**CI/CD process for Snappcoin**

**Contents:**

**A).Prerequisites For deployment**

**1) Launching an Ec2 instance**

**2) Connecting to ec2 instance**

**3) Steps after connecting to Mobaxterm(terminal to connect with ec2 instance )**

**4) Installing Jenkins**

**5) Installing docker**

**6) Installing Kubectl**

**7) Install ingnpm**

**8) Installing Terraform**

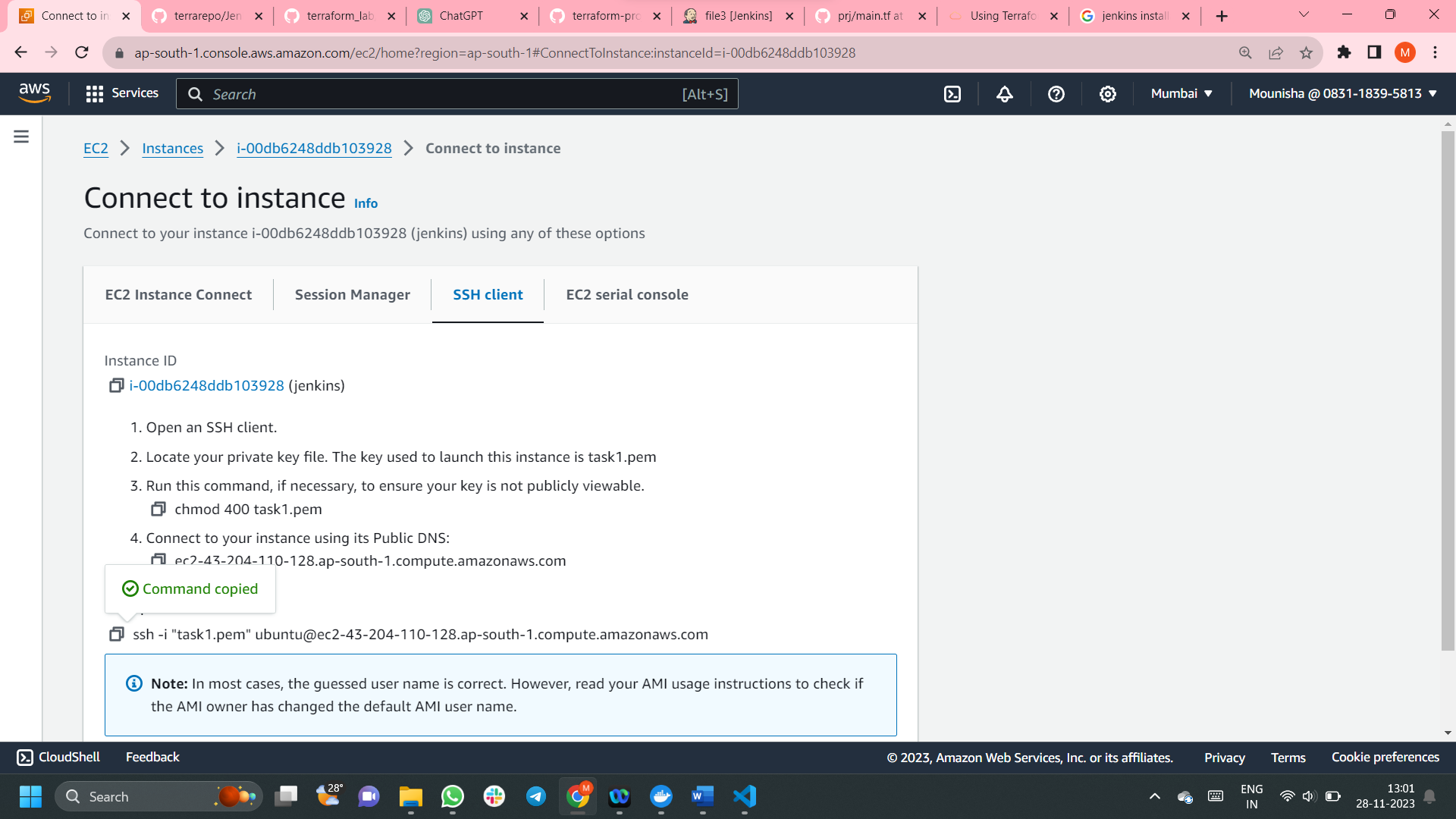
**B)Adding codecommit url in jenkins**

**C)CI-CD pipeline workflow with kuberenetes for Snappcoins application**

**A). Prerequisites / software stack requirements**

1. **Launch an Ec2 instance with,**

* Ubuntu/Amazonlinux (AMI)
* t2.medium(instance-type)(change the instance as per requirements (space/memory) of the project)
* create a key pair(helps in logging into ec2 instance)
* deploy in customised vpc
* add security group rule – add port num 8080(for jenkins)
* configure storage(default- 8gb,if the project requires more storage increase storage capacity)
* launch an instance
* after launched
* click on connect
* copy ssh connect (syntax - ssh -i "jenkins-server.pem" [ec2-user@ec2-3-109-203-7.ap-south-1.compute.amazonaws.com](mailto:ec2-user@ec2-3-109-203-7.ap-south-1.compute.amazonaws.com))

****

1. **Connecting to ec2 instance**

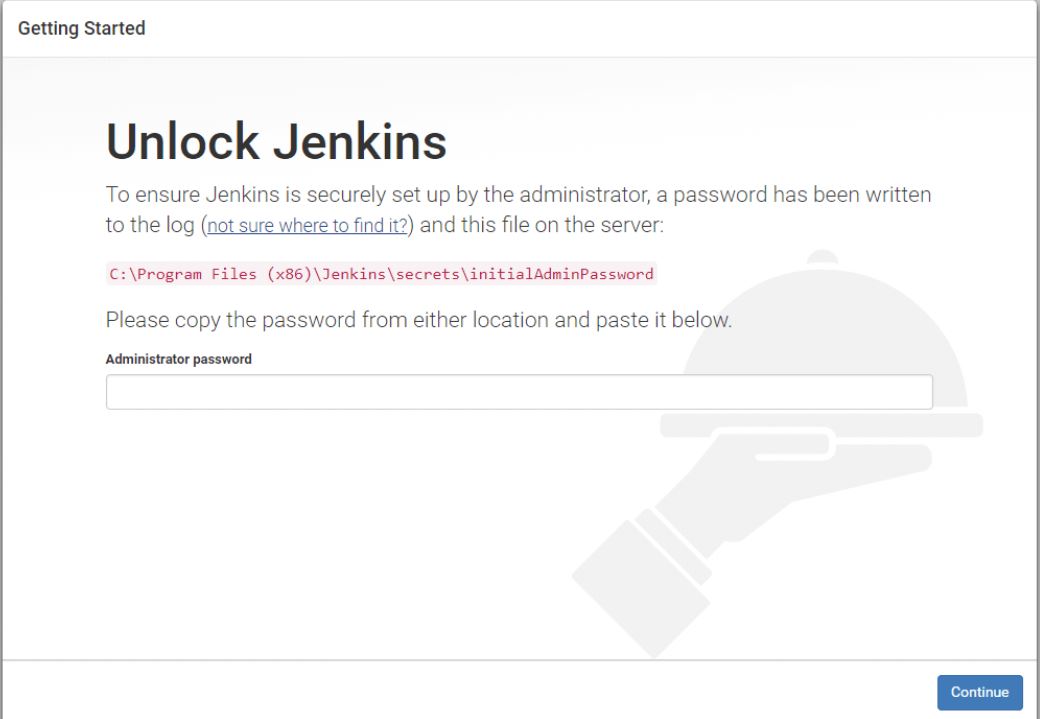
* Use any terminal to connect with ec2 (eg:Mobaxterm)
* (Download Mobaxterm from browser to your PC)

1. **Steps after connecting to Mobaxterm(terminal to connect with ec2 instance )**

* cd path(where your keypair is present)
* paste ssh connect link (eg: ssh -i "jenkins-server.pem" [ec2-user@ec2-3-109-203-7.ap-south-1.compute.amazonaws.com](mailto:ec2-user@ec2-3-109-203-7.ap-south-1.compute.amazonaws.com))
* if it is ubuntu— give commands – sudo apt update -y (to update all packages) and sudo -i

1. **Install Jenkins**

* go to Jenkins.io🡪download🡪select according to your (AMI)
* <https://pkg.jenkins.io/debian-stable/>
* Copy the commands paste it in the Mobaxterm
* After all execution, copy the public ip address of the instance with port num 8080🡪browse it

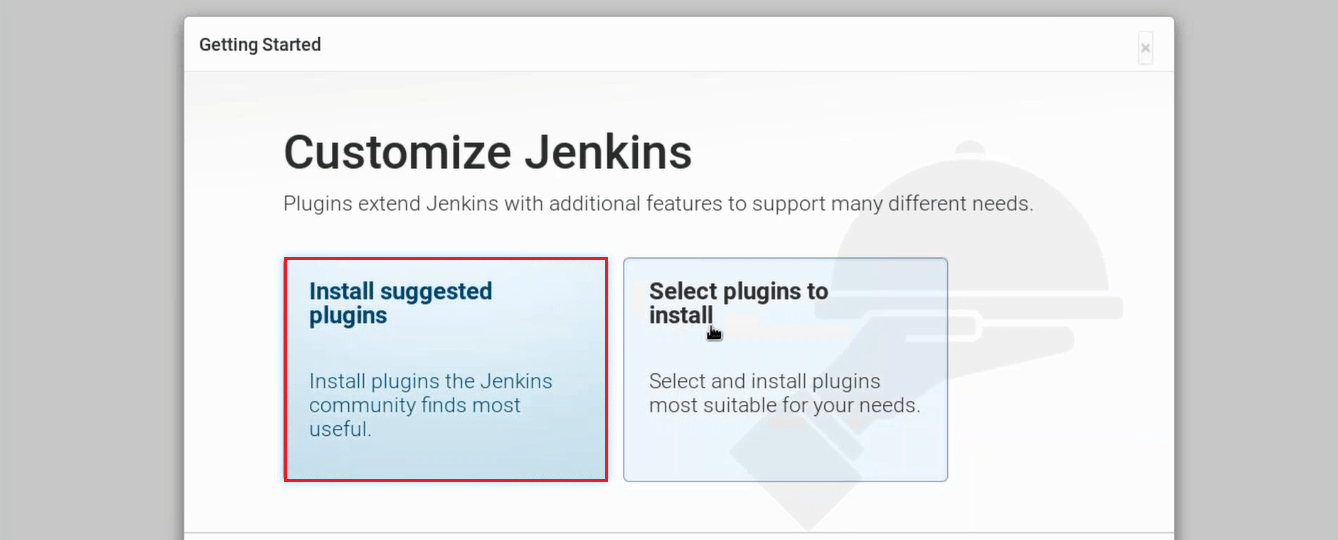
Eg: ip address:8080 ,you will get this web page

* There we get a path (here eg: c:\------)
* Copy the path and using cat command and path (eg: cat c:\---), paste in mobaxterm

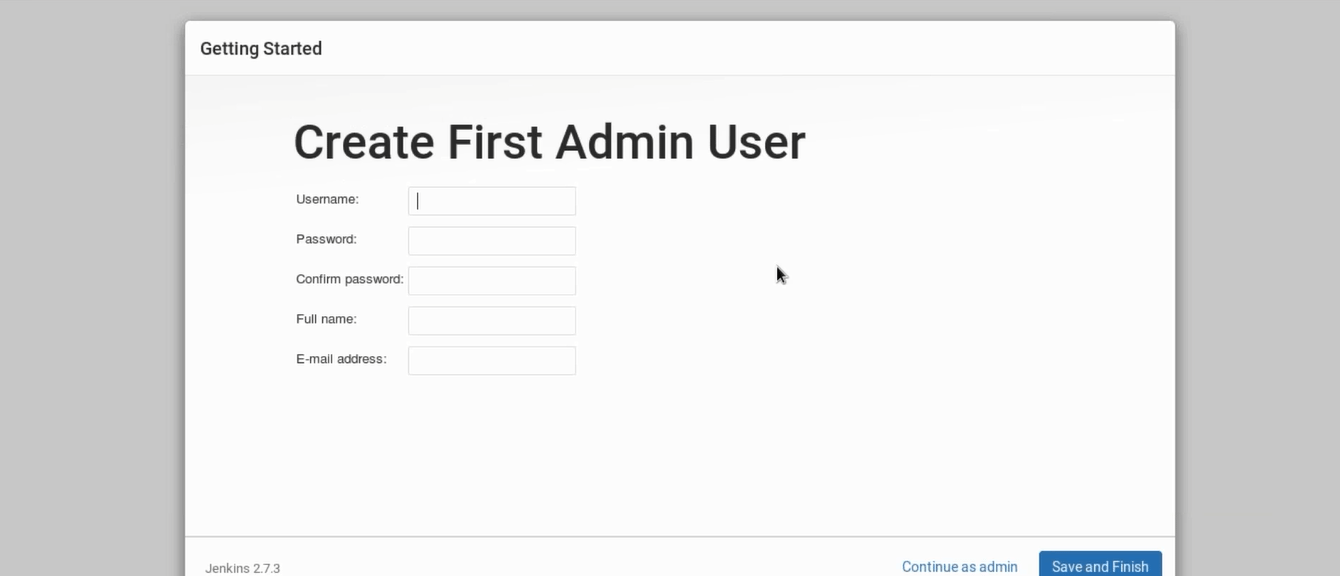
We will get a password(aplhanumeric) 🡪copy the password and place it in text box under administrator password

Download plugins

(select install suggested plugins)



* give admin name , password , gmail



* start Jenkins

1. **Install docker**

-Commands from official docker documentation for installing docker on ubuntu

<https://docs.docker.com/engine/install/ubuntu/>

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

----------- paste it in mobaxterm

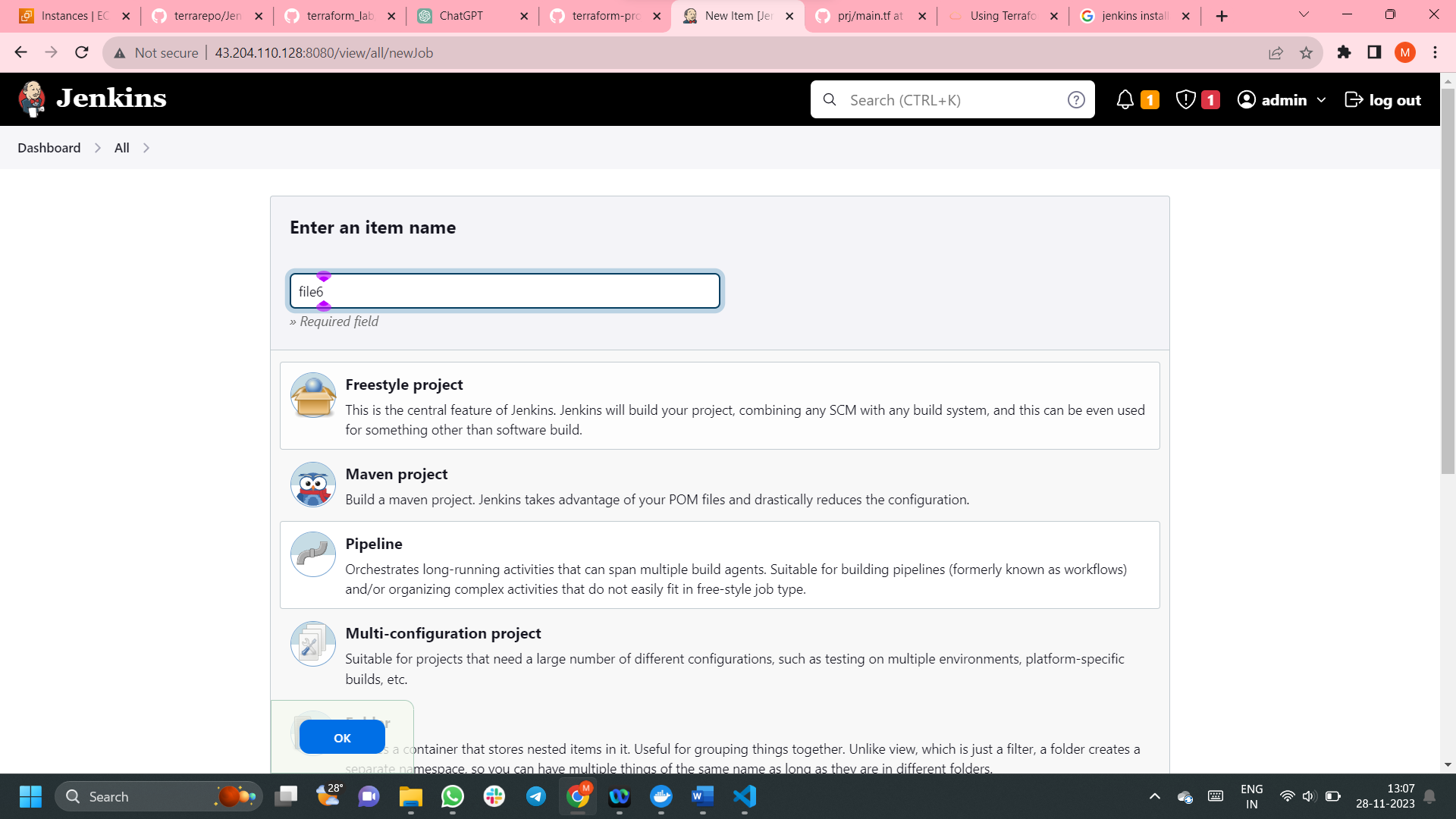
1. **Install Kubectl**

* from this official documentation <https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>
* follow the steps ,copy and paste them in mobaxterm (download 1.28 version)
* kubectl gets downloaded

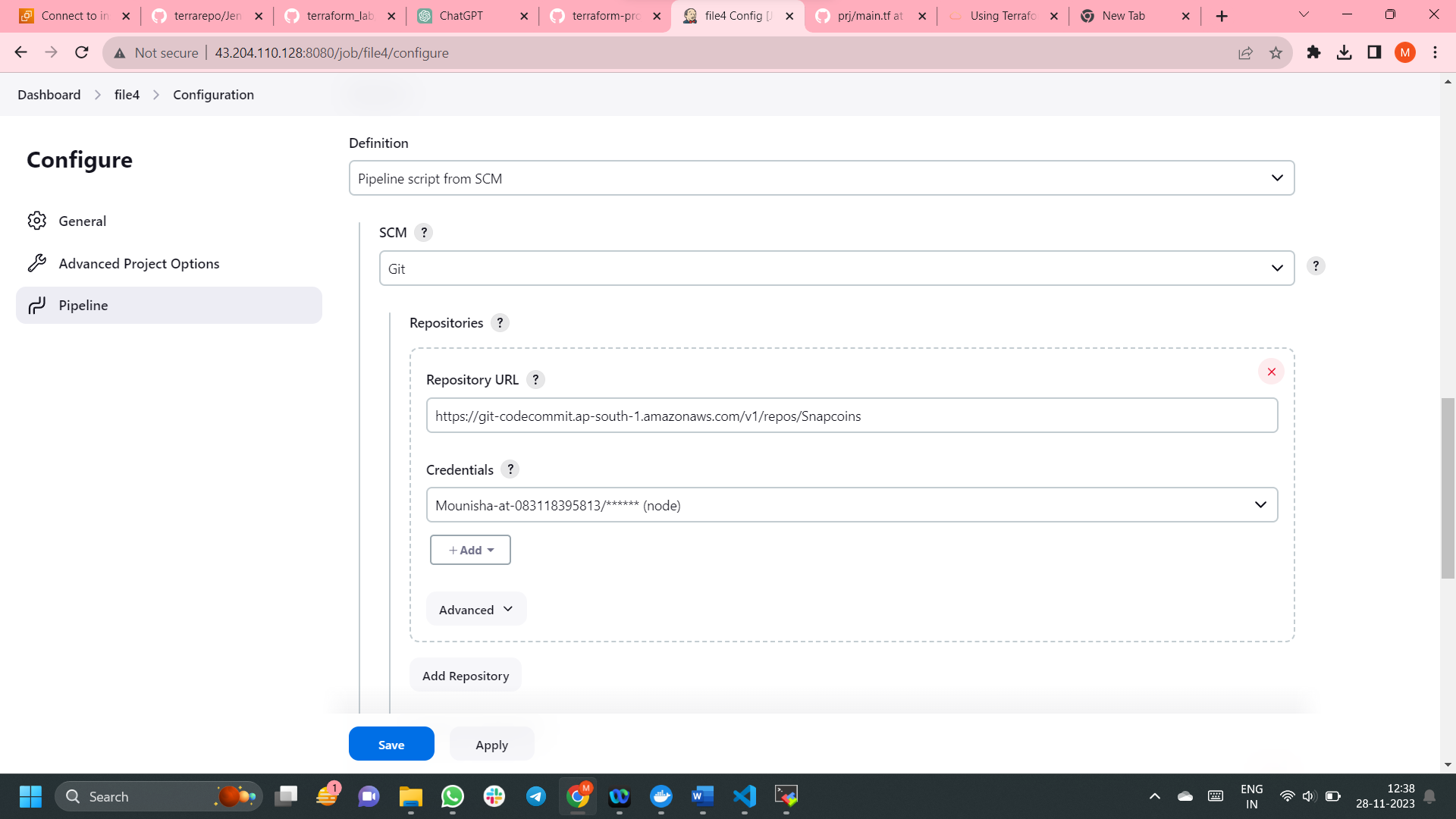
1. **Install npm** (18th version of nodejs)

**8)Install terraform**

* from this official documentation <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>
* select the operating system (as we selected here linux(ubuntu))

**B)ADD CODECOMMIT URL IN JENKINS(ADD CODE HTTP LINK AND ITS CREDENTIALS TO CLONE)--- TO EXTARCT JENKINSFILE AND RUN CI-CD PIPELINE**(prerequisite: Write the Jenkins code that is CI-CD pipeline in a file and name it as ‘Jenkinsfile’ and push it source code repository (for eg : here AWS CodeCommit) 

* Go to dashboard
* Give the name of your job
* Select pipeline
* Click ok

****

* Here select configure then select pipeline
* Give the codecommit repository url
* Give crendentials to access the code in codecommit aws
* Then save

**5) CI-CD pipeline workflow with kuberenetes for Snappcoins application**

# Create a pipeline for build and deployment of frontend **and**  backend service.

1.Authenticate with AWS account with access key and secret key

2.Terraform Init: to initialize the Terraform configuration.

3.Terraform Validate: to check the syntax and validity of the Terraform configuration.

4.Terraform Plan: to show the changes that will be applied to the infrastructure.

5.Terraform Apply: to apply the changes to the infrastructure.

6. Building docker images (attaching version number and build number)

7.Authenticate with ECR

8.Tag and push the images to registry (ECR)

9.Deploy to Kubernetes(using deployment and service yaml files)

10. Print Ingress IP: to retrieve and print the external IP of the Istio Ingress Gateway.

11. Conditional Job Trigger: Ask for user confirmation to trigger the next Jenkins job .The decision is based on user input.

12. Post (Always):Execute cleanup steps:Ask for user confirmation to destroy Terraform resources.

1. stage 1- Authenticate with AWS account with access key and secret key

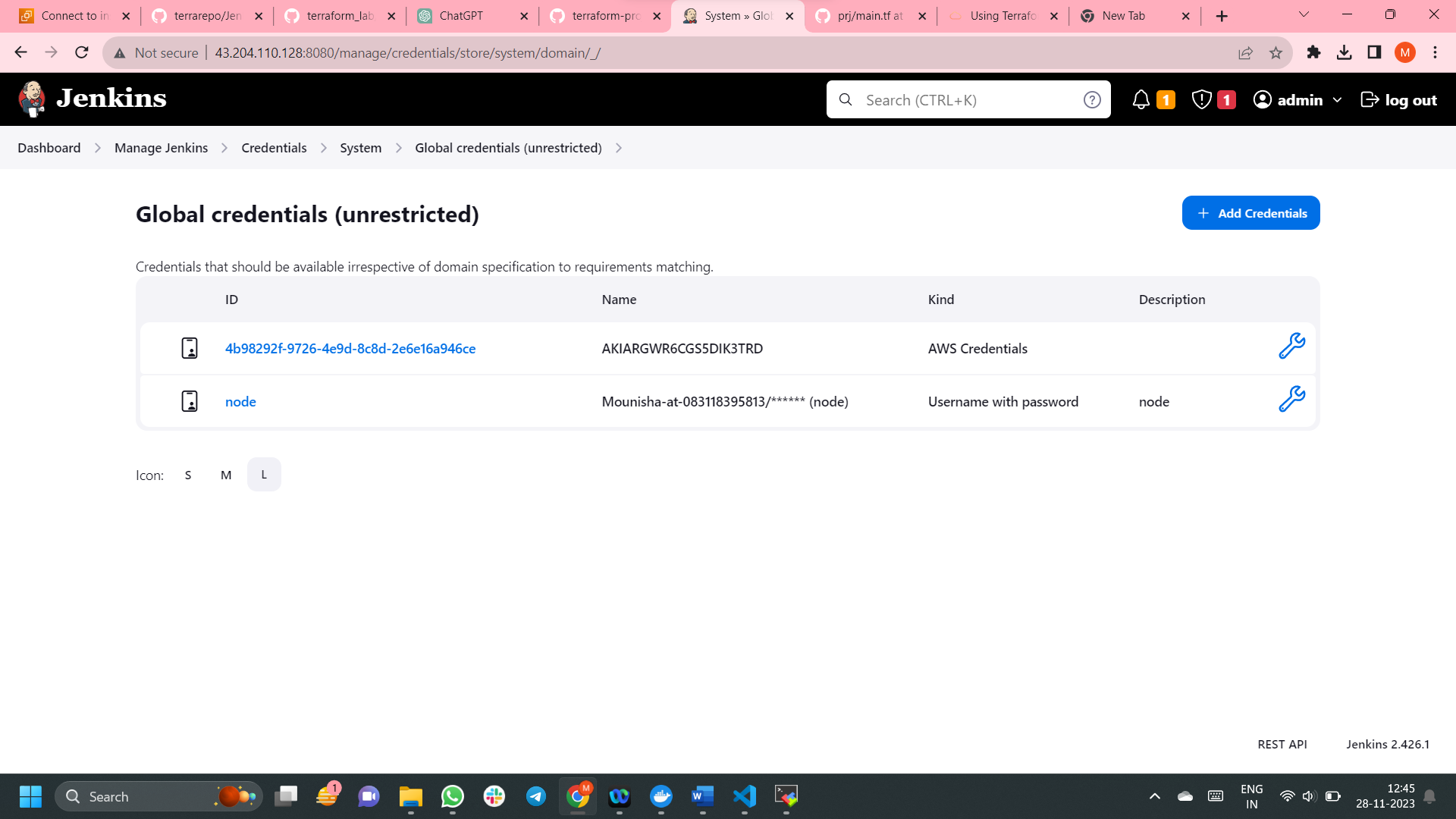
--- login to Jenkins instance

--- In Jenkins dashboard ,go to Manage Jenkins

--- Manage CredentialS

--- set credentials that is give access key and secret key (make sure AWS credentials plugin is downloaded)

--- you will get an credentials ID after saving AWS accesskey and secret keys



--- replace this credentials ID (here eg: [4b98292f-9726-4e9d-8c8d-2e6e16a946ce](http://43.204.110.128:8080/manage/credentials/store/system/domain/_/credential/4b98292f-9726-4e9d-8c8d-2e6e16a946ce) )in Jenkins pipeline

**2. Terraform Init:**

- Purpose: Initializing the Terraform configuration.

- Command: `terraform init`

- Explanation:

- The `terraform init` command is used to initialize a Terraform working directory.

- It downloads the necessary provider plugins and sets up the backend, as specified in the Terraform configuration.

- Initialization is a crucial step before applying any changes to the infrastructure.

- This stage ensures that Terraform is properly set up and has all the required dependencies.

**3. Terraform Validate:**

- Purpose: Checking the syntax and validity of the Terraform configuration.

- Command: `terraform validate`

- Explanation:

- The `terraform validate` command is used to check the syntax and validity of the Terraform configuration files.

- It ensures that the configuration adheres to the correct Terraform language syntax and structure.

- This stage helps catch potential errors early in the process before attempting to apply changes to the infrastructure.

- It is a good practice to validate the configuration before proceeding to the planning and applying stages.

**4. Terraform Plan:**

- Purpose: Showing the changes that will be applied to the infrastructure.

- Command: `terraform plan`

- Explanation:

- The `terraform plan` command generates an execution plan describing the changes that Terraform will make to the infrastructure.

- It doesn't apply any changes but provides a preview of what will happen when the `terraform apply` command is executed.

- The plan includes additions, modifications, and deletions of resources based on the Terraform configuration.

- Reviewing the plan allows operators to understand the impact of the changes before applying them.

**5. Terraform Apply:**

- Purpose: Applying the changes to the infrastructure.

- Command: `terraform apply -var="environment\_name=qa" --auto-approve`

- Explanation:

- The `terraform apply` command is used to apply the changes described in the Terraform configuration to the actual infrastructure.

- The `-var="environment\_name=qa"` flag sets a variable (`environment\_name`) to the value 'qa'. Variables can be used to customize the Terraform configuration for different environments.

- The `--auto-approve` flag is used to automatically approve and apply changes without requiring manual confirmation.

- This stage is where the actual modifications to the infrastructure are made based on the Terraform plan generated in the previous stage.

**6. Building Docker Images (with Version and Build Number):**

Explanation:

The following steps involve building Docker images for different components of the application.

Each Docker image is tagged with a version number and build number, providing versioning information and traceability.

**a. For Each Docker Image:**

Command Example (in a directory): docker build -t <image-name>:<version>-<build-number> .

Explanation:

Change to the directory where the Dockerfile for the respective component is located.

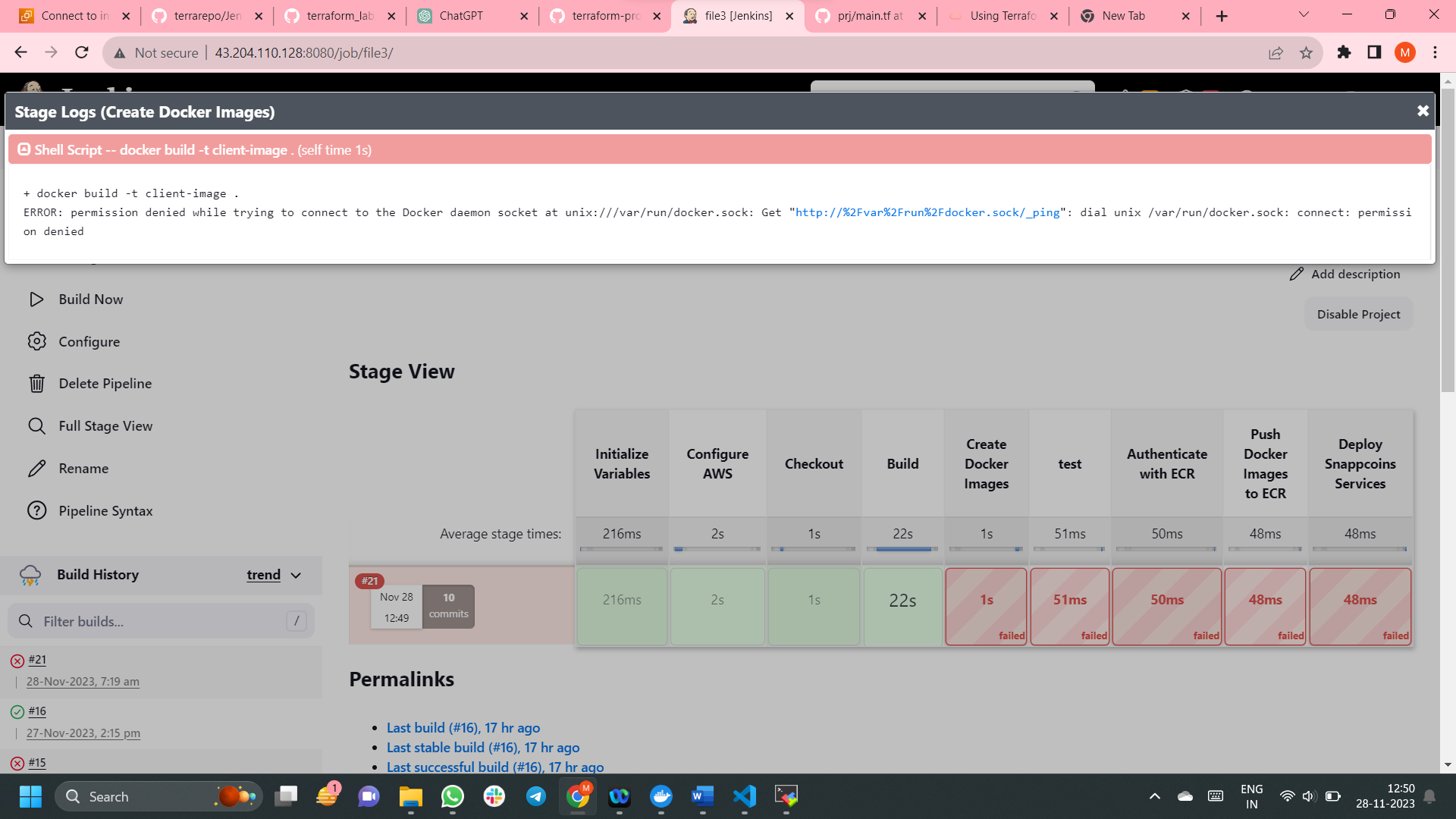
Use the docker build command to build a Docker image.

-t <image-name>:<version>-<build-number> tags the image with a specific name that includes version and build number information.

The . at the end specifies the build context as the current directory.

------------------

**HOW TO SOLVE ERRORS REGARDING DOCKER** :

1. Restart the docker service if the instance get stopped and restarted again
2. Give permissions to /var/run/docker.sock file to solve the below error

-----------------------

**7. Authenticate with ECR**

Purpose:To authenticate the Docker client with the specified Amazon ECR registry, enabling the subsequent stages to push Docker images to this registry.

Steps(script):

The script block allows the execution of a scripted block within the Jenkins Pipeline. In this case, it's used to define a Groovy script that performs the authentication.

**Steps Inside script:**

**1)Defining ecrLogin:**

def ecrLogin = "aws ecr get-login-password --region ${ECR\_REGION}"

This line constructs a command to retrieve an authentication token for the Amazon ECR registry. It uses the aws ecr get-login-password command, which retrieves a Docker login command with an authentication token as its password.

**2)Constructing ecrLoginCommand:**

“def ecrLoginCommand = "${ecrLogin} | docker login --username AWS --password-stdin ${ECR\_REGISTRY}" ”

This line constructs a full Docker login command using the authentication token obtained in the previous step. It uses the docker login command, passing the AWS credentials and the ECR registry URL.

**3)Executing ecrLoginCommand:**

“sh ecrLoginCommand”

This line executes the constructed Docker login command using the sh step in Jenkins. The sh step runs shell commands in the Jenkins environment.

Explanation:

Amazon ECR requires Docker clients to be authenticated before pushing or pulling images. The authentication involves obtaining an authentication token from AWS and using it to log in to the Docker registry.

The aws ecr get-login-password command is used to retrieve this authentication token. The obtained token is then used to log in to the Docker registry using the docker login command.

Once authenticated, the Docker client is authorized to interact with the specified ECR registry, allowing subsequent stages to push Docker images to the registry.

**8.** **Stage: Initialize Variables**

Purpose:

To define and initialize the DOCKER\_IMAGE\_NAMES variable with a list of Docker image names corresponding to various services.

Steps:

script Block:

The script block allows the execution of a scripted block within the Jenkins Pipeline. In this case, it's used to define a Groovy script that initializes the DOCKER\_IMAGE\_NAMES variable.

**Script Steps:**

**Initializing Docker Image Names:**

DOCKER\_IMAGE\_NAMES = ['client-image', 'gamer-module', 'merchant-module', 'gaming-vendor-module', 'general-module']

This line initializes the DOCKER\_IMAGE\_NAMES variable as a Groovy list containing the names of various Docker images. Each name corresponds to a specific service or component

ADDITIONAL\_SUBDOMAINS = ["client-service", "gaming-vendor-module-service", "gamer-module-service", "general-module-service", "merchant-module-service"]

This line initializes the ADDITIONAL\_SUBDOMAINS variable with a list of additional subdomains linked to specific services. Each string in the list represents a subdomain associated with a particular service.

Explanation:

The primary goal of this stage is to set up centralized and easily maintainable lists of Docker image names (DOCKER\_IMAGE\_NAMES) and additional subdomains (ADDITIONAL\_SUBDOMAINS). These variables play a crucial role in subsequent stages, especially tasks like pushing Docker images to Amazon ECR and managing DNS records.

By defining and initializing these variables, the pipeline gains flexibility and adaptability. If there are changes to existing services or the introduction of new ones, updating these lists becomes straightforward. This centralized approach enhances maintainability, avoiding scattered modifications across the pipeline script. It establishes a solid foundation for consistent and efficient management of Docker images and subdomains throughout the CI/CD process.

**9. Stage: Push Docker Images to ECR**

Purpose:

To push Docker images for multiple services to an Amazon ECR registry.

Steps:

script Block:

The script block allows the execution of a scripted block within the Jenkins Pipeline. In this case, it's used to define a Groovy script that performs the necessary Docker image pushing steps.

**Script Steps:**

**Iterating Over Docker Image Names:**

DOCKER\_IMAGE\_NAMES.each { imageName -> ... }: This line iterates over a list of Docker image names. The list is likely defined earlier in the pipeline script.

**Docker Registry Configuration:**

withDockerRegistry(url: "https://${ECR\_REGISTRY}/${imageName}:${VERSION}") { ... }: This block configures the Docker registry URL for the specified Docker image. It allows subsequent docker commands to interact with the specified ECR registry.

**Tagging Docker Image:**

sh "docker tag ${imageName}:${VERSION}-${BUILD\_NUMBER} ${ECR\_REGISTRY}/${imageName}:${VERSION}-${BUILD\_NUMBER}": This line uses the docker tag command to tag the Docker image with a version and build number. This is necessary before pushing it to the ECR registry.

**Pushing Docker Image:**

sh "docker push ${ECR\_REGISTRY}/${imageName}:${VERSION}-${BUILD\_NUMBER}": This line uses the docker push command to push the tagged Docker image to the specified ECR registry.

Explanation:

The purpose of this stage is to automate the process of pushing Docker images to an Amazon ECR registry, which is a managed Docker registry provided by AWS.

The script iterates over a list of Docker image names, and for each image, it performs the following steps:

Configures the Docker registry URL for the specified ECR registry.

Tags the Docker image with the version and build number.

Pushes the tagged Docker image to the ECR registry.

**10.Stage: Deploy snappcoins services**

Purpose:To deploy different components of the Snappcoins application to a Kubernetes cluster.

**Steps:**

**Global Configuration for Kubeconfig:**

sh "export KUBECONFIG=/new/directory/path/config"

This line sets the KUBECONFIG environment variable to specify the path to the Kubernetes configuration file. This is necessary for subsequent kubectl commands to interact with the correct Kubernetes cluster.

**Update Kubeconfig for EKS Cluster:**

sh "aws eks update-kubeconfig --name qa-snappcoins-cluster --region ap-south-1"

This line updates the KUBECONFIG file with the configuration of the specified Amazon EKS (Elastic Kubernetes Service) cluster. This ensures that kubectl commands target the correct Kubernetes cluster.

**Apply Kubernetes Configurations:**

sh 'kubectl apply -f VirtualService.yaml':

Applies the Kubernetes configuration specified in the VirtualService.yaml file. This likely defines a Virtual Service, which is part of service mesh configuration (Istio)🡪gateway and virtual service for each service are configured.

**For Each Service:**

The following steps are repeated for each service (client, gaming-vendor-module, gamer-module, general-module, merchant-module):

**Change to Service Directory:**

dir('service-directory') { ... }: Changes the working directory to the directory containing the Kubernetes configuration files for the specific service.

**Apply Kubernetes Configurations for Service:**

sh 'kubectl apply -f service-serviceaccount.yaml': Applies the Kubernetes configuration specified in the service-serviceaccount.yaml file for the respective service. This includes creating service accounts.

sh 'kubectl apply -f service-deployment.yaml': Applies the Kubernetes configuration specified in the service-deployment.yaml file for the respective service. This includes deploying Kubernetes Deployments.

sh 'kubectl apply -f service-service.yaml': Applies the Kubernetes configuration specified in the service-service.yaml file for the respective service. This includes creating Kubernetes Services.

Explanation:

This stage handles the deployment of various services that make up the Snappcoins application to a Kubernetes cluster.

The script performs necessary configurations, updates the KUBECONFIG file, and then applies the Kubernetes configurations for each service, including service accounts, deployments, and services.

The use of different directories for each service suggests that each service has its own set of configuration files.

**HOW TO SOLVE ERRORS REGARDING KUBECTL** :

1. sometimes while executing kubectl commands we may find this error (**kubectl error: exec plugin: invalid apiVersion “client.authentication.k8s.io/v1alpha1” while using kubectl**)

Then use the following commands

cd

mv kubectl kubectl\_1.24

mv bin bin\_1.24

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.15/2023-01-11/bin/linux/amd64/kubectl

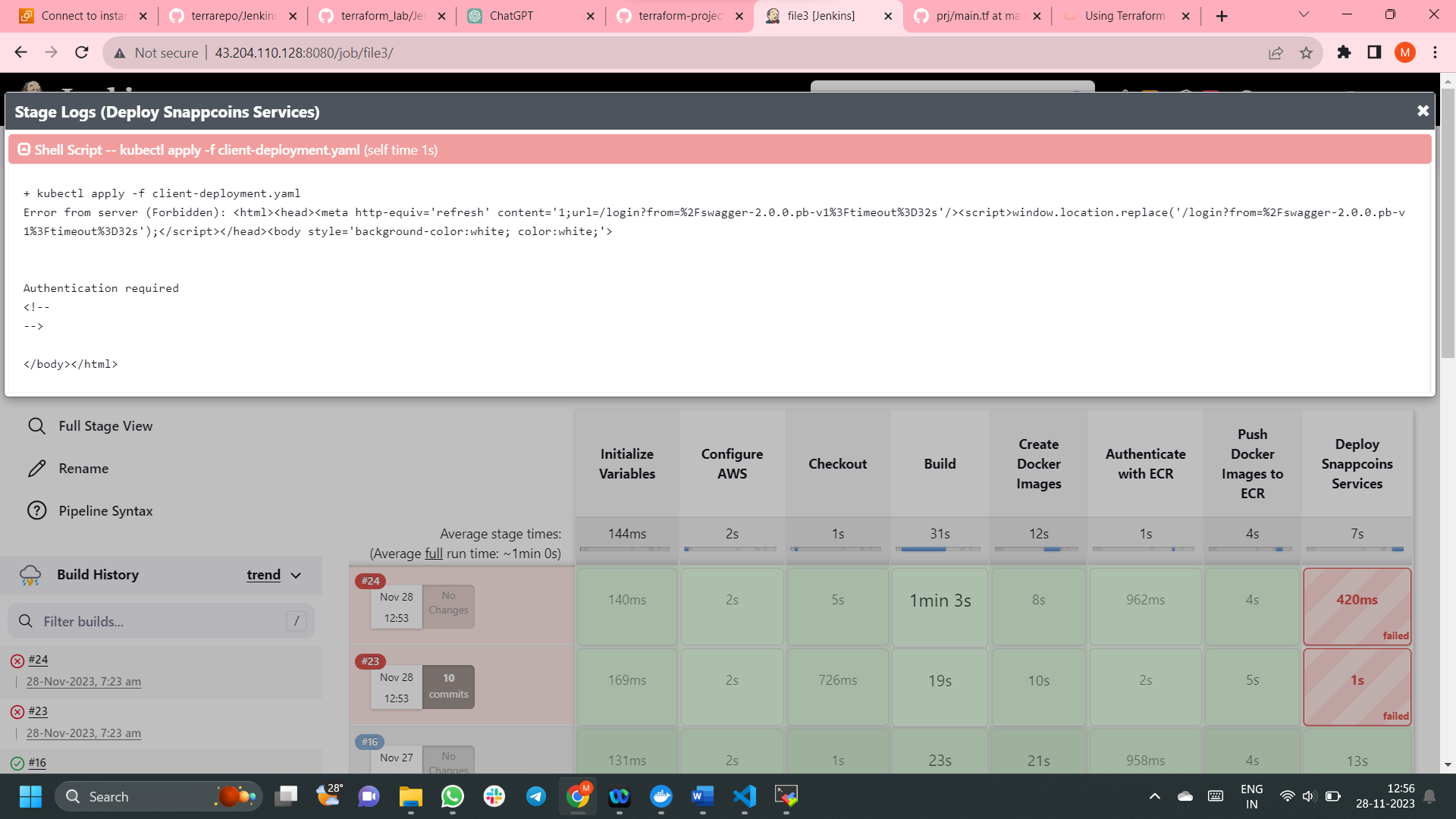
chmod +x ./kubectl

mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

kubectl version

1. authentication error:

create a new directory and move kubectl config file to that directory and give necessary permissions to avoid the below error



**11.** **Stage: Print Ingress IP**

**Purpose:**

To retrieve and display the external IP address of the Istio Ingress Gateway, which is responsible for handling external traffic and routing it to the appropriate services within the Kubernetes cluster.

Steps:

script:

The script block allows the execution of a scripted block within the Jenkins Pipeline. In this case, it's used to define a Groovy script that performs the necessary actions.

Script Steps:

Execute kubectl Command:

def externalIP = sh(script: 'kubectl get svc istio-ingressgateway -n istio-system -o jsonpath="{.status.loadBalancer.ingress[0].hostname}"', returnStdout: true).trim()

This line executes a kubectl command to query the external IP address of the Istio Ingress Gateway. The command uses JSONPath to extract the hostname from the service's status.

**Print External IP:**

echo "Ingress Gateway External IP: ${externalIP}": This line prints the obtained external IP address to the Jenkins console output. The echo statement is used to display a message in the Jenkins build log.

Explanation:

The Istio Ingress Gateway is a critical component that manages external traffic entering the Kubernetes cluster.

The kubectl get svc istio-ingressgateway -n istio-system command retrieves information about the Istio Ingress Gateway service.

The JSONPath expression ("{.status.loadBalancer.ingress[0].hostname}") is used to extract the external hostname (IP address) from the service's status.

The obtained external IP address is then printed to the Jenkins console output using the echo statement.

**12.Stage: Conditional Job Trigger**

Purpose:

To prompt the user for confirmation and, based on the response, either trigger the execution of another job or skip the trigger.

Steps:

script:

The script block allows the execution of a scripted block within the Jenkins Pipeline. In this case, it's used to define a Groovy script that performs the necessary actions.

**Script Steps:**

**User Confirmation Prompt:**

def shouldTriggerNextJob = input(...)

This line uses the input step to prompt the user for confirmation. The user is presented with a dialog box containing a message and a boolean parameter with a default value of false.

**Conditional Execution:**

if (shouldTriggerNextJob) { ... } else { ... }: This block checks the user's response. If the user confirms (selects 'Yes'), the script triggers the execution of another job named 'qa' using the build step. If the user declines (selects 'No'), it prints a message indicating that the next job won't be triggered.

Explanation:

The purpose of this stage is to introduce a manual confirmation step before proceeding with the next job in the pipeline.

The input step is a powerful feature in Jenkins that allows for user interaction within a pipeline. In this case, it presents a dialog box asking the user whether they want to trigger the next job.

The if statement then checks the user's response. If the user confirms by selecting 'Yes' (or the default is set to true), the script triggers the execution of another job named 'qa' with the build step.

If the user declines by selecting 'No' (or the default is set to false), the script prints a message indicating that the next job won't be triggered.

**13. Stage-Post-Build Section:**

Purpose:

To interactively ask the user for confirmation before destroying Terraform resources, and based on the user's response, either proceed with the destruction or skip it.

**Steps:**

**always Block:**

always { ... }: This block ensures that the contained script is executed regardless of the build result (success, failure, or unstable).

**Script Steps:**

**User Confirmation Prompt:**

def userInput = input(...): This line uses the input step to prompt the user for confirmation. The user is presented with a dialog box containing a message and a boolean parameter with a default value of false.

**Conditional Execution:**

if (userInput) { ... } else { ... }: This block checks the user's response. If the user confirms (selects 'Yes'), the script proceeds with the destruction of Terraform resources using the terraform destroy command. If the user declines (selects 'No'), it prints a message indicating that the destruction is skipped.

**Destroy Terraform Resources:**

sh 'terraform destroy -var="environment\_name=dev" --auto-approve': This line executes the terraform destroy command with specific variables, automatically approving the destruction without manual confirmation.

**Skipping Terraform Destruction:**

echo 'Skipping Terraform destruction.': This line prints a message indicating that the Terraform destruction is skipped if the user declines the confirmation.

Explanation:

This post-build section adds an extra layer of safety by requiring user confirmation before destroying Terraform resources. The input step allows users to interactively decide whether to proceed or skip the destruction step.

If the user confirms (selects 'Yes'), the script executes the terraform destroy command, ensuring the destruction of Terraform resources.

If the user declines (selects 'No'), the script skips the destruction step and prints a message to inform users that the destruction is skipped.

**14. Stage- Update Route 53 DNS Records**

This declares a Jenkins pipeline stage titled "Update Route 53 DNS Records"

The steps block contains the scripted pipeline code. The script block is used for writing Groovy script code

This code retrieves the external IP of the Istio Ingress Gateway by running a kubectl command. The obtained IP is stored in the externalIP variable, and it is echoed for informational purposes.

This block iterates through each service specified in the ADDITIONAL\_SUBDOMAINS array and updates the Route 53 DNS records using the AWS CLI (aws route53 change-resource-record-sets). It constructs the necessary parameters dynamically based on the external IP of the Istio Ingress Gateway.

These lines close the loop, the script block, the steps block, and the stage, defining the completion of this Jenkins pipeline stage.

Summary:

This stage retrieves the external IP of the Istio Ingress Gateway, and for each specified service, it updates the corresponding Route 53 DNS record using the AWS CLI. This is useful for dynamically managing DNS records based on the external IP of a service in a CI/CD pipeline.