Amay Sharma's Blog

Follow

The Ultimate CI/CD Deployment using EC2, Jenkins Master-Slave, AWS CLI, GitHub WebHook, Docker, and ECR.

AS Amay Sharma

Dec 26, 2023 · 6 min read

Welcome to the **Spring Boot Deployment Project with GitHub, Docker, Jenkins, and AWS**

Overview:

In this project, we aim to deploy a basic Spring Boot application using modern DevOps tools and practices. The application's source code, Docker configuration, and build settings are managed in a GitHub repository. The deployment pipeline is orchestrated by Jenkins, leveraging Docker containers, Amazon ECR (Elastic Container Registry), and AW

4 1

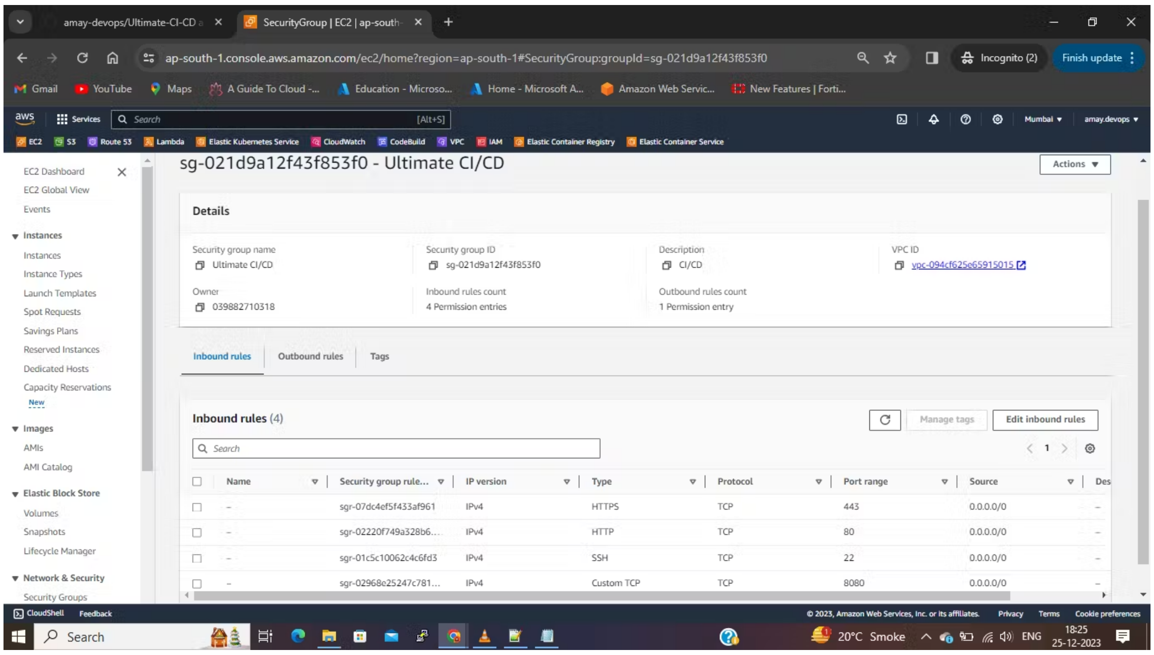
GitHub - amay-devops/Ultimate-CI-CD: This is the ultimate CI/CD project using Amazon Linux EC2, Jenkins, Maven, Docker, ECR.

This is the ultimate CI/CD project using Amazon Linux EC2, Jenkins, Maven, Docker, ECR. - GitHub - amay-devops/Ultimate-CI-CD: This is the ultimate CI...

github.com

First, let's begin with the AWS setup:-

1. Go to the EC2 area of the AWS console after logging in, then choose Security Group. Since this is a testing environment, create a security group with the inbound rules **"80,443,22, and 8080"** and the outbound rules allowing anything.



ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#SecurityGroup:groupId=sg-021d9a12f43f853f0

sg-021d9a12f43f853f0 - Ultimate CI/CD

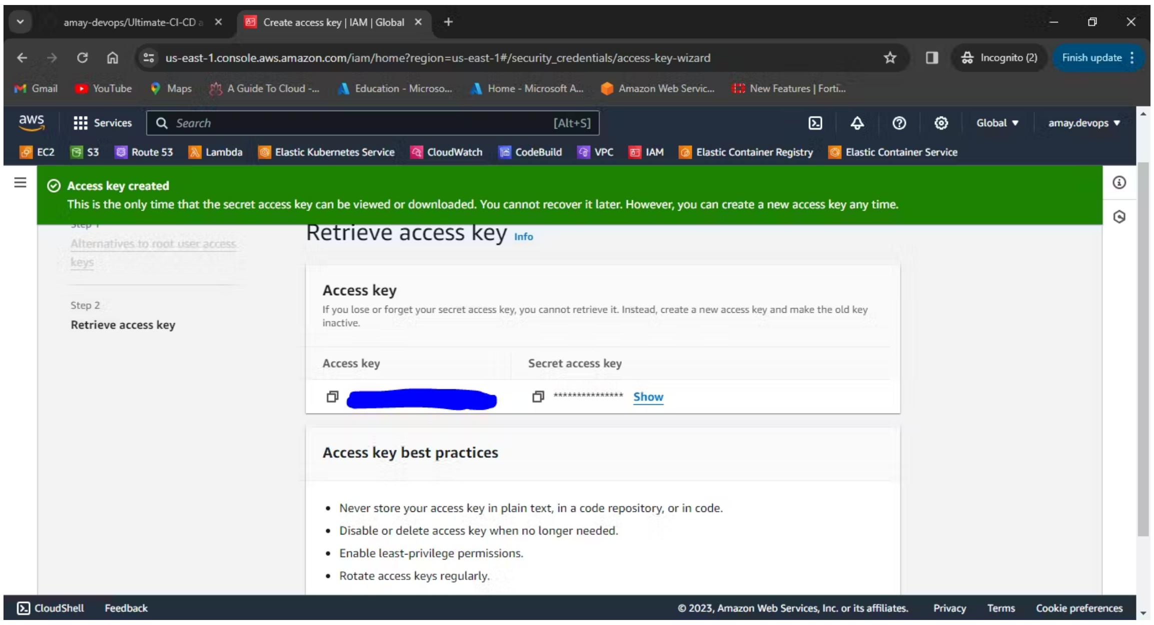
Details

Security group name	Security group ID	Description	VPC ID
Ultimate CI/CD	sg-021d9a12f43f853f0	CI/CD	vpc-094cf625e65915015
Owner	**Inbound rules count**	**Outbound rules count**	
039882710318	4 Permission entries	1 Permission entry	

Inbound rules (4)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Des
–	sgr-07dc4ef5f433af961	IPv4	HTTPS	TCP	443	0.0.0.0/0	–
–	sgr-02220f749a328b6...	IPv4	HTTP	TCP	80	0.0.0.0/0	–
–	sgr-01c5c10062c4c6fd3	IPv4	SSH	TCP	22	0.0.0.0/0	–
–	sgr-02968e25247c781...	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	–

2. Navigate to the AWS dashboard and select the security credentials under the profile area on the right-hand side. Create a new Access Key by going to Access Key. Copy the Security Key and Access Key; we'll need them for AWS CLI configuration.



us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/security_credentials/access-key-wizard

Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 2
Retrieve access key

Retrieve access key

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
[redacted]	*************** Show

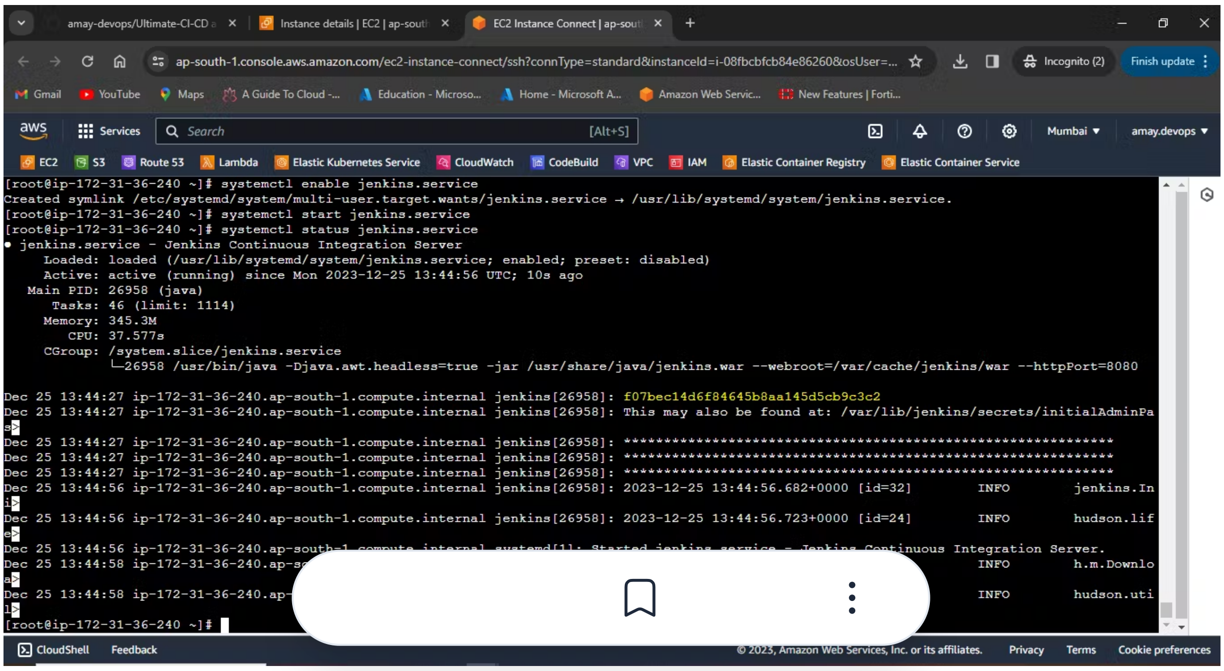
Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

3. Launch two instances (t2.large) with the new security group, Ultimate CI/CD, and choose storage for each instance that is at least 20GB in size. Because we need to install a lot of applications on those instances, including Docker, Apache-Maven, Java-11, and Jenkins, we c es have been created successfully, run the following command:-

```
yum update

wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

yum update -y

yum install git -y

yum install java-17-amazon-corretto

yum install jenkins -y

systemctl enable jenkins

systemctl start jenkins

systemctl status jenkins
```

Copy the one-time admin password for the jenkins server from the above command.

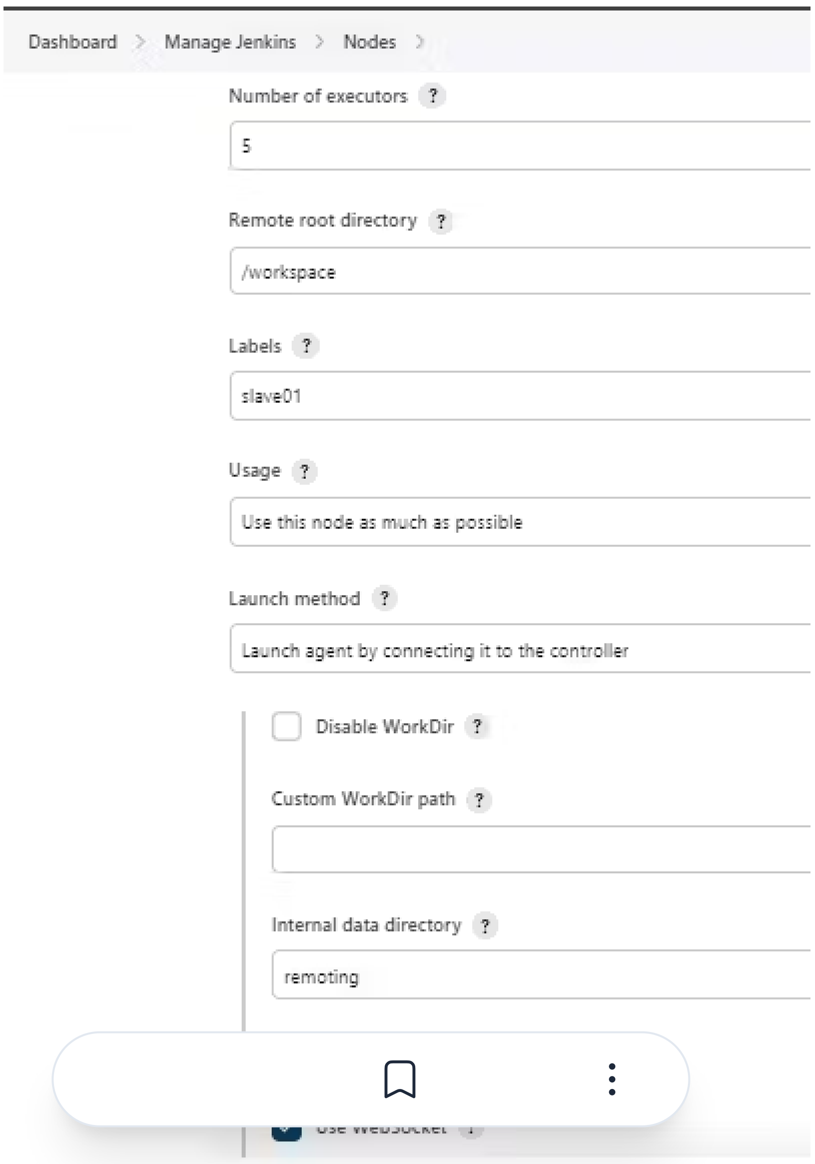


```
[root@ip-172-31-36-240 ~]# systemctl enable jenkins.service
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
[root@ip-172-31-36-240 ~]# systemctl start jenkins.service
[root@ip-172-31-36-240 ~]# systemctl status jenkins.service
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: disabled)
     Active: active (running) since Mon 2023-12-25 13:44:56 UTC; 10s ago
   Main PID: 26958 (java)
      Tasks: 46 (limit: 1114)
     Memory: 345.3M
        CPU: 37.577s
     CGroup: /system.slice/jenkins.service
             └─26958 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Dec 25 13:44:27 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: f07bec14d6f84645b8aa145d5cb9c3c2
Dec 25 13:44:27 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPa
s>
Dec 25 13:44:27 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: *************************************************************
Dec 25 13:44:27 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: *************************************************************
Dec 25 13:44:27 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: *************************************************************
Dec 25 13:44:56 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: 2023-12-25 13:44:56.682+0000 [id=32]       INFO       jenkins.In
i>
Dec 25 13:44:56 ip-172-31-36-240.ap-south-1.compute.internal jenkins[26958]: 2023-12-25 13:44:56.723+0000 [id=24]       INFO       hudson.lif
e>
Dec 25 13:44:56 ip-172-31-36-240.ap-south-1.compute.internal systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Dec 25 13:44:58 ip-172-31-36-240.ap-s                                                                          INFO       h.m.Downlo
a>
Dec 25 13:44:58 ip-172-31-36-240.ap-                                                                           INFO       hudson.uti
l>
[root@ip-172-31-36-240 ~]#
```

Let's begin with the Jenkins configuration.

1. After starting the server, go to http://<AWS_EC2_PUBLIC_IP>:8080 to log in. After pasting the admin password, install the suggested plugin and log in. After installing every suggested plugin, provide your username, password, email address, and other details. Go to **"Manage Jenkins → Nodes → Add Node → Enter the slave's name → Select Permanent Agent"** after logging into the dashboard.

2. Set the **number of executors to 5** in the following section (this refers to the maximum number of concurrent jobs that can be executed simultaneously). Additionally, configure the **remote root directory to "/workspace**" (this is the directory from which Jenkins will pull the code to create the artifact from GitHub).


Dashboard > Manage Jenkins > Nodes >

Number of executors: 5

Remote root directory: /workspace

Labels: slave01

Usage: Use this node as much as possible

Launch method: Launch agent by connecting it to the controller

Disable WorkDir

Custom WorkDir path:

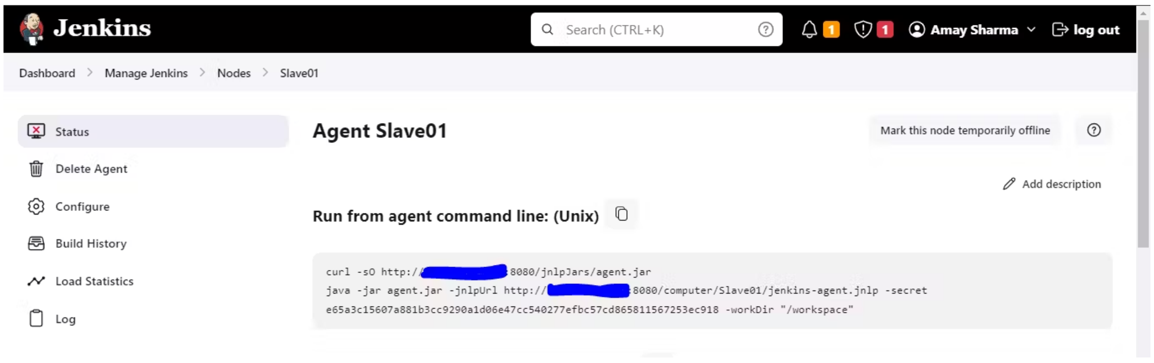
Internal data directory: remoting

Use WebSocket

3. Set the **agent's label** (Note: The agent's label serves as its identifier). When defining the node in which a task needs to be completed, we must provide the label name. Select "**use a web socket**" from the Launch method section.

4. Right now, our slave01 is not online in the node section. When you select the slave01, a few commands that must be executed in the slave instance are displayed.

```
curl -sO http://<Jenkins_Server_IP>:8080/jnlpJars/agent.jar

java -jar agent.jar -jnlpUrl \

http://<Jenkins_Server_IP>:8080/computer/Slave01/jenkins-agent.jnlp -
secret \
ac25f8b9a748ae8cd43f5e0a3b81d55a85605c0c284f5a954751eed884d3bafb -
workDir \ "/workspace"
```



Agent Slave01

Run from agent command line: (Unix)

```
curl -sO http://[redacted]8080/jnlpJars/agent.jar
java -jar agent.jar -jnlpUrl http://[redacted]8080/computer/Slave01/jenkins-agent.jnlp -secret
e65a3c15607a881b3cc9290a1d06e47cc540277efbc57cd865811567253ec918 -workDir "/workspace"
```

5. Visit **Manage Jenkins → Plugin → Available Plugin** once more. Look for **Blue Ocean, Maven Integration, Maven Info, and Maven Invoker**. After installation, select Return to Page.

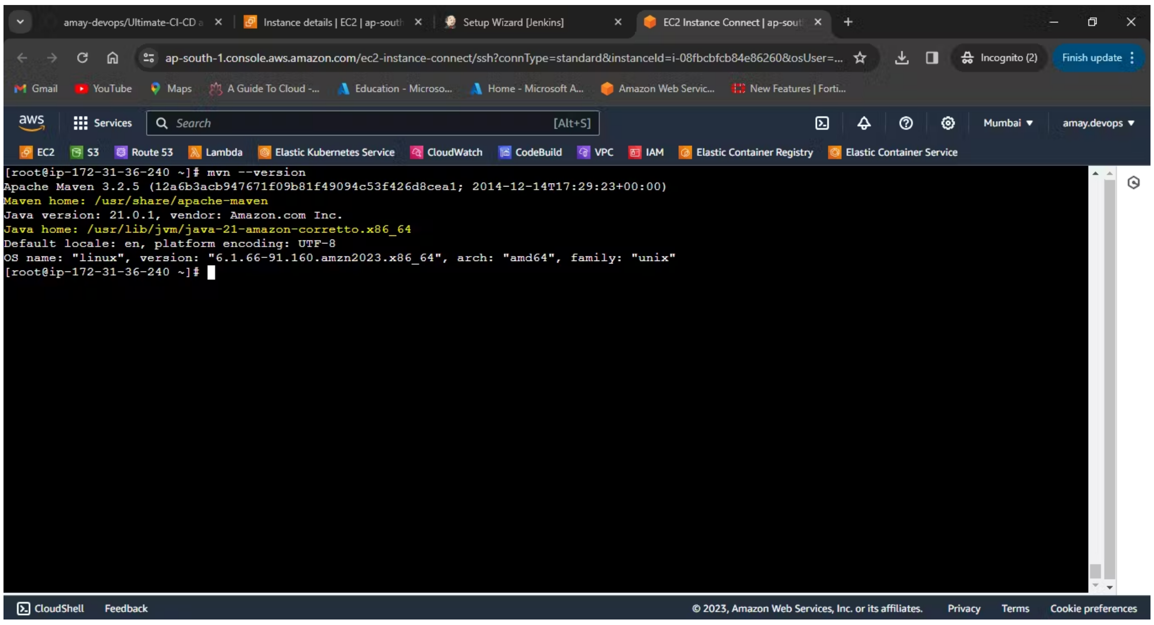
Install Apache-maven on the Master and Slave now, so that Jenkins will generate the artifact as soon as it receives the code from the GitHub webhook. Also, Install Docker on the Slave.

```
wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-
maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo

sed -i s/$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
```

```
yum install -y apache-maven

yum install docker -y(to be run only on slave)

systemctl enable docker(to be run only on slave)

systemctl status docker(to be run only on slave)

mvn --version
```

Adding Build Home Location

1. Navigate to **Manage Jenkins → Tools → Maven Integration, at the bottom. To add Maven, select Add → Uncheck Install Automatically → Type MAVEN_HOME** → Copy the Maven home location on Apache, which can be found by using the **mvn --version** command.
2. There is a space to add JDK at the beginning of the page as well. Select **Add JDK, type JAVA_HOME** as the name, and then copy the Java home address found in the **mvn --version** command output.



```
[root@ip-172-31-36-240 ~]# mvn --version
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-14T17:29:23+00:00)
Maven home: /usr/share/apache-maven
Java version: 21.0.1, vendor: Amazon.com Inc.
Java home: /usr/lib/jvm/java-21-amazon-corretto.x86_64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.1.66-91.160.amzn2023.x86_64", arch: "amd64", family: "unix"
[root@ip-172-31-36-240 ~]#
```

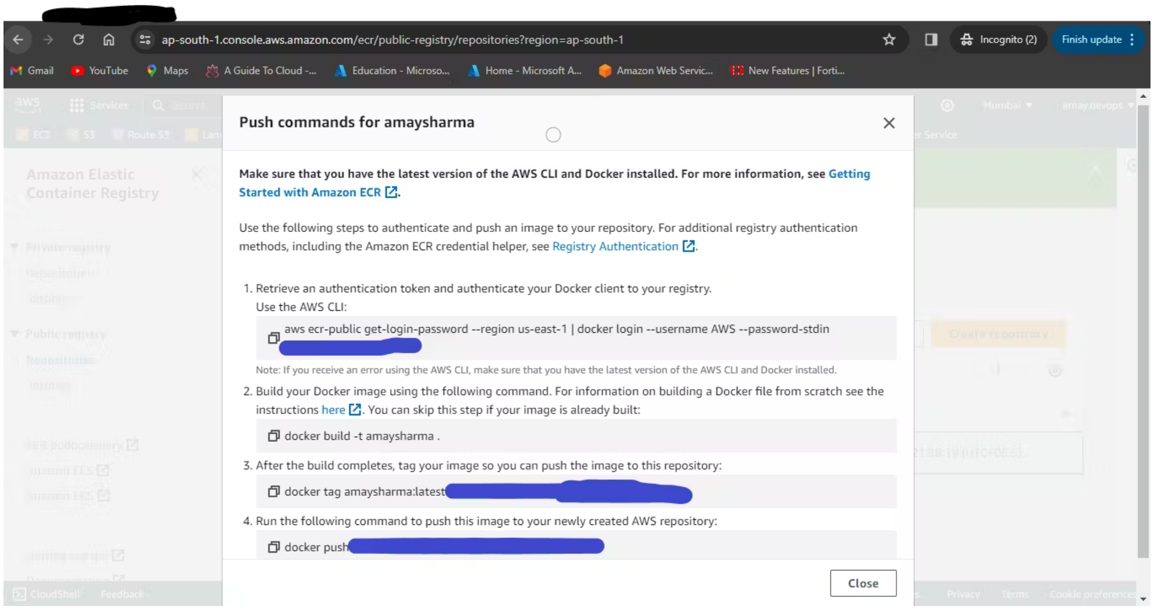
Simply provide the path to the git execution in the git section. This can be done by running **"which git"** on the slave or master ec2. When everything is ready, simply save and go.

AWS ECR Setup

1. Access the AWS console by logging in. (console.aws.amazon.com) → Navigate to ECR, select Create Repository, provide the repository's name, select Public, and then select Create Repository.

2. Choose the just created repository under the public repository area, then click View Push Command.

[redacted]



Push commands for amaysharma

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry.
   Use the AWS CLI:

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin [redacted]
```

   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

```
docker build -t amaysharma .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag amaysharma:latest[redacted]
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push[redacted]
```

3. Set up the AWS CLI on the slave instance so that the aforementioned command runs on slave01 and pushes the docker image from the AWS CLI.

4. Use the following command if the Amazon CLI is not configured:-

```
aws configure
```

The command above will prompt for some input.

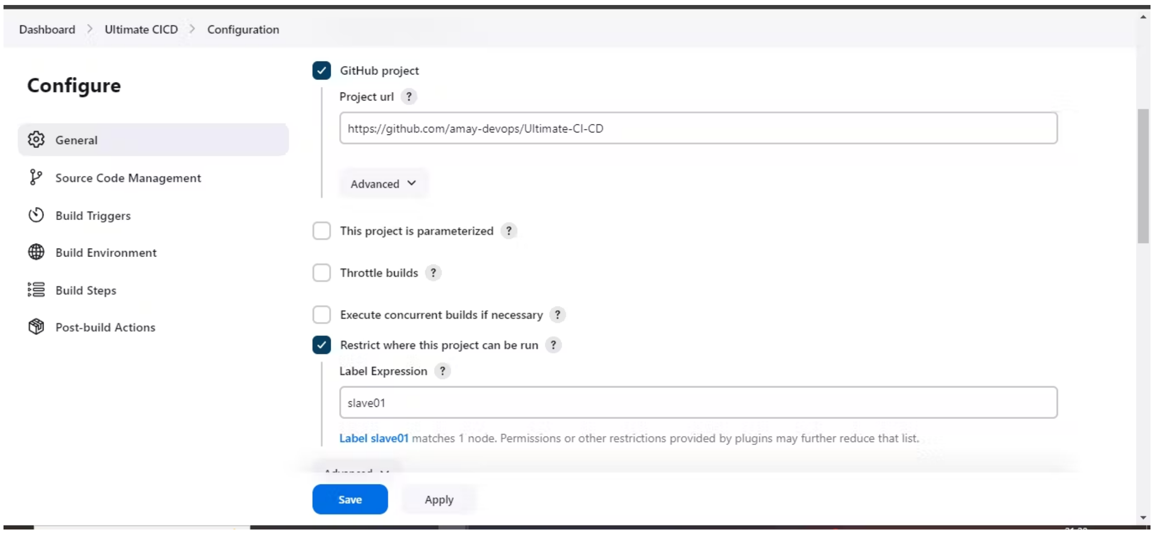
```
AWS Access Ke                      ated earlier)
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
(Generated earlier)
Default region name [None]: us-west-2
Default output format [None]: json
```

5. Following a successful command run. Execute only the first command. Following the successful execution of the first instruction, it will display "Login succeeded."

```
aws ecr get-login-password --region region | docker login --username
AWS --password-stdin aws_account_ id.dkr.ecr.region.amazonaws.com
```

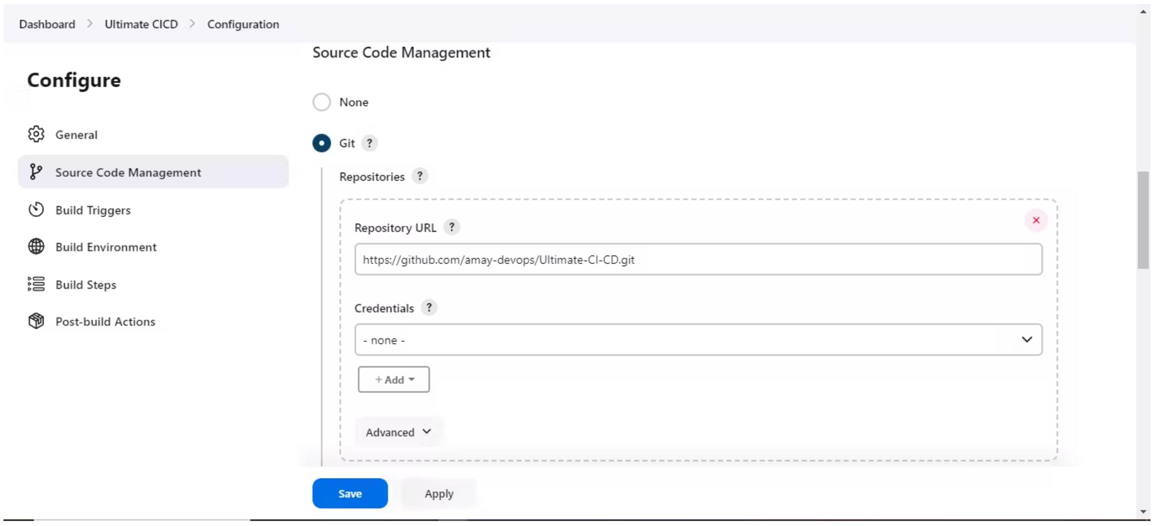
Project Set-up

1. To access the **Freestyle Project, navigate to Dashboard → New Item. Select GitHub Project** in the setup area and enter the project URL:- https://github.com/amay-devops/Ultimate-CI-CD.
2. Select **"Restrict where this project can be run"** Enter the label name that was previously set when we added the node.



GitHub project ✓
Project url: https://github.com/amay-devops/Ultimate-CI-CD
This project is parameterized
Throttle builds
Execute concurrent builds if necessary
Restrict where this project can be run ✓
Label Expression: slave01
Label slave01 matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

3. The Git repository URL should be added in the SCM (Source Code Management Section) by clicking on Git and entering https://githu[illegible].git. Additionally, indicate which [illegible]osen.



Repository URL: https://github.com/amay-devops/Ultimate-CI-CD.git

Credentials: - none -

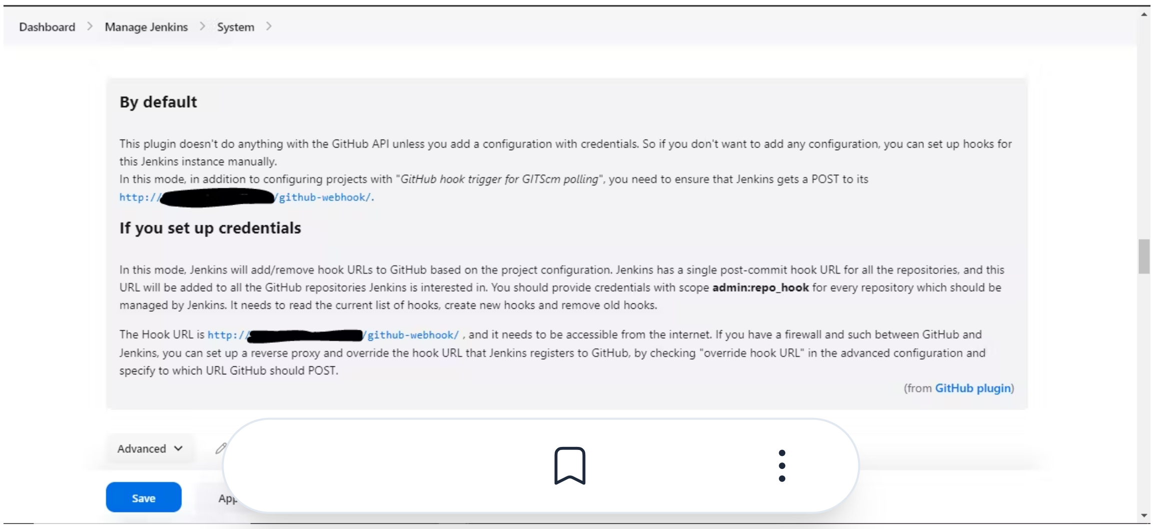
4. Select the **GitHub hook trigger for GITScm polling** in the Build Trigeer section.

5. Choose **Execute Shel**l under the Build Setp section, then enter the following command.

```
mvn clean package (This will generate the artifact in the target
folder)
docker build -t amaysharma .;
docker tag amaysharma:latest
public.ecr.aws/********/<Public_ECR_name>:latest ; docker push
public.ecr.aws/********/<Public_ECR_name>:latest ;
```

6. Save and exit.

7. The webhook URL can be copied and pasted from Manage Jenkins → System → On the GitHub Servers, click on?



By default

This plugin doesn't do anything with the GitHub API unless you add a configuration with credentials. So if you don't want to add any configuration, you can set up hooks for this Jenkins instance manually.
In this mode, in addition to configuring projects with *"GitHub hook trigger for GITScm polling"*, you need to ensure that Jenkins gets a POST to its http:/[redacted]/github-webhook/.

If you set up credentials

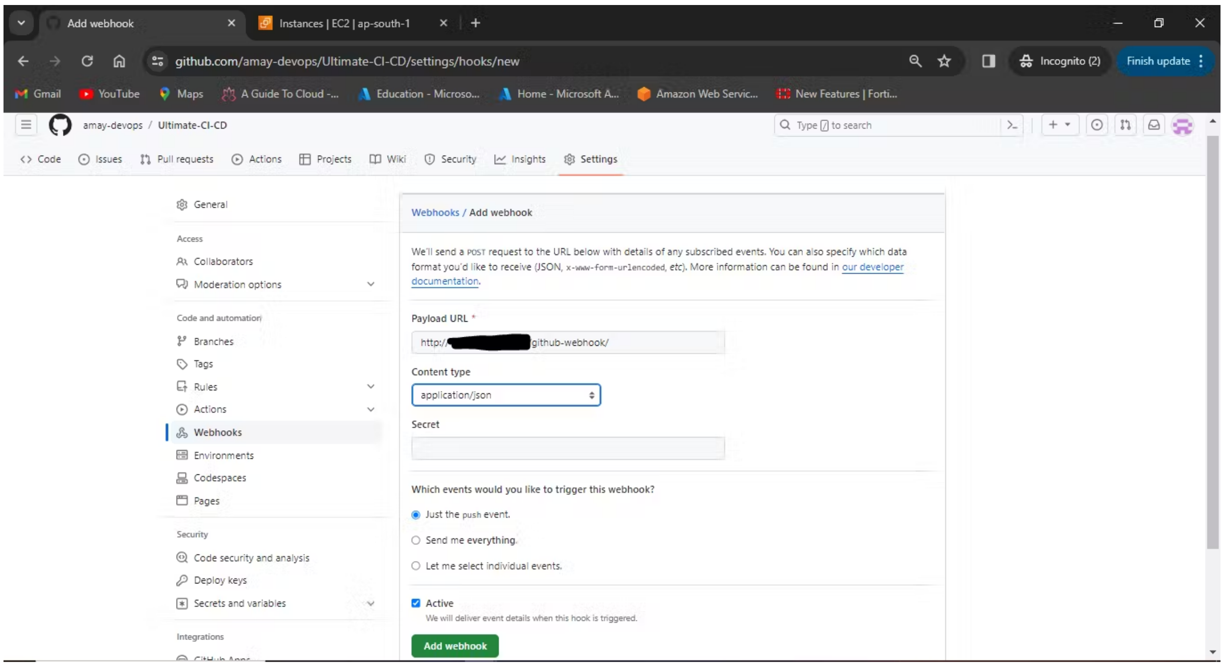
In this mode, Jenkins will add/remove hook URLs to GitHub based on the project configuration. Jenkins has a single post-commit hook URL for all the repositories, and this URL will be added to all the GitHub repositories Jenkins is interested in. You should provide credentials with scope **admin:repo_hook** for every repository which should be managed by Jenkins. It needs to read the current list of hooks, create new hooks and remove old hooks.

The Hook URL is http:/[redacted]/github-webhook/ , and it needs to be accessible from the internet. If you have a firewall and such between GitHub and Jenkins, you can set up a reverse proxy and override the hook URL that Jenkins registers to GitHub, by checking "override hook URL" in the advanced configuration and specify to which URL GitHub should POST.

(from GitHub plugin)

GitHub WebHook Set-up

Navigate to your GitHub repository and select **Settings → Webhook → Add Webhook. paste the copied URL. Choose the application type (JSON/application) → Select "just push event"** and Add Webhook.



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

Payload URL *
http:/[redacted]github-webhook/

Content type
application/json

Secret

Which events would you like to trigger this webhook?
[✓ Just the push event.]
Send me everything.
Let me select individual events.

[✓ Active]
We will deliver event details when this hook is triggered.

Add webhook

Code Commit

Navigate to the local repository on the code-hosting PC and execute the following commands:

```
git init
```

```
git remote add origin  https://github.com/amay-devops/Ultimate-CI-CD.git
(In place of my repository link just paster your repository link.
```

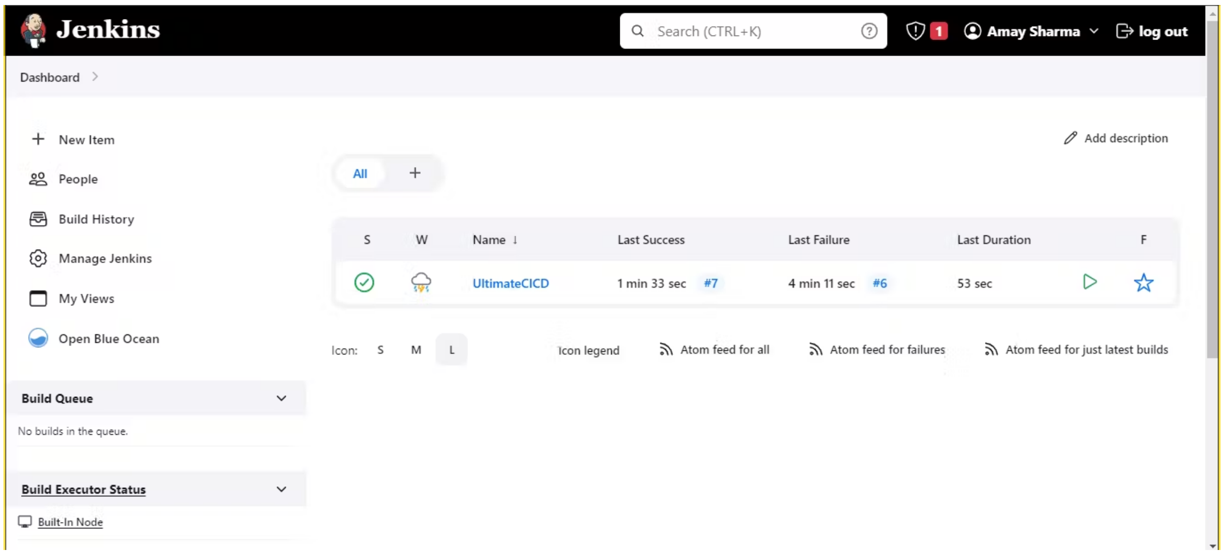
```
git status
```

```
git add .
```

```
git commit -m "Initial Commit"
```

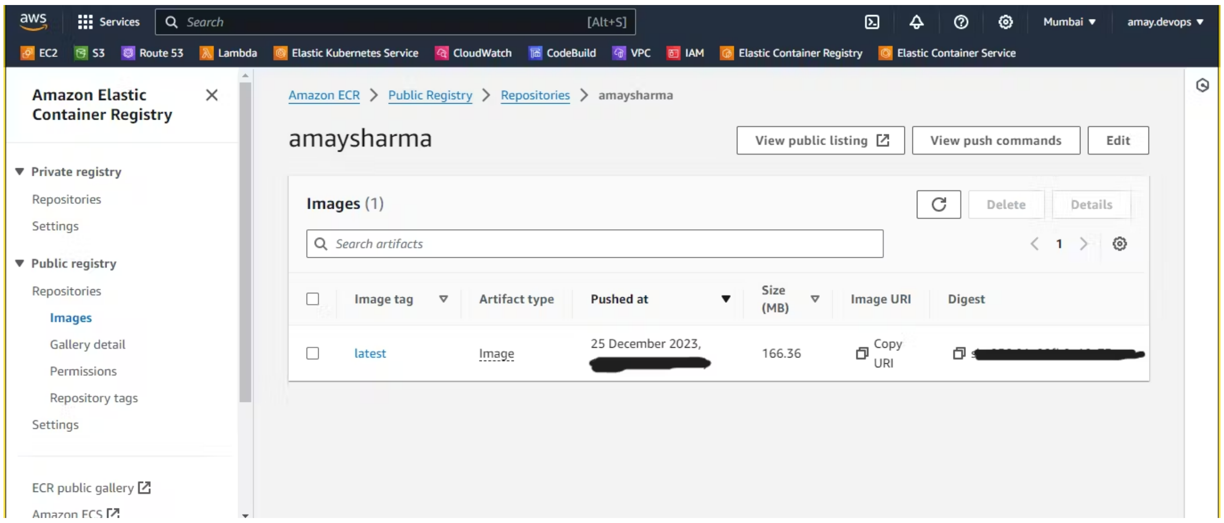
```
git push origin master
```

Results



No builds in the queue.

S	W	Name ↓	Last Success	Last Failure	Last Duration	F
✓		UltimateCICD	1 min 33 sec #7	4 min 11 sec #6	53 sec	

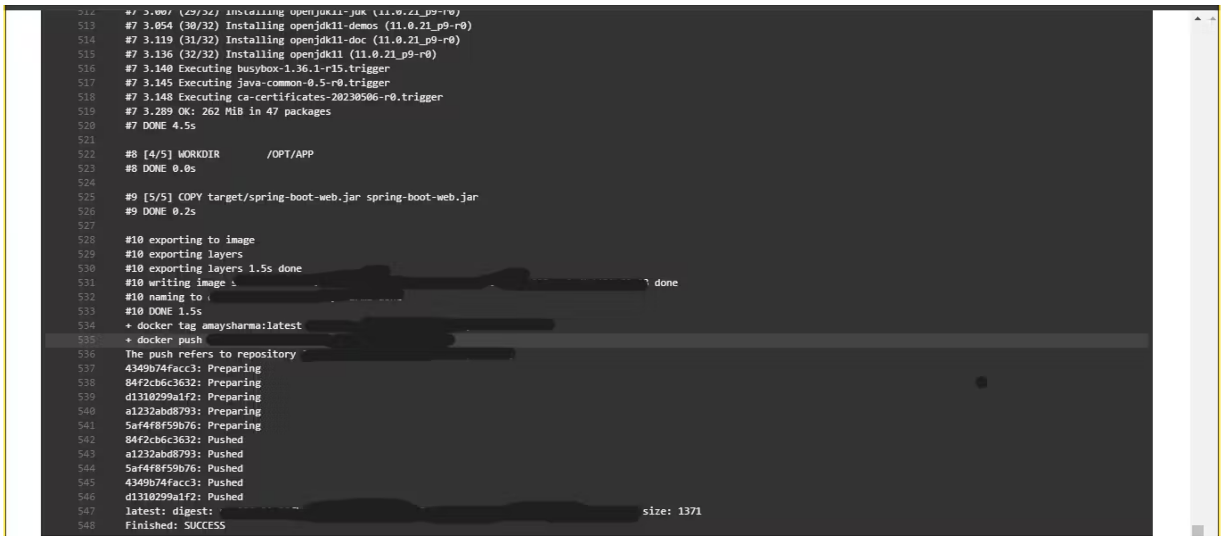


Amazon ECR > Public Registry > Repositories > amaysharma

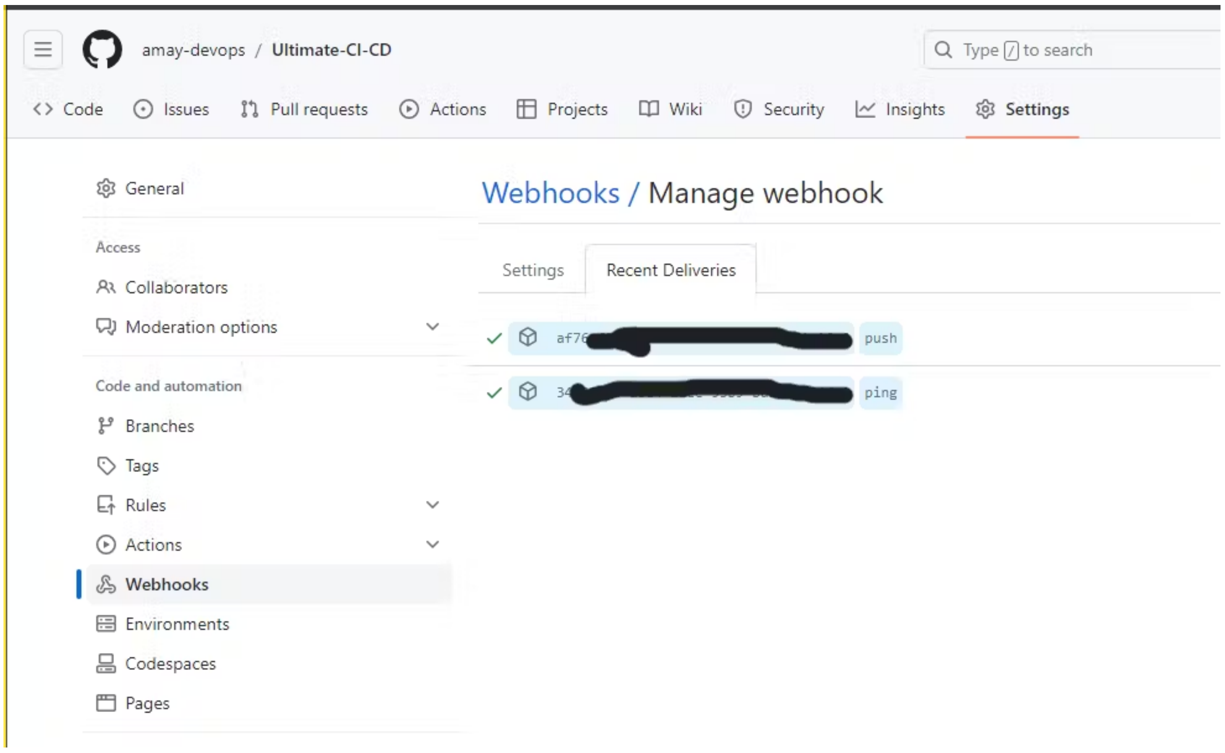
amaysharma

Images (1)

	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
	latest	Image	25 December 2023, [redacted]	166.36	Copy URI	[redacted]



```
#7 3.007 (29/32) Installing openjdk11-jdk (11.0.21_p9-r0)
#7 3.054 (30/32) Installing openjdk11-demos (11.0.21_p9-r0)
#7 3.119 (31/32) Installing openjdk11-doc (11.0.21_p9-r0)
#7 3.136 (32/32) Installing openjdk11 (11.0.21_p9-r0)
#7 3.140 Executing busybox-1.36.1-r15.trigger
#7 3.145 Executing java-common-0.5-r0.trigger
#7 3.148 Executing ca-certificates-20230506-r0.trigger
#7 3.289 OK: 262 MiB in 47 packages
#7 DONE 4.5s

#8 [4/5] WORKDIR        /OPT/APP
#8 DONE 0.0s

#9 [5/5] COPY target/spring-boot-web.jar spring-boot-web.jar
#9 DONE 0.2s

#10 exporting to image
#10 exporting layers
#10 exporting layers 1.5s done
#10 writing image [redacted] done
#10 naming to [redacted]
#10 DONE 1.5s
+ docker tag amaysharma:latest [redacted]
+ docker push [redacted]
The push refers to repository [redacted]
4349b74facc3: Preparing
84f2cb6c3632: Preparing
d1310299a1f2: Preparing
a1232abd8793: Preparing
5af4f8f59b76: Preparing
84f2cb6c3632: Pushed
a1232abd8793: Pushed
5af4f8f59b76: Pushed
4349b74facc3: Pushed
d1310299a1f2: Pushed
latest: digest: [redacted] size: 1371
Finished: SUCCESS
```



✓ af76[redacted] push

✓ 34[redacted] ping

SonarQube will be included in the future between the pipeline. Additionally, we'll utilize Amazon Lambda to call a function that deploys an Amazon EKS cluster taking AWS ECR pushed image. In addition, we'll attempt to use Terraform for the infrastructure as a code in addition to a few other continuous integration tools like ArgoCD and GitHub CI.

AWS Jenkins Devops AWS ECR Springboot GitHub

Kubernetes Docker IAM ec2 Pipeline

Written by

AS Amay Sharma

Follow