Part 1:

Reop: <https://github.com/daniel7491/hello-world-war>

Docker file:

FROM maven:3.8.4-openjdk-11 AS build

WORKDIR /app

COPY pom.xml .

COPY src ./src

RUN mvn clean package

FROM tomcat:9.0

COPY --from=build /app/target/\*.war /usr/local/tomcat/webapps/

EXPOSE 8082

ENTRYPOINT ["catalina.sh", "run"]

Docker build command: docker build -t hello-world-pj .

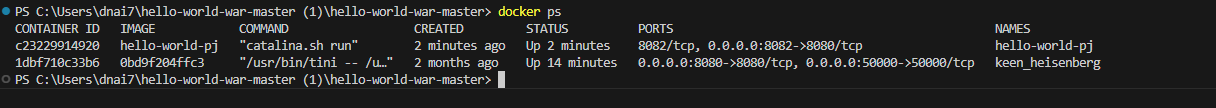
Docker run: docker run -d -p 8082:8080 --name hello-world-pj hello-world-pj

Docker tag: docker tag hello-world-pj daniel745/hello-world-pj:1

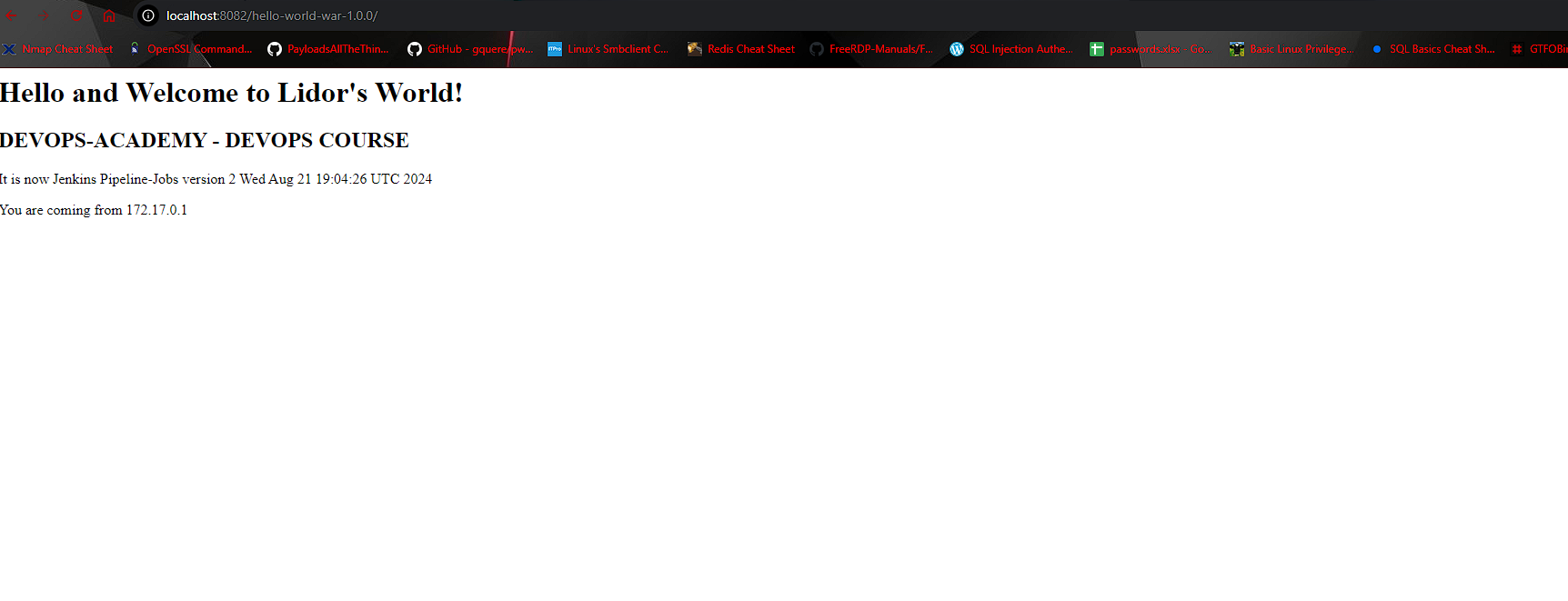
Docker push: docker push daniel745/hello-world-pj:1

Docker repo: https://hub.docker.com/repository/docker/daniel745/hello-world-pj/general

Docker ps:



Working app:



The docker file builds the image and exposes port 8082 so the address to reach the app will be

<http://localhost:8082/hello-world-war-1.0.0/>

in the commands you build the image, run it, tag it, and push it to docker hub.

Part 2:

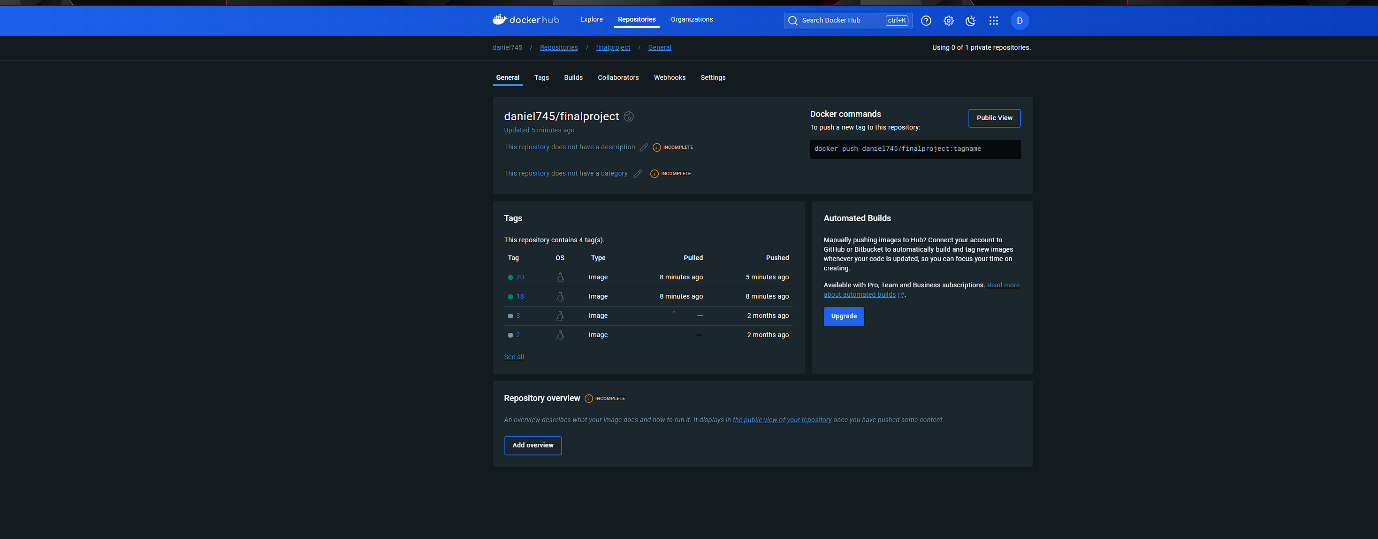
Jenkins builds the maven war file and sends it to sonarqube to get checked once its done it then builds the docker image and pushes it to docker hub.

The build triggers automatically using github webhooks that are detected in Jenkins using cms polling.

Once a commite is made the build triggers and a new image is uploaded to dockerhub.

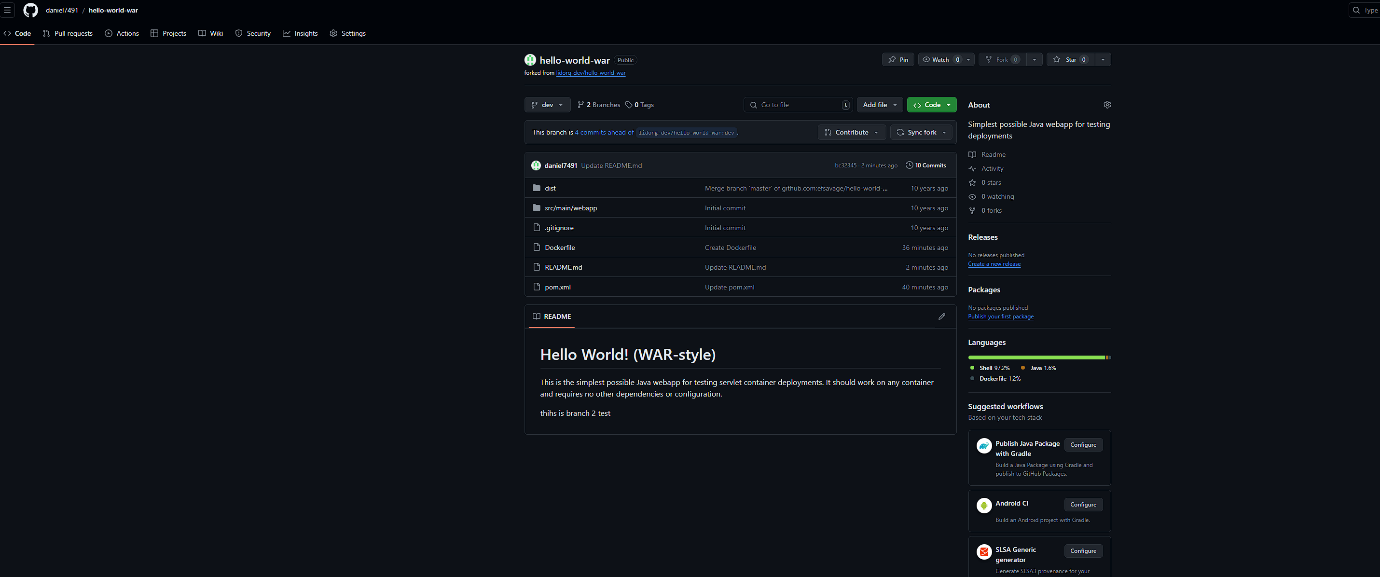
Dockerhub ripo:

https://hub.docker.com/repository/docker/daniel745/finalproject/general

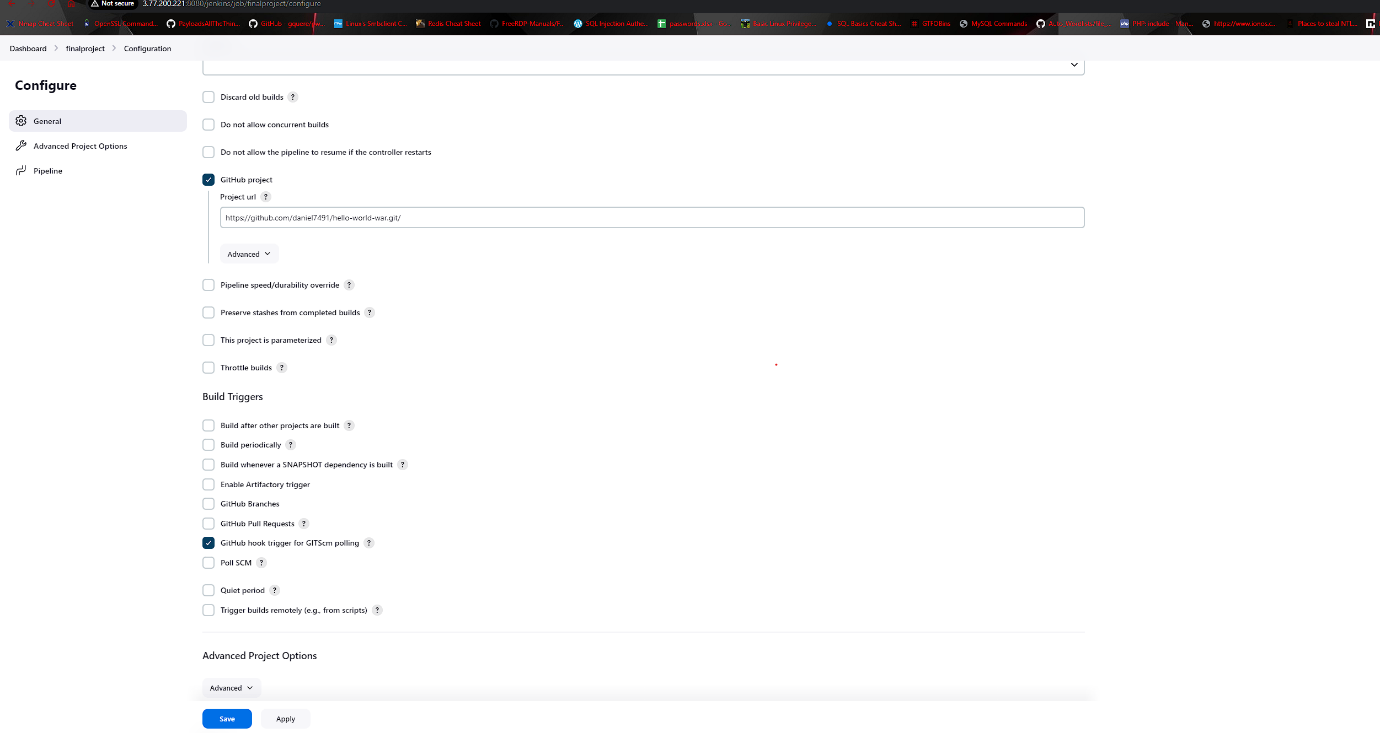


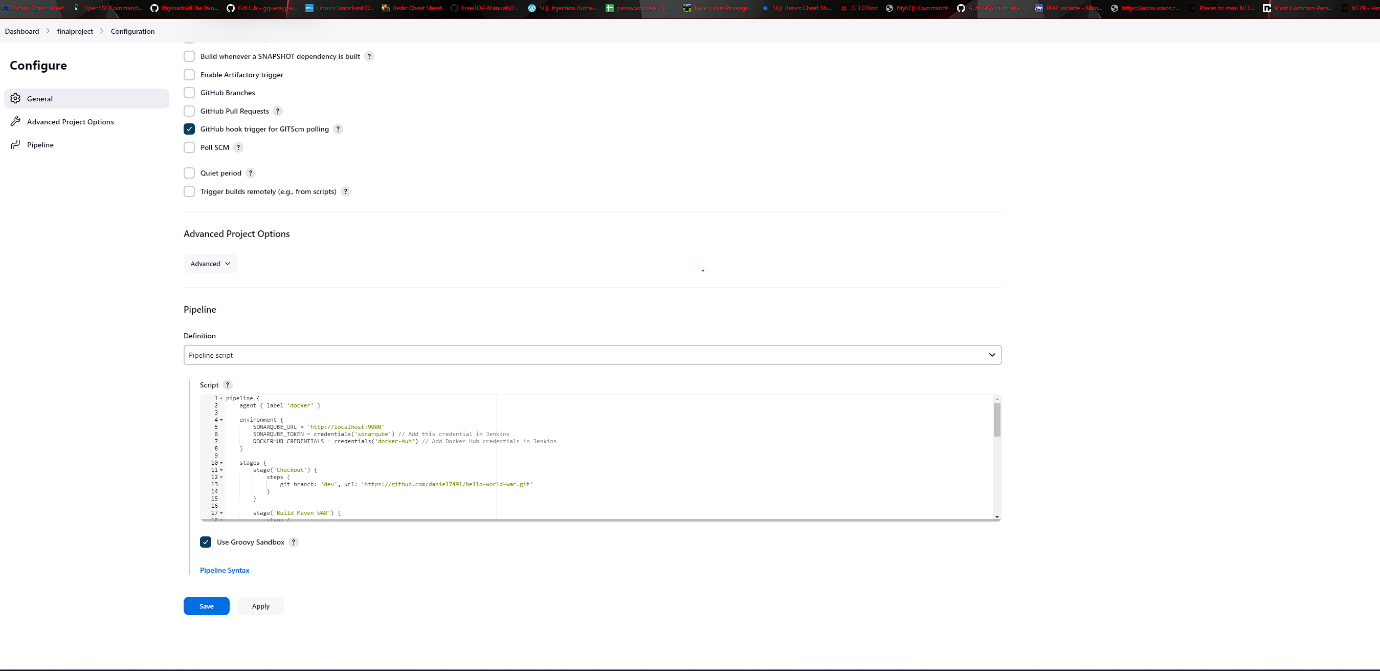
Github ripo:

https://github.com/daniel7491/hello-world-war

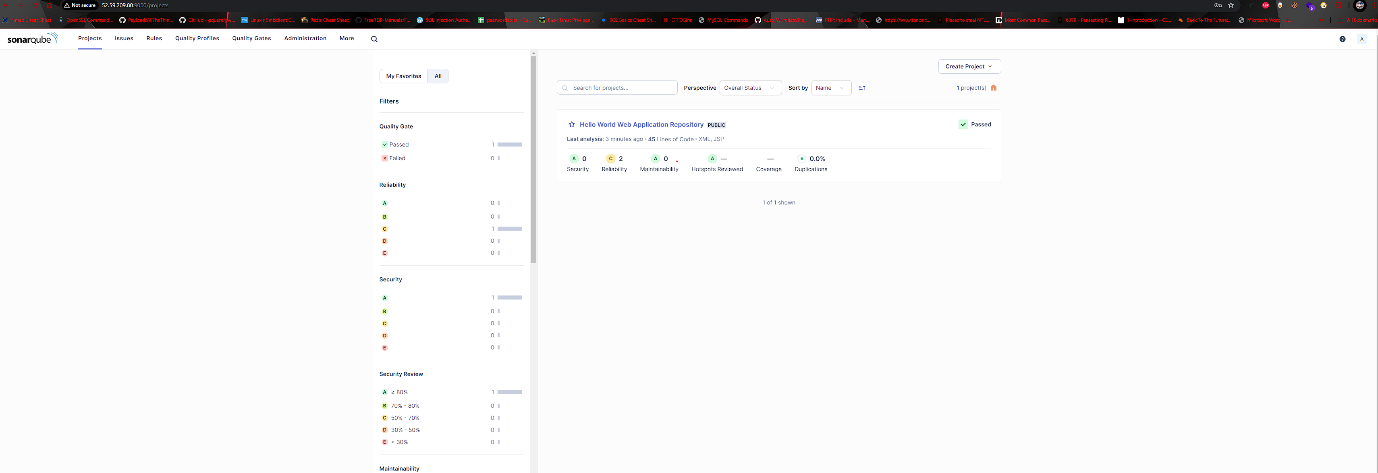


Jenkins configuration:

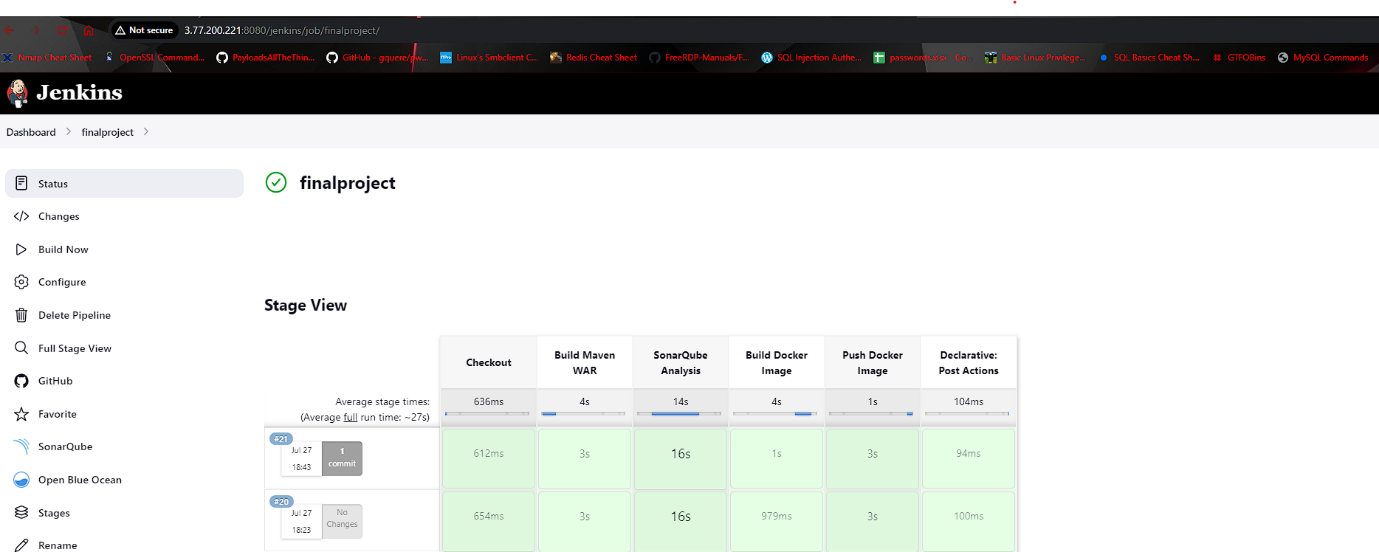




Sonarqube:



Build stages in Jenkins:



Groovy:

pipeline {

agent { label 'docker' }

environment {

SONARQUBE\_URL = 'http://localhost:9000'

SONARQUBE\_TOKEN = credentials('sonarqube')

DOCKERHUB\_CREDENTIALS = credentials('docker-hub')

}

stages {

stage('Checkout') {

steps {

git branch: 'dev', url: 'https://github.com/daniel7491/hello-world-war.git'

}

}

stage('Build Maven WAR') {

steps {

sh 'mvn clean package'

}

}

stage('SonarQube Analysis') {

steps {

withSonarQubeEnv('SonarQube') {

sh 'mvn sonar:sonar -Dsonar.projectKey=hello-world-war'

}

}

}

stage('Build Docker Image') {

steps {

script {

def buildId = env.BUILD\_ID

sh "docker build -t daniel745/finalproject:${buildId} ."

}

}

}

stage('Push Docker Image') {

steps {

script {

def buildId = env.BUILD\_ID

docker.withRegistry('https://index.docker.io/v1/', 'docker-hub') {

sh "docker push daniel745/finalproject:${buildId}"

}

}

}

}

}

post {

always {

cleanWs()

}

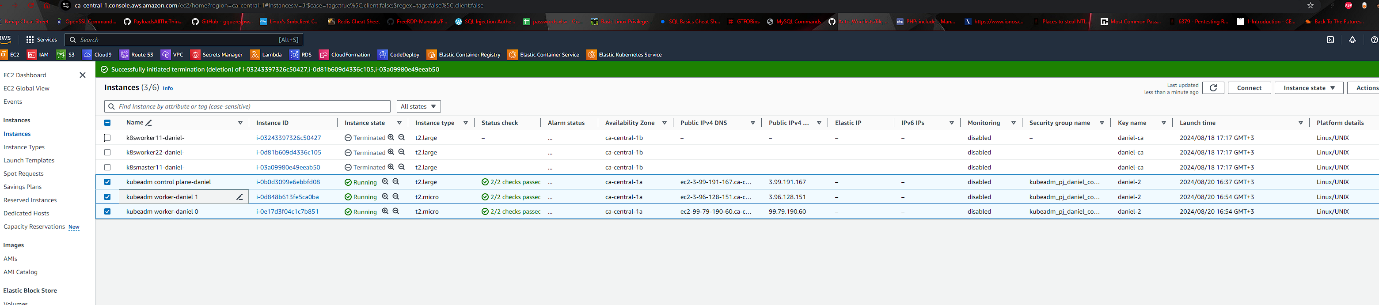
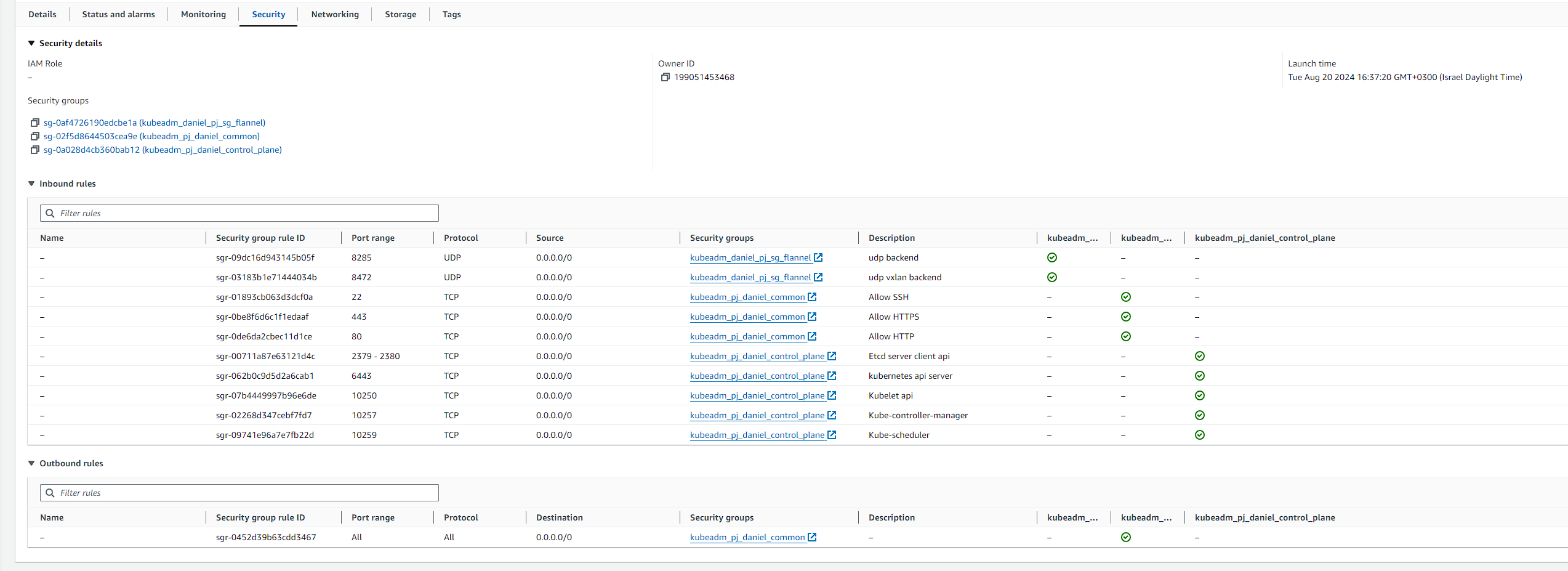
}

}

Part 3:

Repo for Terraform and Ansible: <https://github.com/daniel7491/kubeadm-with-terraform-ansible> The repo contains all the Terraform files, Ansible playbooks, and inventory file. We create 3 machines with Terraform: 1 master and 2 workers. Also, all the necessary security groups are created, along with the VPC, subnet, gateway, route table, and route table associations. We save their public IPs in a host file for use with Ansible. Ansible then installs Kubernetes on all the machines and, with separate files, configures the master and the workers.

Aws:

Commands to run:

In the project directory:

Terraform init.

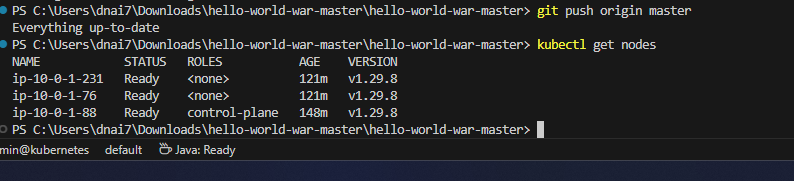
Terraform apply

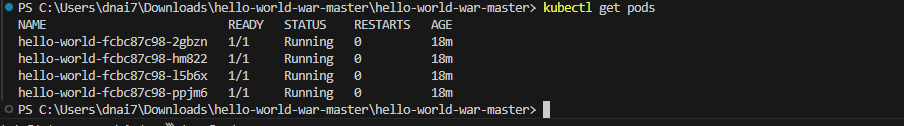
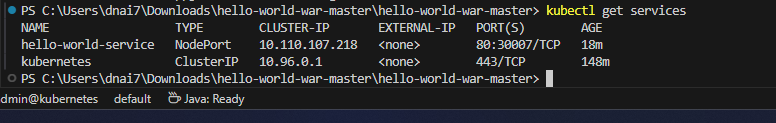
Then ansible commands :

ansible-playbook -i inventory kubeadm-setup.yml

ansible-playbook -i inventory kube\_master.yml

ansible-playbook -i inventory kube\_workers.yml



app repo that was used: <https://github.com/daniel7491/hello-world-war>

the deployment pulls an image from ecr and deploys it on the cluster that was built in the described proccess above.

Im using node port to access the app.

