Deploy a highly available Sprint application

Revision History

Version	Date	Description	Author
1.0	03/10/2024	Initial version	Mohamed El Eraki
1.2	03/10/2024	Append pre-code section	Mohamed El Eraki
1.3	04/10/2024	Append Gradel build steps	Mohamed El Eraki
1.4	05/10/2024	Appebd Python build steps	Mohamed El Eraki

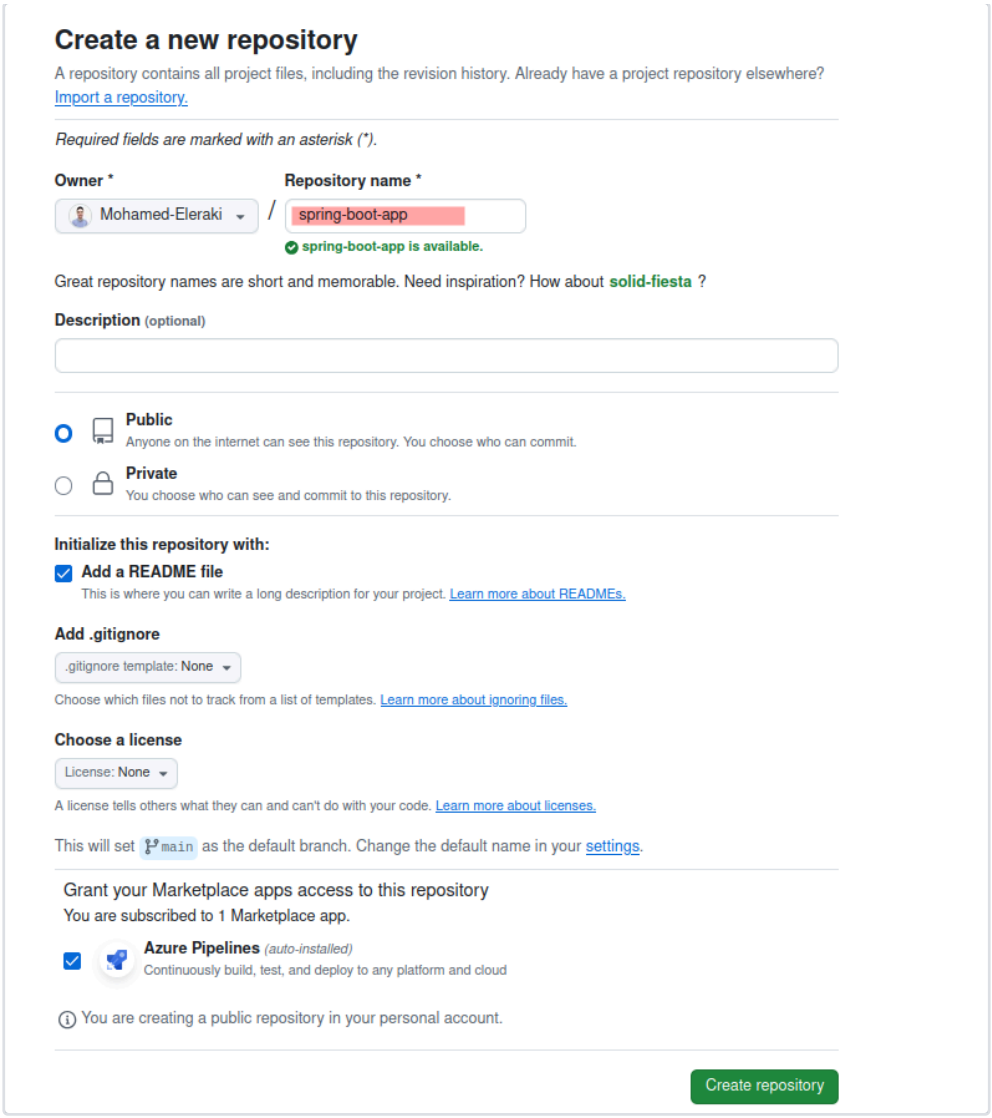
Resources

- Architecture Diagram.drawio
- Mohamed-Eleraki/spring-boot-app

Pipeline Implementations steps

Repository creation

Create a GitHub public repository called `spring-boot-app` as follows *Screenshot-01*



Screenshot-01

AWS Pipeline implementation

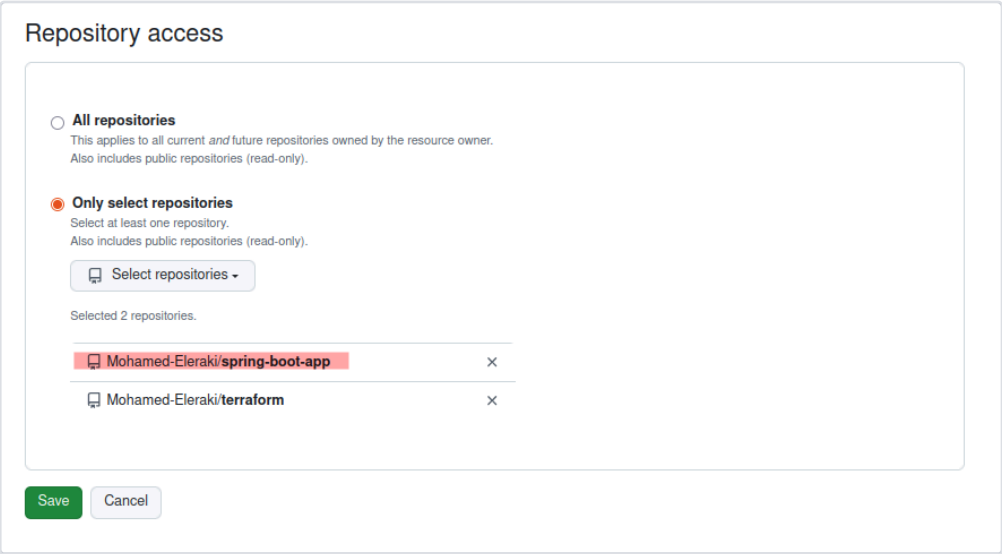
Craft an AWS pipeline integrated with The GitHub repository created, the pipeline will be triggered automatically based on tags assigned with the pushed commits that include “release*” value.

The pipeline will include multiple stages as the fetching source code stage, Plan stage, Manual approval stage, and Apply stage.

Implementations steps

- Open-up AWS console, in the search bar search for **CodePipeline.**
- From the CodePipeline main page press **Create Pipeline.**
- Set **pipeline name** as `eraki_springApp_us1_pipeline`.
- Set the **Execution mode** as **Superseded** avoiding overlapping deployments, leave all as defaults then **Next.**
- For **Source provider** specify **GitHub version 2,** Then **Connect to GitHub.**
- on the Pop-up page, for **Connection name** set `eraki_springApp_us1_connection`, Then **Connect to GitHub.**

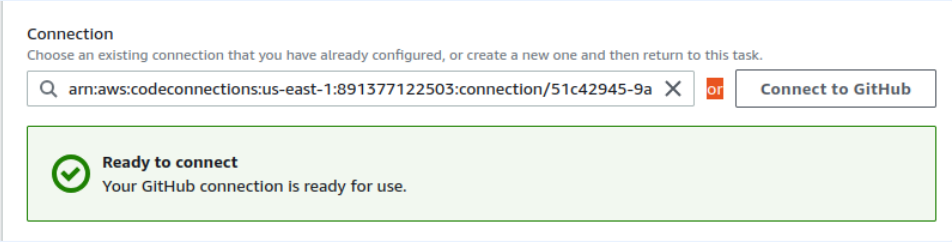
- press **Install a new app,** This will open-up the GitHub configuration page, specify the created branch, Or you can set it for all repositories.



Screenshot-02

- Press **Save,** Will revert you back to the connection creation page, press **connect.**

You should see **Ready to connect** as follows



Screenshot-03

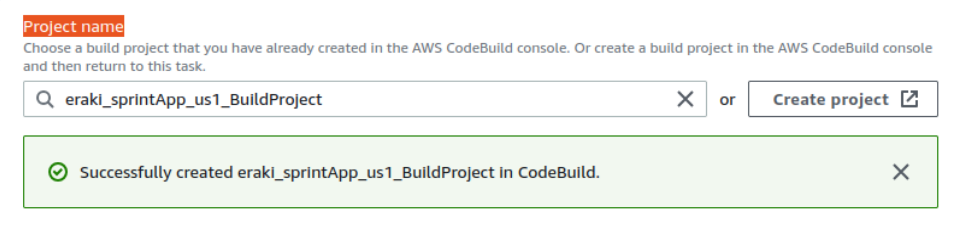
- on the list of **Repository name,** specify “spring-boot-app” repository.
- on the list of **Default branch,** specify “main”.
- For **Triggers,** Set **Trigger type** as Specify filter, **Event type** as push, and **Filter type** as Tags and for **Include** type “release-*” Then **Next.**

this ensures that the pipeline will fireup when a commit include **release-*** instead of setup it for a specific branch

- In **Build Stage** for **Build provider** Specify AWS **CodeBuild.**
- For **Project name** Press **Create project.**
- On the Pop-up page, for **Project name** set “eraki_sprintApp_us1_BuildProject”
- Leave all as default, and for **Build spec** specify “Use a buildspec file” Then **continue to pipeline.**

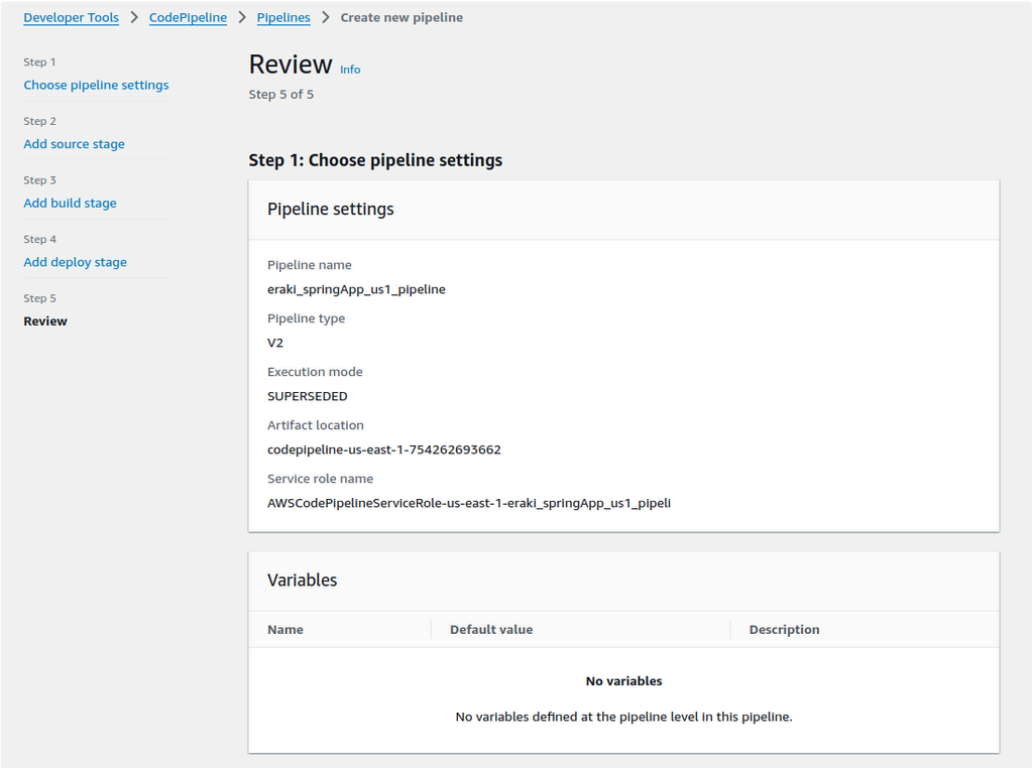
here we’re specifing the machine specs that the pipeline stages will run on it.

The result should be as follows:



Screenshot-04

- Press **Next**, and **skip deploy stage.**
- Review and Press **Create pipeline,** The pipeline configurations should be as follows *Screenshot-05, Screenshot-06, and Screenshot-07*



Screenshot-05

Step 2: Add source stage

Source action provider

Source action provider
GitHub (Version 2)

OutputArtifactFormat
CODE_ZIP

DetectChanges
false

ConnectionArn
arn:aws:codeconnections:us-east-1:891377122503:connection/51c42945-9a8b-466e-b1b8-aa0c11689607

FullRepositoryId
Mohamed-Eleraki/spring-boot-app

Default branch
main

Trigger configuration

You can add additional pipeline triggers after the pipeline is created.

Trigger type
Specify filter

Event type
Push

Filter type
Tags

Include tags
release-*

Exclude tags
-

Screenshot-06

Step 3: Add build stage

Build action provider

Build action provider
AWS CodeBuild

ProjectName
eraki_sprintApp_us1_BuildProject

Step 4: Add deploy stage

Deploy action provider

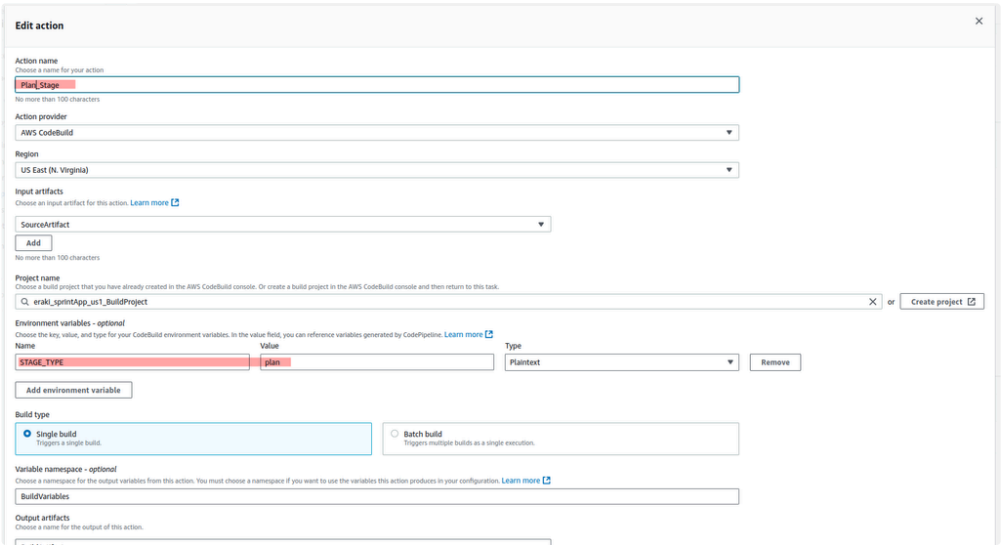
Deployment stage
No deploy

Cancel | Previous | Create pipeline

Screenshot-07

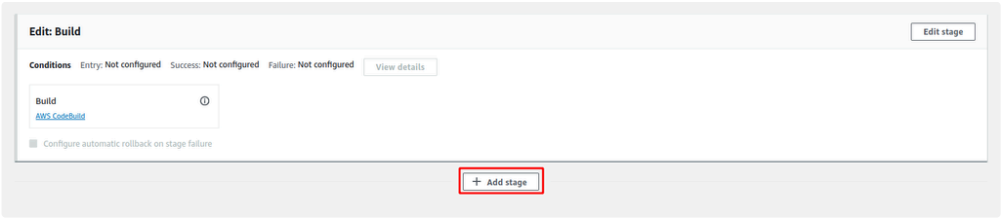
Now, that we have created the pipeline, let's create the pipeline stages in the following steps.

- Open your created pipeline, Then **Edit.**
- Edit **Build stage** as follows.
  Create a variable for your stages so you can differentiate them into your `buildspec.yml` file. The plan stage will include **terrafrom plan** command.



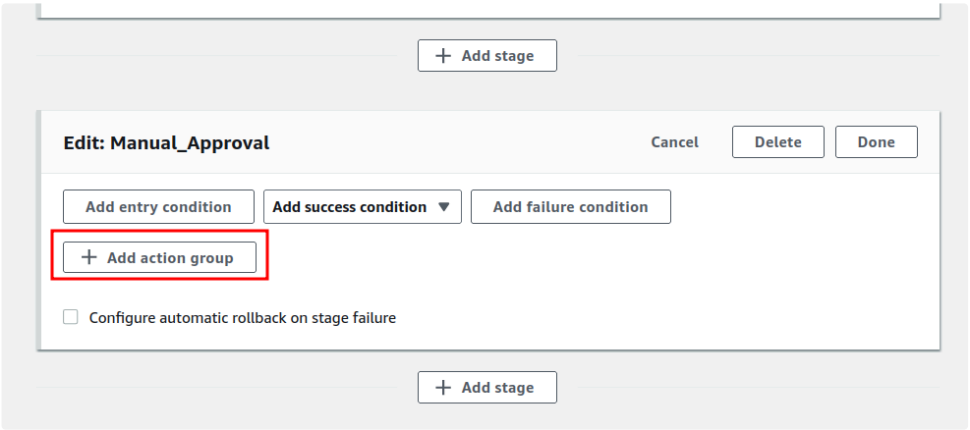
Screenshot-08

- Under build stage, **Add stage.**

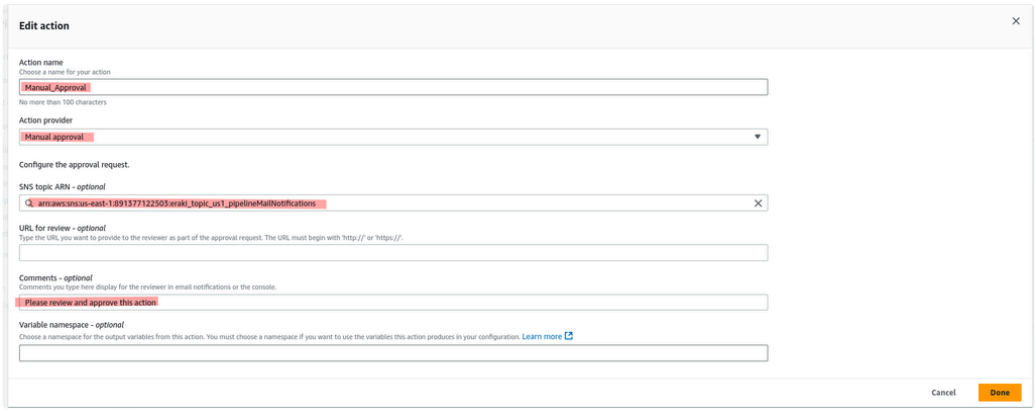


Screenshot-09

- For **stage name** set Manual_Approval.



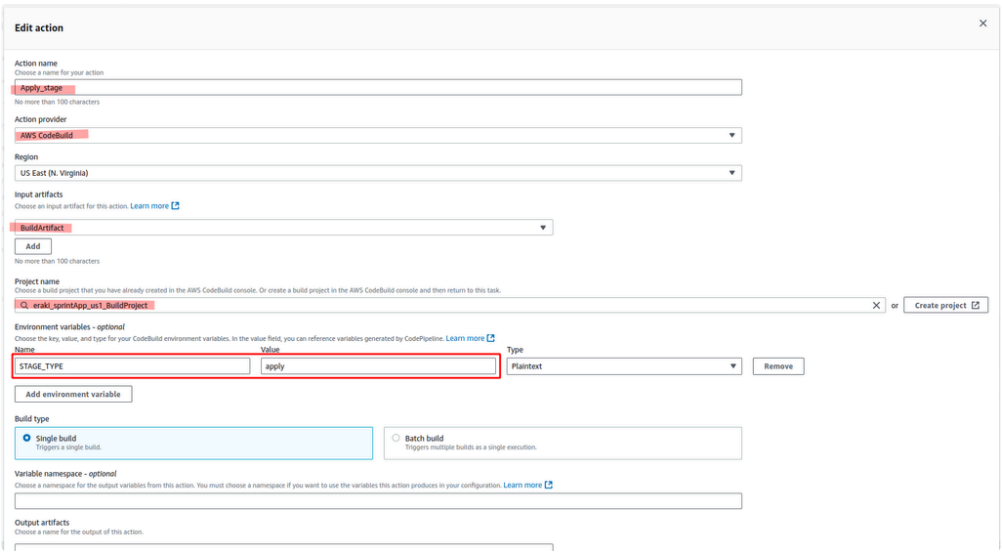
Screenshot-10



Screenshot-11

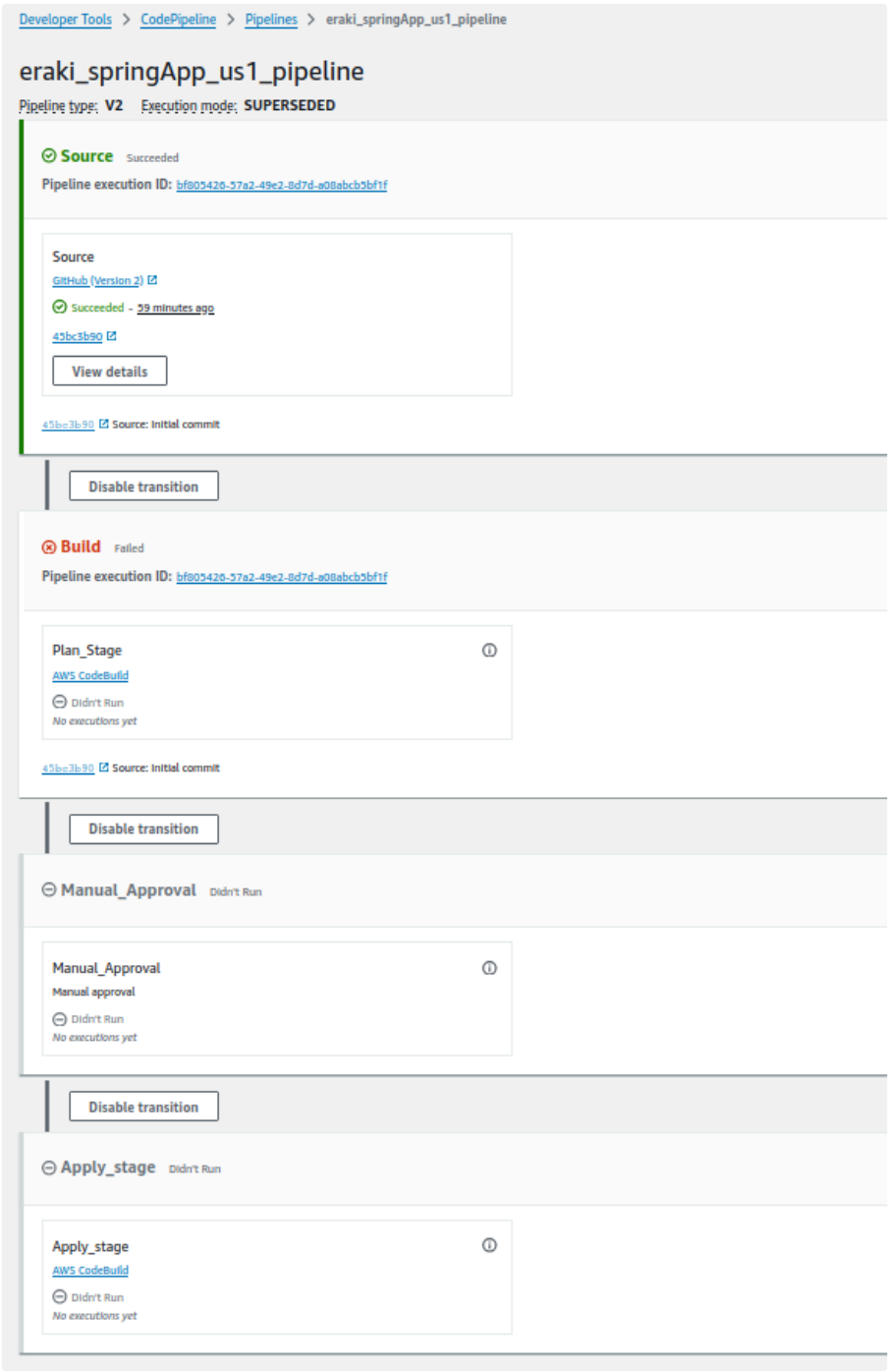
I have configured an SNS topic holds my e-mail for reciving notifications from pipeline

- Add a new stage for applying terraform code as follows.
  Create a variable for your stages so you can differentiate them into your `buildspec.yml` file. The apply stage will include **terrafrom apply** command.



Screenshot-12

The Pipeline stages should be as follows:



Fetch repos and configure `buildspec.yaml`

Now, let's configure the pipeline line YAML file `buildspec.yml` as follows.

- Clone down the repo into your local by running the following command:

```
git clone https://github.com/Mohamed-Eleraki/spring-boot-app.git
```

- Open the repo into your editor, and create the `buildspec.yml` file as follows:

```
version: 0.2
```

```
env:
  variables:
    TERRAFORM_VERSION: "1.5.6"
    #key: "value"
#
#  parameter-store:
#    key: "value"
#
#  secrets-manager:
#    key: "value"

phases:
  install:
    runtime-versions:
      python: 3.12

    on-failuer: ABORT
    commands: |
      echo "Installing terraform"
      yum install -y wget unzip
      yum clean all
      tf_version=$TERRAFORM_VERSION
      wget https://releases.hashicorp.com/terraform/"$TERRAFORM_VERSION"/terraform_"$TERRAFORM_VERSION"_linux_amd64.zip
      unzip terraform_"$TERRAFORM_VERSION"_linux_amd64.zip
      chmod 775 terraform
      mv terraform /usr/local/bin/
      terraform --version
      rm terraform_"$TERRAFORM_VERSION"_linux_amd64.zip
      ls -al /usr/local/bin/terraform

  pre_build:
    on-failure: ABORT
    commands: |
      if [ "$STAGE_TYPE" = "plan" ]; then
        echo "Filter on Plan stage";
        echo terraform plan started on `date`
        #cd "CODEBUILD_SRC_DIR/AWS_Demo/37-build-test-pipeline";
        cd "infra";
        ls -lathr;
        terraform init;
        terraform validate;
        terraform plan -out tfplan;
      else
        echo "No Plan stage";
      fi

  build:
    on-failure: ABORT
    commands: |
      if  [ "$STAGE_TYPE" = "apply" ]; then
        echo  "Filter on Apply stage";
        echo terraform execution started on `date`;
        ls -lathr;
        #cd "$CODEBUILD_SRC_DIR/AWS_Demo/37-build-test-pipeline";
        cd "infra";
        ls -lathr;
        terraform apply tfplan;
```

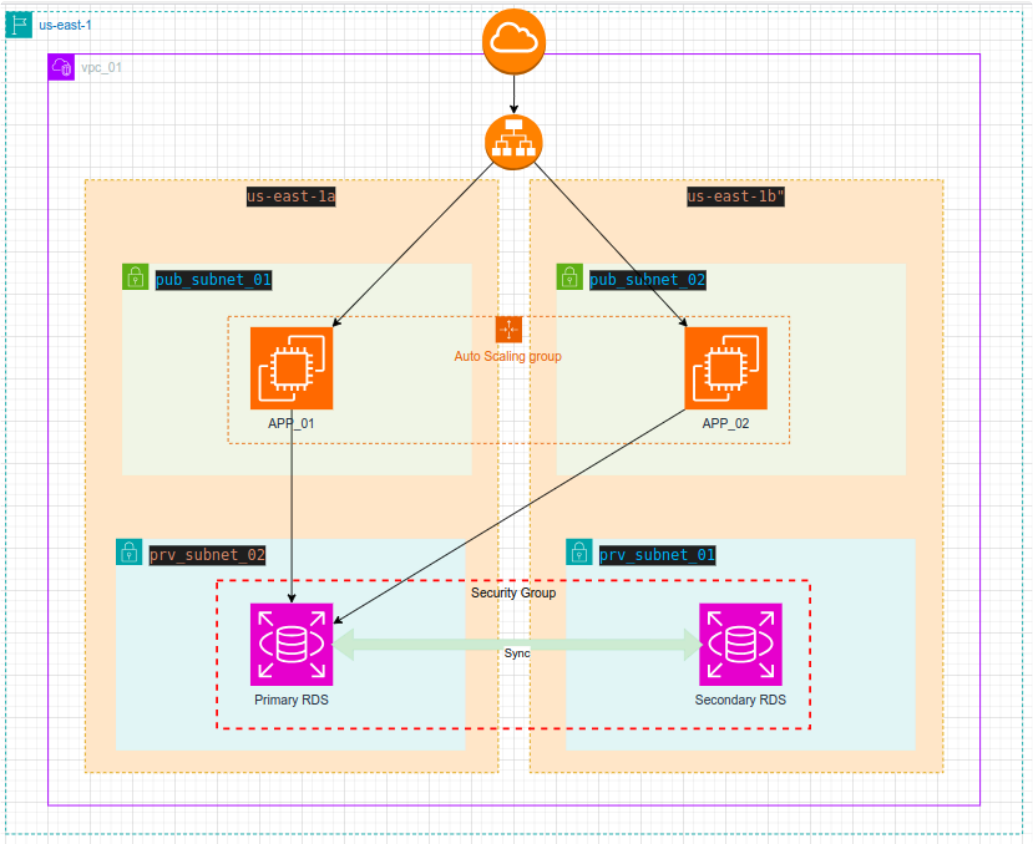
```
      else
        echo "No Apply stage";
      fi

  post_build:
    on-failure: CONTINUE
    commands: |
      echo "Fetching provisioning details"
      terraform show -json tfplan > tfplan.json
      yum install -y jq
      echo "print out terrafrom version and json format version"
      jq '.terraform_version, .format_version' tfplan.json
      echo ""
      echo "print out provider config"
      jq '.configuration.provider_config' tfplan.json
      echo ""
      echo "print out resource config"
      jq '.configuration.root_module.resources' tfplan.json
      echo ""
      echo "print out outputs"
      jq '.outputs' tfplan.json
      echo ""
      echo "print out resource changes"
      jq '.resource_changes' tfplan.json
      echo ""
      echo "print out resource config"
      jq '.configuration.root_module.resources' tfplan.json
      echo ""
      echo "print out provider config"
      jq '.configuration.provider_config' tfplan.json
      echo ""
      echo "print out provider config"
      jq '.configuration.provider_config' tfplan.json
      echo ""
      echo "print out lock file configuration"
      jq '.configuration.lock_version' tfplan.json
```

now the pipeline is ready, and will be triggered by setting a tag on commit using the following example command:

```
git log  # to get the commit hash
git tag release-1 1234  # 1234 refer to commit has
git push origin release-test01  # push the applied tag
```

Architecture diagram

build a highly available application using autoscaling group and load balancer besides building a PostgreSQL RDS that is highly available as an RDS Multi-az instance.



Pre-Code Preparation

Here we go, will build up the entire infrastructure as the diagram above in the following repository: https://github.com/Mohamed-Eleraki/spring-boot-app

- Create a Dev Branch, deploying the infrastructure components.

```
                                    /spring-boot-app$ git push origin release-test01
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Mohamed-Eleraki/spring-boot-app.git
 * [new tag]         release-test01 -> release-test01
                                    /spring-boot-app$ git checkout -b Dev
Switched to a new branch 'Dev'
                                    /spring-boot-app$ git branch -a
* Dev
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
                                   :/spring-boot-app$
```

Gradel application build steps

To build Gradel first you need to install the compatible Java jdk version using the following commands:

Check the compatible version of Gradel by viewing the following file

```
$ tree
.
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── HELP.md
├── settings.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── example
    │   │           └── demo
    │   │               └── DemoApplication.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── com
                └── example
                    └── demo
                        └── DemoApplicationTests.java

14 directories, 10 files

```

```
grep "sourceCompatibility" build.gradle
sourceCompatibility = '11'
```

Install the required version

Ubuntu

```
sudo apt-get install -y openjdk-11-jdk
```

Amazon linux 2

```
sudo yum install -y java-11-openjdk-devel
```

ensure the required Java version is set successfully

```
jave --version
```

if not set to 11, alternative by utilizing:

```
sudo update-alternatives --config java
```

ensure the JAVE_HOME variable is configured to use the Jave compatible version

- Check the variable value

```
$JAVE_HOME
```

- Adjust it to use Java 11 version

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Now, you're ready to build your Gradel application as follows:

```
./gradlew build
```

Run and test your application:

```
java -jar build/libs/<your-jar-file>.jar

Or with a specific port
java -jar build/libs/demo-0.0.1-SNAPSHOT.jar --spring.profiles.active=dev

Or
./gradlew bootRun
```

Check application default port configuration:

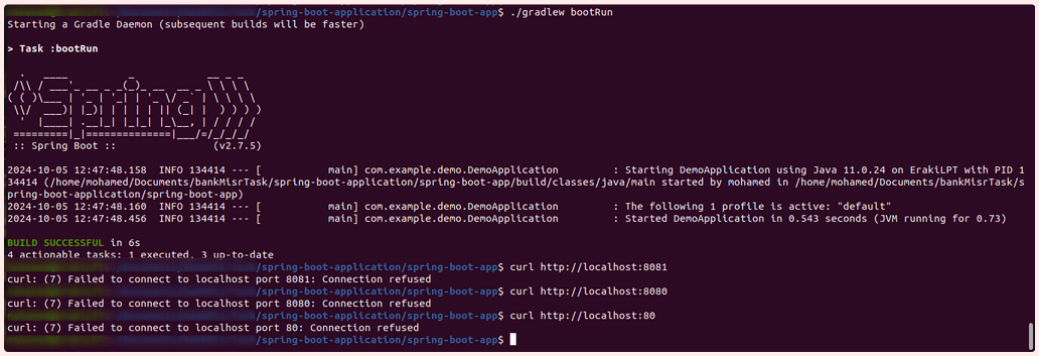
```
cat src/main/resources/application.properties

# Server configuration
server.port=8080
server.address=0.0.0.0

# Database configuration
spring.datasource.url=jdbc:postgresql://<rds-endpoint>:5432/<database-name>
spring.datasource.username=<your-username>
spring.datasource.password=<your-password>
spring.datasource.driver-class-name=org.postgresql.Driver

# JPA / Hibernate settings (optional, but commonly used)
spring.jpa.hibernate.ddl-auto=update

spring.profiles.active=default
```

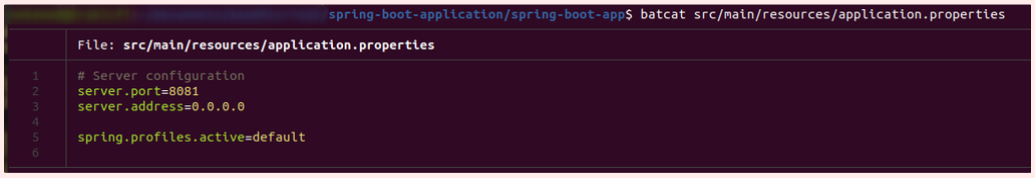
you should rebuild after adjusting this file!

To stop Gradel, kill the process

```
ps aux | grep -i gradel
kill -9 <PID>
```

the application has run successfully as below screenshots. However, the webapp won't work, and not accessible from the browser. even after configuring the application.properties file as below.





Python application build steps

Will use a simple flask app ensuing the RDS database accessible to application load balancer

This stps included into Iac code

Install the prerequisite:

```
sudo yum update -y
sudo yum install python3 -y
sudo yum install python3-pip -y
sudo yum install git -y
sudo pip3 install psycopg2-binary Flask
```

```
mkdir ~/webapp
cd ~/webapp
nano app.py
```

Flask App configuration

```python
from flask import Flask
import psycopg2

app = Flask(__name__)

def connect_db():
    try:
        conn = psycopg2.connect(
            host="terraform-20241005095652655600000004.cjeqis2oi24n.us-east-1.rds.amazonaws.com",  # Replace with your RDS endpoint
            database="postgres",  # Replace with your database name
            user="postgres",      # Replace with your username
            password="vIv8GrbF}WqKOk|lyAJh2e:]%YHR"   # Replace with your password
        )
        return "Connected to the database!"
    except Exception as e:
        return f"Failed to connect: {str(e)}"

@app.route('/')
def index():
    return connect_db()

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

Run the simple app

```
sudo python3 app.py
```

Accessing the load balancer