

Chanugomula Shiva Kumar

Milestone Assessment-2

Project Documentation

Document Version / Details: Ver. 2.1/ 02-Jan-2023

Problem Statement

create an end-to-end CI/CD pipeline in aws platform using Jenkins as the orchestration tool, GitHub as scm, maven as the build tool, deploy in a docker instance and create a docker image, store the docker image in ECR, achieve Kubernetes deployment using ECR image. Build sample java web app using maven.

Solution:

Requirements:

* CI/CD pipeline System
* Git - local version control system.
* GitHub - As Distributed version control system.
* Jenkins - Continuous Integration tool.
* Maven - As a Build Tool.
* Ansible - Configuration Management & Deployment tool.
* docker -Containerization
* Kubernetes - As Container Management Tool.

Step-1:

Setup CI/CD with GitHub, Jenkins, Maven & Tomcat.

Setup Jenkins

Setup & Configure Maven , Git.

Setup Tomcat Server.

Integrating GitHub, Maven, Tomcat Server with Jenkins

Create a CI and CD Job.

Test the Deployment

Step-2:

Setup CI/CD with GitHub, Jenkins, Maven & Docker.

Setting up the docker Environment.

Write DockerFile.

Create an Image and Container on Docker Host.

Integrate Docker Host with Jenkins.

Create CI/CD Job on Jenkins to build and deploy on container.

Step-3:

Build and Deploy on Container.

CI/CD with GitHub, Jenkins, Maven, Ansible & Kubernetes.

Setting up the Kubernetes (EKS).

Write pod service and deployment manifest file.

Integrate Kubernetes with Ansible.

Ansible playbook to create deployment & service.

CI/CD Job to build code on Ansible & Deploy it on Kubernetes.

Step-4:

Deploy artifacts on the kubernetes

Launch an instance called developer-server and Jenkins-server.

Connect to the servers using ssh client.

Set up git, tomcat on the developer-server and Jenkins on the Jenkins-server using some installation commands.

A screenshot of a computer

Description automatically generated

Fig.1 developer-server instance

A screenshot of a computer

Description automatically generated

Fig.2 developer-server instance ssh connection

A screenshot of a computer program

Description automatically generated

Fig.3 Jenkins-server ssh connection

After installing Git on the developer-server, generate the SSH key using the ssh-keygen command.

Paste the generated key on GitHub for the connection.

A screenshot of a computer screen

Description automatically generated

Fig.4 SSH key generation

A screenshot of a computer

Description automatically generated

Fig.5 Authentication between Jenkins and git using ssh key

Create a directory using mkdir and initialize Git in that directory.

Write Java applications in the directory and push them into the Git repository using Git commands.

These changes will be reflected in the remote Git repository.

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Fig.6 git operations

Install Jenkins on the Jenkins-server using some commands.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

After installing Jenkins, paste the public IP of Jenkins with port number 8080 to access the Jenkins dashboard.

Copy the path and run the cat command on the Jenkins server to retrieve the password for login.

After entering the password, you will gain access to the Jenkins dashboard.

I have added a webhook to GitHub using the Jenkins URL and token generated in the Jenkins dashboard.

By adding the webhook, Git is successfully integrated with Jenkins.

A screenshot of a computer

Description automatically generated

Fig. 7 Jenkins dashboard

A screenshot of a computer

Description automatically generated

Fig. 8 connection between git and Jenkins using webhook.

Install Maven Plugin in Jenkins:

* Open your Jenkins dashboard.
* Navigate to Manage Jenkins from the left-hand side menu.
* Click on Manage Plugins under the System Configuration section.
* In the Plugin Manager, switch to the Available tab and search for the Maven plugin.
* Install the Maven plugin.

Configure Maven in Jenkins

Build the Java Application

A screenshot of a computer

Description automatically generated

Fig.9 Maven configuration

A screenshot of a computer

Description automatically generated

Fig.10 build success of java project.

Install Apache tomcat on developer-server and integrate it with Jenkins using commands and can access the tomcat server using port 8080 after adding roles and users to the tomcat-users.xml file.

Install Deploy to container plugin from Jenkins available plugin.

Integrate tomcat with Jenkins by adding global credentials in the Jenkins credentials section.

A screenshot of a computer

Description automatically generated

A black screen with red and blue text

Description automatically generated

Fig.11 tomcat-users.xml file

Integrate tomcat with Jenkins by adding global credentials in the Jenkins credentials section.

A screenshot of a computer

Description automatically generated

After adding global credentials, create a new Maven project. Choose Git as the source code management and add the Git repository link. Set the build trigger to ‘poll SCM,’ which triggers after any code changes. In the build section, specify the pom.xml file and use ‘clean install’ as the goals and options. For post-build options, enter \*\*/\*.war in the war file field (this considers any war file present on the system). Next, add a container and select Tomcat 9.x from the dropdown. In the credentials section, choose the Tomcat credentials and provide the Tomcat URL. Save the configuration. Your Java web application will run on the Tomcat server under the /webapp context.

A screenshot of a computer

Description automatically generated

Fig.12 Tomcat Server

A screenshot of a computer

Description automatically generated

Fig.13 Index.jsp

Launch an instance of Ansible-server and connect to it using an SSH client. Install Ansible-core. Navigate to /opt and create a directory called ‘docker,’ then install Docker in that directory.

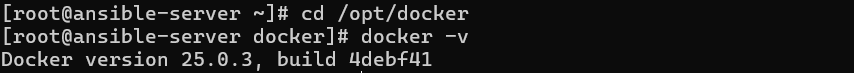
A screenshot of a computer

Description automatically generated

Fig.14 Ansible Server

A screen shot of a computer

Description automatically generated



Integrate Ansible with Jenkins. Install the ‘Publish Over SSH’ plugin in Jenkins. Enable password-based authentication by making changes to the sshd\_config file located in /etc/ssh/sshd\_config. Provide the credentials in the SSH server on Jenkins for test configuration purposes to integrate Ansible with Jenkins.

A screenshot of a computer

Description automatically generated

Fig.15 Publish Over SSH plugin.

A screenshot of a computer

Description automatically generated

Fig.16 SSH Server

Create a Docker file in which the Tomcat image should be pulled. Next, create a playbook containing the code to build the image and finally, push the image to Docker Hub.

A black background with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

Fig.17 docker file

Jenkins job to build and copy the artifacts on to the ansible server.

Configure ssh server.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fig.18 Configuring artifact onto Ansible.

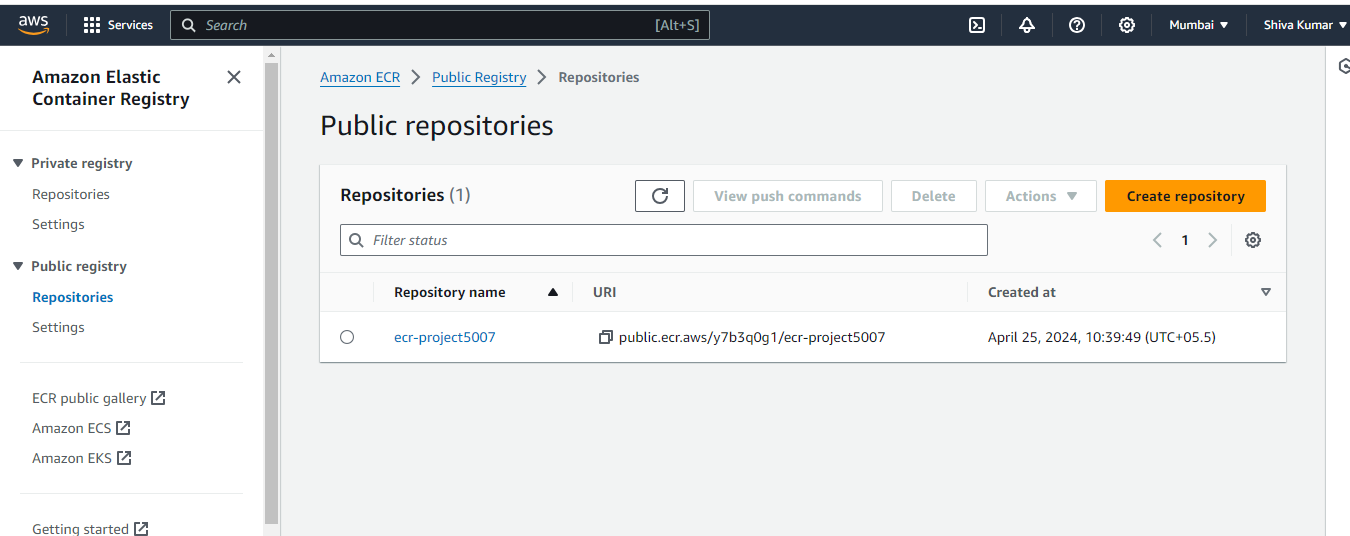
Generate ssh key.

A computer screen shot of a black screen

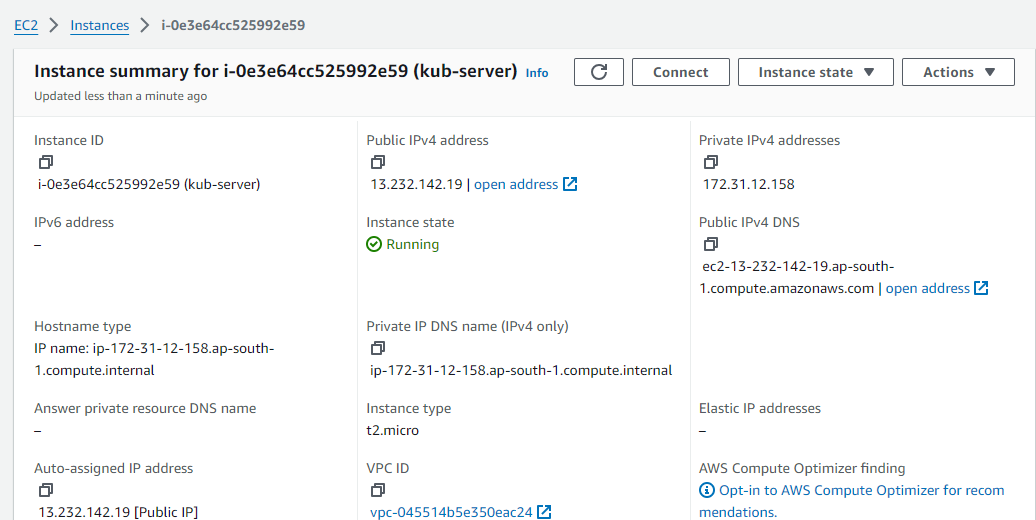
Description automatically generated

Fig.19 ssh-keygen

Create a ECR public repository in ansible server.



Launch an instance called kub-server for the Kubernetes cluster and connect to it using an SSH client. Install Kubernetes with deployment tools, the latest version of AWS CLI, and set up kubectl and eksctl.



A black screen with white text

Description automatically generated

Fig.19 kub-server instance

Create an IAM role.

Assign this role to the kub-server instance.

Add permissions to the role.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fig.21 IAM Role and permissions

Create the cluster with the command eksctl create cluster and specify the region and node type.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fig.22 Clusters

Now we can observe that the cluster is created. We need to create a manifest file as shown below. One is for the deployment, and the other is for the service. In the service, we use a load balancer as it is the best service when compared to other services.

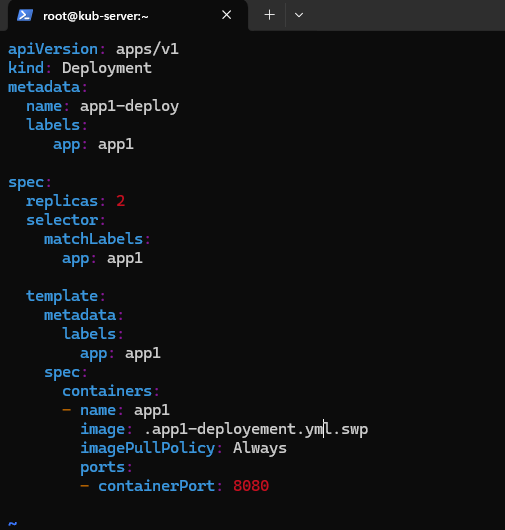


Fig.23 deployement.yml

A screen shot of a computer

Description automatically generated

Fig.24 Service.yml

Now run kubectl apply for the service; then the service will be created. Similarly, for the deployment, we can check the output, including pods, deployment, and service, by using kubectl get all.

After that, we need to integrate the bootstrap server with Ansible. To do this, we’ll add the user to the sudoers list and copy the SSH key to the authorized keys file. This integration allows us to work with eksctl from Ansible. We’ll create an Ansible playbook for deploying eksctl

A screenshot of a computer

Description automatically generated

Fig.25 app1.yml

Now run the ansible book as shown below.

A screenshot of a computer screen

Description automatically generated

Fig.26 ansible-playbook execution

Create a Jenkins deployment job for Kubernetes. In Jenkins, navigate to the Continuous Integration (CI) section and provide the Git repository link. Specify the branch specifier and set the trigger to ‘poll SCM’ so that it runs whenever a commit is made to the repository. In the goals and options, use ‘clean install’ for execution. For the post step, choose ‘run regardless of build result.’ Additionally, configure it to trigger other projects (CD) only if the build is stable. Finally, add a post-build action to send build artifacts over SSH. Specify the server’s name and EXEC command, then apply and save the configuration.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fig.27 app1-ci-job

A screenshot of a computer program

Description automatically generated

In the same way, go to the CD section, select the post-build action ‘Send build artifacts over SSH.’ Provide the SSH server name and EXEC command, then apply and save.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Fig.28 app1-cd-job

Now, after setting up the CI/CD pipeline, go to the source code and make some changes. Next, add, commit, and push to Git. When we check in Jenkins, the created projects are built automatically.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

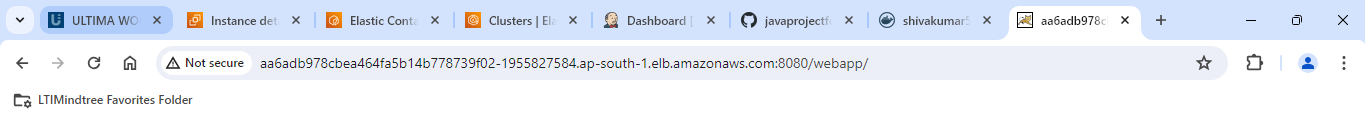
Fig.29 build success of all files

After that, go to the EKSCTL bootstrap server and run the command kubectl get all. This will display all the pods, services, deployments, and replicas. Now, in the service table, copy the load balancer’s external IP and paste it into a new window. You’ll observe that the changes are automatically reflected in the application.

A black screen with white text

Description automatically generated

Copy the external-Ip of service/app1-service and go to port 8080/webapp then you can access the web application.

A screenshot of a computer

Description automatically generated

The above approach outlines the creation of an end-to-end CI/CD pipeline on the AWS platform using DevOps tools such as GitHub, Jenkins, Ansible, Docker, Kubernetes, and Maven for a web application.

