

TECHNOLOGY

DATA SCIENCE



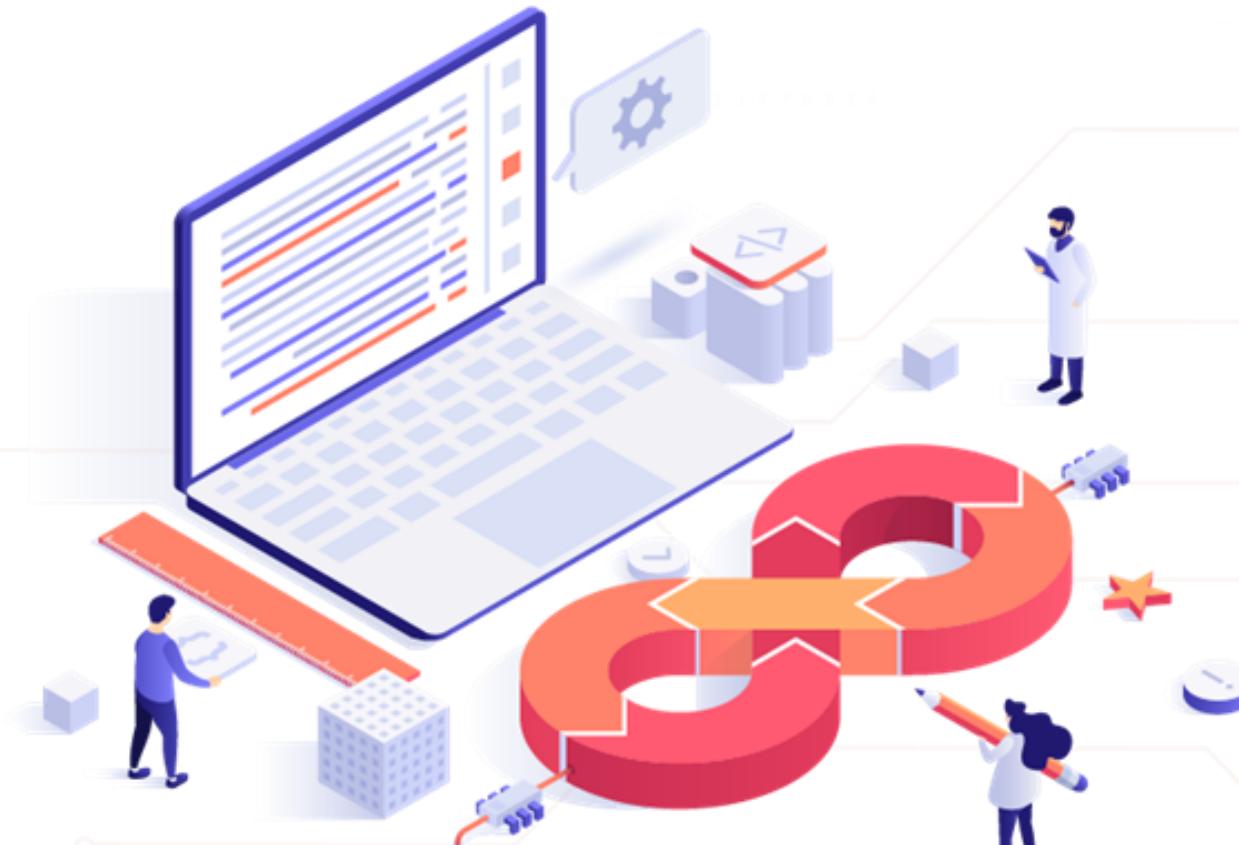
Caltech

Center for Technology & Management Education

Configure AWS for Back End

TECHNOLOGY

DATA SCIENCE
PROGRAMMING



Caltech

Center for Technology & Management Education

Set Up AWS for Java Back End

You Already Know

Before we begin, let's recall what we have covered till now:



Agile



Git



SQL



Angular



HTML



CSS



JavaScript



CoreJava

You Already Know

Before we begin, let's recall what we have covered till now:



JDBC



JSP



Servlets



MongoDB



Maven

You Already Know

Before we begin, let's recall what we have covered till now:

JUnit



JUnit

Spring

Spring Boot



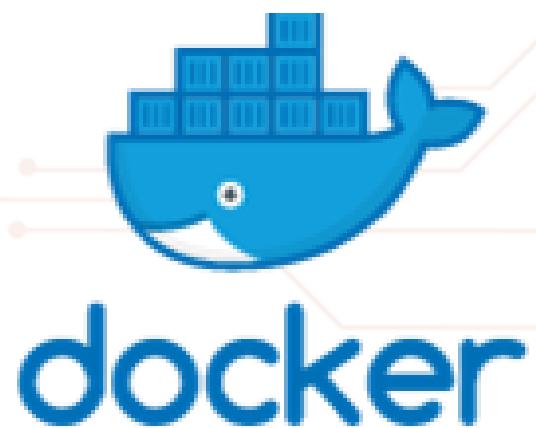
Microservices

Webservices

Microservices

You Already Know

Before we begin, let's recall what we have covered till now:



Docker



Jenkins



AWS

Front-End Backend Communication

- Used HTTP Client in Angular to communicate with Java back end
- Implemented communication for both admin and end-user projects

Jenkins Pipeline

- Used Jenkins build automation server to build angular apps and Java back end
- Created Pipeline with multiple stages



Docker

- Used Docker to build images for Angular apps and Java back end
- Used Jenkins to dockerize the Angular apps and Java back end with Jenkins

Configure Angular Apps on EC2

- Created and configured EC2 instance for Angular apps
- Ran Jenkins pipeline to dockerize the Angular apps and deploy them



A Day in the Life of a Full Stack Developer

As a full stack web developer, our key role is to develop both client and server software.



Angular and Node can be used to build front end of the web page.



Spring Boot, Java, and MySQL or MongoDB can be used to build at the back end.



A Day in the Life of a Full Stack Developer

Bob needs to configure AWS for Front end. He brainstorms a bit and finds a solution.

Let me use Jenkins, Docker, and AWS to build CI/CD Pipelines, containerize the apps, and finally host them to AWS EC2 instance.



In this lesson, we will host the Java Backend app on EC2 instance, use Jenkins and Docker as DevOps tools to make it live and help Bob to complete his task effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the steps to create and configure EC2 instance on AWS
- Discuss the steps to create MySQL RDS database on AWS
- Understand the use Jenkins on EC2 for CI/CD
- Discuss how to integrate Docker in Jenkins to build and release the images as containers on EC2



Create and Configure EC2 in AWS

Amazon Web Services

Amazon Web Services (AWS) is a cloud platform, which offers more than 200 fully featured services.



Angular web apps on EC2 instance is deployed using Jenkins and Docker.

Create EC2 Instance

Open the EC2 dashboard.

The screenshot shows the AWS EC2 Instances dashboard. On the left, there is a sidebar with various navigation options. The 'EC2 Dashboard' option is highlighted with a red box. The main area displays a table with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP. A message at the top of the table says, "You do not have any instances in this region". At the bottom of the main area, there is a section titled "Select an instance".

Launch Instance

Launch a new instance.

The screenshot shows the AWS EC2 Instances page. The URL in the browser bar is `console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances:`. The left sidebar shows navigation options like EC2 Dashboard, EC2 Global View, Events, Tags, Limits, Instances, Instances (New), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances (New), Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images, Elastic Block Store, Network & Security, Security Groups, Elastic IPs, and Placement Groups. The main content area displays 'Instances (1/2)' with a table. The table has columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Pub. Two instances are listed: 'BackEndEC2' (Instance ID: i-04a7ae5c9bae70b61, State: Running, Type: t2.micro) and 'FrontEndEC2' (Instance ID: i-0b2d99ad1267484be, State: Running, Type: t2.micro). A red box highlights the first instance. A green callout box with the text 'Use the AWS Cloud Shell to perform the tasks.' points to the instance table. Another green callout box with the text 'Create service group and key pair to access the instance from terminal.' points to the instance details. The instance details page for 'BackEndEC2' shows tabs for Details, Security, Networking, Storage, Status checks, Monitoring, and Tags. Under Details, there's an 'Instance summary' section with fields for Instance ID (i-04a7ae5c9bae70b61), Public IPv4 address (3.80.74.39), Private IPv4 addresses (172.31.21.117), Instance state (Running), Public IPv4 DNS (ec2-3-80-74-39.compute-1.amazonaws.com), and Private IP DNS name (ip-172-31-21-117.ec2.internal).

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Pub
BackEndEC2	i-04a7ae5c9bae70b61	Running	t2.micro	2/2 checks passed	No alarms	us-east-1d	ec2-
FrontEndEC2	i-0b2d99ad1267484be	Running	t2.micro	2/2 checks passed	No alarms	us-east-1c	ec2-

Instance: i-04a7ae5c9bae70b61

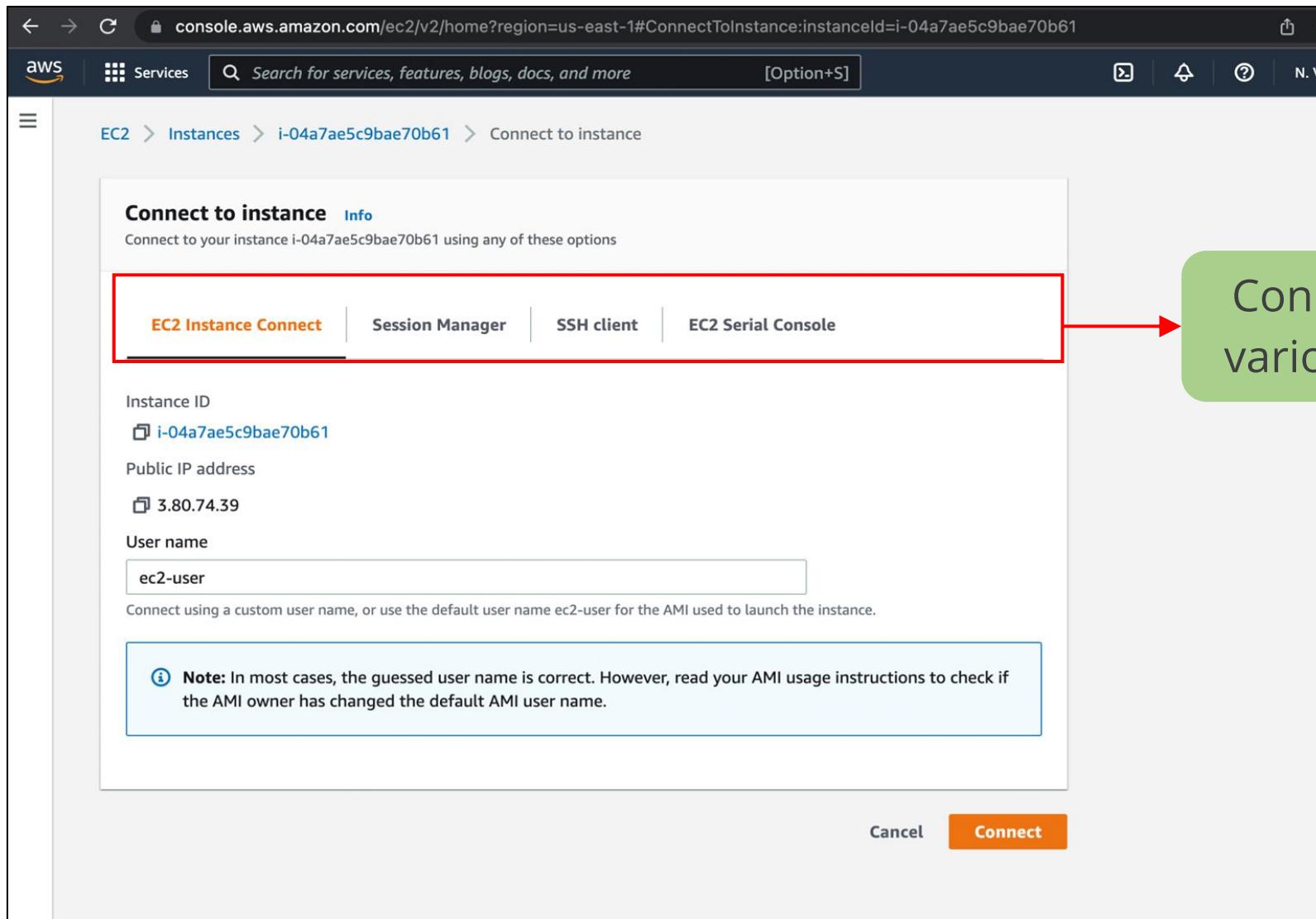
Details Security Networking Storage Status checks Monitoring Tags

Instance summary

Instance ID i-04a7ae5c9bae70b61	Public IPv4 address 3.80.74.39 open address	Private IPv4 addresses 172.31.21.117
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-3-80-74-39.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-21-117.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-21-117.ec2.internal	Answer private resource DNS name -

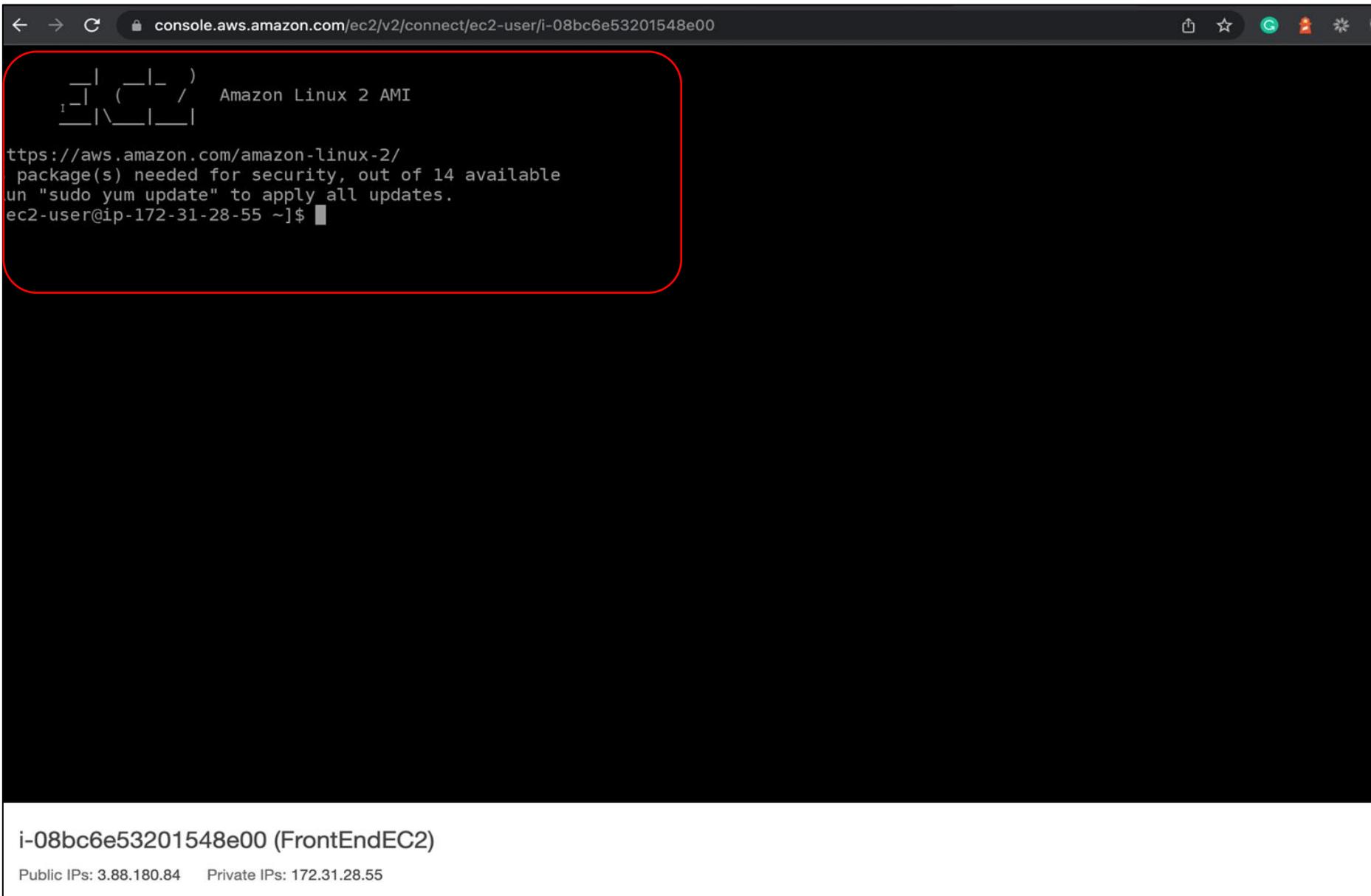
Connect to EC2 Instance

Connect through AWS Cloud Shell, which is the first option on clicking connect, that is EC2 Instance Connect.



Launch the EC2 Shell and Configure the Tools

Install the tools and software required to deploy Angular apps and configure Node, Git, Docker, and Jenkins.



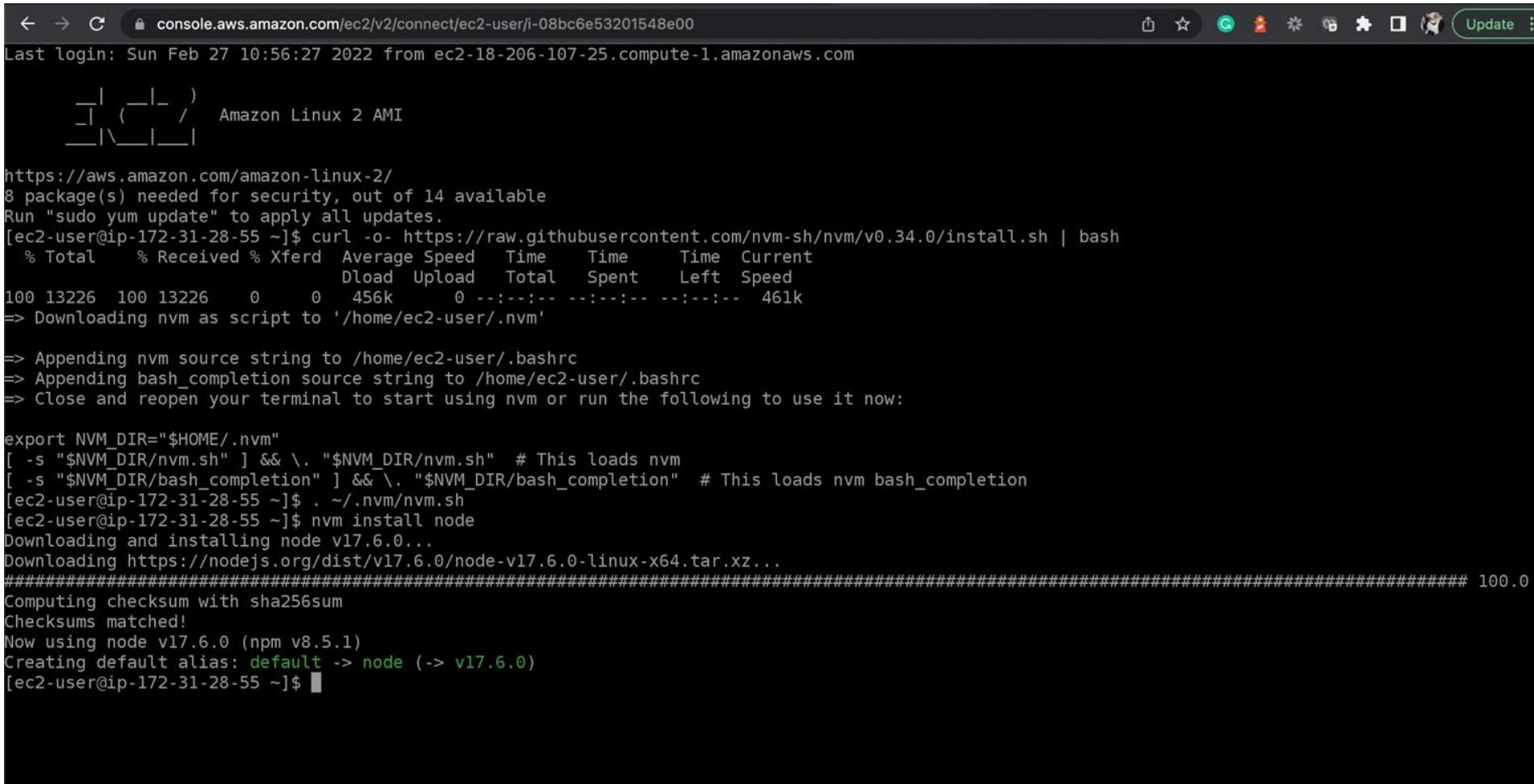
A screenshot of a terminal window from the AWS CloudShell interface. The URL in the address bar is `console.aws.amazon.com/ec2/v2/connect/ec2-user/i-08bc6e53201548e00`. The terminal shows the following output:

```
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
package(s) needed for security, out of 14 available
run "sudo yum update" to apply all updates.
ec2-user@ip-172-31-28-55 ~]$
```

The terminal window has a red rounded rectangle highlighting the first few lines of the output. At the bottom of the terminal window, there is a footer bar with the text "i-08bc6e53201548e00 (FrontEndEC2)" and "Public IPs: 3.88.180.84 Private IPs: 172.31.28.55".

Configure Node on EC2

Configure the Jenkins Pipeline Project by passing GitHub repository URL.



The screenshot shows a terminal window on an Amazon Linux 2 AMI instance. The user is running a Jenkins pipeline script to install Node.js using the nvm (Node Version Manager) tool. The process involves curling the nvm installation script from GitHub, appending it to the .bashrc file, installing node, and finally creating a default alias for node. The terminal output includes progress bars for file downloads and checksum verification.

```
Last login: Sun Feb 27 10:56:27 2022 from ec2-18-206-107-25.compute-1.amazonaws.com
[ec2-user@ip-172-31-28-55 ~]$ curl -o https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
  % Total    % Received % Xferd  Average Speed   Time     Time  Current
          Dload  Upload   Total Spent    Left  Speed
100 13226  100 13226    0      0  456k      0 --:--:-- --:--:-- 461k
=> Downloading nvm as script to '/home/ec2-user/.nvm'

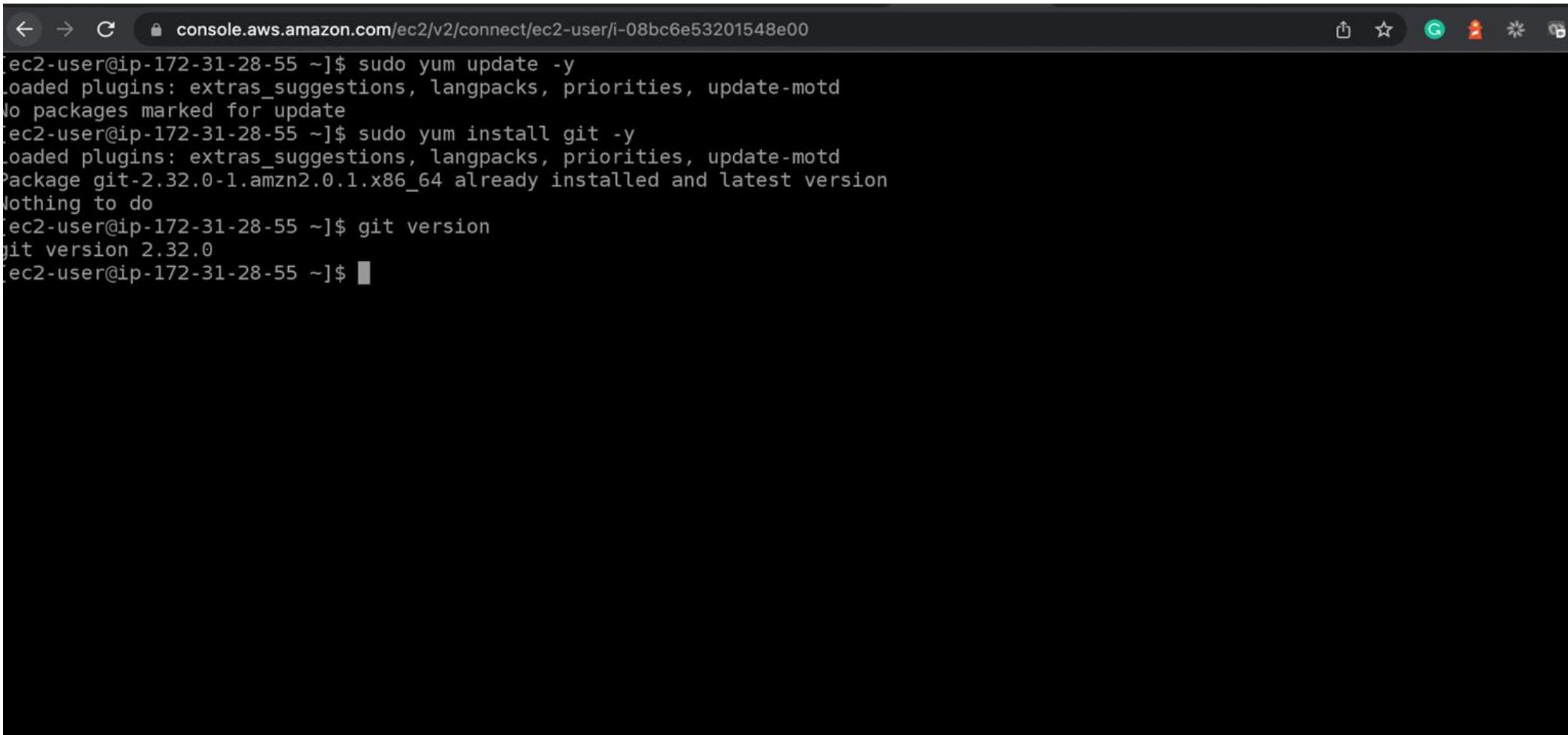
=> Appending nvm source string to /home/ec2-user/.bashrc
=> Appending bash_completion source string to /home/ec2-user/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
[ec2-user@ip-172-31-28-55 ~]$ . ~/.nvm/nvm.sh
[ec2-user@ip-172-31-28-55 ~]$ nvm install node
Downloading and installing node v17.6.0...
Downloaded https://nodejs.org/dist/v17.6.0/node-v17.6.0-linux-x64.tar.xz...
Computing checksum with sha256sum
Checksums matched!
Now using node v17.6.0 (npm v8.5.1)
Creating default alias: default -> node (-> v17.6.0)
[ec2-user@ip-172-31-28-55 ~]$
```

Reference link on AWS documentation: <https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html>

Configure Git on EC2 Instance

Configure Git on EC2 Instance.



The screenshot shows a terminal window from the AWS CloudShell interface. The URL in the address bar is `console.aws.amazon.com/ec2/v2/connect/ec2-user/i-08bc6e53201548e00`. The terminal session is as follows:

```
ec2-user@ip-172-31-28-55 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
ec2-user@ip-172-31-28-55 ~]$ sudo yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package git-2.32.0-1.amzn2.0.1.x86_64 already installed and latest version
Nothing to do
ec2-user@ip-172-31-28-55 ~]$ git version
git version 2.32.0
ec2-user@ip-172-31-28-55 ~]$ █
```

Configure Java on EC2 Instance

Use the following commands to install Java:

Install JDK 8

- sudo yum update -y
- sudo yum install java-1.8.0-openjdk

Install JDK 11 - Java

- sudo amazon-linux-extras install java-openjdk11

Configure Java on EC2 Instance

Use the following commands to install Java:

Install JDK 11 Development - javac

- sudo yum install java-11-openjdk-devel

Configure Java Version 8 or 11

- sudo alternatives --config java

Configure Java on EC2 Instance

The screenshot shows a terminal window titled "console.aws.amazon.com/ec2/v2/connect/ec2-user/i-04a7ae5c9bae70b61". The user runs the command `sudo alternatives --config java`. The output shows two Java installations:

Selection	Command
*+ 1	java-1.8.0-openjdk.x86_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b07-1.amzn2.0.2.x86_64/jre/bin/java)
2	java-11-openjdk.x86_64 (/usr/lib/jvm/java-11-openjdk-11.0.13.0.8-1.amzn2.0.3.x86_64/bin/java)

The user selects option 2 and enters "2" at the prompt. The terminal then displays the output of `java -version`, which shows:

```
openjdk version "11.0.13" 2021-10-19 LTS  
OpenJDK Runtime Environment 18.9 (build 11.0.13+8-LTS)  
OpenJDK 64-Bit Server VM 18.9 (build 11.0.13+8-LTS, mixed mode, sharing)
```

It also shows the output of `javac -version`:

```
javac 11.0.13
```

Configure Docker on EC2 Instance

Now, install Docker for containerizing the Angular apps. Here are some useful commands:

1

Install Docker:
sudo yum install docker

2

Add the ec2-user to the docker group to execute Docker commands without using sudo, exit the terminal, and re-login to make the change effective.

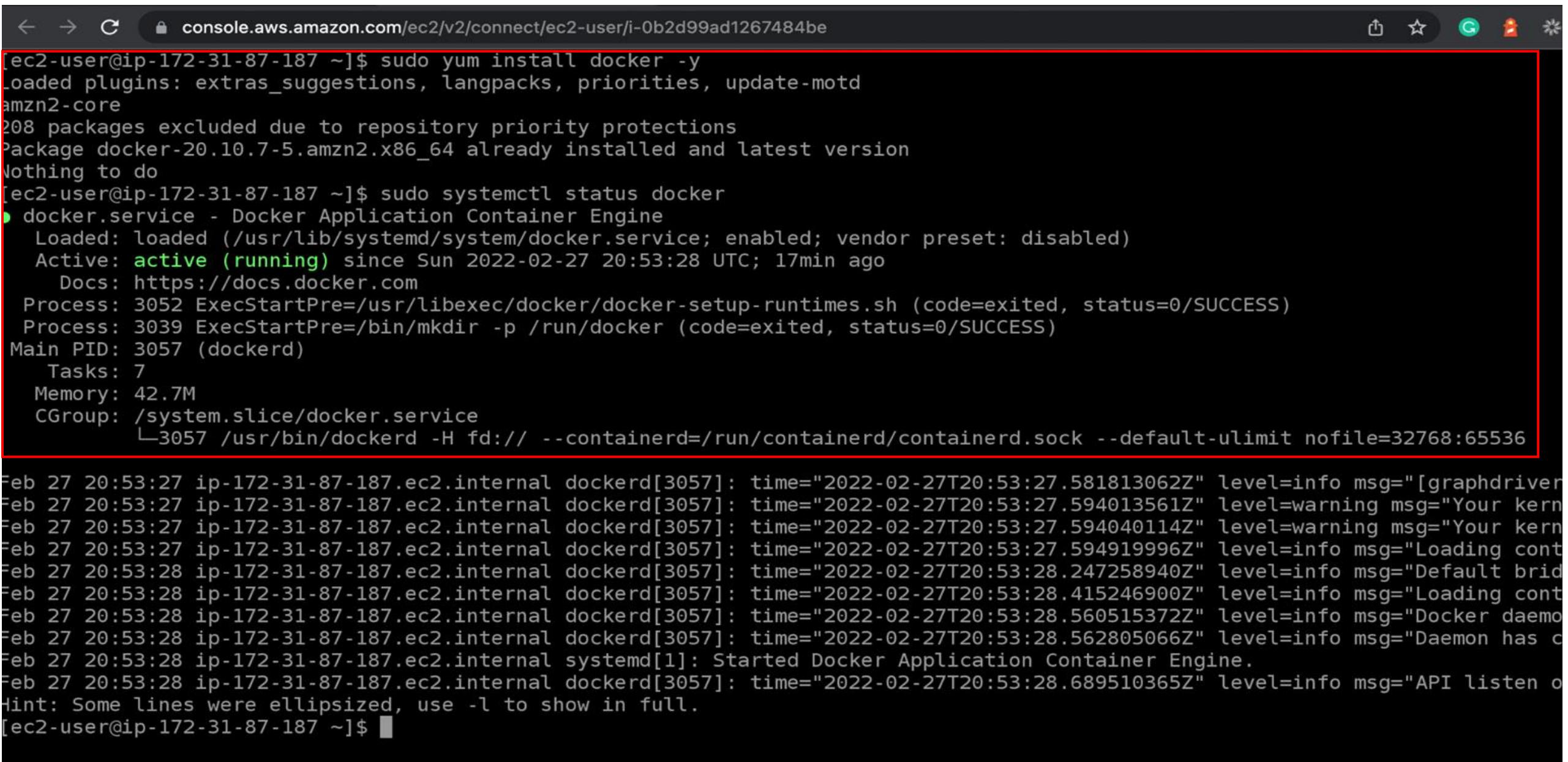
sudo usermod -a -G docker ec2-user
exit

Configure Docker on EC2 Instance

- 3
Enable docker service:
`sudo systemctl enable docker`
- 4
Start docker service:
`sudo systemctl start docker`
- 5
Check the docker service:
`sudo systemctl status docker`

Configure Docker on EC2 Instance

The image below shows the visual of the commands:



A screenshot of a terminal window titled "console.aws.amazon.com/ec2/v2/connect/ec2-user/i-0b2d99ad1267484be". The terminal displays the following command-line session:

```
[ec2-user@ip-172-31-87-187 ~]$ sudo yum install docker -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
208 packages excluded due to repository priority protections
Package docker-20.10.7-5.amzn2.x86_64 already installed and latest version
Nothing to do
[ec2-user@ip-172-31-87-187 ~]$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
  Active: active (running) since Sun 2022-02-27 20:53:28 UTC; 17min ago
    Docs: https://docs.docker.com
   Process: 3052 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Process: 3039 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 3057 (dockerd)
   Tasks: 7
  Memory: 42.7M
    CGroup: /system.slice/docker.service
            └─3057 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

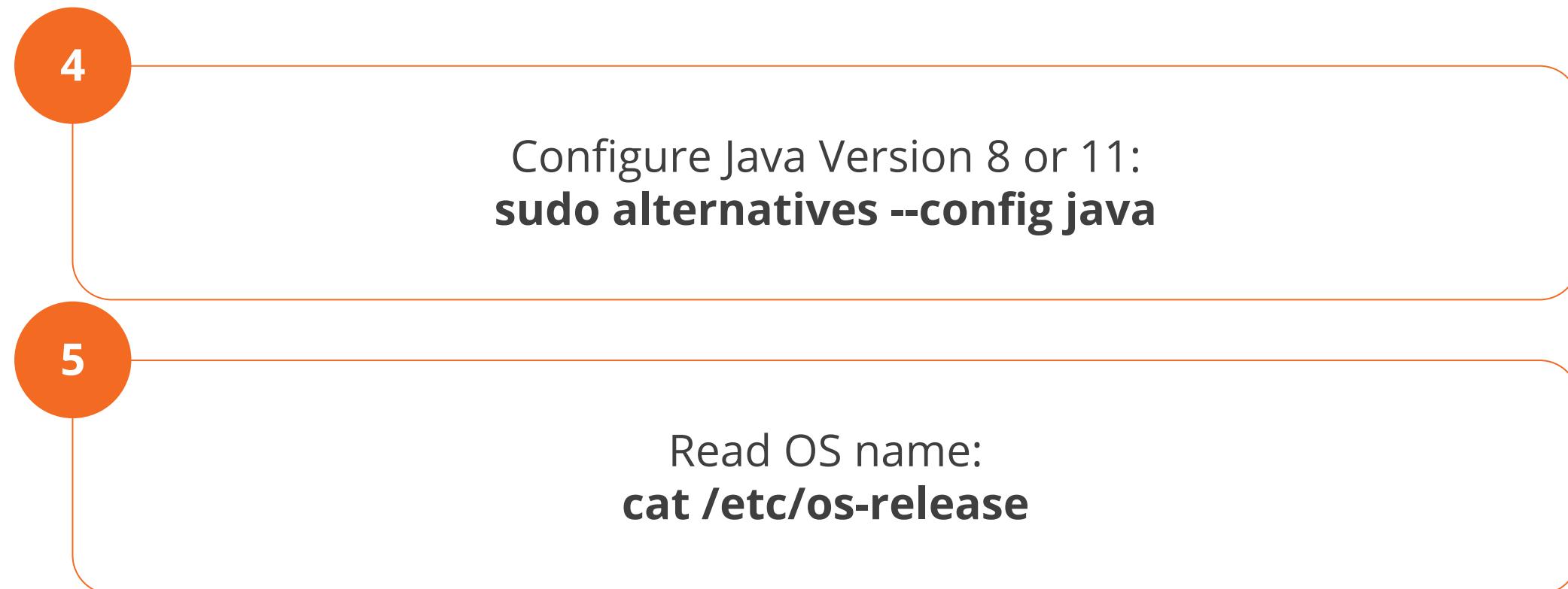
Feb 27 20:53:27 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:27.581813062Z" level=info msg="[graphdriver"
Feb 27 20:53:27 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:27.594013561Z" level=warning msg="Your kern
Feb 27 20:53:27 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:27.594040114Z" level=warning msg="Your kern
Feb 27 20:53:27 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:27.594919996Z" level=info msg="Loading cont
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:28.247258940Z" level=info msg="Default brid
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:28.415246900Z" level=info msg="Loading cont
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:28.560515372Z" level=info msg="Docker daemo
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:28.562805066Z" level=info msg="Daemon has c
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal systemd[1]: Started Docker Application Container Engine.
Feb 27 20:53:28 ip-172-31-87-187.ec2.internal dockerd[3057]: time="2022-02-27T20:53:28.689510365Z" level=info msg="API listen o
Hint: Some lines were ellipsized, use -l to show in full.
[ec2-user@ip-172-31-87-187 ~]$ █
```

Configure Jenkins on EC2 Instance

To configure Jenkins on EC2 instance, install Java first. Use the following commands:

- 1 Install JDK 8:
sudo yum update -y
sudo yum install java-1.8.0-openjdk
- 2 # Install JDK 11 – java:
sudo amazon-linux-extras install java-openjdk11
- 3 # Install JDK 11 Development – javac:
sudo yum install java-11-openjdk-devel

Configure Jenkins on EC2 Instance



Configure Jenkins on EC2 Instance

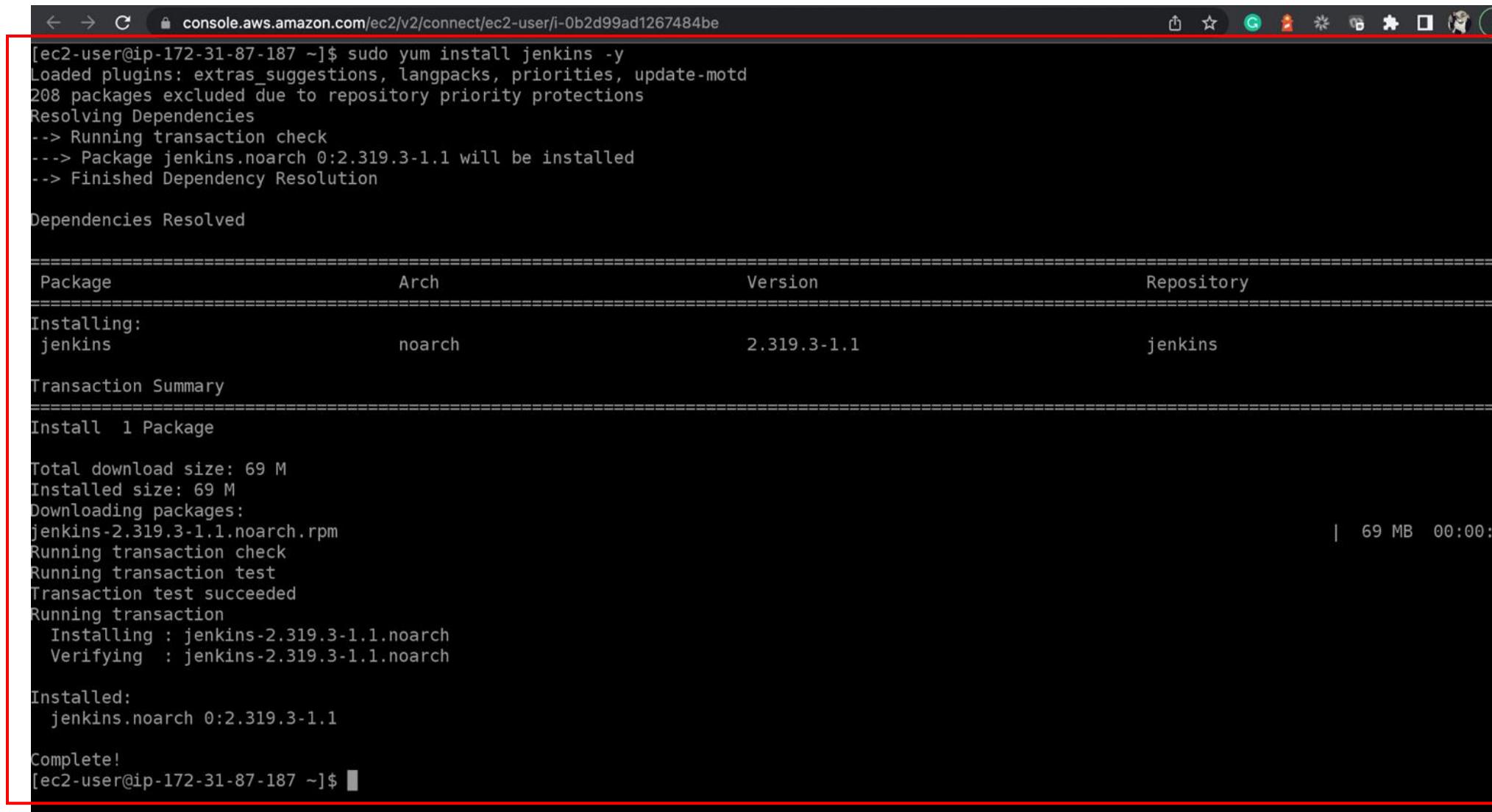
6

From Jenkins Installation document follow instructions:
[<https://www.jenkins.io/doc/book/installing/linux/>]

```
sudo amazon-linux-extras install epel
sudo yum-config-manager --enable epel
sudo yum install daemonize -y
sudo yum install jenkins -y
sudo systemctl daemon-reload
```

```
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
```

Configure Jenkins on EC2 Instance



The screenshot shows a terminal window on an AWS EC2 instance. The user has run the command `sudo yum install jenkins -y`. The output shows the package being downloaded and installed from the `jenkins` repository. The transaction summary indicates the installation of 1 package, totaling 69 MB. The process includes steps like running transaction check, transaction test, and installing the `jenkins-2.319.3-1.1.noarch.rpm` file.

```
[ec2-user@ip-172-31-87-187 ~]$ sudo yum install jenkins -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
208 packages excluded due to repository priority protections
Resolving Dependencies
--> Running transaction check
--> Package jenkins.noarch 0:2.319.3-1.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package           Arch      Version       Repository
=====
Installing:
jenkins          noarch   2.319.3-1.1  jenkins

Transaction Summary
=====
Install 1 Package

Total download size: 69 M
Installed size: 69 M
Downloading packages:
jenkins-2.319.3-1.1.noarch.rpm | 69 MB  00:00:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : jenkins-2.319.3-1.1.noarch
  Verifying  : jenkins-2.319.3-1.1.noarch

Installed:
  jenkins.noarch 0:2.319.3-1.1

Complete!
[ec2-user@ip-172-31-87-187 ~]$
```

Configure Jenkins on EC2

Edit the Inbound rules of security group associated to EC2 instance by opening port 8080 so that Jenkins can run.

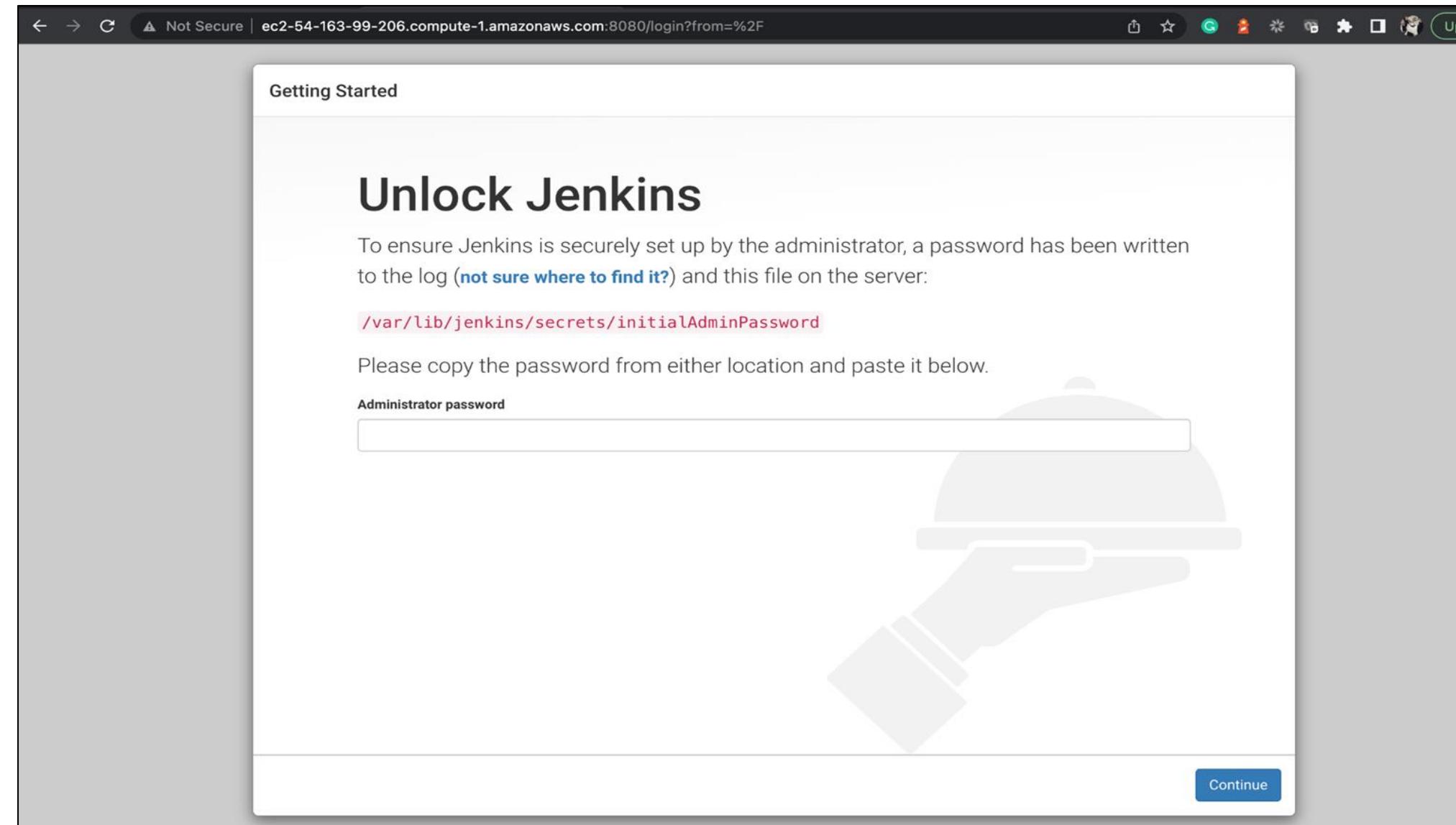
The screenshot shows the AWS Management Console interface for modifying inbound security group rules. The URL in the browser is `console.aws.amazon.com/ec2/v2/home?region=us-east-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-02a6344fb3c0f2d30`. The page title is "Edit inbound rules". The main content area displays a table of existing inbound rules:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-085e477d16ef7f171	SSH	TCP	22	Custom	0.0.0.0/0
-	Custom TCP	TCP	8080	Anywh...	0.0.0.0/0

A red box highlights the bottom row of the table, which represents the new rule being added. Below the table is a button labeled "Add rule". At the bottom right of the page are three buttons: "Cancel", "Preview changes", and "Save rules".

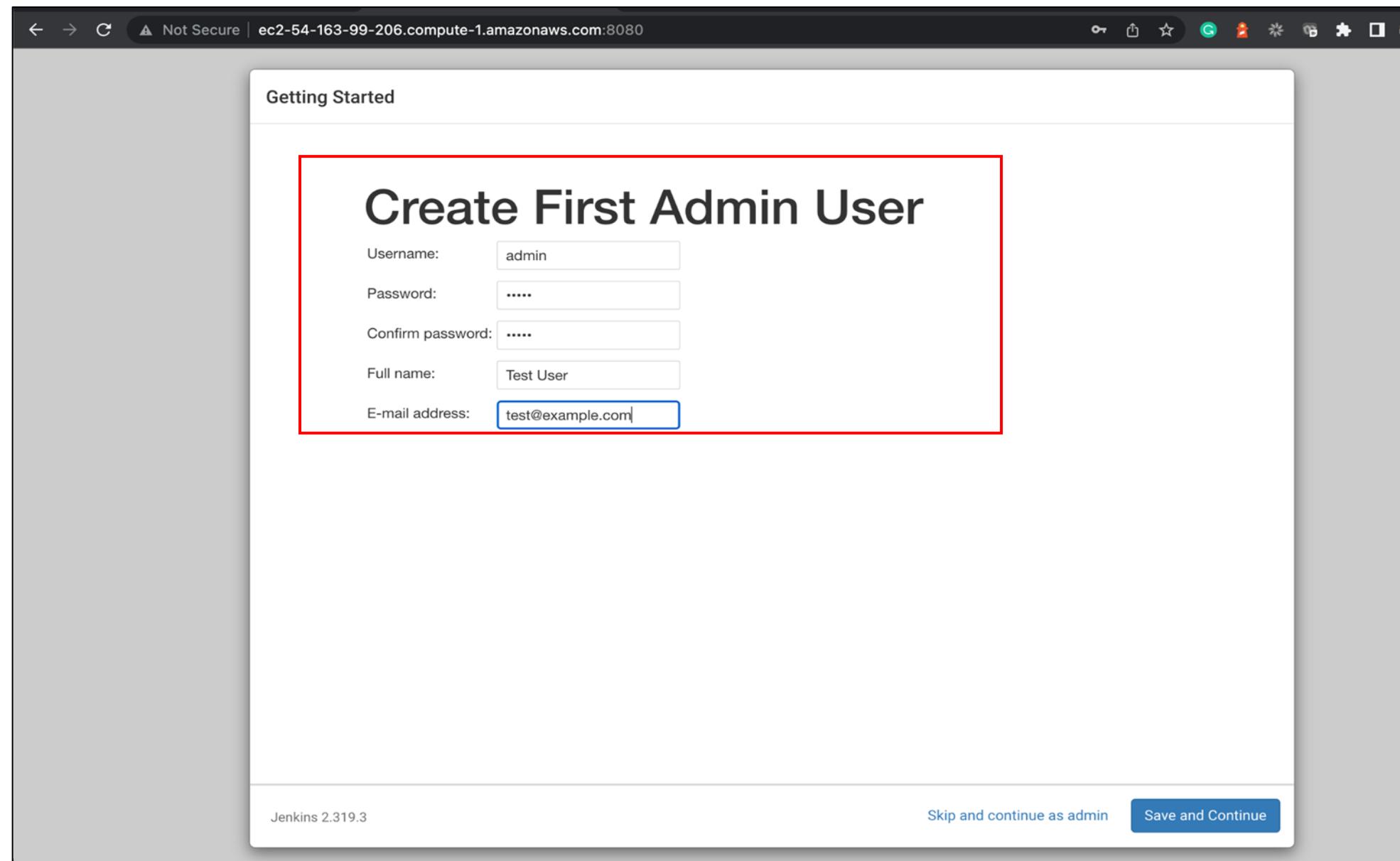
Access Jenkins on Port 8080 of EC2

From here, proceed by logging into the Jenkins with the **initial password**.



Access Jenkins on Port 8080 of EC2

Create a **new user**.



Configure Jenkins on EC2

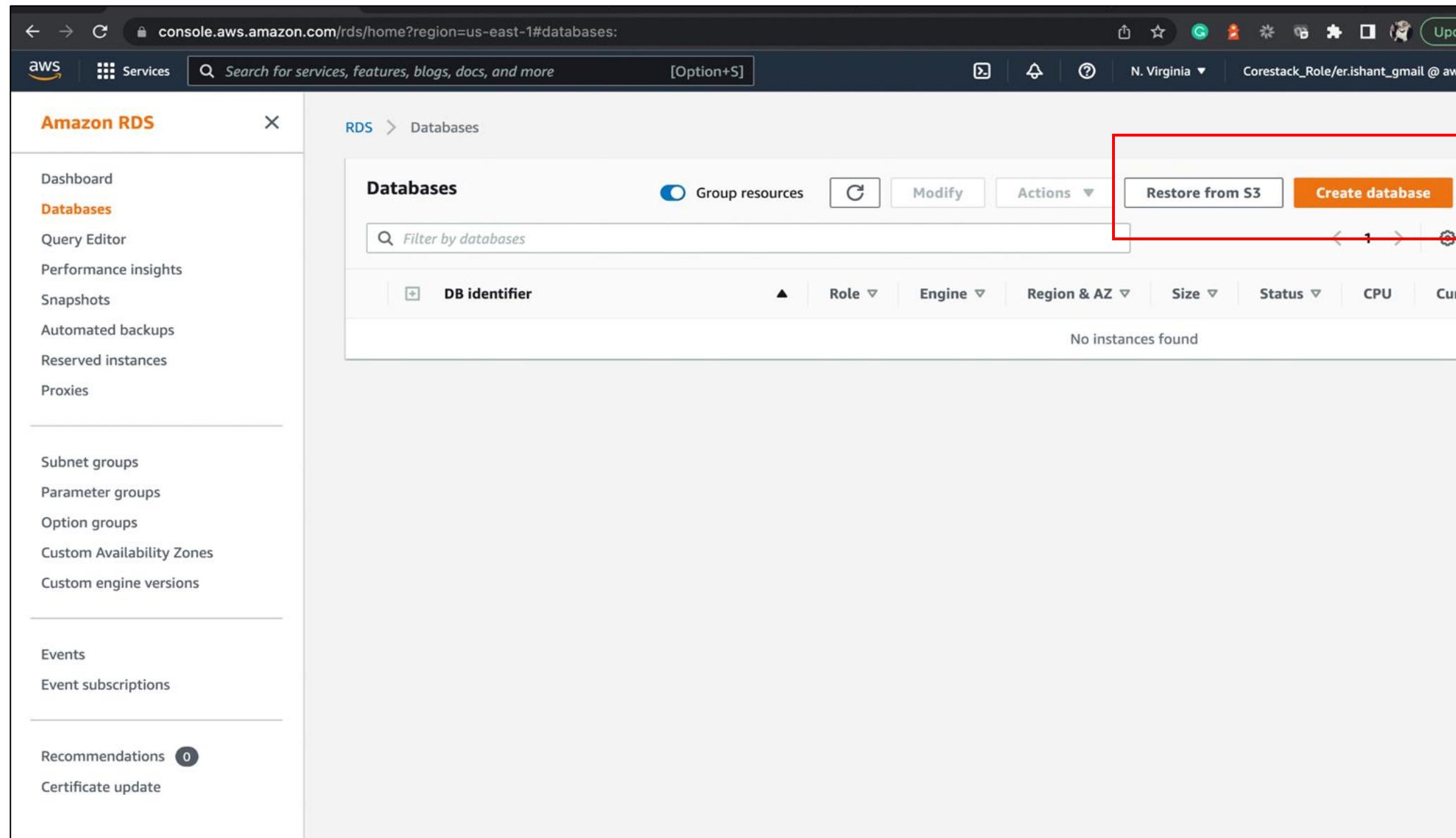
Jenkins is ready to be used on EC2 now.

The screenshot shows the Jenkins dashboard at <http://ec2-54-163-99-206.compute-1.amazonaws.com:8080>. The page title is "Jenkins". The left sidebar includes links for "New Item", "People", "Build History", "Manage Jenkins", "My Views", "Lockable Resources", and "New View". Under "Build Queue", it says "No builds in the queue.". Under "Build Executor Status", it lists "1 Idle" and "2 Idle". The main content area features a "Welcome to Jenkins!" message, a "Start building your software project" section with a "Create a job" button, and a "Set up a distributed build" section with "Set up an agent", "Configure a cloud", and a "Learn more about distributed builds" link.

Create AWS MySQL RDS Instance

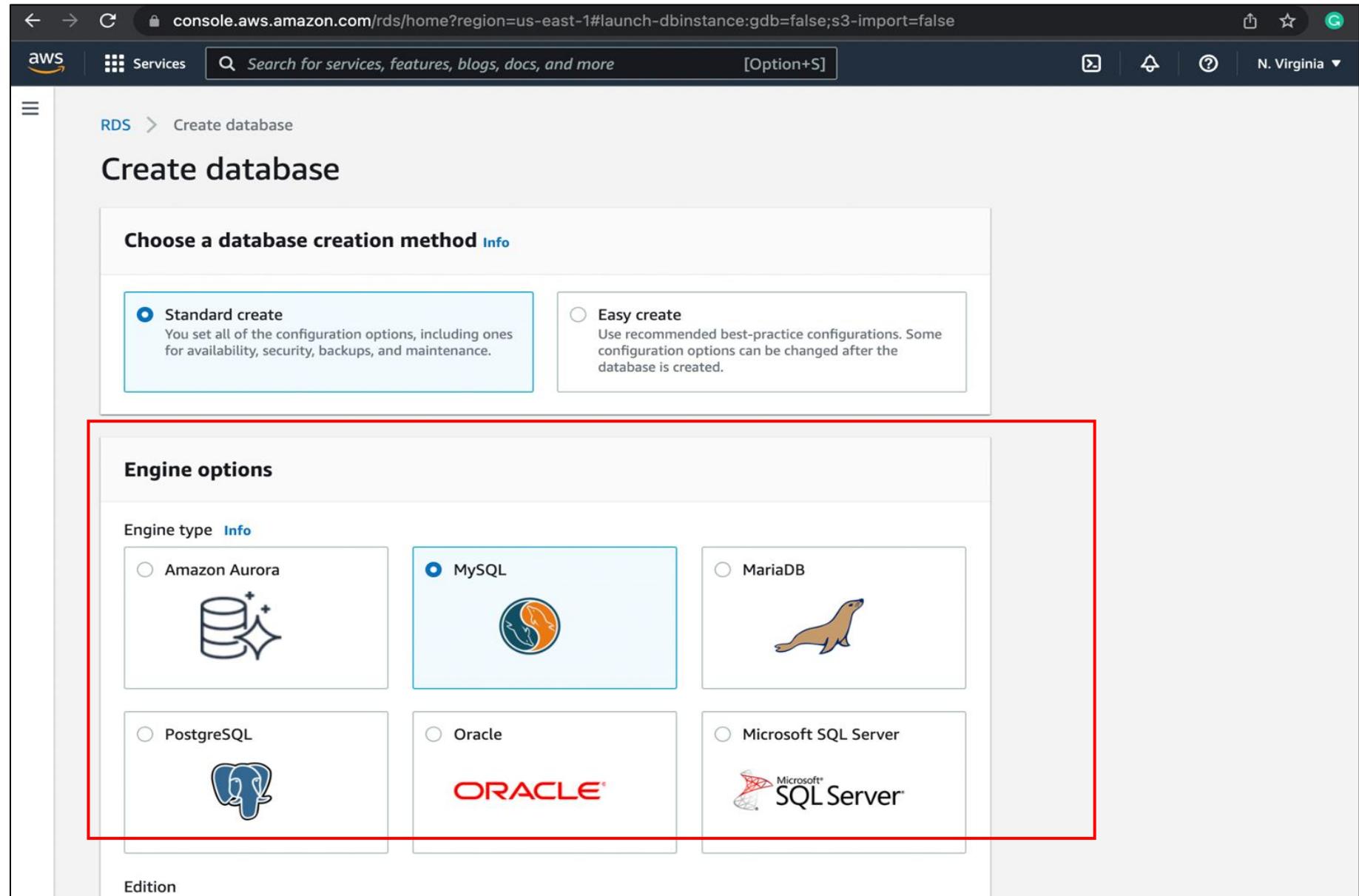
Create MySQL Database

From the Amazon RDS, click on the **Create database** button.



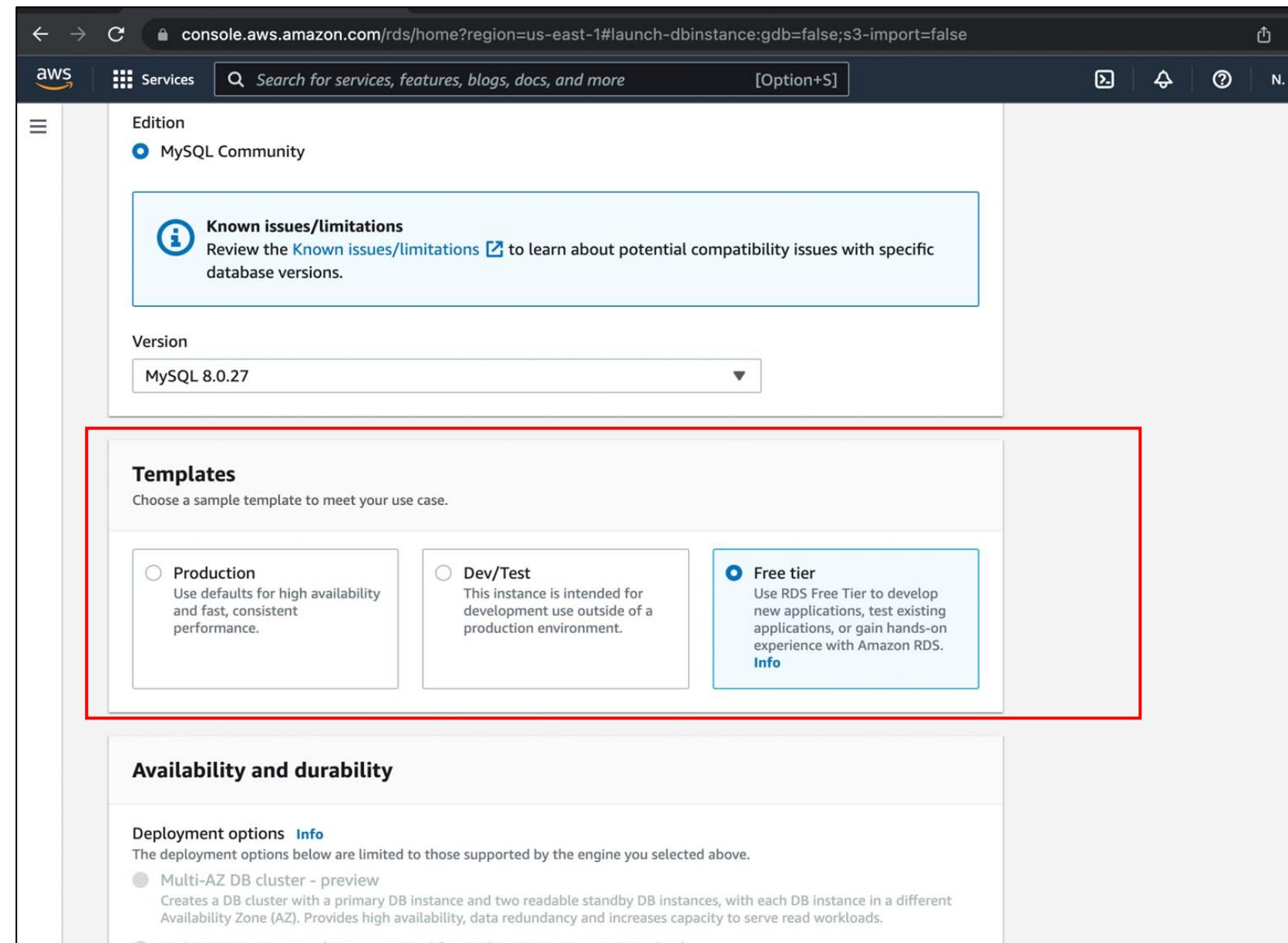
Create MySQL Database

Choose MySQL from the database selection.



Create MySQL Database

From Templates, select the **Free tier** option.



Create MySQL Database

In the **settings**, provide a name, username, and password for the database.

The screenshot shows the AWS RDS 'Create MySQL Database' settings page. The 'DB instance identifier' is set to 'estoredb'. The 'Master username' is 'admin'. The 'Master password' and 'Confirm password' fields both contain '.....'. The 'DB instance class' is set to 'Standard classes (includes m classes)'. The 'Credentials Settings' section is highlighted with a red box.

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter.
 Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

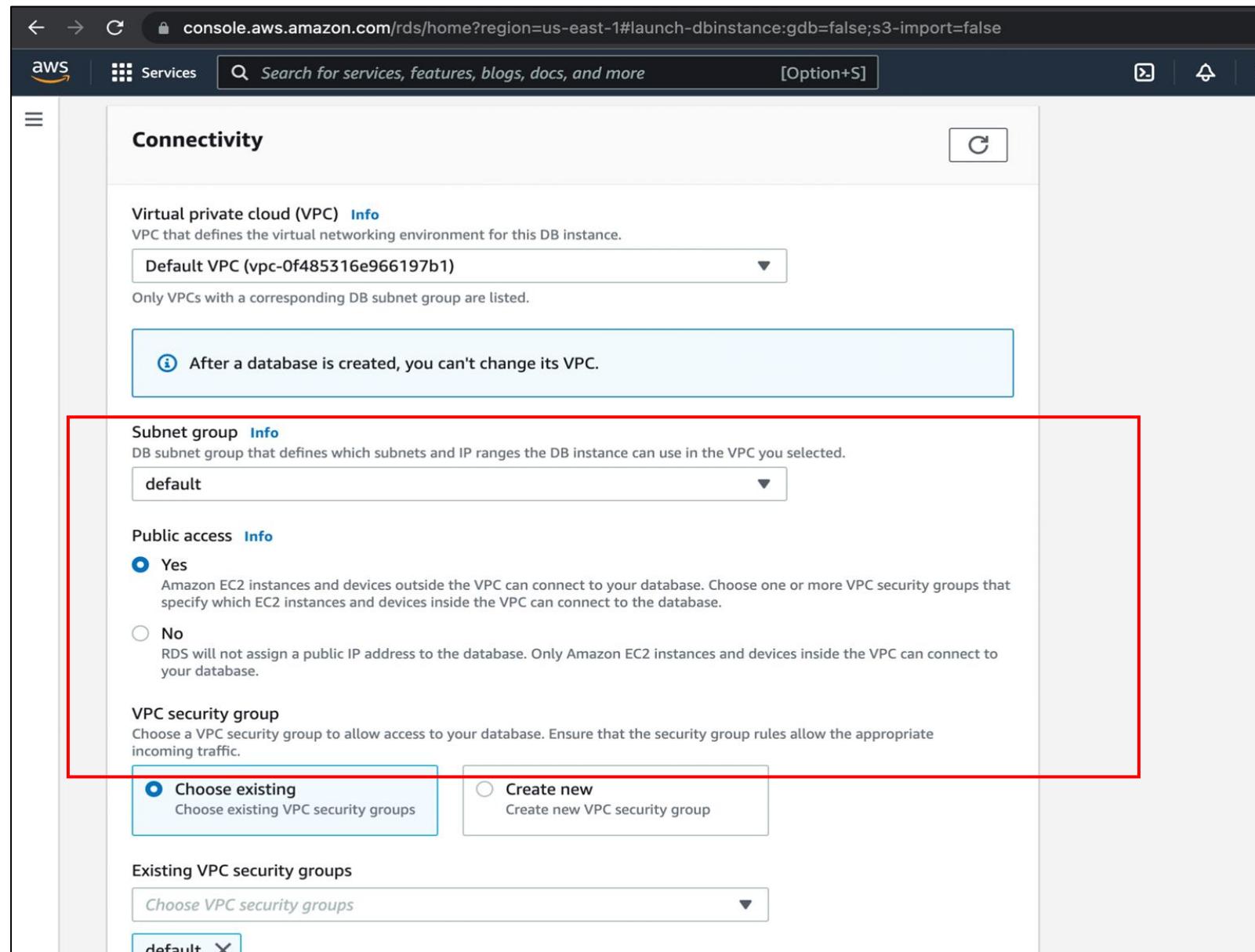
Confirm password [Info](#)

DB instance class

DB instance class [Info](#)
 Standard classes (includes m classes)

Create MySQL Database

On the **Connectivity** page, in the **Public access** section, select **Yes** and create the database.



Create MySQL Database

The database is **ready**.

The screenshot shows the AWS RDS (Relational Database Service) console. On the left, there's a sidebar with the following navigation:

- Dashboard
- Databases** (selected)
- Query Editor
- Performance insights
- Snapshots
- Automated backups
- Reserved instances
- Proxies

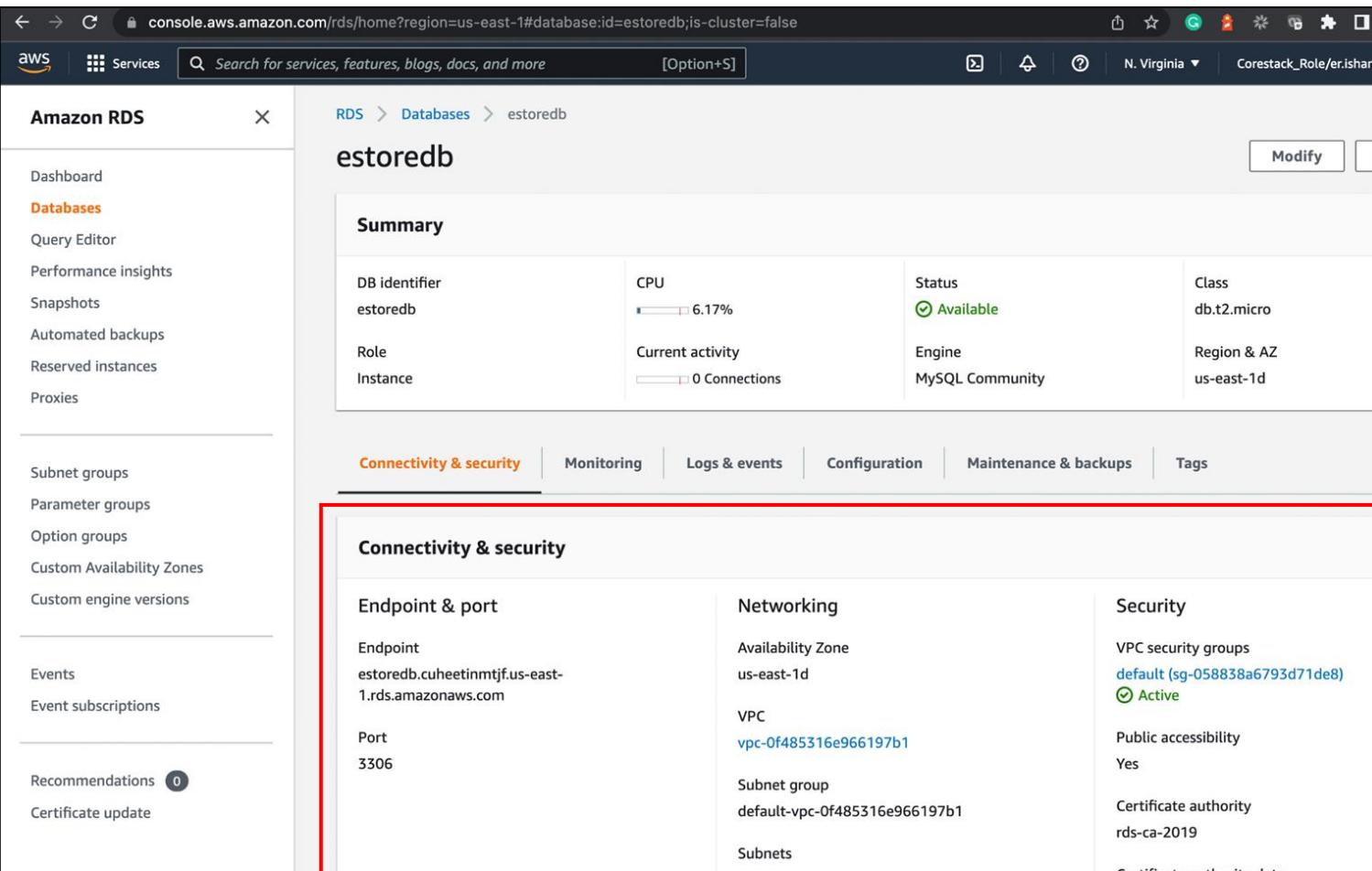
Below these are sections for Subnet groups, Parameter groups, Option groups, Custom Availability Zones, and Custom engine versions.

The main content area is titled "Databases" and shows a table of existing databases. A red box highlights the first row of the table, which contains the following information:

DB identifier	Role	Engine	Region & AZ	Size	Status
estoredb	Instance	MySQL Community	-	db.t2.micro	Creating

Create MySQL Database

Click the **Connectivity & security** tab to note the end point and port numbers. Use the same numbers in the application.properties file of your Java back-end project.



Do not forget to connect to MySQL and create the estoredb database manually.

TECHNOLOGY

Machine Learning
Data Science



Caltech

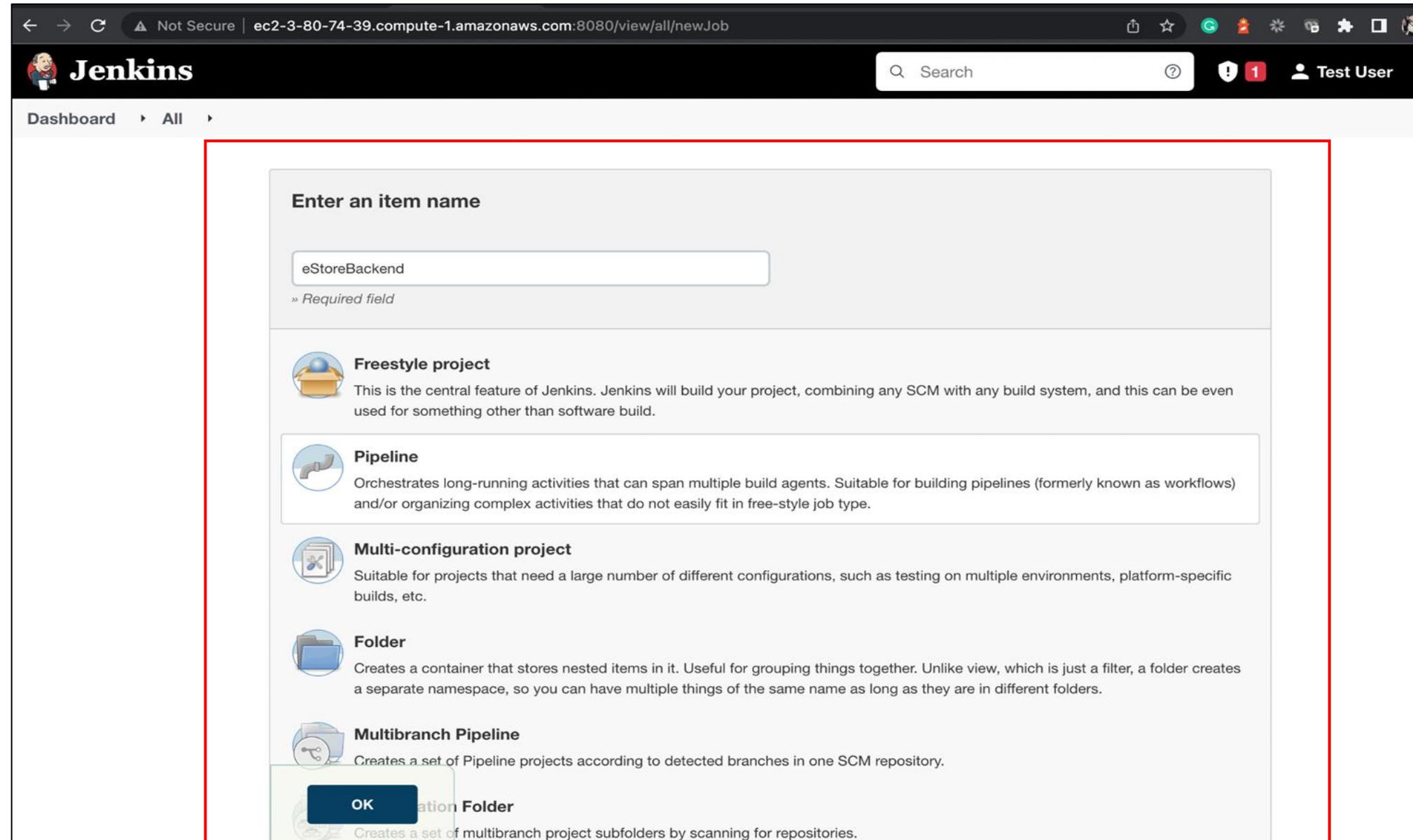
Center for Technology & Management Education

Build and Deploy with Jenkins Docker

Set Up Jenkins for the Java Backend Project

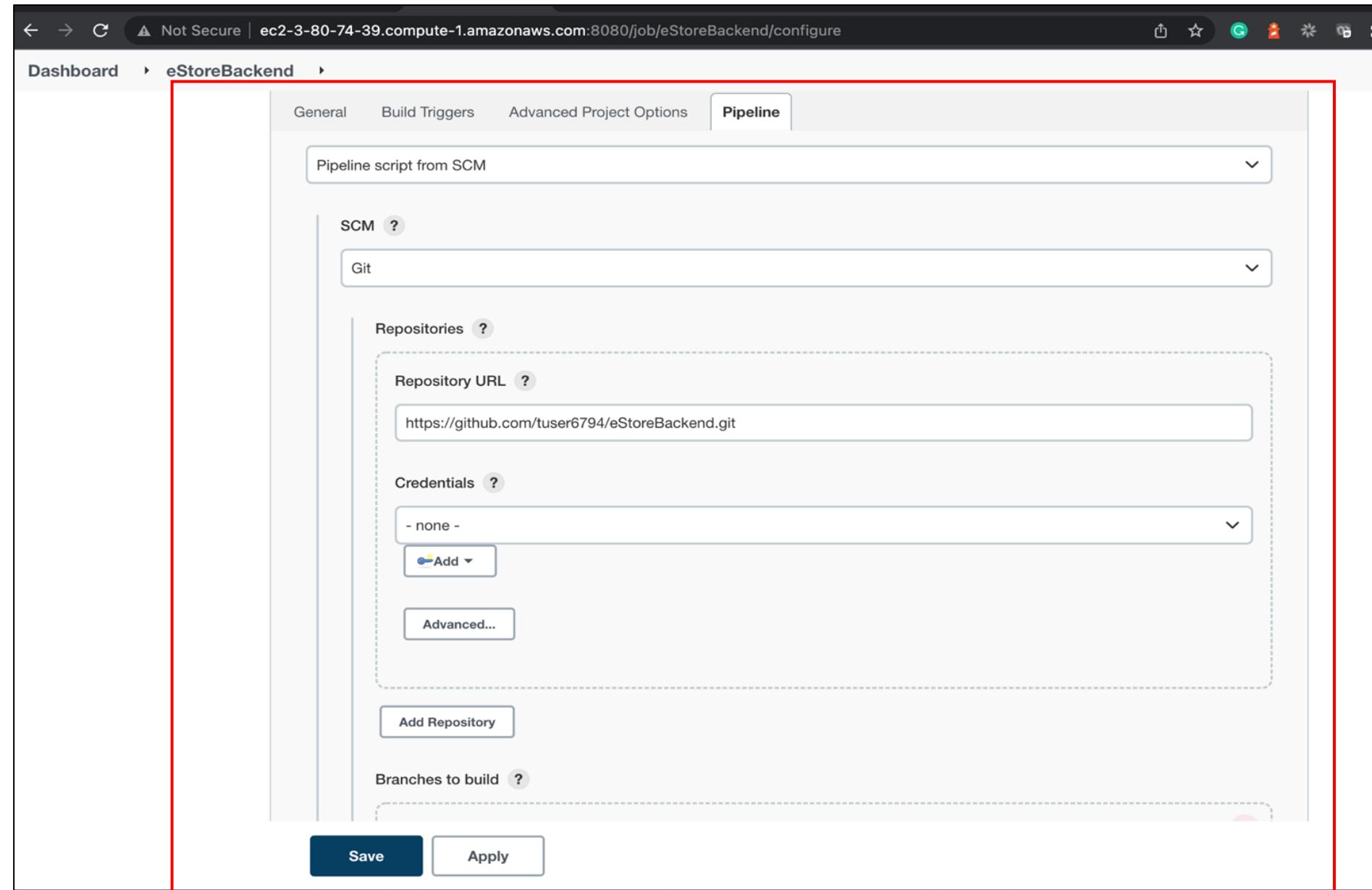
Create a Jenkins Pipeline Project for the Java Backend Project

Create a new project in Jenkins of type **Pipeline**.



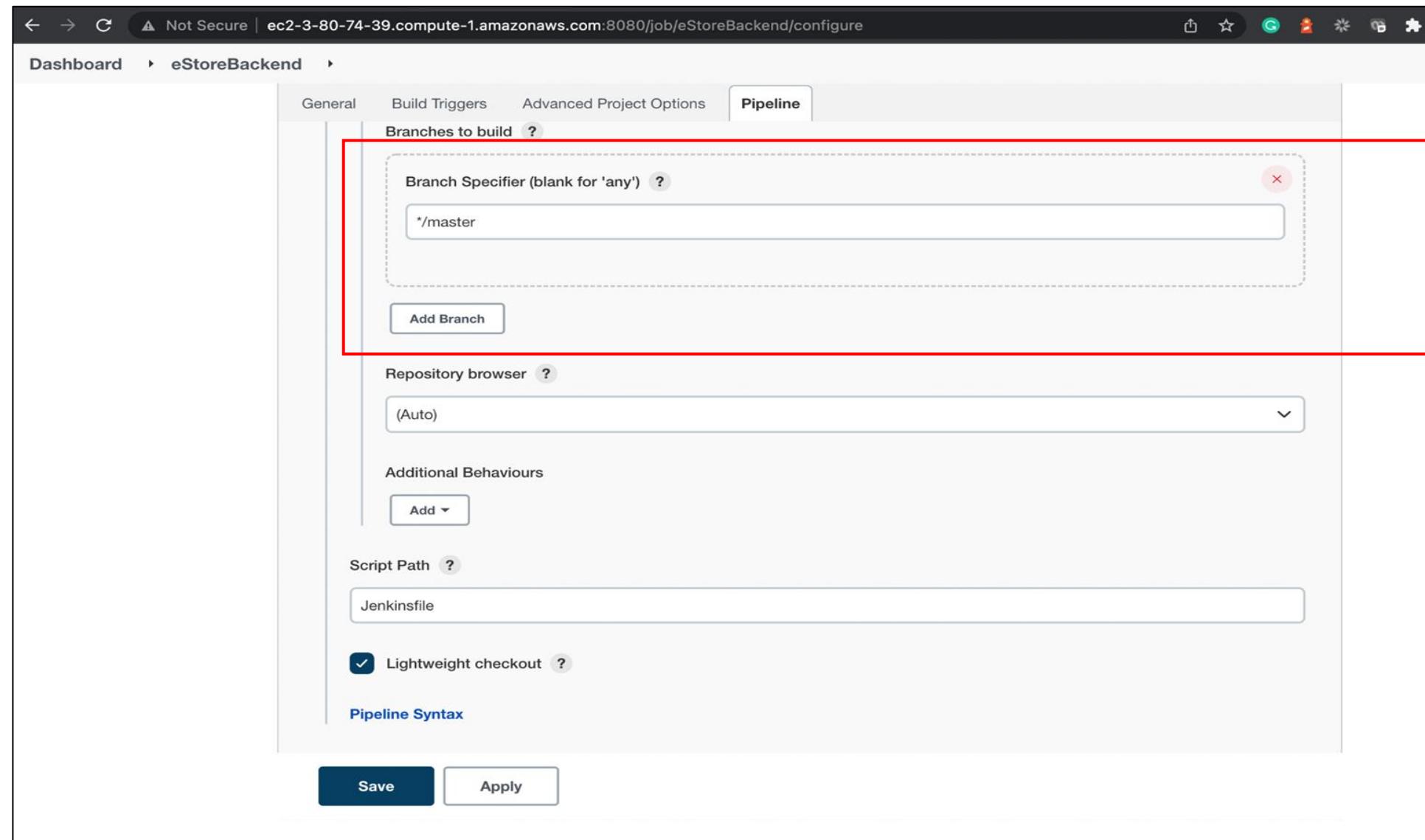
Create a Jenkins Pipeline Project for the Java Backend Project

Configure the Jenkins Pipeline Project by passing GitHub repository URL.



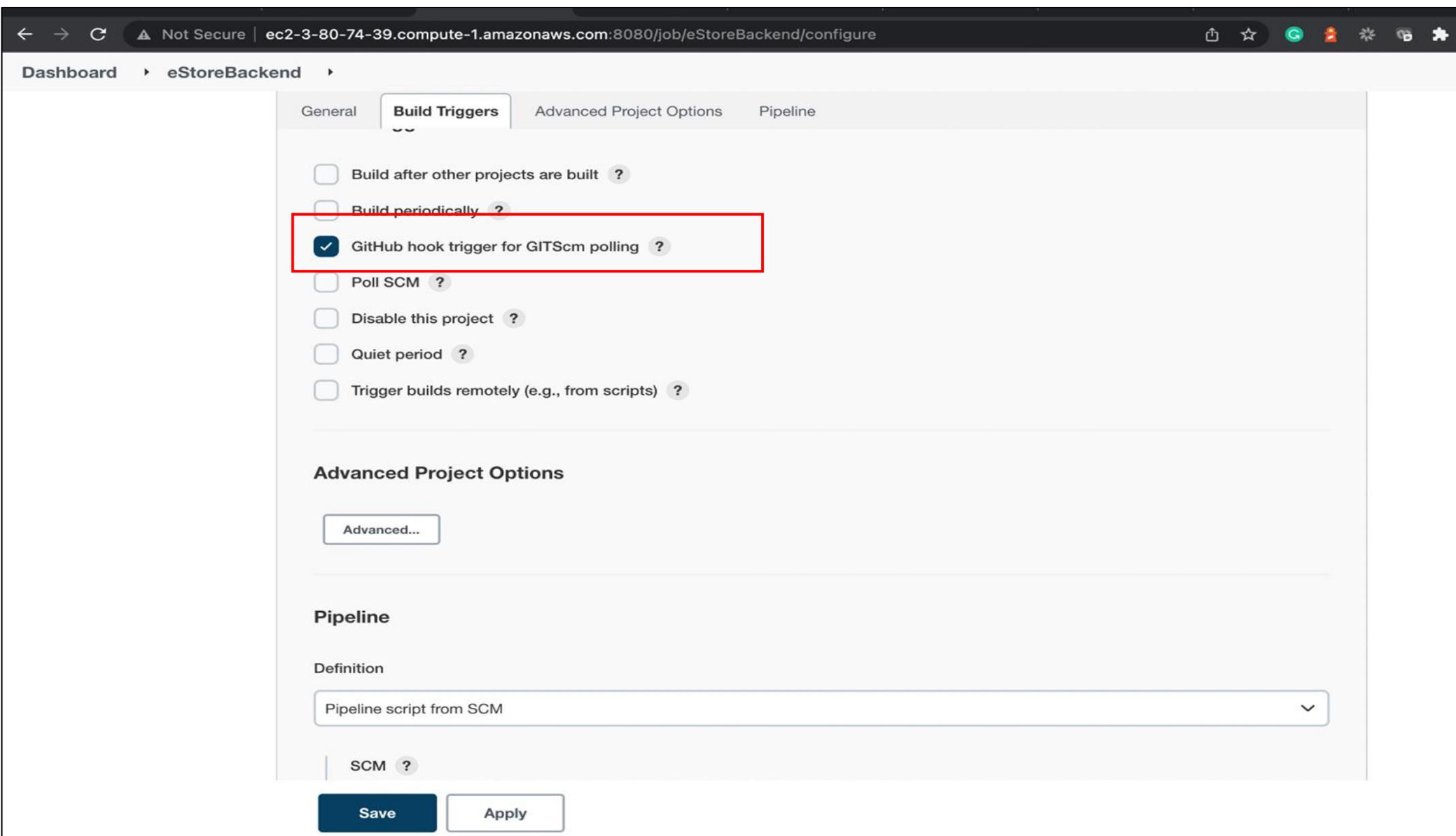
Create a Jenkins Pipeline Project for the Java Backend Project

From the branch specifier, select the **default branch**, which is master. The Script Path contains the Jenkinsfile, which is created in the root directory of our project.



Optional: GitHub Hook Trigger for the Java Backend Project

Configure **GitHub trigger** for GITScm Polling. Users can configure the same using ngrok.



Run the Jenkins Pipeline Project for the Java Backend Project

Notice stages appearing as mentioned in the Jenkinsfile.
Source > Test > Build > Containerize > Deploy

The screenshot shows the Jenkins interface for the 'eStoreBackend' pipeline project. On the left, a sidebar lists various actions: Back to Dashboard, Status, Changes, Build Now (which is highlighted with a red box), Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. Below this is the Build History section, which shows two builds: #2 (Feb 27, 10:00 AM) with 3 commits and #1 (Feb 27, 05:27 AM) with No Changes. At the bottom of the sidebar are links for Atom feed for all and Atom feed for failures.

The main content area is titled 'Pipeline eStoreBackend'. It features a 'Recent Changes' icon and a 'Stage View' section. The Stage View table has columns for Declarative: Checkout SCM, Build, Test, Package, Containerize, and Deploy. The table shows the average stage times for the most recent run (#2). Build #2 took 269ms for Checkout SCM, 3s for Build, 11s for Test, 11s for Package, 1s for Containerize, and 887ms for Deploy. Build #1 took 261ms for Checkout SCM, 3s for Build, 11s for Test, 12s for Package, and 1s for Containerize. The Deploy column for both builds is empty.

A green callout bubble with the text 'Click on Build Now and build project.' points to the 'Build Now' button in the sidebar.

Key Takeaways

- EC2 instances is created and configured on AWS.
- MySQL RDS database is created on AWS.
- Jenkins Pipeline is built to dockerize the Java backend.
- Docker is integrated in Jenkins to build and release the images as containers on EC2.

