DevOps Engineer - Case Study

Submitted By -
Empid:760210
Name: Ashish Singh Baghel
eMail: Ashish.SBaghel@tcs.com

TCS wants you to analyze, recommend and perform a DevOps Practices implementation. Automate and streamline sample application with Devops Engineering Practices, including continuous integration, configuration management, continuous delivery, and automated quality assurance. The key objectives of your analysis and recommendations are:

– Create an Spring Boot Application and configure the inbuilt server port to 8885.
  Status: Complete
– Write Junit5 test cases for Spring Boot Application.
  Status: Complete
– Create a GitHub Repository and push the code to GitHub repository.
  Status: Complete
– Integrate Git plugin in jenkins pipeline to checkout the code.
  Status: Complete
– Implement Continuous Integration and Automation of Build, Test and Package.
  Status: Complete
– Implement a tool to analyse the quality of code (Sonar plugin)
  Status: Complete
– Implement Automation of Docker image creation.
  Status: Complete
– Implement authentication on docker registry.
  Status: Complete
– Implement Automation of pushing the docker image to docker registry.
  Status:Complete
– Implement email alerts for bad builds.
  Status: Complete
– Implement auto configuration of tomcat or aws ec2 instance.(using Ansible)
  Status: Complete
– Implement Automation of deployment on tomcat or aws ec2 instance.
  Status: Complete
– Implement Auto trigger of the jenkins cicd pipeline.
  Status: Complete

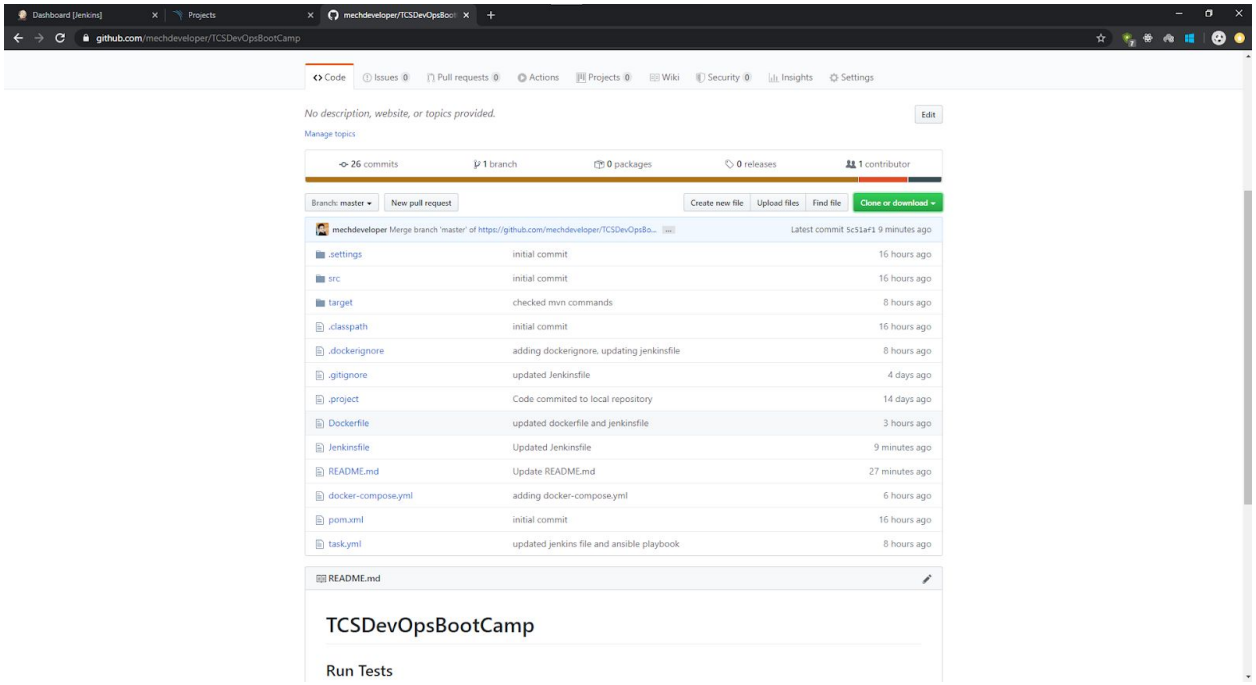
Note : The evaluation will be done on the provided Jenkin's environment. Once you are ready with the application on your local, please configure and test it according to the Jenkins Environment.

Date of Submission : 11th May 2020

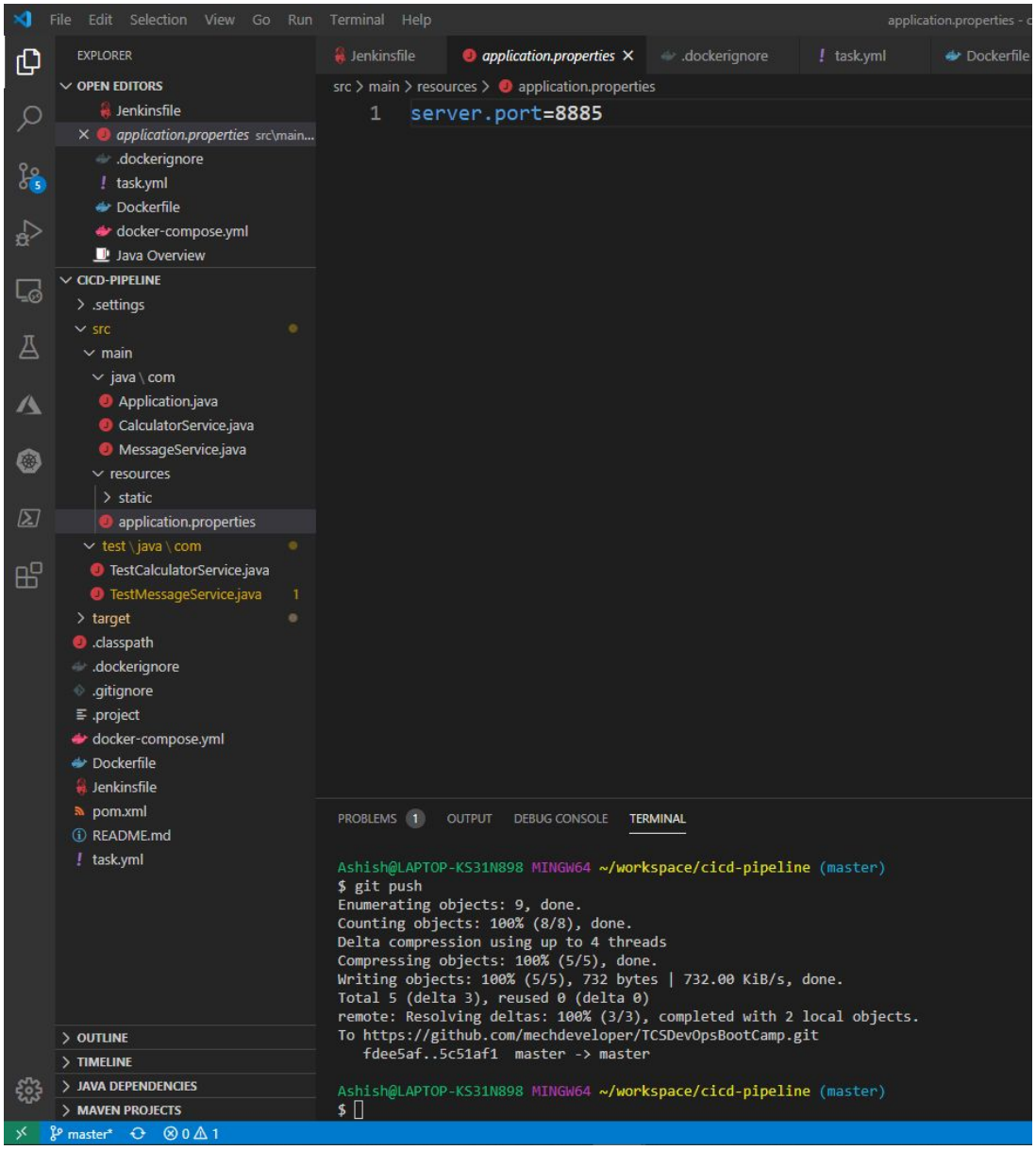
Setup Details and screenshots :

- ## Java Spring Boot Application -

  Github repo for Spring Boot Application -
  https://github.com/mechdeveloper/TCSDevOpsBootCamp.git

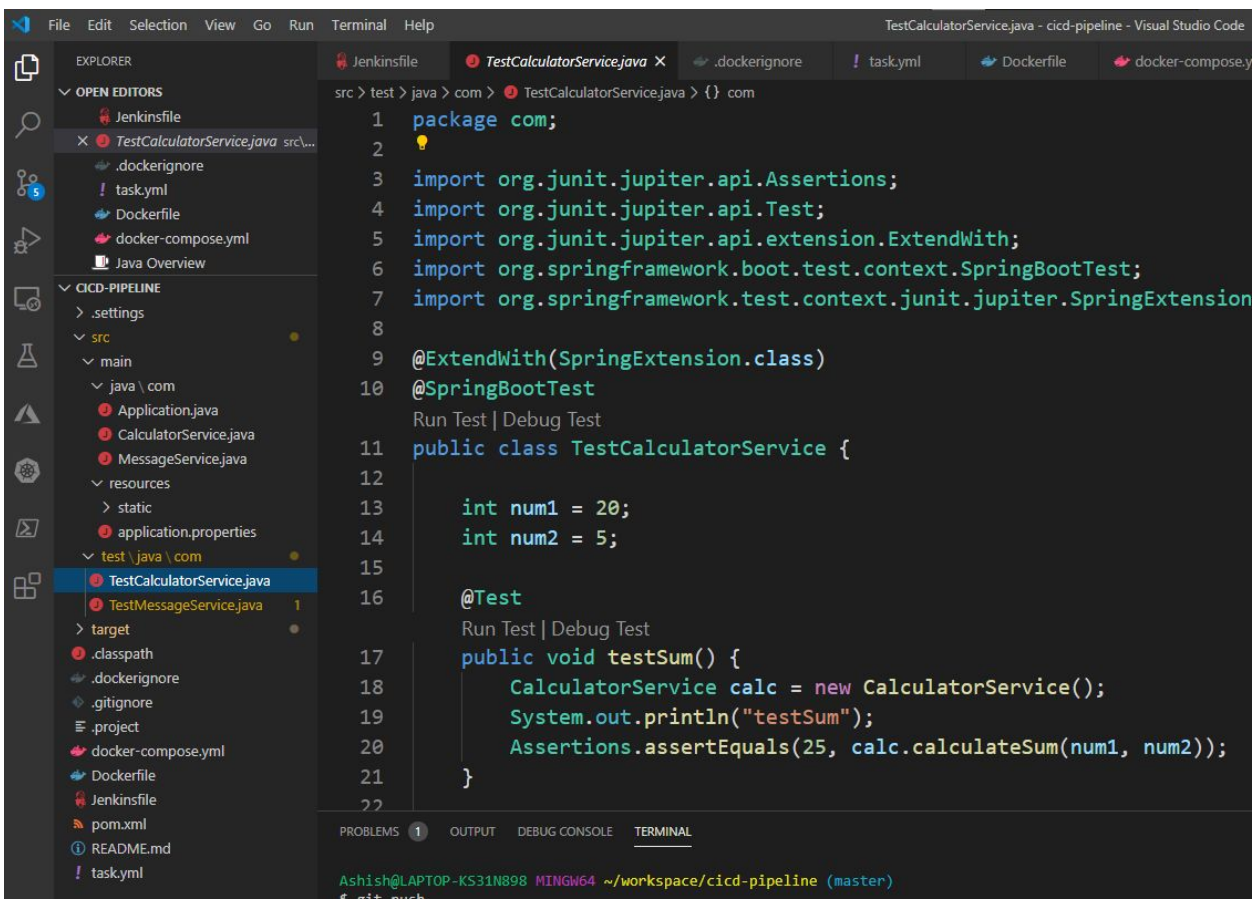


Inbuilt server port 8885 defined in application.properties file -



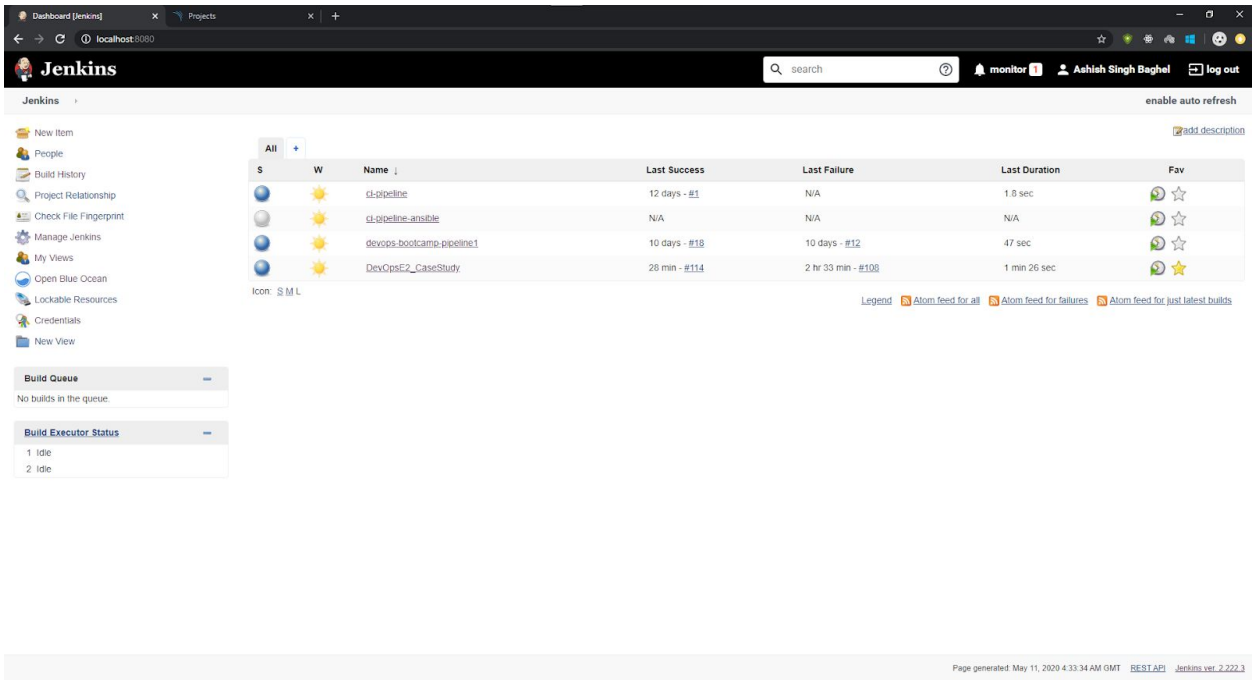
JUnit5 Testcases -

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.TestMessageService
testWelcome
testGreet
testMessage
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.02 s - in com.TestMessageService
[INFO] Running com.TestCalculatorService
testDiff
testMul
testMod
testInc
testDec
testIsGreater
testIsLower
testIsEqual
testSum
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in com.TestCalculatorService
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
```



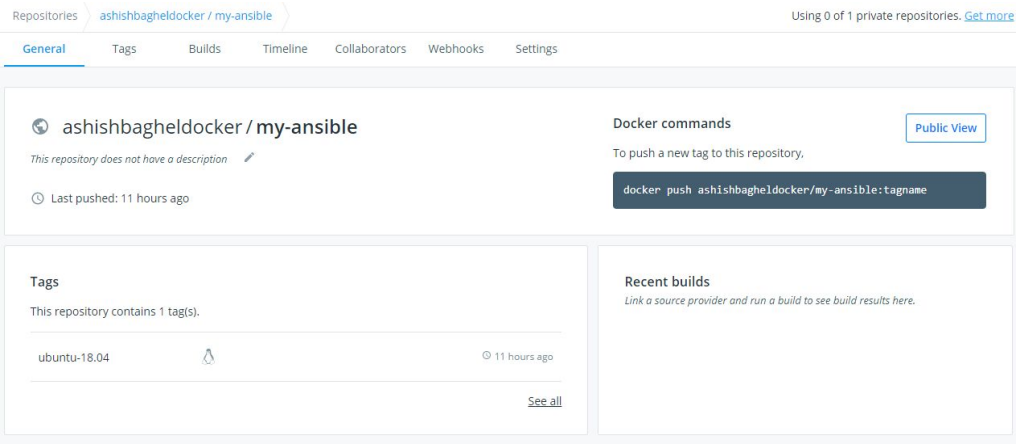
- ## Jenkins Setup -

Running jenkins on docker container.
Github repo with docker-compose.yml file to run jenkins as docker container -
https://github.com/mechdeveloper/jenkins-docker.git



SpringBoot Application Jenkinsfile explained -

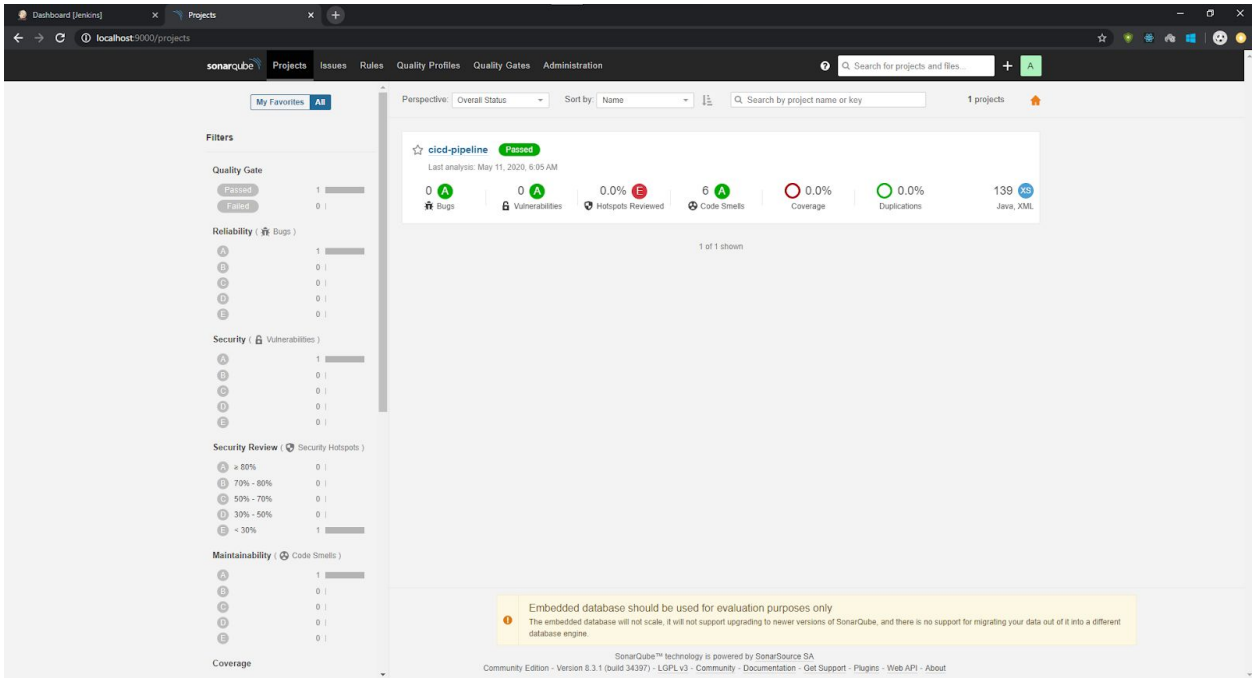
- Git checkout (source code checkout from github)
- mvn clean (cleanup)
- mvn package (build,test and package)
- SonarQube analysis
- Build Docker image (withCredentials, using dockerhub credentials to mask username and password for dockerhub)
- Push image to dockerhub
- Ansible create EC2 Instance (runs taks.yml file to create EC2 instance along with its security group to allow ssh and http port 80 access over internet)
  Note: Pulling my own ansible docker image from dockerhub and running the ansible commands to spin up resources in AWS cloud -



- EC2 instance Install docker (ssh into EC2 instance using ip and privatekey)
- EC2 instance Start docker (ssh into EC2 instance using ip and privatekey)
- EC2 instance initiate docker swarm mode for docker stack deployment (ssh into EC2 instance using ip and privatekey)
- EC2 instance copy docker-compose.yml for deployment of application on EC2 Instance
- EC2 instance Deploy application as a stack of service in EC2 Docker swarm (ssh into EC2 instance using ip and privatekey)
- Entire pipeline is enclosed in try catch block to catch any build step failure and report/send notification via email.

- Sonarqube server Setup -

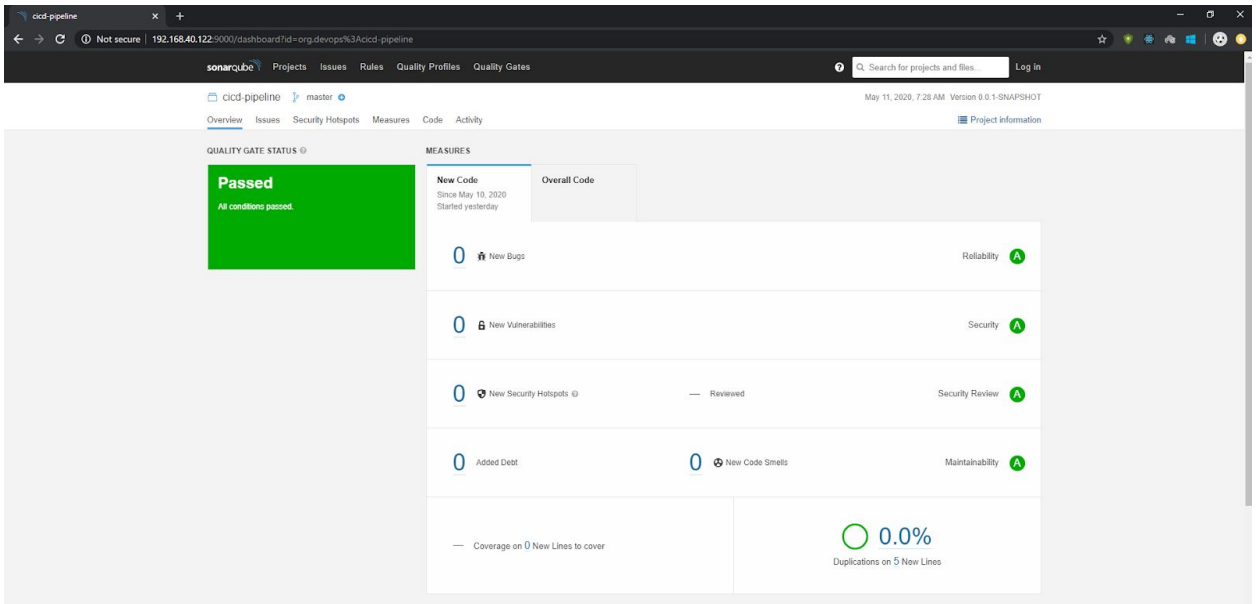
Running Sonarqube server on docker container -
Github repo with docker-compose.yml file to run sonarqube server as docker container -
https://github.com/mechdeveloper/sonarqube-docker.git



SonarQube build success -

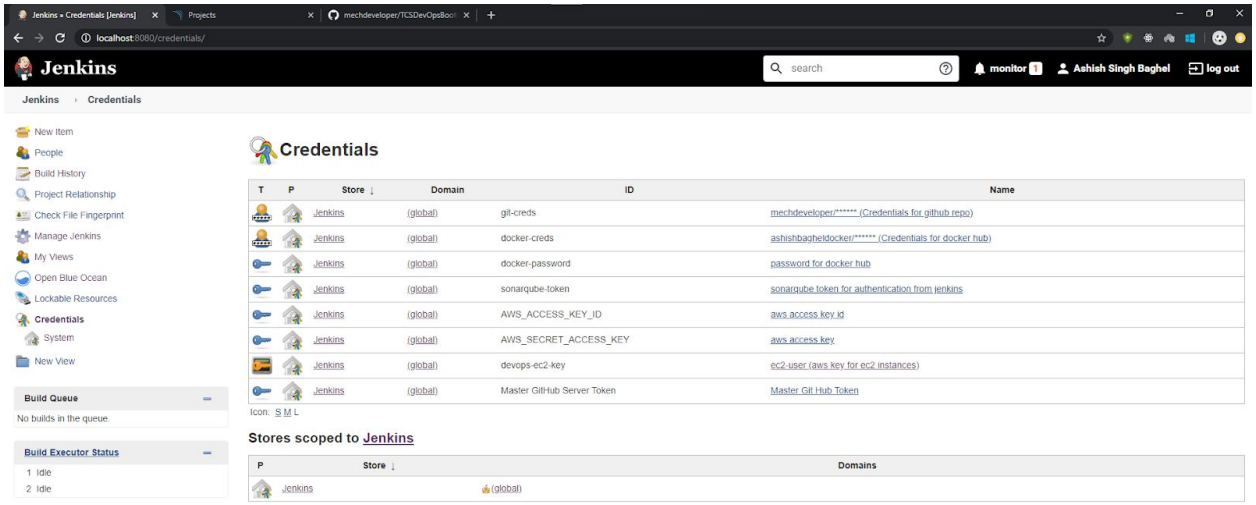
```
[INFO] ------------- Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=13ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=20ms
[INFO] CPD Executor 1 file had no CPD blocks
[INFO] CPD Executor Calculating CPD for 2 files
[INFO] CPD Executor CPD calculation finished (done) | time=10ms
[INFO] Analysis report generated in 102ms, dir size=97 KB
[INFO] Analysis report compressed in 43ms, zip size=21 KB
[INFO] Analysis report uploaded in 98ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://192.168.40.122:9000/dashboard?id=org.devops%3Acicd-pipeline
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://192.168.40.122:9000/api/ce/task?id=AXICNIlGYqcBmUEPWcqf
[INFO] Analysis total time: 8.721 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.860 s
[INFO] Finished at: 2020-05-11T05:28:17Z
[INFO] ------------------------------------------------------------------------
```

SonarQube Dashboard after code analysis -



- Jenkins configuration and Integration explained

Credentials configuration on Jenkins -

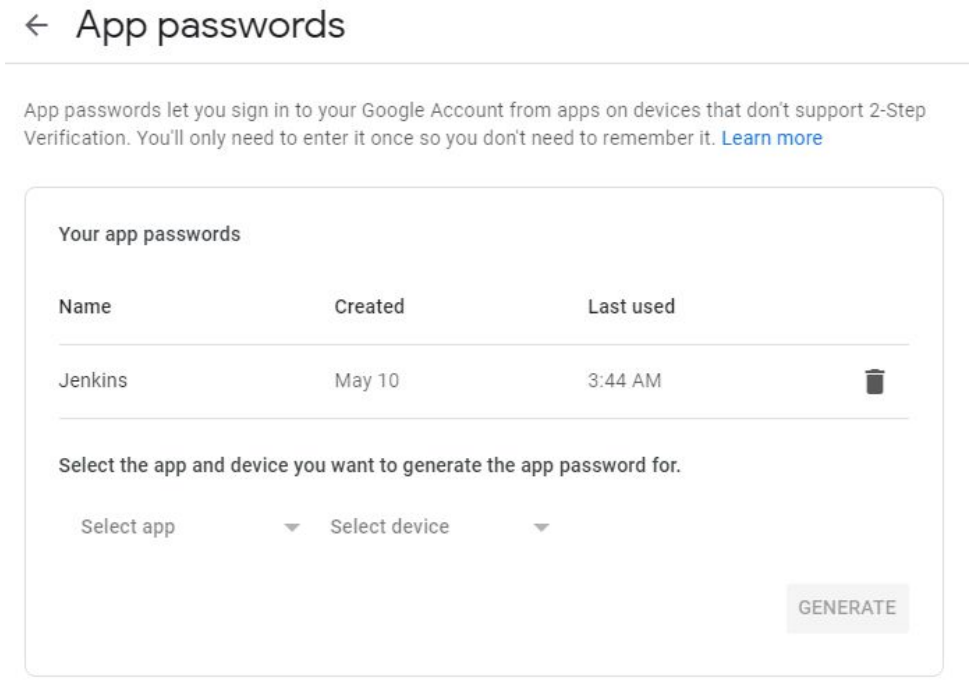


- git-creds : Github credentials
- docker-creds : Dockerhub credentials
- sonarqube-token : Token for sonarqube server authentication from jenkins server
- AWS_ACCESS_KEY_ID : AWS access key ID to allow AWS access for EC2 configuration
- AWS_SECRET_ACCESS_KEY: AWS secret access key to allow AWS access for EC2 configuration
- devops-ec2-key: private key configured in AWS to enable ssh into E2 Instances
- Master Github Token: Github integration for auto trigger of builds

- Email Notification Jenkins:

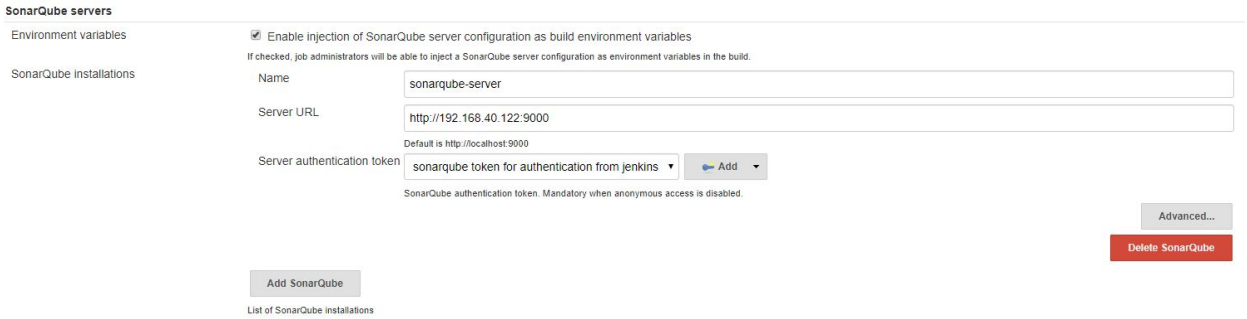
Enabled email Notification via gmail SMTP server -

Created app password in personal gmail account for integration with Jenkins

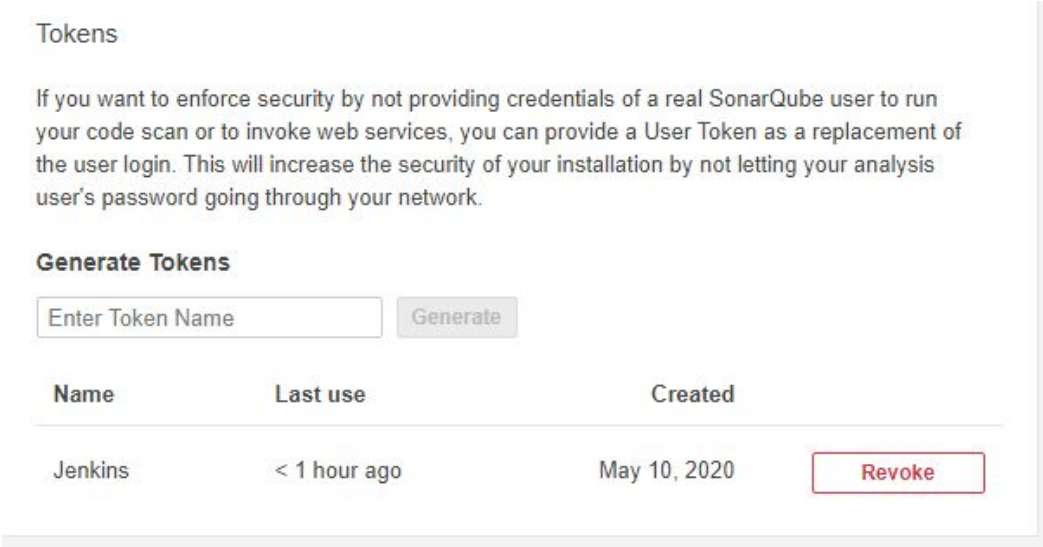


Failed Builds are notified via email -

- **SonarQube Server** configuration in Jenkins -



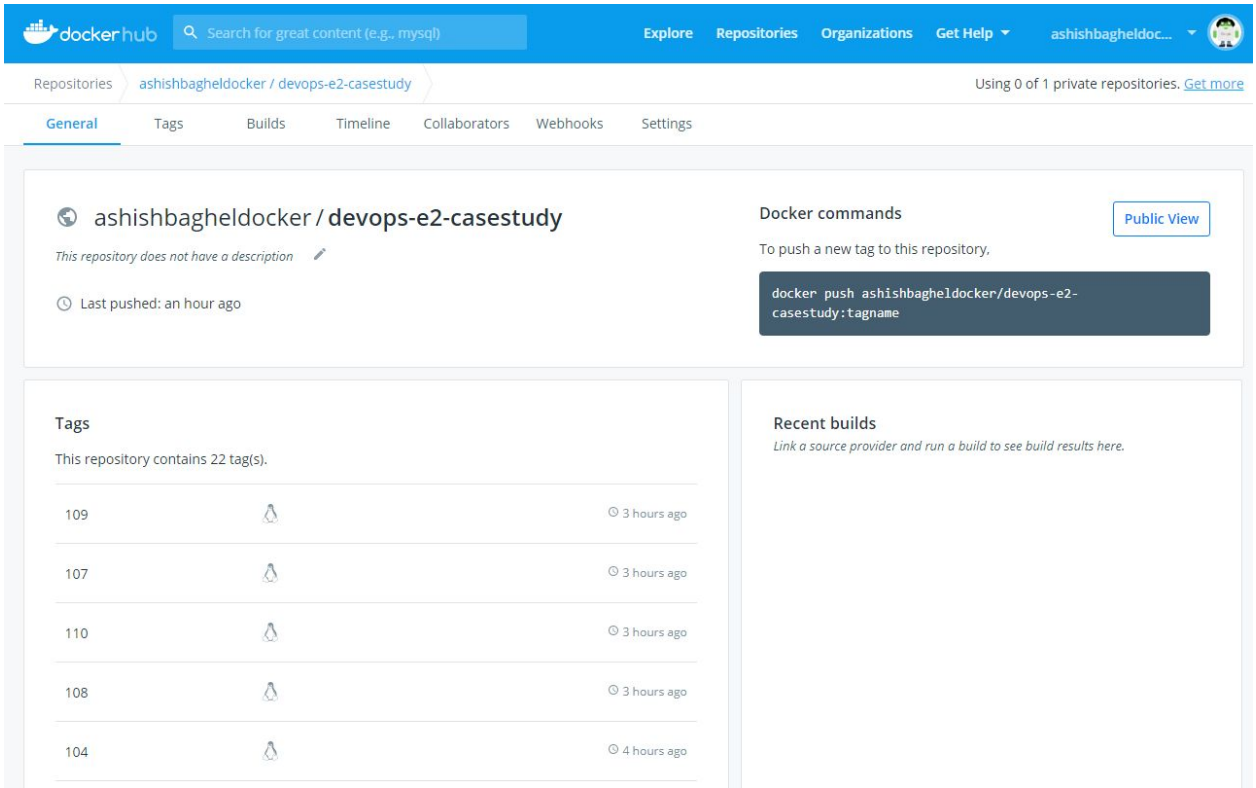
SonarQube Token generated for Jenkins authentication -



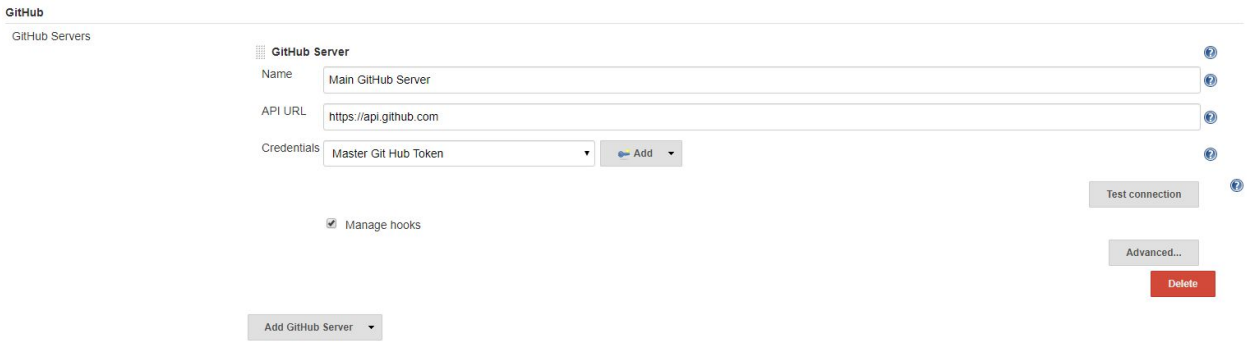
SonarQube server URL in jenkins is IPv4 address of local machine as sonarqube is running on a docker container similarly jenkins is also running on its own docker container both cannot talk to each other over network as they have their own isolated docker network. SonarQube Token is required for Jenkins to access the SonarQube server over the ip address of local machine

- ## Docker Hub Image Registry-

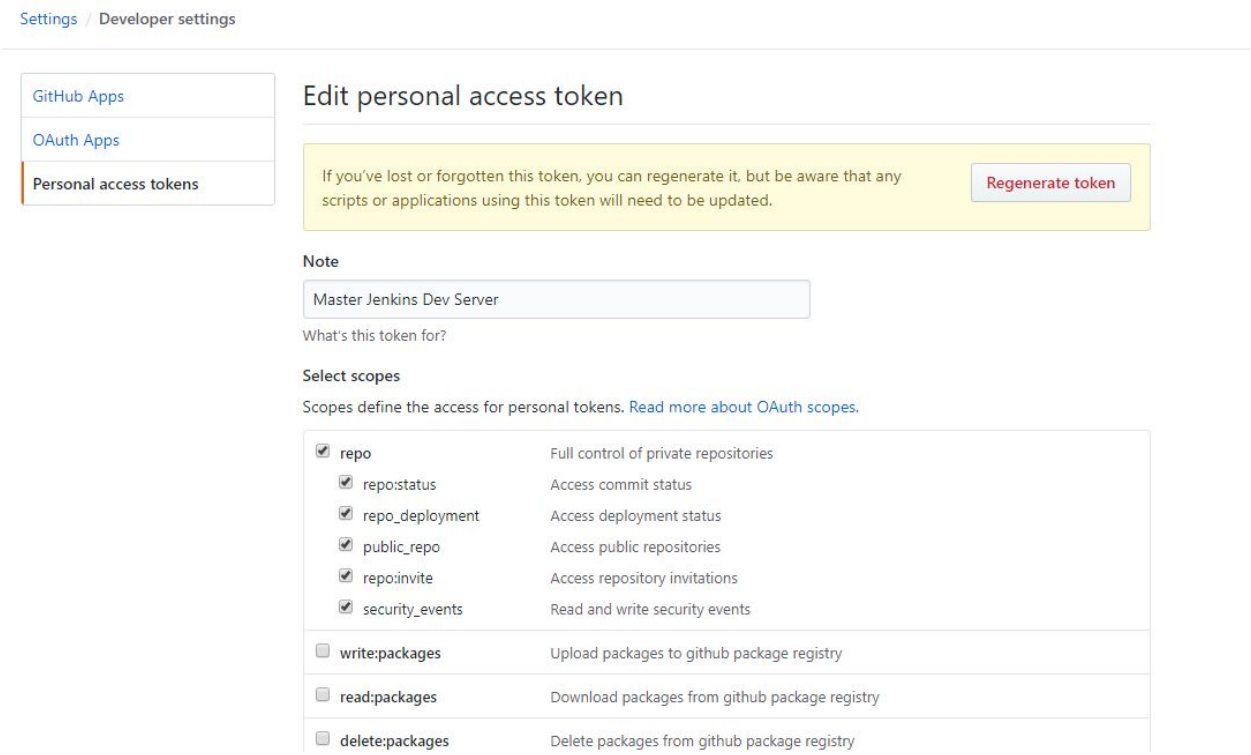
Created docker hub account to push docker images created from jenkins pipeline -



- Github Integration on Jenkins for auto trigger of builds



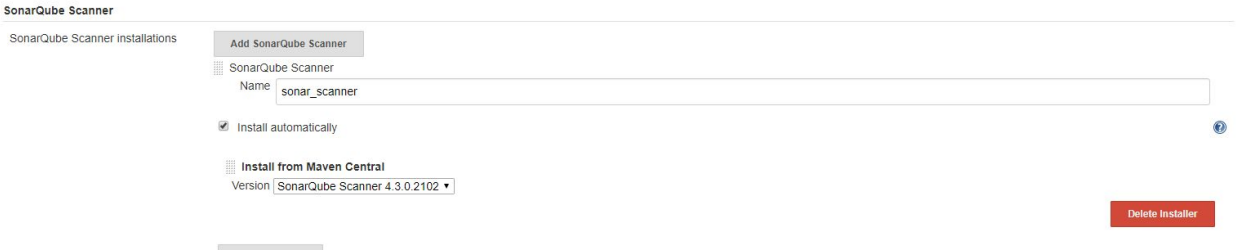
Created Personal Access token on Github for Jenkins integration -



- Jenkins Global Tool Configuration

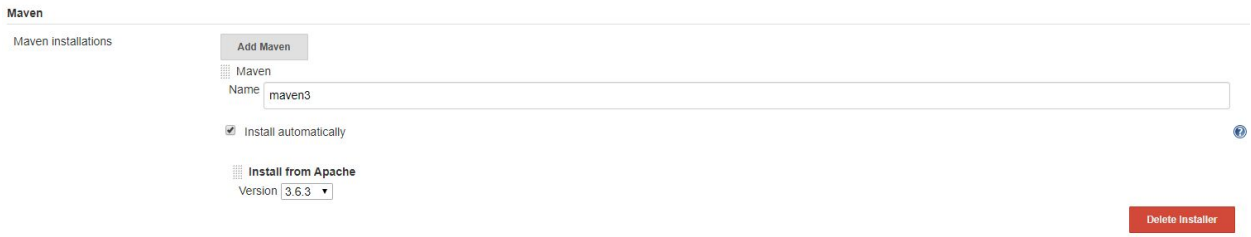
SonarQube Scanner installation -

Configured to install automatically

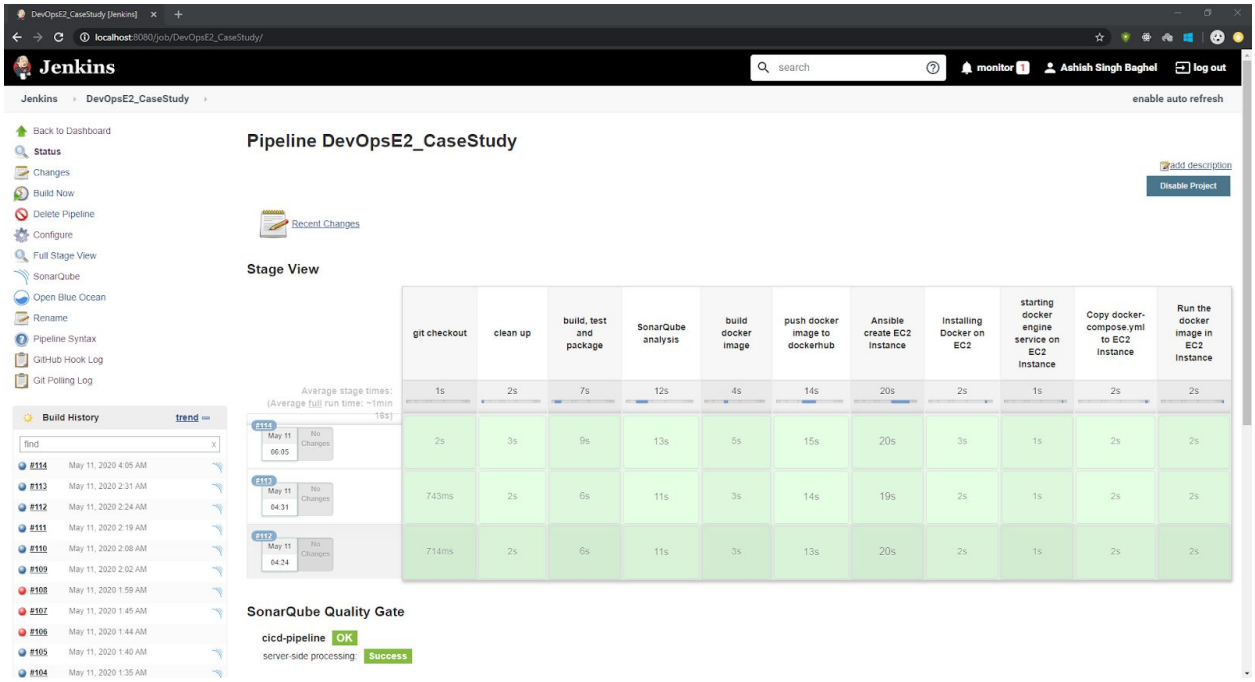


Maven Installation -

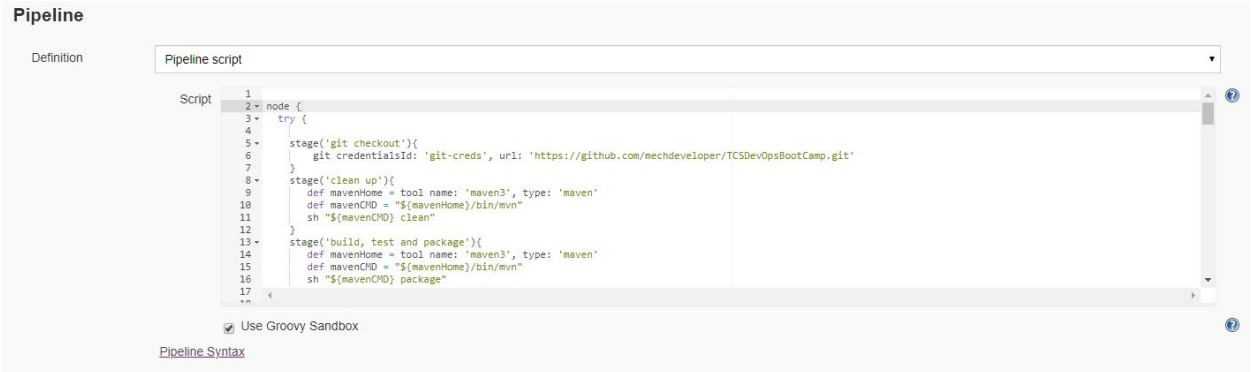
Configured to install automatically



- **Jenkins Build Pipeline** to trigger build and deployment of Spring Boot application



Entire pipeline is written as Groovy Script



Automatic Build triggers

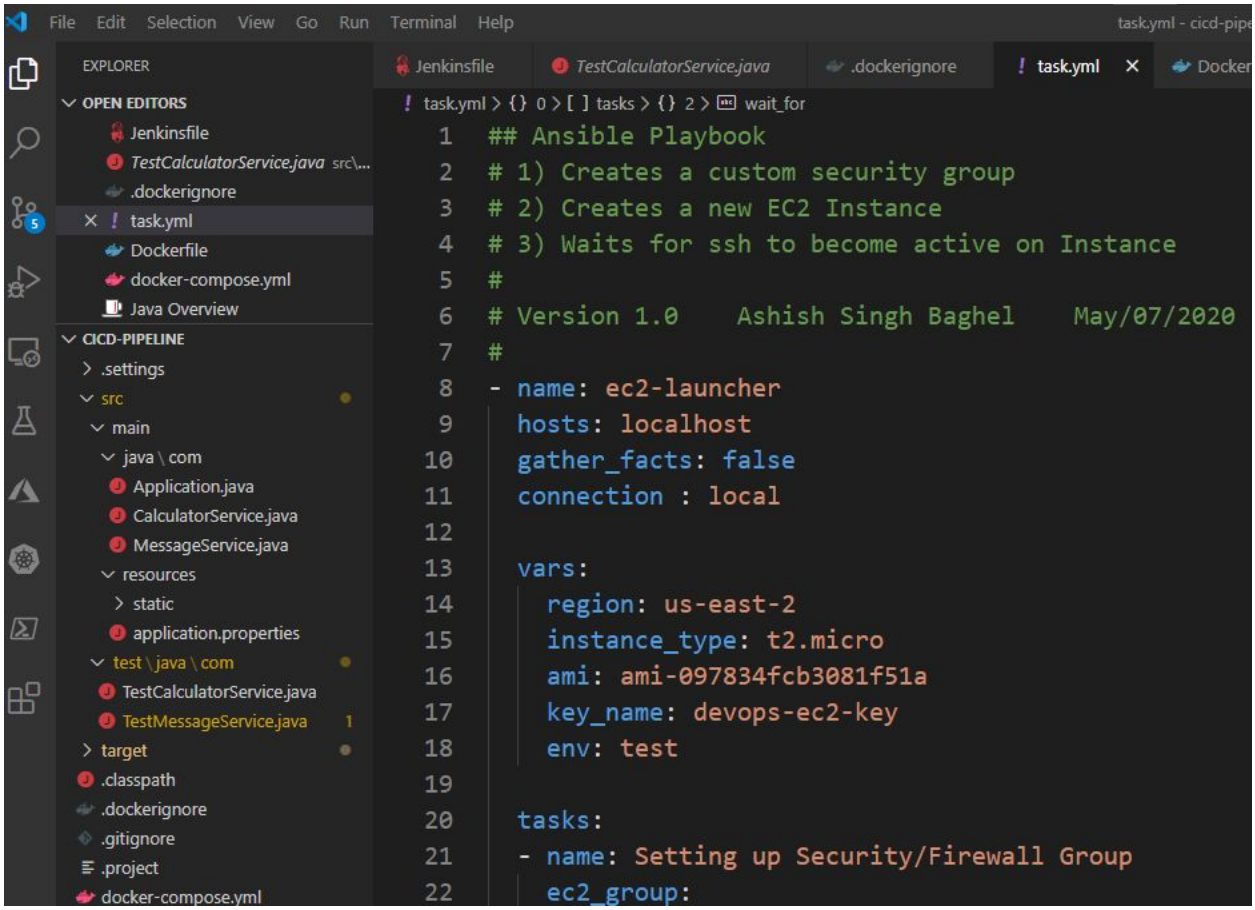
- Auto configuration of AWS EC2 Instances via Ansible -

Running ansible command on a docker container with ansible installed -

```
// Run Ansible playbook here
stage ('Ansible create EC2 Instance'){
    withCredentials([string(credentialsId: 'AWS_ACCESS_KEY_ID', variable: 'AWS_ACCESS_KEY_ID'), string(credentialsId: 'AWS_SECR
        def ansibleCMD = "ansible-playbook task.yml"
        sh "docker run --rm -v \$(pwd):/ansible/playbooks --env AWS_ACCESS_KEY_ID=\${AWS_ACCESS_KEY_ID} --env AWS_SECRET_ACCESS
    }
}
```

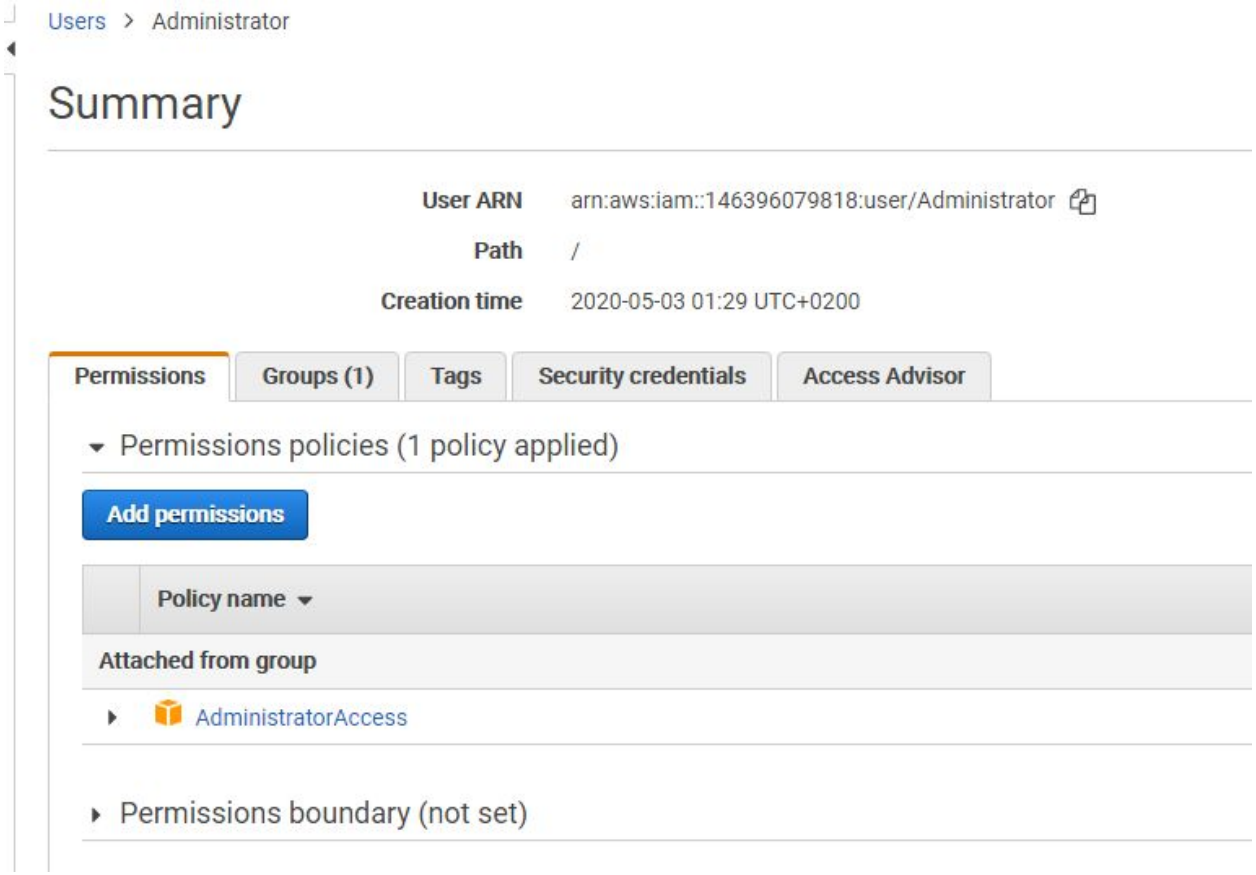
This Build step runs task.yml playbook to configure EC2 Instance -
task.yml explained -

- First task sets up a new security group to allow access over port 80 and 22
- Second task launches the AWS EC2 instance with key devops-ec2-key to allow ssh access to server via this key.
- Third task waits untill ssh starts working for the AWS EC2 instance just created via ansible playbook.

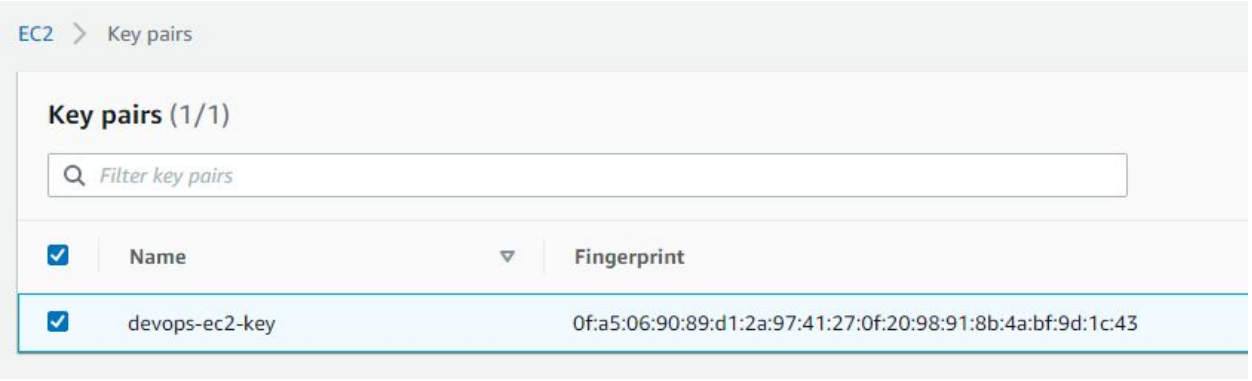


- AWS Configuration -

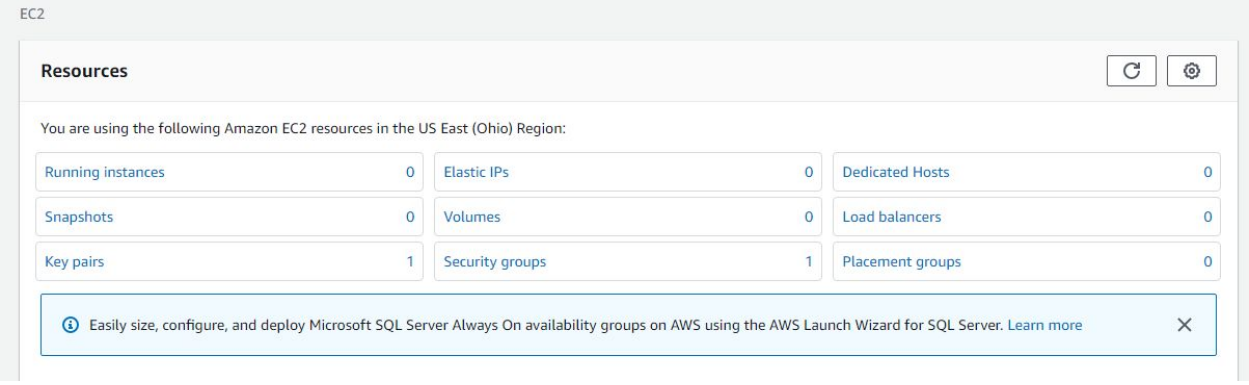
Created non root Administrator user with admin access to AWS for Jenkins Integration via Access Key ID and and Secret Access Key



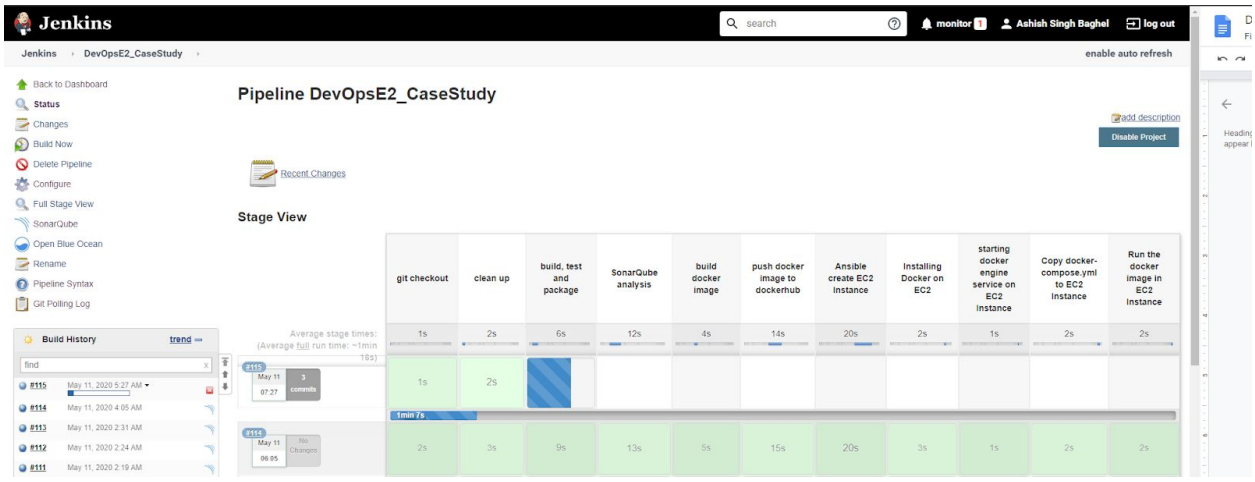
Created private key pair for AWS Region us-east-2 to enable access to EC2 machines deployed in that region



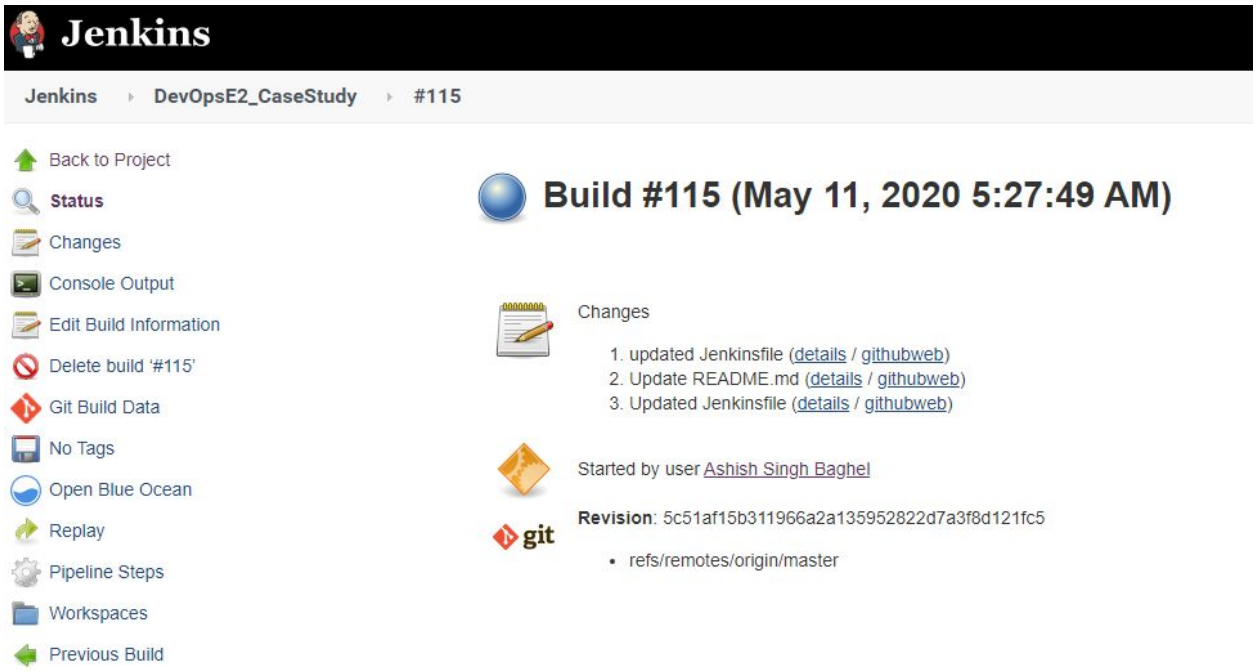
AWS EC2 Dashboard before deployment region us-east-2



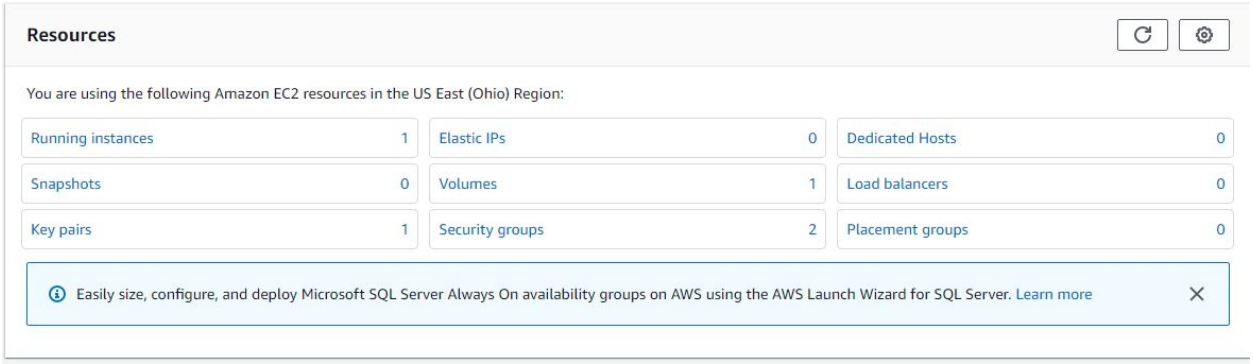
Triggering the build from jenkins -



Jenkins Build Successful -

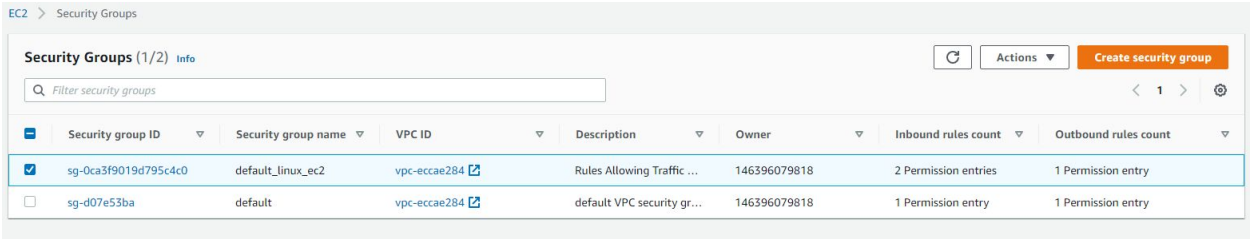


Checking AWS EC2 Dashboard after successful build from Jenkins pipeline build -

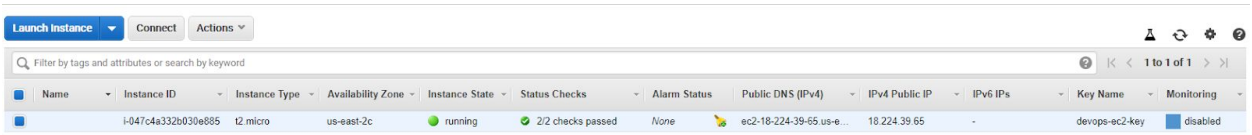


1 New running instance along with 1 New Volume and a brand new Security Group for our Instance -

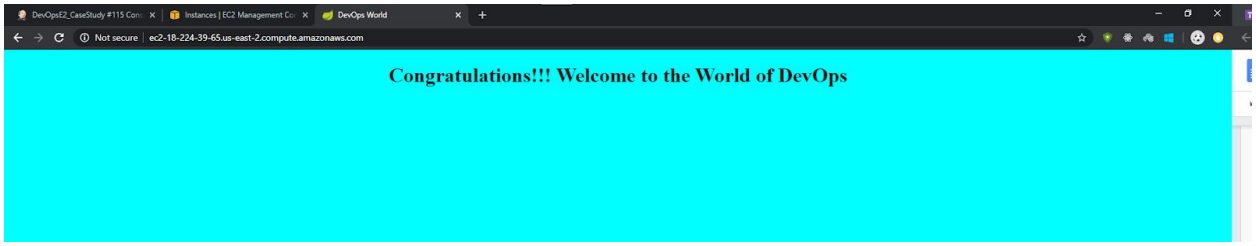
Checking the Newly created security group for our ec2 instance -



Checking new Running EC2 Instance -



- Accessing public DNS name of the newly created AWS EC2 instance



Our Application is successfully deployed via end to end automated jenkins pipeline.