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ gcloud container clusters get-credentials logix-gke --region us-central1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for logix-gke.
```

- Apply the Kubernetes manifests using `kubectl` to deploy the backend pod .

`kubectl apply -f deployment.yaml`

- Apply the Kubernetes manifests using `kubectl` to expose the backend via a LoadBalancer service:

`kubectl apply -f service.yaml`

```
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ kubectl apply -f deployment.yaml
deployment.apps/logix-app created
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ kubectl apply -f service.yaml
service/logix-service created
```

- To check if the pods are running:

`kubectl get pods`

```
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
logix-app-75b8f7459-472gc	1/1	Running	0	90s
logix-app-75b8f7459-wvl2h	1/1	Running	0	90s

- To check service details including external IP:

`kubectl get services`

- To confirm that the service correctly maps to the pods:

`kubectl get endpoints`

- Check that your service has an external IP address.

`kubectl get svc logix-service`

- Test it in the browser:

<http://<EXTERNAL-IP>>

User Management

Name Email [Add User](#)

All Users

ID	Name	Email	Actions
----	------	-------	---------

8. Deploy to Cloud Run and Testing the Cloud Run Deployment

Goal: Deploy a serverless container. Test and verify the deployment.

What you create: Cloud Run service

- Deploy the container image to Cloud Run with public access using the CLI. Use `gcloud run deploy` to deploy the image from Artifact Registry.
- Allow unauthenticated access. Set the platform to “managed”.
- Use region `us-central1`. Configure the container to listen on port 8081.
- To deploy the Docker image to Cloud Run with public access:

```
gcloud run deploy logix-cloudrun --image=us-central1-docker.pkg.dev/guidedlabs-1686748563544/logix-repo/backend:latest --platform=managed --region=us-central1 --allow-unauthenticated --port=8081
```

```
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ gcloud run deploy logix-cloudrun --image=us-central1-docker.pkg.dev/guidedlabs-1686748563544/logix-repo/backend:latest --platform=managed --region=us-central1 --allow-unauthenticated --port=8081
Deploying container to Cloud Run service [logix-cloudrun] in project [guidedlabs-1686748563544] region [us-central1]
✓ Deploying... Done.
✓ Creating Revision...
✓ Routing traffic...
✓ Setting IAM Policy...
Done.
Service [logix-cloudrun] revision [logix-cloudrun-00003-5pn] has been deployed and is serving 100 percent of traffic.
Service URL: https://logix-cloudrun-546196871088.us-central1.run.app
```

- Use `gcloud run services describe` to get the public URL of the Cloud Run service.

```
gcloud run services describe logix-cloudrun --region=us-central1 --format='value(status.url)'
```

- Run `curl <url>` to verify that the service is live and accessible which send a request to the service URL to check if it returns the expected response

curl https://logix-cloudrun-abcde12345-uc.a.run.app

```
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ gcloud run services describe logix-cloudrun --region=us-central1 --format='value(status.s.url)'
https://logix-cloudrun-rbqg7chhgq-uc.a.run.app
labuser@ip-172-31-18-245:~/Desktop/Project/k8s$ curl https://logix-cloudrun-rbqg7chhgq-uc.a.run.app
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management</title>
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
  }
  .container {
    max-width: 800px;
    margin: 50px auto;
    padding: 20px;
    background: #fff;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  }
```

Testcases

1. Checking Docker image is pushed to Artifact Registry under the correct project and repository. [20 marks]
2. Verify that the GCS bucket logix-terraform-state exists in the correct region (us-central1). [10 marks]
3. Verify the Repository name. [10 marks]
4. Ensure the Kubernetes cluster logix-gke is created in the specified region (e.g., us-central1). [20 marks]
5. Check that pods are running in the cluster (kubectl get pods). [20 marks]
6. Confirm Cloud Run service logix-cloud run is deployed in us-central1. [20 marks]