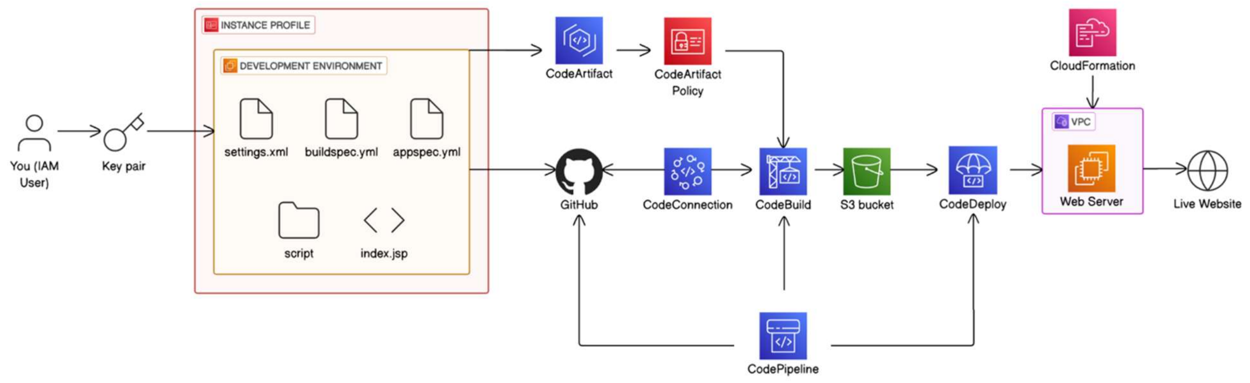
Vashrith Vinodh

Creating a CI/CD Pipeline from Scratch

Overview

Project Basics

The aim of this project is to build a fully function CI/CD pipeline from scratch. I don't have any strong experience in DevOps frameworks or technical skills, but I will learn the fundamentals and implement them into this project.



Skills I Know

SSH, VS Code, Git, GitHub

Skills I Learned

AWS, Maven, Amazon, Corretto, CodeArtifact, CodeBuild, CodeDeploy, CloudFormation

How I Faced Problems

I had an initial error when trying to SSH into my environment on VS Code, it would always disconnect within a couple seconds and stay stuck on reconnecting. I figured out that instance was not sufficiently powered to host Java and SSH, so I upgraded the instance to t3 micro.

Another error I encountered was when I was unable to connect to the EC2 via SSH. It had been working just fine the night before so I thought of the factors that had changed since then. I realized that I was at a different location using a different network, meaning my inbound rule did not let the new IP connect. I then added the IP I was connected to and successfully connected to the EC2.

I also faced a build error for my CodeDeploy, stating it was unable to access the files necessary. I eventually noticed that there was an error in my install_dependencies.sh script in the proxy declaration. A small typo was causing the issue, however, once that was fixed, the build still failed. I then realized that I had forgotten to push the changes to GitHub, which then led to a successful build.

Abstract

AWS Setup and EC2 Development Environment (Stage 1)

1. **AWS Console and IAM**
    - Created an AWS account and an IAM user
    - Configured permissions to securely access AWS services
2. **Launched EC2 Instance**
    - Created a virtual machine and set OS, instance type, key pair, and network settings
3. **VS Code and SSH Integrations**
    - Configured .pem key permissions.
    - Connected to the EC2 instance in VS Code using the remote -SSH extension
4. **Installed Development Tools**
    - Amazon Corretto 8
    - Apache Maven
5. **Created a Web Application**
    - Used Maven archetype to generate a basic web applicatio
    - Edited project files directly from VS Code

Launch Ec2 Instance (Stage 2)

1. **Installed Git on EC2**
2. **Connected Project to GitHub Repository**
    - Initialized repository and added remote origin.
    - Commited and pushed changes.
3. **Authentication**
    - Used GitHub personal access token for secure authorization

AWS CodeArtifact (Stage 3)

1. **Created CodeArtifact Repository**
    - Managed project dependencies with Maven
    - Configured upstream repository and domain
2. **IAM Policies and Roles**
    - Granted EC2 instance permissions to access CodeArtifact packages
3. **Configured Maven**
    - Created a settings file with repository credentials
    - Compiled the project
4. **Verified Packages Appear in CodeArtifact Console**

AWS CodeBuild (Stage 4)

1. **Created CodeBuild Project**
    - Connected to the GitHub repository using AWS CodeStar Connection
    - Configured build environment, buildspec, and artifact output to S3
2. **BuildSpec.yml**
    - Defined install, pre-build, build, and post-build phases
    - Specified files to save as artifacts
3. **Ran the Build and Logs**
    - CloudWatch monitored logs for debugging
    - Successful builds were verified in S3

Deployment With CodeDeploy (Stage 5)

1. **Prepared deployment scripts**
2. **AppSpec.yml**
    - Instructions for CodeDeploy on how to deploy the app
3. **CodeDeploy Application and Deployment Group**
    - Attached IAM Role
    - Specified S3 revision location
    - Deployed build artifacts
4. **Verified Deployment**
    - Accessed the web app via the EC2 public DNS address

CloudFormation Automation (Stage 6)

1. **Generated CloudFormation Template**
2. **Launched Template**
    - Automatically deploys all resources and infrastructure

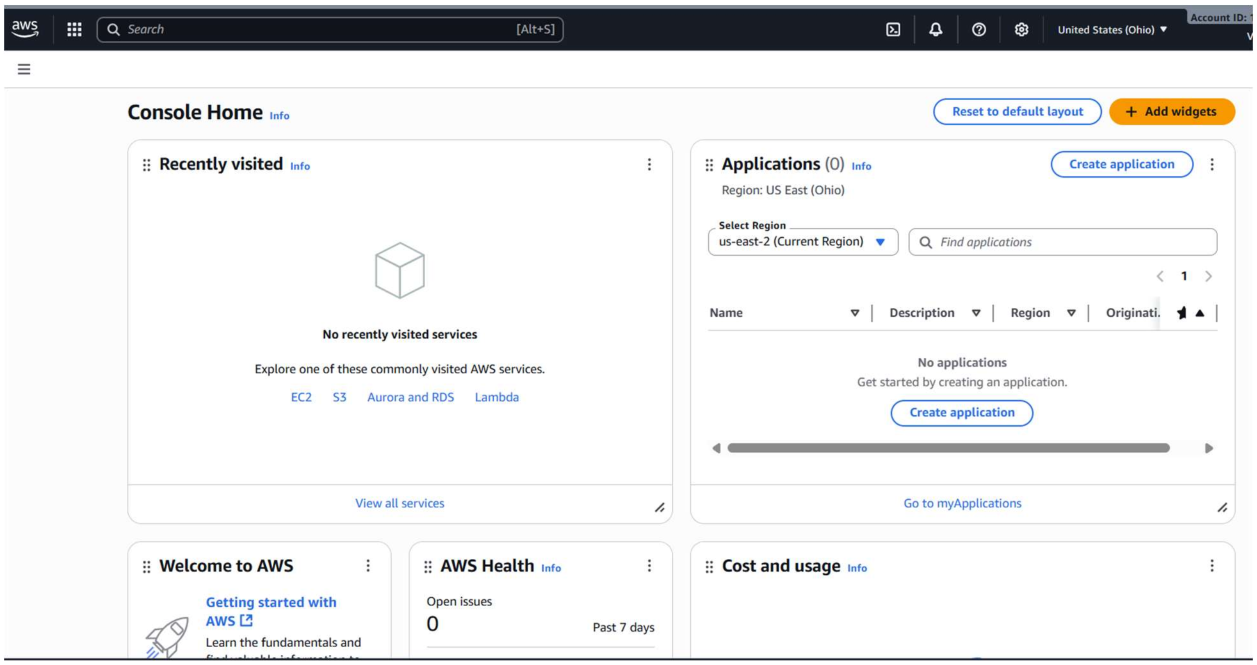
Stage 1

Setting up AWS Console

This stage will demonstrate how to build a CI/CD pipeline that will automate the build and deployment of a web app.

The first step is to create an AWS Console account, then create an IAM user. This is done because initial access to the console is granted to the root user, in the real world, we don't want root privileges for daily activities like this, the least privilege method. We should give the minimum number of capabilities as necessary for the task.

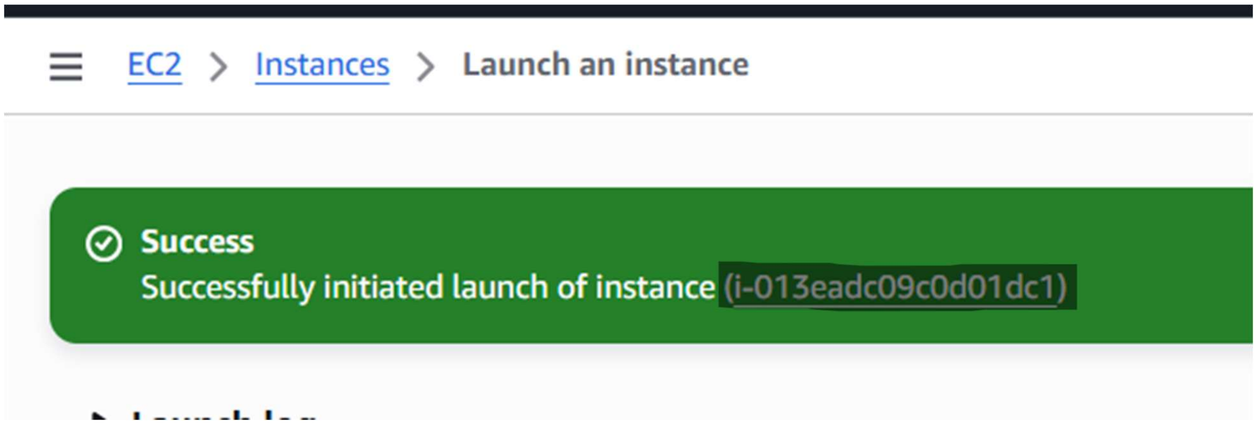
Once the user credentials are configured, the next step is to set the permissions of the user. Now we use the link provided and enter the credentials we set up, so that we can access the console from the user account.



Launching an EC2 Instance

EC2 is essentially a virtual machine hosted on Amazon's cloud network. We can personalize the instance through scaling and customizability, allowing for a viable option to create cloud-based networks. This EC2 will be the home for our development work.

Once we configure our EC2 with its name, OS, instance type, key pair, and network settings, our EC2 is up and running.



EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance [redacted]

Configuring VS Code

Our first step here is to navigate to the folder containing our .pem key. Then, we can change its permission with chmod. We use the following commands,

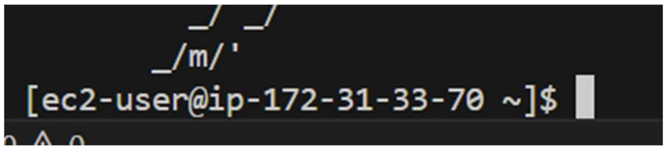
icacls "filename" /reset
icacls "filename" /grant:r "username:R"
icacls "filename" /inheritance:r

This allows only the owner to read the file while restricting access to anyone else on the server.

Connecting to the EC2 Instance

We need to go back to the AWS console and find the public IPv4 DNS. Using this DNS will show how the local computer can connect to our virtual machine. Now we need to connect to our instance using ssh. Ssh stands for secure shell, it allows an encrypted connection between users and a server. We can connect to our instance with the following command,

ssh -i "PATH TO .PEM FILE" ec2-user@"PUBLIC IPV4 DNS"



```
          _/ _/
        _/m/'
 [ec2-user@ip-172-31-33-70 ~]$ 
```

Install Apache Maven and Amazon Corretto 8

Now that our connection is successful, our next step is to set up a web app inside our instance. We will install Apache Maven and Amazon Corretto 8, these applications will help build Java web apps.

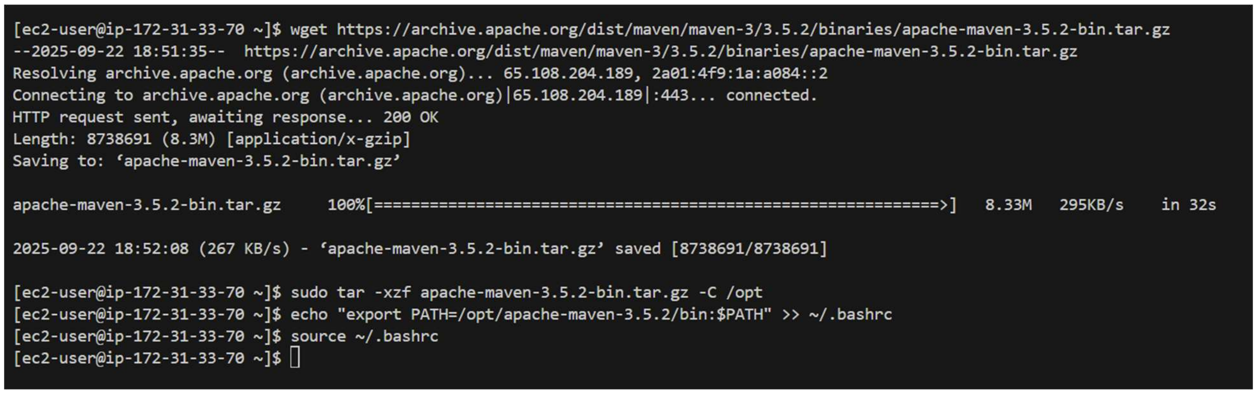
Apache Maven is a tool that allows users to build and organize Java software projects. It has archetypes, which essentially means templates, to lay foundations for projects. It can be installed with the following commands:

wget https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz

sudo tar -xzf apache-maven-3.5.2-bin.tar.gz -C /opt

echo "export PATH=/opt/apache-maven-3.5.2/bin:$PATH" >> ~/.bashrc

source ~/.bashrc



```
[ec2-user@ip-172-31-33-70 ~]$ wget https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz
--2025-09-22 18:51:35--  https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f9:1a:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8738691 (8.3M) [application/x-gzip]
Saving to: ‘apache-maven-3.5.2-bin.tar.gz’

apache-maven-3.5.2-bin.tar.gz     100%[=============================================================>]   8.33M   295KB/s    in 32s

2025-09-22 18:52:08 (267 KB/s) - ‘apache-maven-3.5.2-bin.tar.gz’ saved [8738691/8738691]

[ec2-user@ip-172-31-33-70 ~]$ sudo tar -xzf apache-maven-3.5.2-bin.tar.gz -C /opt
[ec2-user@ip-172-31-33-70 ~]$ echo "export PATH=/opt/apache-maven-3.5.2/bin:$PATH" >> ~/.bashrc
[ec2-user@ip-172-31-33-70 ~]$ source ~/.bashrc
[ec2-user@ip-172-31-33-70 ~]$ 
```

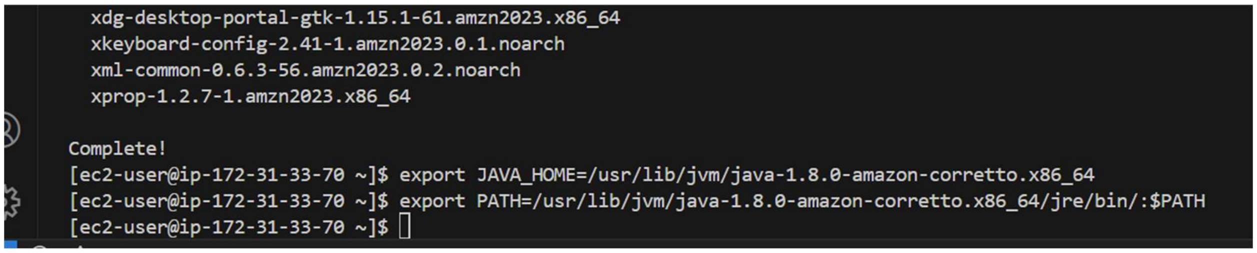
The first command downloads the setup file from the url, the second command extracts the package and saves it to the folder /opt, the third and fourth commands save a path to the package so that we can run Maven commands from any directory.

Amazon Corretto 8 is a version of the Java programming language. Maven needs Java to operate, meaning we can’t build the web app without it. It can be installed with the following commands:

sudo dnf install -y java-1.8.0-amazon-corretto-devel

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64

export PATH=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre/bin/:$PATH

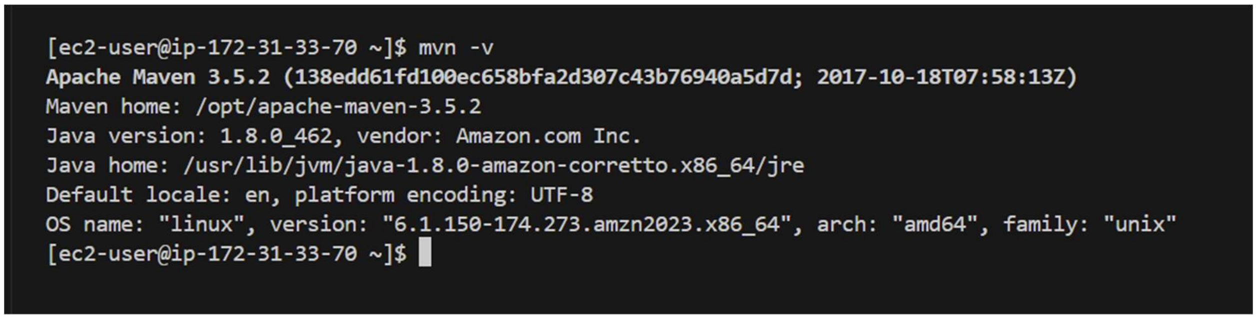


```
  xdg-desktop-portal-gtk-1.15.1-61.amzn2023.x86_64
  xkeyboard-config-2.41-1.amzn2023.0.1.noarch
  xml-common-0.6.3-56.amzn2023.0.2.noarch
  xprop-1.2.7-1.amzn2023.x86_64

Complete!
[ec2-user@ip-172-31-33-70 ~]$ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64
[ec2-user@ip-172-31-33-70 ~]$ export PATH=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre/bin/:$PATH
[ec2-user@ip-172-31-33-70 ~]$ 
```

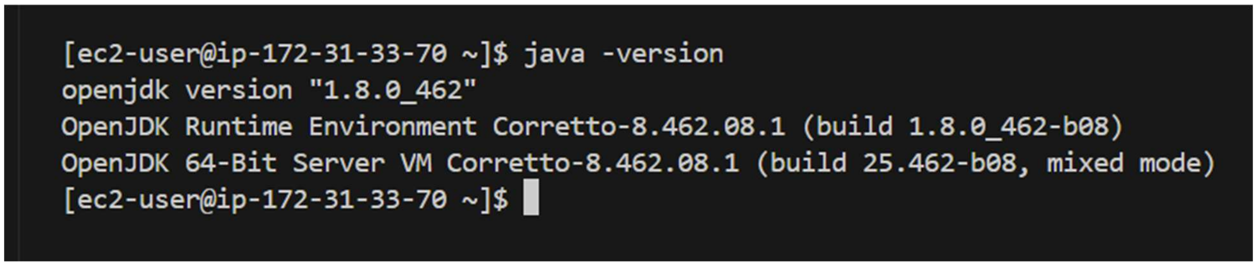
The first command install Java Amazon Corretto 8, the second command tells the EC2 how to find Java, the third command saves the location of Java so that it can be run from anywhere in the instance.

To verify that Maven is installed, we can run ***mvn -v***



```
[ec2-user@ip-172-31-33-70 ~]$ mvn -v
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T07:58:13Z)
Maven home: /opt/apache-maven-3.5.2
Java version: 1.8.0_462, vendor: Amazon.com Inc.
Java home: /usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.1.150-174.273.amzn2023.x86_64", arch: "amd64", family: "unix"
[ec2-user@ip-172-31-33-70 ~]$
```

To verify that java is installed, we can run ***java -version***



```
[ec2-user@ip-172-31-33-70 ~]$ java -version
openjdk version "1.8.0_462"
OpenJDK Runtime Environment Corretto-8.462.08.1 (build 1.8.0_462-b08)
OpenJDK 64-Bit Server VM Corretto-8.462.08.1 (build 25.462-b08, mixed mode)
[ec2-user@ip-172-31-33-70 ~]$
```

Creating the Application

Now that Maven and Java have been integrated to the EC2 instance, we can generate the web app. We can do that with the following command,

mvn archetype:generate \

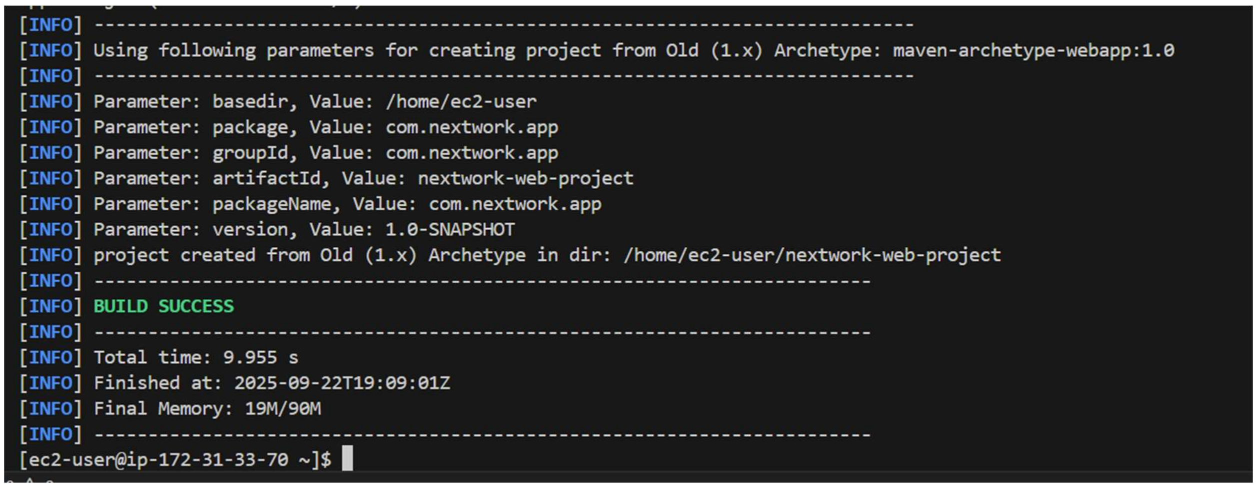
-DgroupId=com.work.app \

- DartifactId=web-project \

-DarchetypeArtifactId=maven-archetype-webapp \

-DinteractiveMode=false

Mvn commands tell maven to perform tasks. The archetype:generate tells Maven to create a new project from a template (archetype), essentially setting up the basic structure for this project. DartifactId names the projects, DarchetypeArtifactId specifies that a web application is being created, DinteractiveMode tells the computer to whether or not it should wait for user confirmation.



```
[INFO] ------------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-webapp:1.0
[INFO] ------------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: /home/ec2-user
[INFO] Parameter: package, Value: com.nextwork.app
[INFO] Parameter: groupId, Value: com.nextwork.app
[INFO] Parameter: artifactId, Value: nextwork-web-project
[INFO] Parameter: packageName, Value: com.nextwork.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/ec2-user/nextwork-web-project
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.955 s
[INFO] Finished at: 2025-09-22T19:09:01Z
[INFO] Final Memory: 19M/90M
[INFO] ------------------------------------------------------------------------
[ec2-user@ip-172-31-33-70 ~]$
```

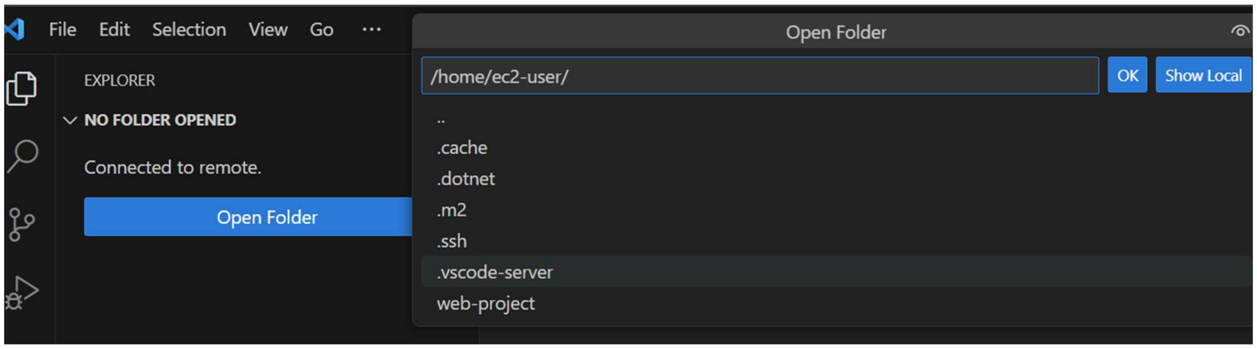
Connecting VS Code with the EC2 Instance

The first thing to do is check if the Remote -SSH extension is installed in VS Code. If not, it can be installed by installing Remote Development. We can than open the Remote -SSH and select Connect to Host, then + Add new SSH Host. We can input the same command we used to connect to our EC2 instance from the terminal,

ssh -i "PATH TO .PEM FILE" ec2-user@"PUBLIC IPV4 DNS"

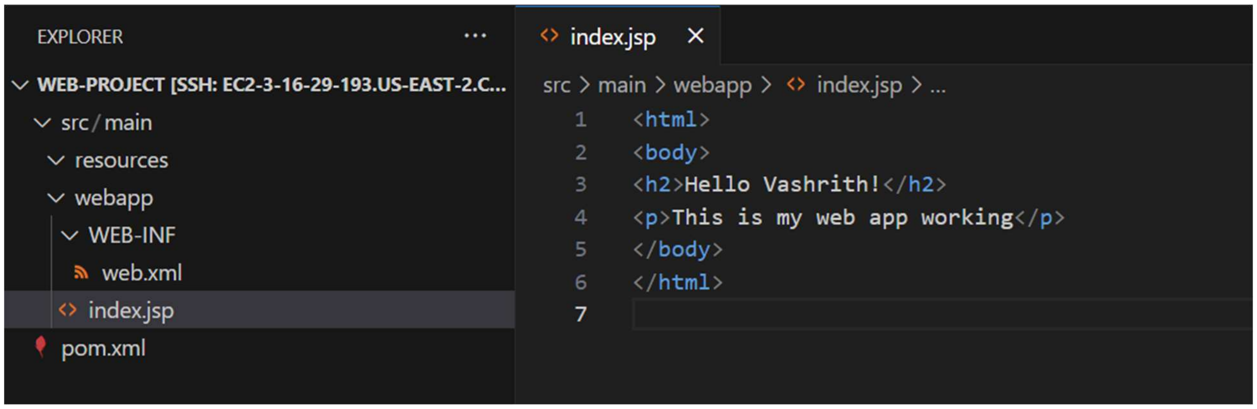
Then select the config file containing the name of local machine user. A file containing the Host (matching with the EC2 IPv4 DNS, an identity file (matching with the location of the .pem file), and a user (saying ec2-user).

Now if we open the extension and click on "connect to host", we should get an option to connect to the EC2 instance. It will take a second and it should connect successfully. Next is to open the folder where we have our web app.



```
/home/ec2-user/
..
.cache
.dotnet
.m2
.ssh
.vscode-server
web-project
```

We can then open the index.jsp file in the folder, this is a HTML file for our web app.


EXPLORER
WEB-PROJECT [SSH: EC2-3-16-29-193.US-EAST-2.C...
src / main
resources
webapp
WEB-INF
web.xml
index.jsp
pom.xml
index.jsp
src > main > webapp > index.jsp > ...

```
<html>
<body>
<h2>Hello Vashrith!</h2>
<p>This is my web app working</p>
</body>
</html>
```

Stage 2

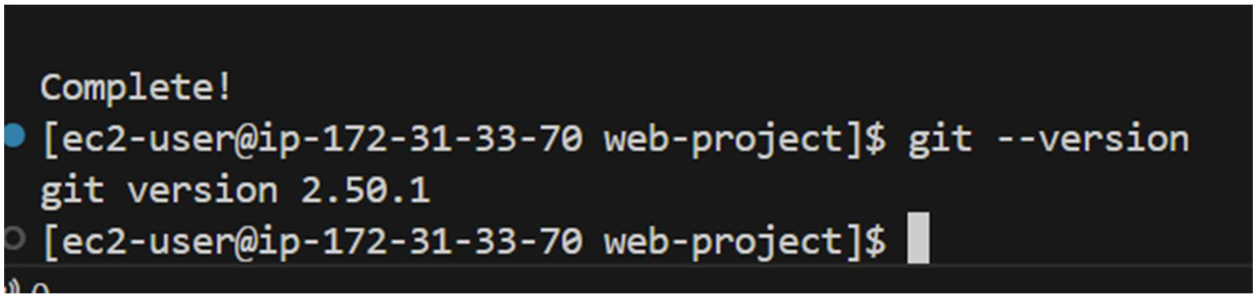
Installing Git and Setting up GitHub

To install git on the EC2 instance, we need to open our terminal and run the following commands,

sudo dnf update -y

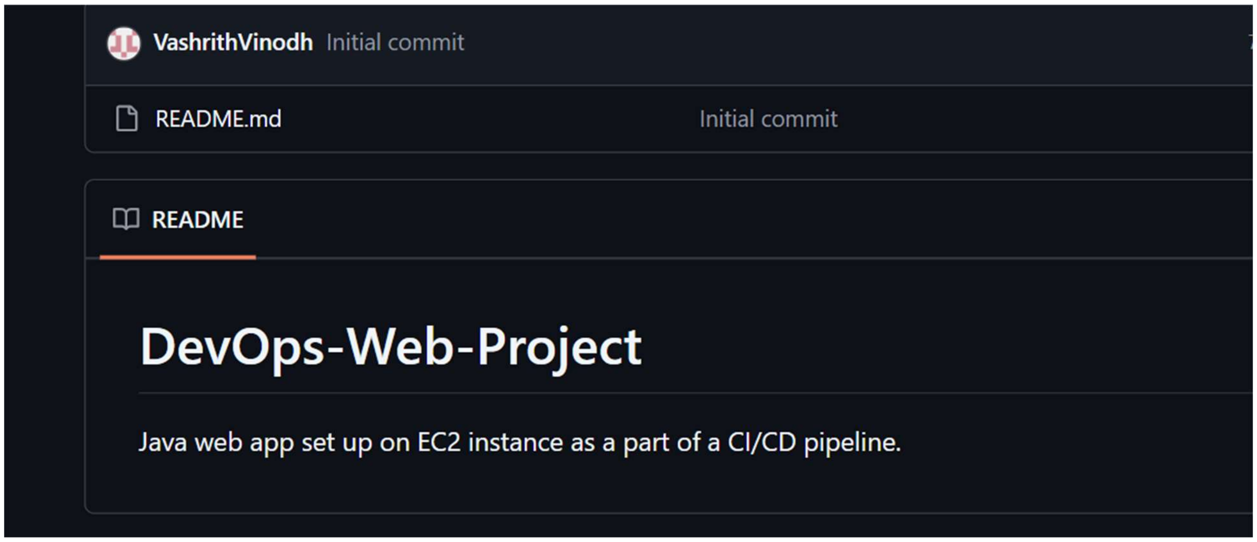
sudo dnf install git -y

The first command tells the instance to install the latest updates of all the software currently on the system. The second command tells the instance to install git, with the -y being advanced approval to download. The reason behind updating software before downloading anything new is to ensure that there are no compatibility issues. We can verify the installation with ***git –version.***



```
Complete!
[ec2-user@ip-172-31-33-70 web-project]$ git --version
git version 2.50.1
[ec2-user@ip-172-31-33-70 web-project]$ 
```

Then we need to log into GitHub and create a new repository.



VashrithVinodh Initial commit

README.md Initial commit

README

DevOps-Web-Project

Java web app set up on EC2 instance as a part of a CI/CD pipeline.

Committing and Pushing Changes to GitHub

Now that the repository has been set up, we need to connect it to the web app on the EC2 instance. Navigate to the web app folder and run the command ***git init.*** This command sets up the directory as a local Git repository. Head back to GitHub and copy the HTTP clone link. In the terminal, run the command ***git remote add origin "HTTP clone link".*** This command essentially tells Git where the GitHub repository is located. Next we run the following commands:

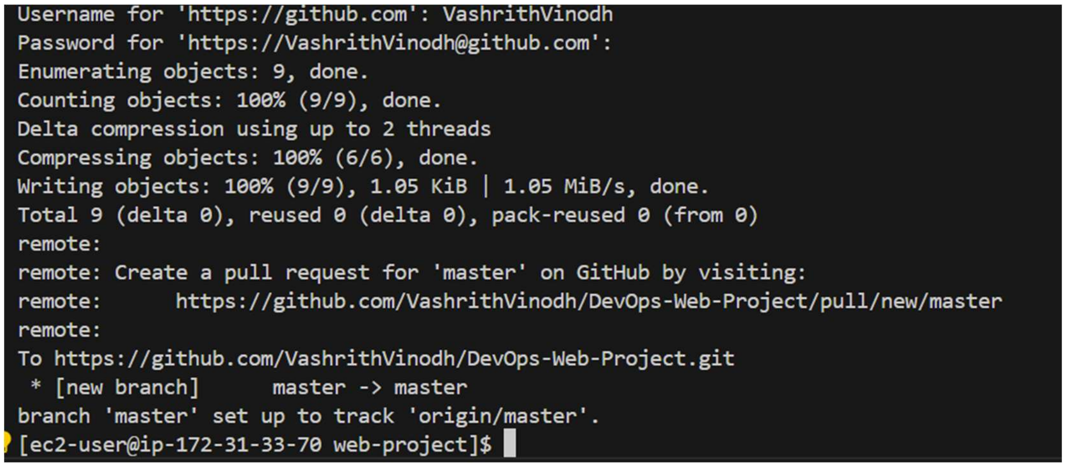
git add .

git commit -m "Updated index.jsp with new content"

git push -u origin master

The first command stages all the files in the folder, the second command gives the staged files a message, for some additional information, the third command pushes the staged files with the message to our GitHub repository. The -u sets the “push to master” as default, the next time we need to push something, ***git push*** is enough. The terminal will prompt for a username and password as authentication, but once entering those credentials, it will give an error response of “password is not valid for authentication”. This is a security measure in place; we need to generate a token and use that to access the repository.

Head to developer settings in GitHub settings and generate a new personal token. Select repo as the scope and hit Generate Token. Run the git push process in the terminal again and for the password, paste the token.



```
Username for 'https://github.com': VashrithVinodh
Password for 'https://VashrithVinodh@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 1.05 KiB | 1.05 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/VashrithVinodh/DevOps-Web-Project/pull/new/master
remote:
To https://github.com/VashrithVinodh/DevOps-Web-Project.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
[ec2-user@ip-172-31-33-70 web-project]$
```

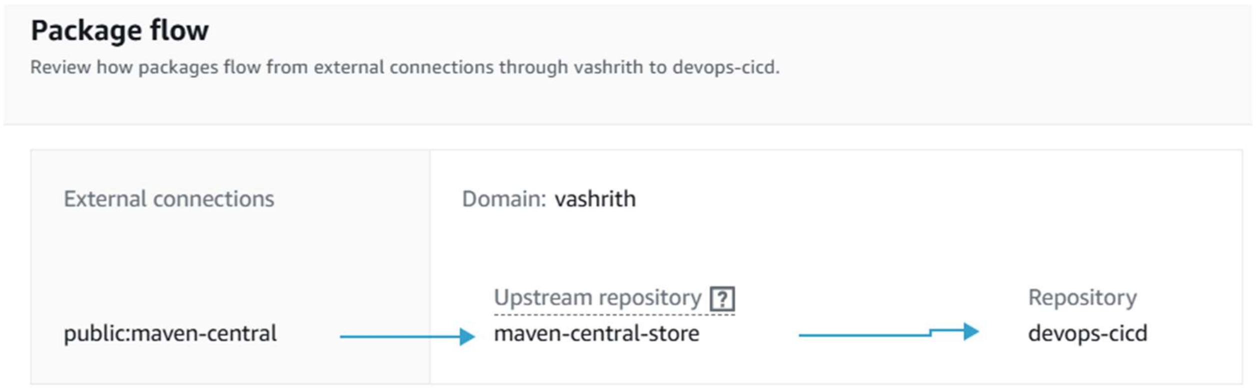
Now in the GitHub repository, we should be able to see the changes committed.

Stage 3

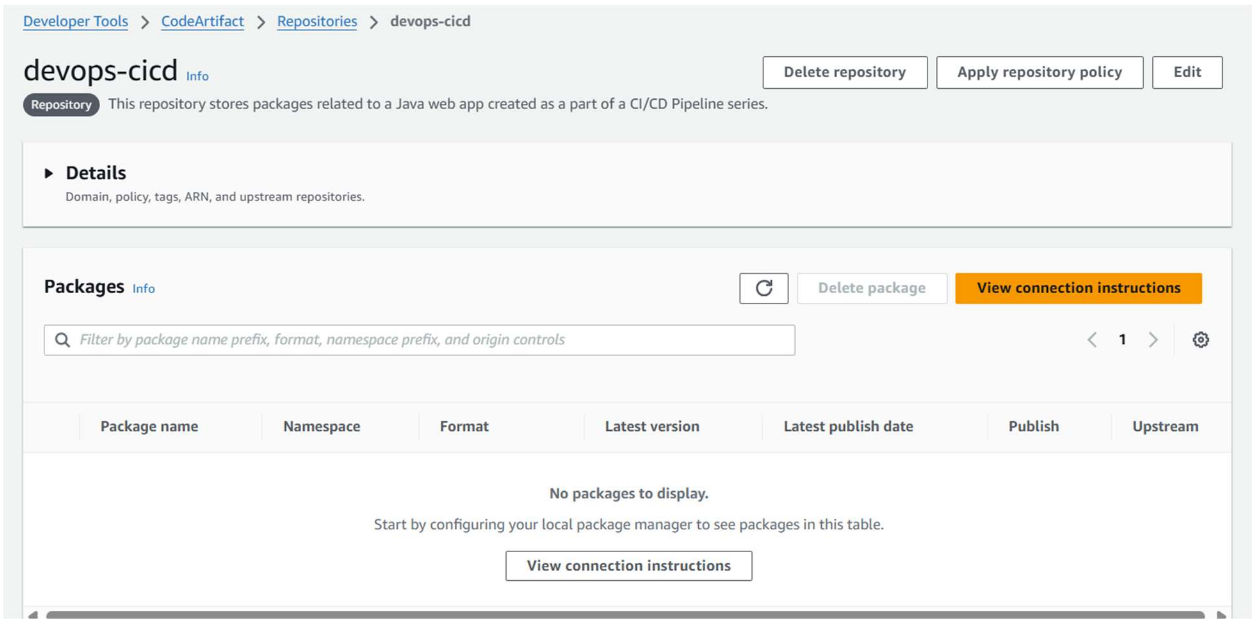
Setting up AWS CodeArtifact Repository

We need to set up CodeArtifact to store and manage our dependencies, this ensures easy and reliable access to the dependencies.

Head to the CodeArtifact in Developer tools and create a new repository. Enter the basic information and make sure to select maven-central-stream for public upstream repository, this acts as an external library that CodeArtifact will check. Then configure the domain settings, a hub where all repositories are stored and their permissions can be set.



This diagram shows us how the dependencies will travel to the application. Ensure all the details are correct and click create repository.


Developer Tools > CodeArtifact > Repositories > devops-cicd
devops-cicd Info
Delete repository
Apply repository policy
Edit
Repository
Details
Domain, policy, tags, ARN, and upstream repositories.
Packages Info
Delete package
View connection instructions
Filter by package name prefix, format, namespace prefix, and origin controls
1
View connection instructions

This repository stores packages related to a Java web app created as a part of a CI/CD Pipeline series.

Package name	Namespace	Format	Latest version	Latest publish date	Publish	Upstream

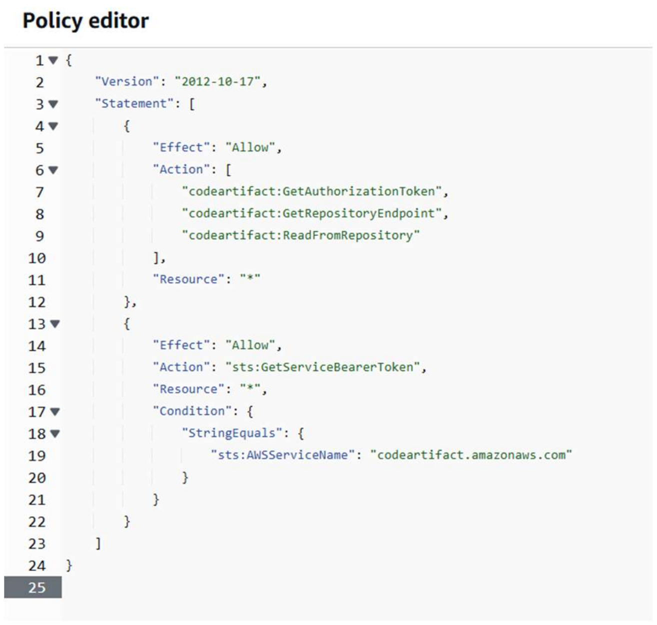
No packages to display.

Start by configuring your local package manager to see packages in this table.

Creating an IAM Policy for CodeArtifact Access

To ensure that Maven can work with CodeArtifact, we have to set up an IAM role and IAM policy that allows EC2 the permission to access CodeArtifact.

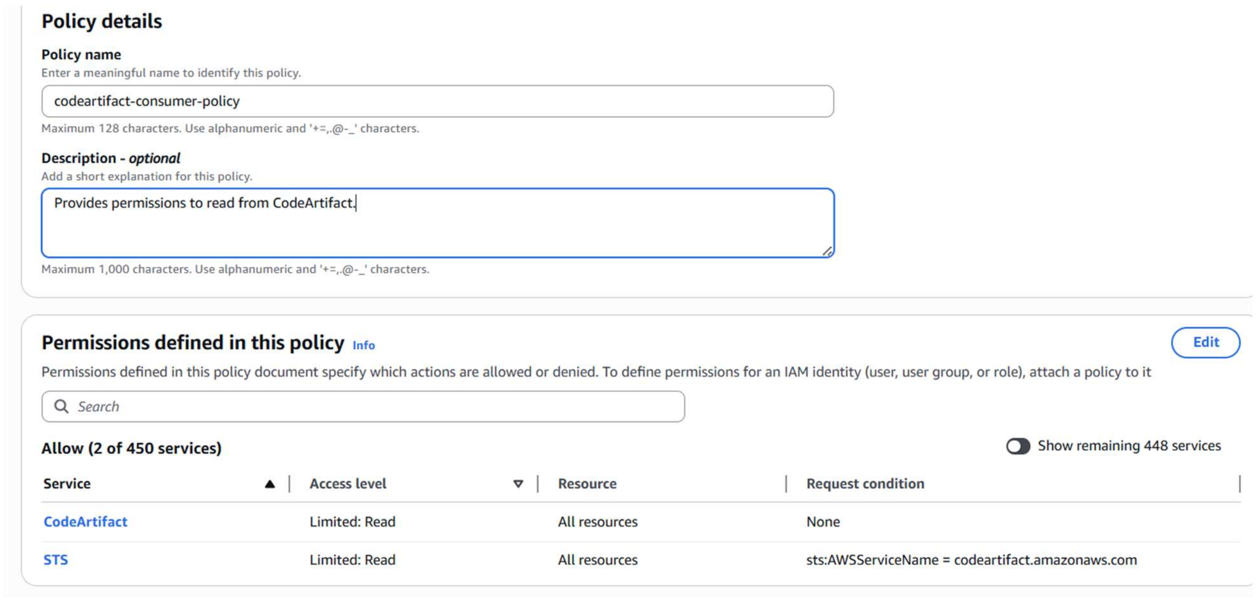
Navigate to IAM policies and select JSON, the input the following code.



Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeartifact:GetAuthorizationToken",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ReadFromRepository"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

The first block grants permission to get the authentication tokens, find repository locations, and read packages form repositories. The second block allows temporary elevated access to CodeArtifact operations.



Policy details

Policy name
Enter a meaningful name to identify this policy.
codeartifact-consumer-policy
Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

Description - *optional*
Add a short explanation for this policy.
Provides permissions to read from CodeArtifact.
Maximum 1,000 characters. Use alphanumeric and '+=,.@-_' characters.

Permissions defined in this policy
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (2 of 450 services)

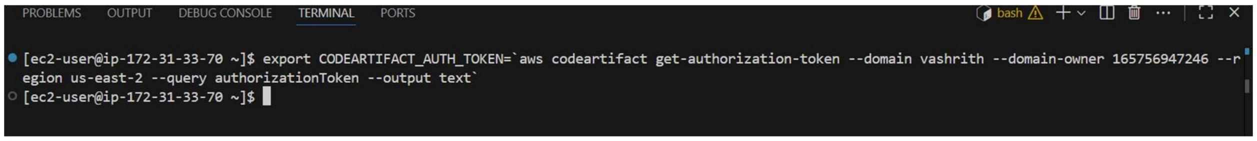
Service	Access level	Resource	Request condition
CodeArtifact	Limited: Read	All resources	None
STS	Limited: Read	All resources	sts:AWSServiceName = codeartifact.amazonaws.com

Attach IAM Policy and Verify CodeArtifact Connection

Now we need to create an IAM role with our policy. Head to roles and click on create new role. Select the entity type and use case, then select the policies to add. Configure the descriptions and create the role.

Head to the instances tab, click on the EC2 instance and select security > modify IAM role. In the dropdown menu, select the role to add and hit update to attach it.

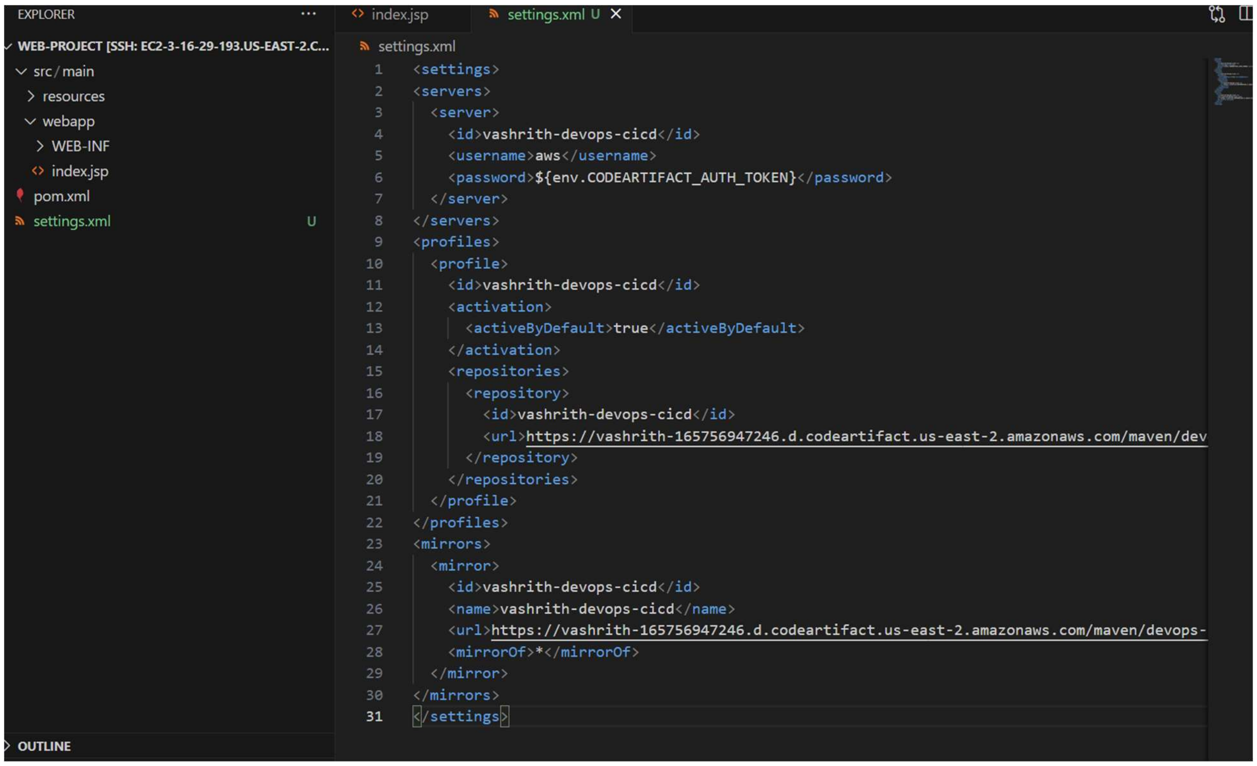
Click on view connection instructions in our repository details page. Then, make sure to select the OS of your EC2 and mvn for the package manager client. There should be commands displayed on how to connect maven to the CodeArtifact repository.



```
[ec2-user@ip-172-31-33-70 ~]$ export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain vashrith --domain-owner 165756947246 --r
egion us-east-2 --query authorizationToken --output text`
[ec2-user@ip-172-31-33-70 ~]$ 
```

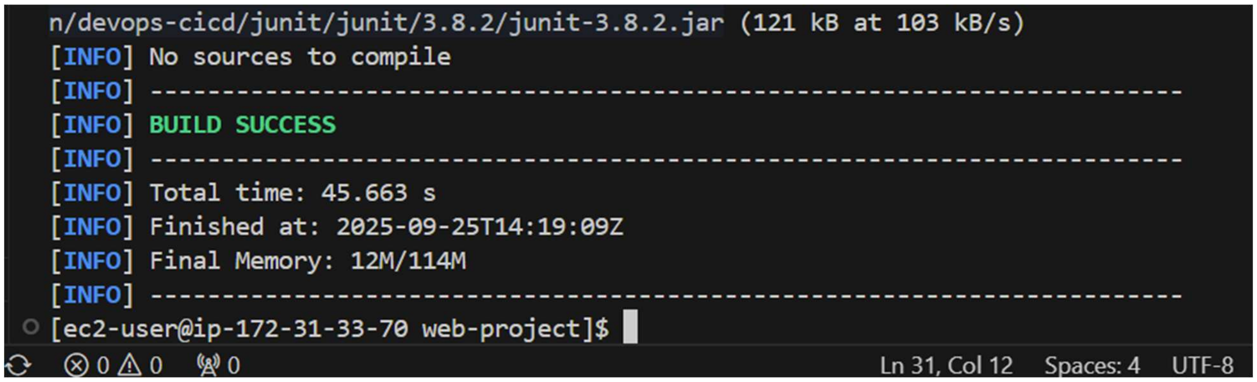
See Packages in CodeArtifact

First, we need to create a settings.xml file at the root of the project directory. This file will store the settings directing Maven on how to behave across all projects by showing where to find the dependencies and how to gain access to the needed repositories. Add a settings tag to the file and from the connection instructions tab from earlier, add the code in steps 4, 5, and 6.



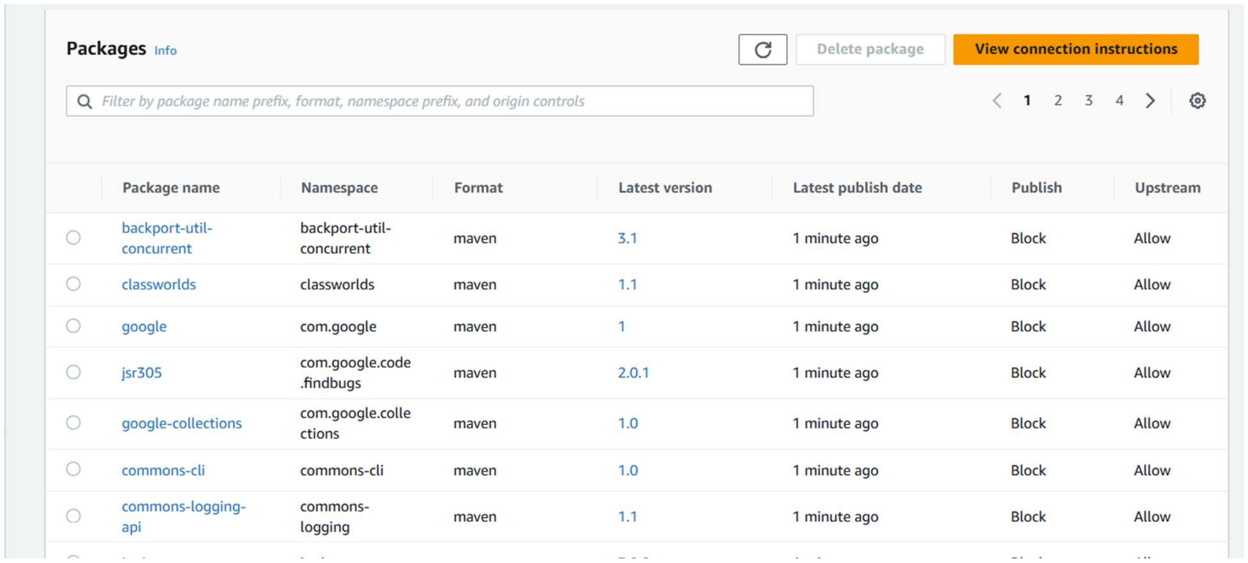
```xml
<settings>
<servers>
  <server>
    <id>vashrith-devops-cicd</id>
    <username>aws</username>
    <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
  </server>
</servers>
<profiles>
  <profile>
    <id>vashrith-devops-cicd</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>vashrith-devops-cicd</id>
        <url>https://vashrith-165756947246.d.codeartifact.us-east-2.amazonaws.com/maven/dev
      </repository>
    </repositories>
  </profile>
</profiles>
<mirrors>
  <mirror>
    <id>vashrith-devops-cicd</id>
    <name>vashrith-devops-cicd</name>
    <url>https://vashrith-165756947246.d.codeartifact.us-east-2.amazonaws.com/maven/devops-
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>
</settings>
```

Now it’s time to compile the code with the command ***mvn -s settings.xml compile.***



```
n/devops-cicd/junit/junit/3.8.2/junit-3.8.2.jar (121 kB at 103 kB/s)
[INFO] No sources to compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 45.663 s
[INFO] Finished at: 2025-09-25T14:19:09Z
[INFO] Final Memory: 12M/114M
[INFO] ------------------------------------------------------------------------
[ec2-user@ip-172-31-33-70 web-project]$
```

Now if we look at the CodeArtifact console, we can see a list of Maven packages.



Packages Info

Filter by package name prefix, format, namespace prefix, and origin controls

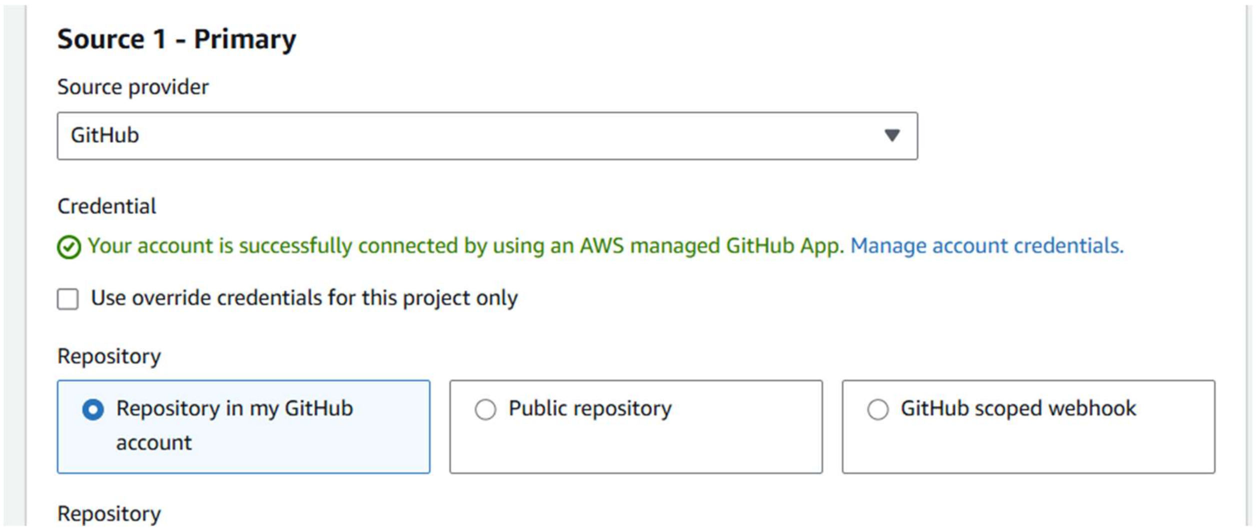
Package name	Namespace	Format	Latest version	Latest publish date	Publish	Upstream
backport-util-concurrent	backport-util-concurrent	maven	3.1	1 minute ago	Block	Allow
classworlds	classworlds	maven	1.1	1 minute ago	Block	Allow
google	com.google	maven	1	1 minute ago	Block	Allow
jsr305	com.google.code.findbugs	maven	2.0.1	1 minute ago	Block	Allow
google-collections	com.google.collections	maven	1.0	1 minute ago	Block	Allow
commons-cli	commons-cli	maven	1.0	1 minute ago	Block	Allow
commons-logging-api	commons-logging	maven	1.1	1 minute ago	Block	Allow

Stage 4

Create a CodeBuild Project and Connect it to the GitHub Repository

CodeBuild is a tool that takes source code, compiles it, runs tests, and packages it. This removes the manual need to set up build servers. A CodeBuild Project is the setup for the Continuous Integration (CI) process. We provide AWS with a blueprint on how to the application with the location of the code, the environment needed, what commands to run, and where to store the results.

In AWS, navigate to CodeBuild, select Build Projects and configure the settings. Now it’s time to connect it to GitHub using AWS CodeConnection. Select create a new GitHub connection and link your GitHub by following the directions.


Source 1 - Primary

Source provider

GitHub

Credential

Your account is successfully connected by using an AWS managed GitHub App. Manage account credentials.

☐ Use override credentials for this project only

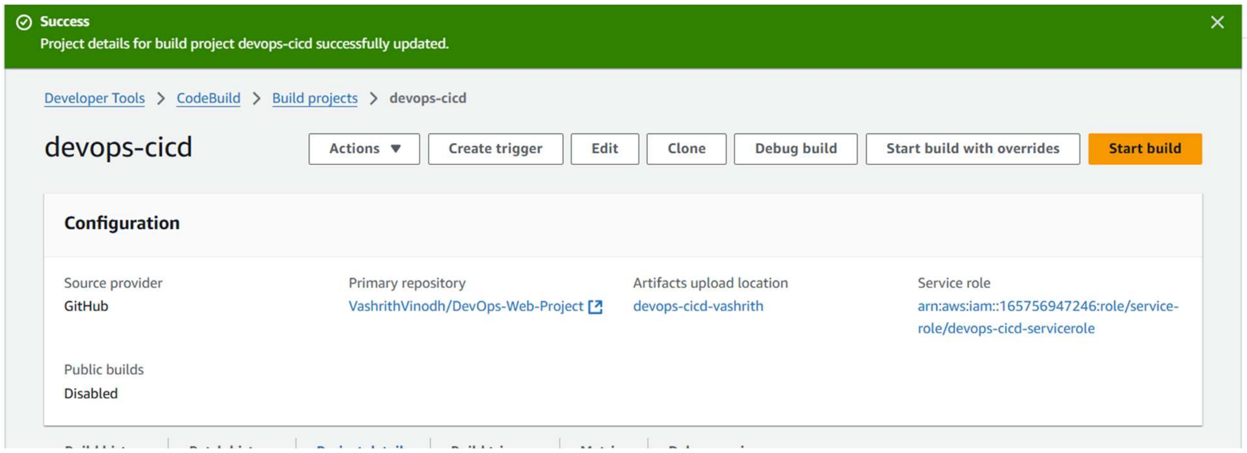
Repository

● Repository in my GitHub account ○ Public repository ○ GitHub scoped webhook

Repository

It should be possible to connect to the repository now. Configure the settings for the Environment and BuildSpec. Head over to the S3 console and create a bucket. Head back to the Build Project console and select the newly created bucket. Scroll down to logs and make sure CloudWatch Logs is selected, then set a name and hit create build project.

CloudWatch Logs is an AWS tool that monitors and collects all the data on the AWS services. This can be a useful tool for debugging.


Success
Project details for build project devops-cicd successfully updated.

Developer Tools > CodeBuild > Build projects > devops-cicd

devops-cicd

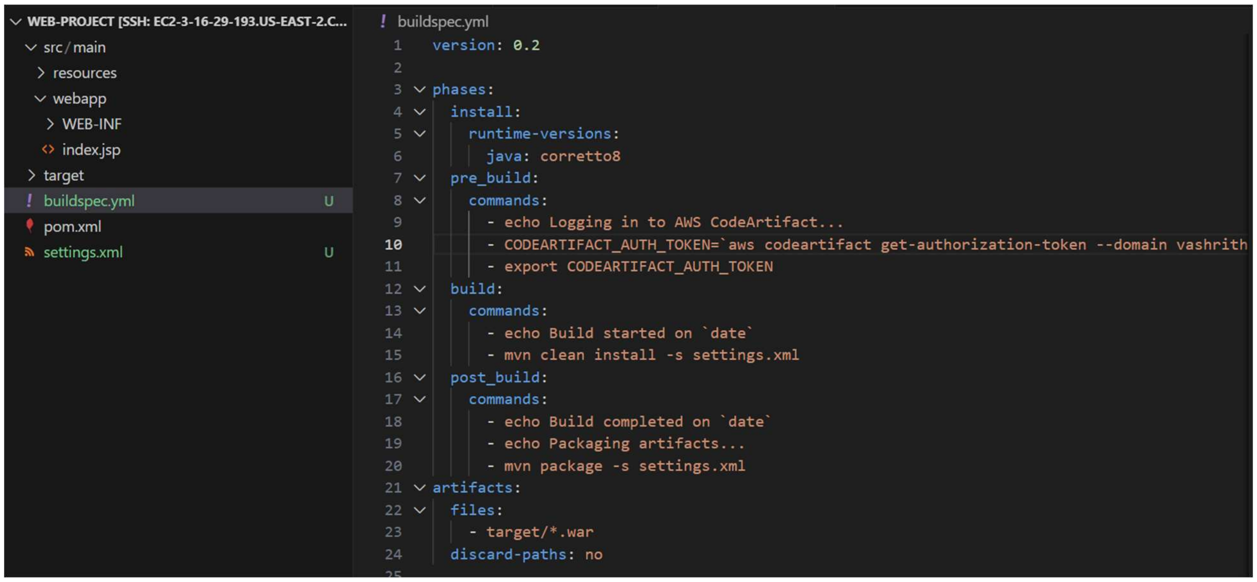
Actions | Create trigger | Edit | Clone | Debug build | Start build with overrides | Start build

Configuration

Source provider GitHub	Primary repository VashrithVinodh/DevOps-Web-Project	Artifacts upload location devops-cicd-vashrith	Service role arn:aws:iam::165756947246:role/service-role/devops-cicd-servicerole
Public builds Disabled			

Running the Build

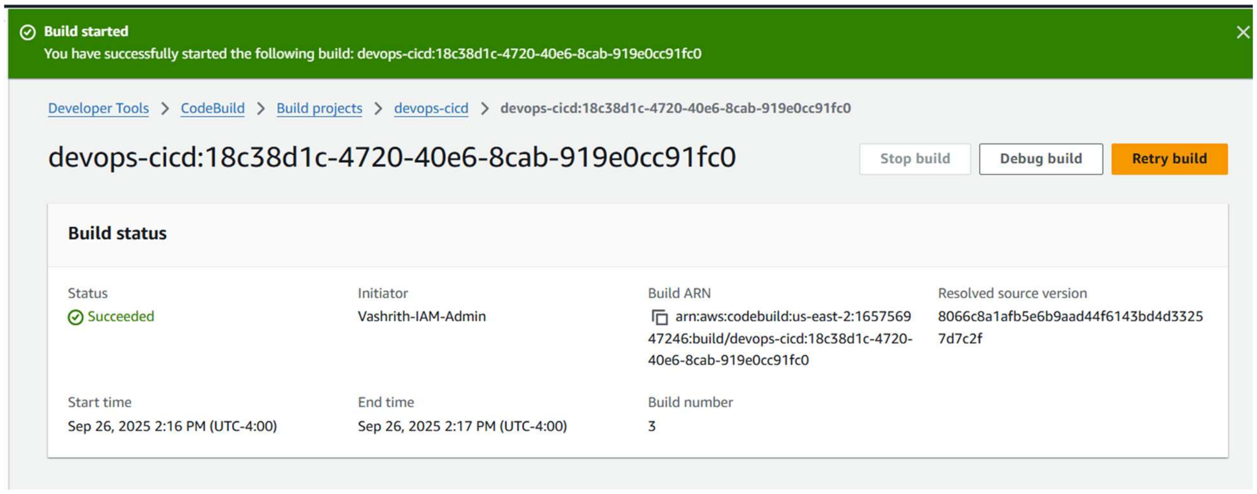
Before we can start the build, we need to create the buildspec.yml file in the root directory. This file will contain the essentials on how to build the Java web app, the version, and the following phases: install, pre-build, build, and post-build. Closing off with artifacts, which files to save from the output. The, push the files to the GitHub repo.



```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto8
  pre_build:
    commands:
      - echo Logging in to AWS CodeArtifact...
      - CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain vashrith
      - export CODEARTIFACT_AUTH_TOKEN
  build:
    commands:
      - echo Build started on `date`
      - mvn clean install -s settings.xml
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Packaging artifacts...
      - mvn package -s settings.xml
artifacts:
  files:
    - target/*.war
  discard-paths: no
```

Now we need to grant permissions for the build to access files, this can be done in the IAM console using roles. Now we should be able to run the build successfully.


Developer Tools > CodeBuild > Build projects > devops-cicd > devops-cicd:18c38d1c-4720-40e6-8cab-919e0cc91fc0
Stop build
Debug build
Retry build

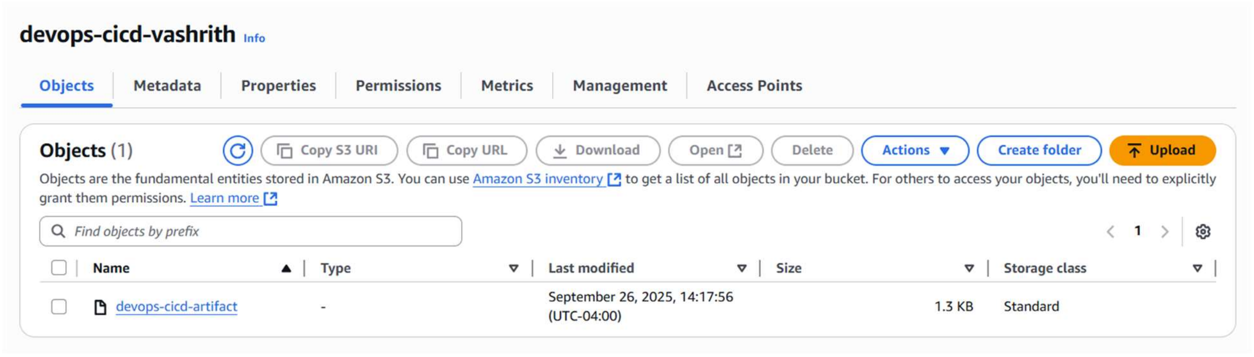
Build started
You have successfully started the following build: devops-cicd:18c38d1c-4720-40e6-8cab-919e0cc91fc0

devops-cicd:18c38d1c-4720-40e6-8cab-919e0cc91fc0

Build status

Status	Initiator	Build ARN	Resolved source version
Succeeded	Vashrith-IAM-Admin	arn:aws:codebuild:us-east-2:165756947246:build/devops-cicd:18c38d1c-4720-40e6-8cab-919e0cc91fc0	8066c8a1afb5e6b9aad44f6143bd4d33257d7c2f
Start time	**End time**	**Build number**	
Sep 26, 2025 2:16 PM (UTC-4:00)	Sep 26, 2025 2:17 PM (UTC-4:00)	3	

We can verify the build by looking at the S3 buckets console.


devops-cicd-vashrith Info

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

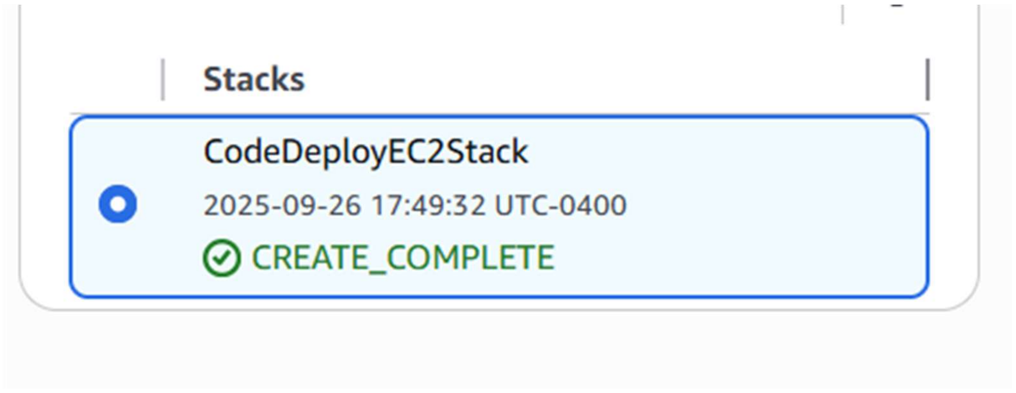
Name	Type	Last modified	Size	Storage class
devops-cicd-artifact	-	September 26, 2025, 14:17:56 (UTC-04:00)	1.3 KB	Standard

Stage 5

Launching an EC2 with CloudFormation

CouldFormation is an infrastructure as code tool (IaC), template that describes the EC2 instances, security groups, databases, and other things. IaC is a type of software that lets you write code to create servers, networks, and other infrastructure, essentially a script.

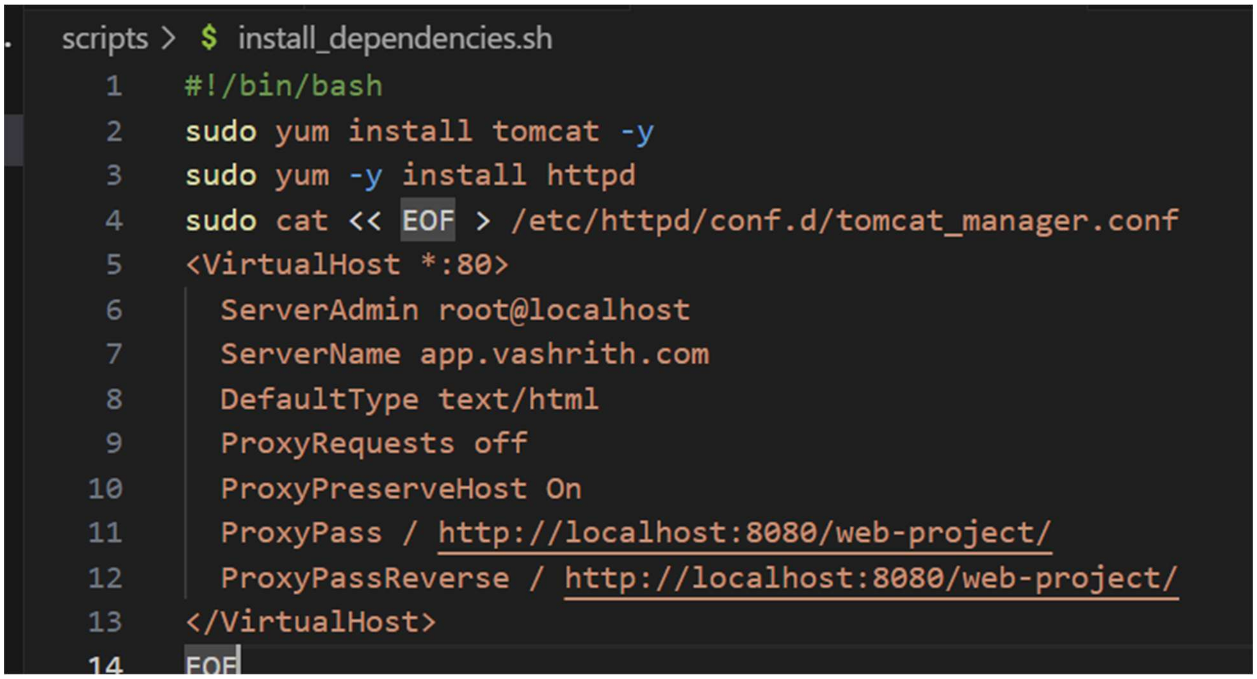
In the CloudFormation Console, create a new stack and configure the settings.


Stacks

CodeDeployEC2Stack
2025-09-26 17:49:32 UTC-0400
CREATE_COMPLETE

Preparing Deployment Scripts and App Specs

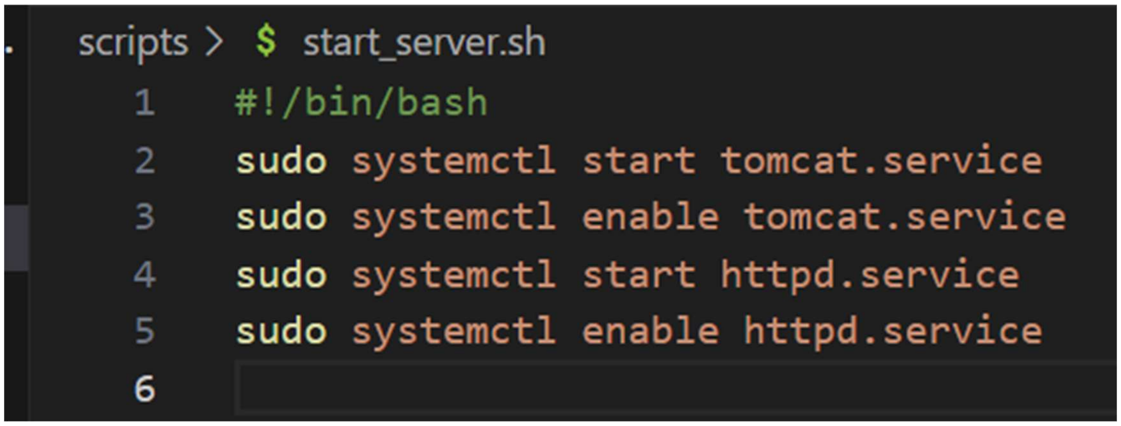
Create a scripts folder in the root of the project directory and write the following script in install_dependencies.sh.



```bash
#!/bin/bash
sudo yum install tomcat -y
sudo yum -y install httpd
sudo cat << EOF > /etc/httpd/conf.d/tomcat_manager.conf
<VirtualHost *:80>
  ServerAdmin root@localhost
  ServerName app.vashrith.com
  DefaultType text/html
  ProxyRequests off
  ProxyPreserveHost On
  ProxyPass / http://localhost:8080/web-project/
  ProxyPassReverse / http://localhost:8080/web-project/
</VirtualHost>
EOF
```

This script sets up the software needed to run the website by installing the programs that handle internet traffic and host the application. The it creates settings to let these programs work together.

Create a new script start_server.sh with the following code.

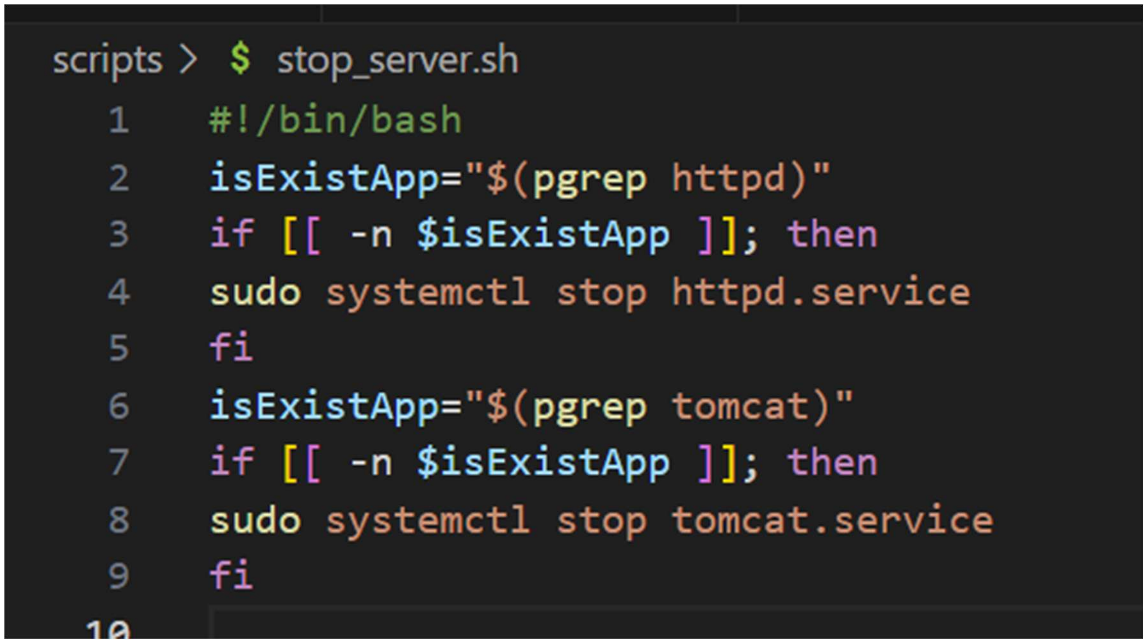


```bash
#!/bin/bash
sudo systemctl start tomcat.service
sudo systemctl enable tomcat.service
sudo systemctl start httpd.service
sudo systemctl enable httpd.service

```

This script starts tomcat (Java application server) and Apache (web server) and makes sure they restart if the EC2 instance reboots.

Create a new script stop_server.sh with the following code.

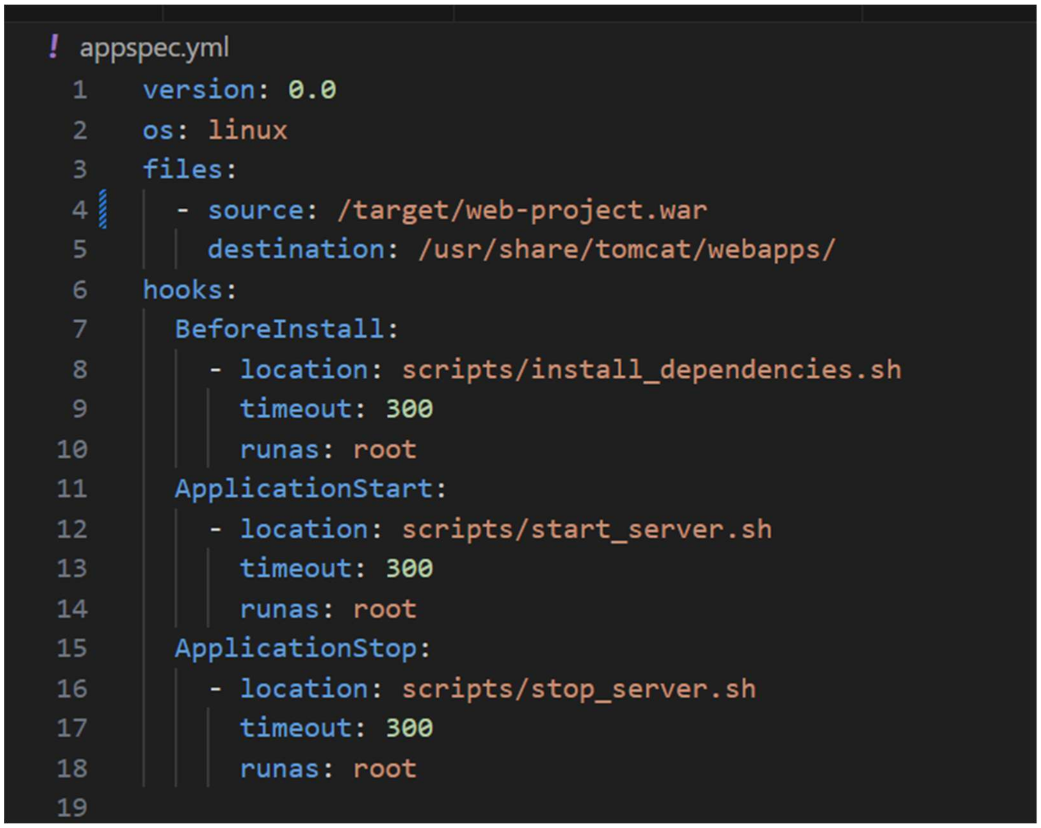


scripts > $ stop_server.sh

```bash
#!/bin/bash
isExistApp="$(pgrep httpd)"
if [[ -n $isExistApp ]]; then
sudo systemctl stop httpd.service
fi
isExistApp="$(pgrep tomcat)"
if [[ -n $isExistApp ]]; then
sudo systemctl stop tomcat.service
fi
```

This script ensure that the web server is safely stopped by checking if they are running, done by using pgrep.

Now is the root of the project, create the file appspec.yml with the following code.

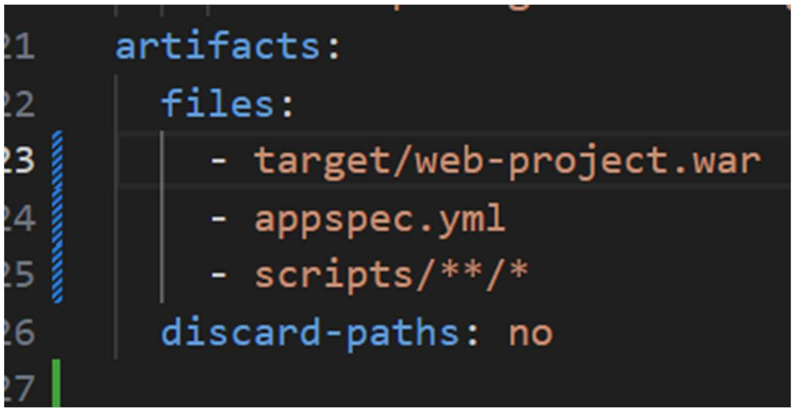


! appspec.yml

```yaml
version: 0.0
os: linux
files:
  - source: /target/web-project.war
    destination: /usr/share/tomcat/webapps/
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

This file essentially acts as a manual for CodeDeploy.

Modify the artifacts section in the buildspec.yml file with the following code.

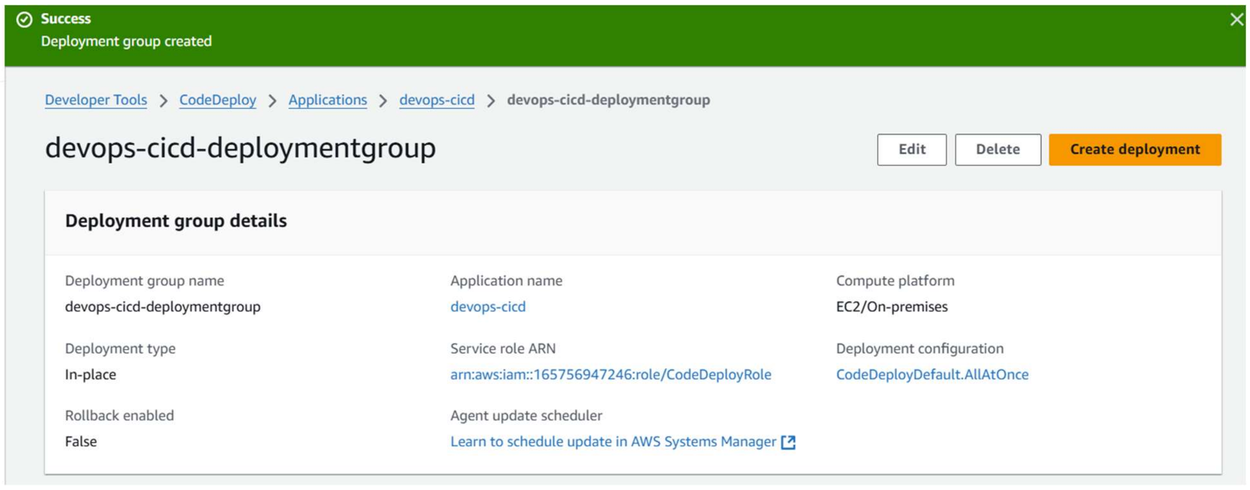


```
artifacts:
  files:
    - target/web-project.war
    - appspec.yml
    - scripts/**/*
  discard-paths: no
```

Push the changes to GitHib.

Setting up CodeDeploy

Head to CodeDeploy and create an application, then head to the IAM console and create a new role with CodeDeploy as the service and use case. Now head to CodeDeploy again and create a deployment group with the new rule.



Success
Deployment group created

Developer Tools > CodeDeploy > Applications > devops-cicd > devops-cicd-deploymentgroup

devops-cicd-deploymentgroup

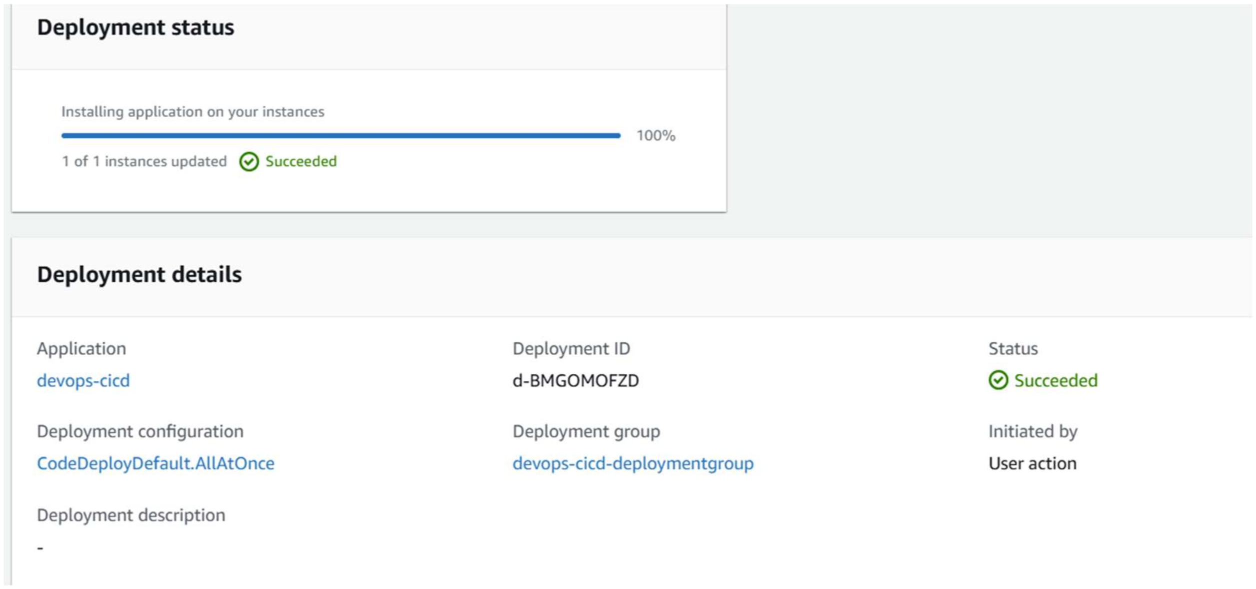
Deployment group details

Deployment group name devops-cicd-deploymentgroup	Application name devops-cicd	Compute platform EC2/On-premises
Deployment type In-place	Service role ARN arn:aws:iam::165756947246:role/CodeDeployRole	Deployment configuration CodeDeployDefault.AllAtOnce
Rollback enabled False	Agent update scheduler Learn to schedule update in AWS Systems Manager	

Create and Verify Deployment

Head to the S3 bucket and copy the S3 url. Then hit create a new deployment and paste the url in the Revision location, the revision file type will be .zip. The revision location is where

CodeDeploy will look to try and find the application's build artifacts. Before we can deploy successfully, we need to run the CodeBuild.



Deployment status

Installing application on your instances

100%

1 of 1 instances updated ✓ Succeeded

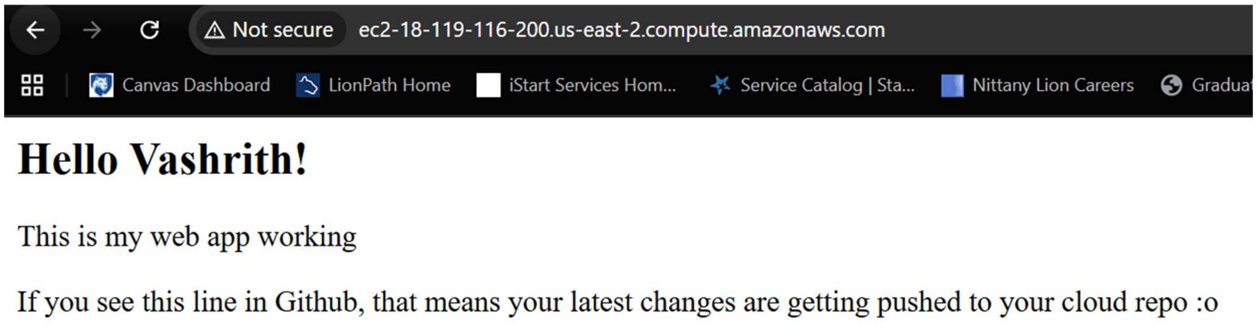
Deployment details

Application	Deployment ID	Status
devops-cicd	d-BMGOMOFZD	✓ Succeeded
Deployment configuration	**Deployment group**	**Initiated by**
CodeDeployDefault.AllAtOnce	devops-cicd-deploymentgroup	User action

Deployment description

-

Now if we open a web page with our EC2 instance with CloudFormation's Public IPv4 DNS, we can see the webpage.



Hello Vashrith!

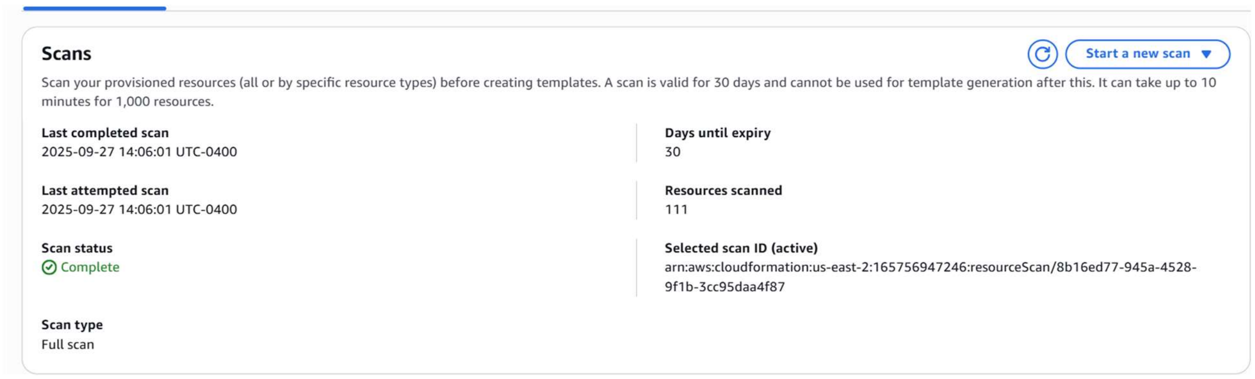
This is my web app working

If you see this line in Github, that means your latest changes are getting pushed to your cloud repo :o

Stage 6

Generating and Launching a CloudFormation Template

Head to IaC generator in the CloudFormation console and complete a scan.


Scans

Start a new scan

Scan your provisioned resources (all or by specific resource types) before creating templates. A scan is valid for 30 days and cannot be used for template generation after this. It can take up to 10 minutes for 1,000 resources.

Last completed scan
2025-09-27 14:06:01 UTC-0400

Last attempted scan
2025-09-27 14:06:01 UTC-0400

Scan status
Complete

Scan type
Full scan

Days until expiry
30

Resources scanned
111

Selected scan ID (active)
arn:aws:cloudformation:us-east-2:165756947246:resourceScan/8b16ed77-945a-4528-9f1b-3cc95daa4f87

Create and configure a template with the scanned resources, make sure to download the template. Now delete all the individual instances of the resources to get rid of overlap. Now it’s time to launch the CloudFormation template. Create a new stack and upload the template file, then submit it. This should automatically deploy everything that this project has deployed manually up until this point.