DevSecOps-Demo Project

Step by step for pratices

**Prerequisites**

Before diving into the project, make sure you have the following prerequisites in place:

1. **Local Environment Setup**:

* **Terraform and AWS CLI**: Install and configure Terraform and AWS CLI on your local machine. Basic knowledge of these tools is necessary.
* **Basic Knowledge**: Ensure basic knowledge of Terraform, AWS CLI, and familiarity with cloud concepts.

2. **Jenkins Server Deployment:**

* **Git**: Basic knowledge of Git commands is required.
* **AWS EC2**: Understanding of AWS EC2 instances and security groups.

**3. Jenkins Configuration:**

* **Jenkins**: Familiarity with Jenkins and basic Jenkins pipeline concepts.
* **Docker, Sonarqube, Terraform, Kubectl, AWS CLI, Trivy**: Basic knowledge of these tools is necessary.

**4. ArgoCD Setup:**

* **Kubernetes**: Basic knowledge of Kubernetes concepts.
* **ArgoCD**: Familiarity with ArgoCD and understanding of continuous deployment concepts.

**5. Pipeline Configuration:**

* **Jenkins Plugins:** Understanding of Jenkins plugins, especially AWS Credentials, and Pipeline: AWS Steps.
* **Tools Configuration:** Basic knowledge of configuring tools like Docker, NodeJS, OWASP Dependency-Check, and SonarQube in Jenkins.

**GitHub Repository for the Project-** <https://github.com/nhuluong2024/DevSecOps-Demo>

**We will install Terraform & AWS CLI to deploy our Jenkins Server(EC2) on AWS**

Install & Configure Terraform and AWS CLI on your local machine to create Jenkins Server on AWS Cloud

**Terraform Installation Script**

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg - dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg  
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list  
sudo apt update  
sudo apt install terraform -y

**AWSCLI Installation Script**

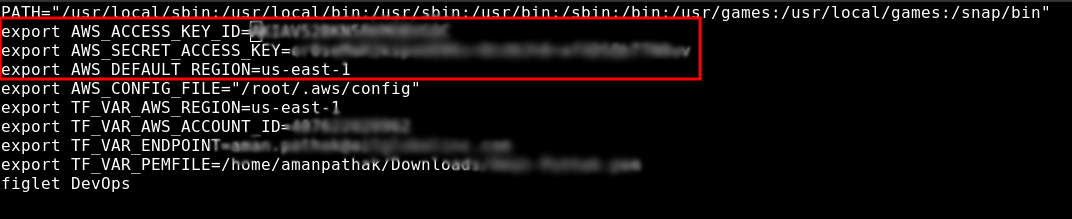
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"  
sudo apt install unzip -y  
unzip awscliv2.zip  
sudo ./aws/install

Now, Configure both the tools

**Configure Terraform**

Edit the file /etc/environment using the below command and add the highlighted lines and add your keys at the blur space.

sudo vim /etc/environment

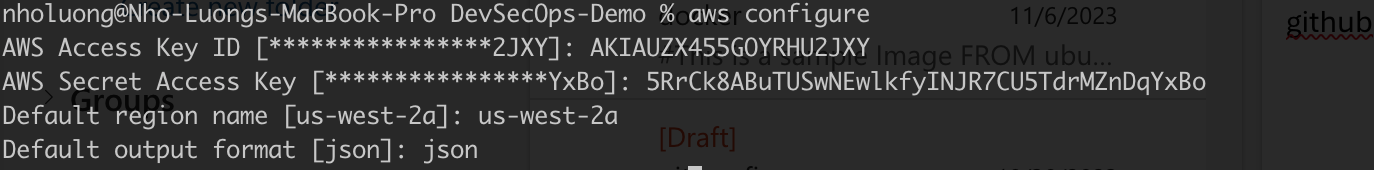


After doing the changes, restart your machine to reflect the changes of your environment variables.

**Configure AWS CLI**

Run the below command, and add your keys

aws configure

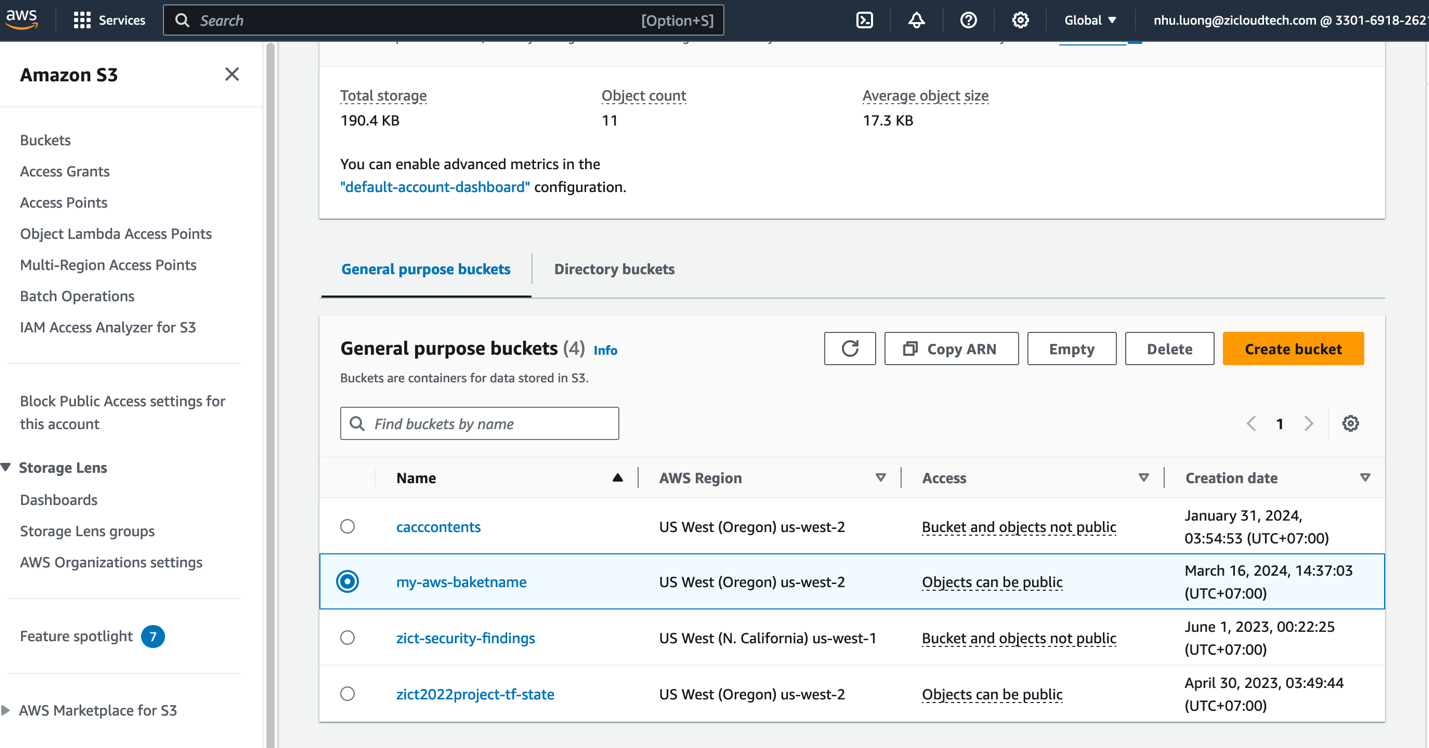


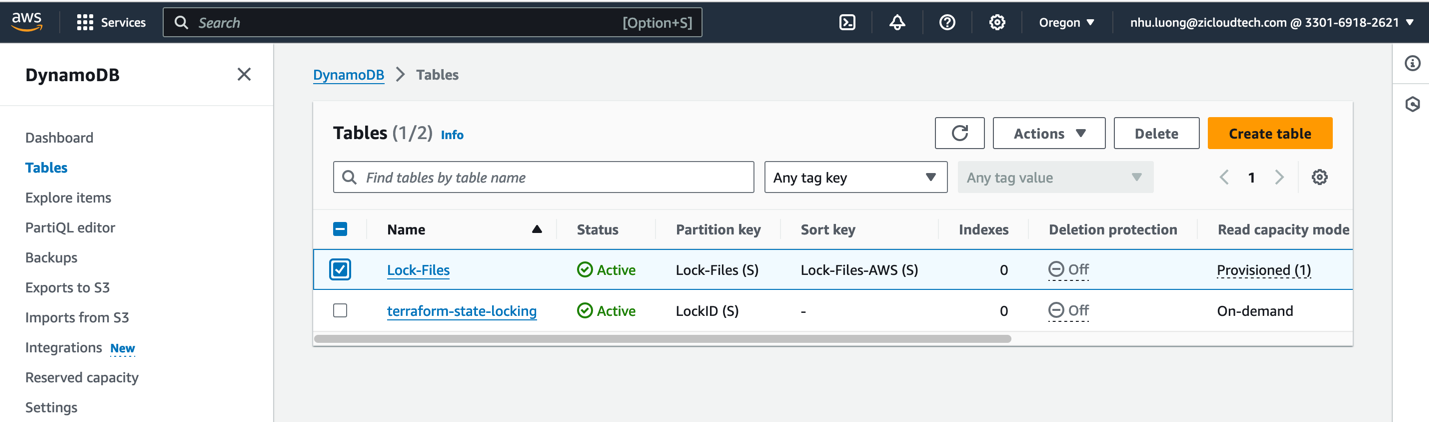
# ****Deploy the Jenkins Server(EC2) using Terraform****

Clone the Git repository

Navigate to the **Jenkins-Server-TF**

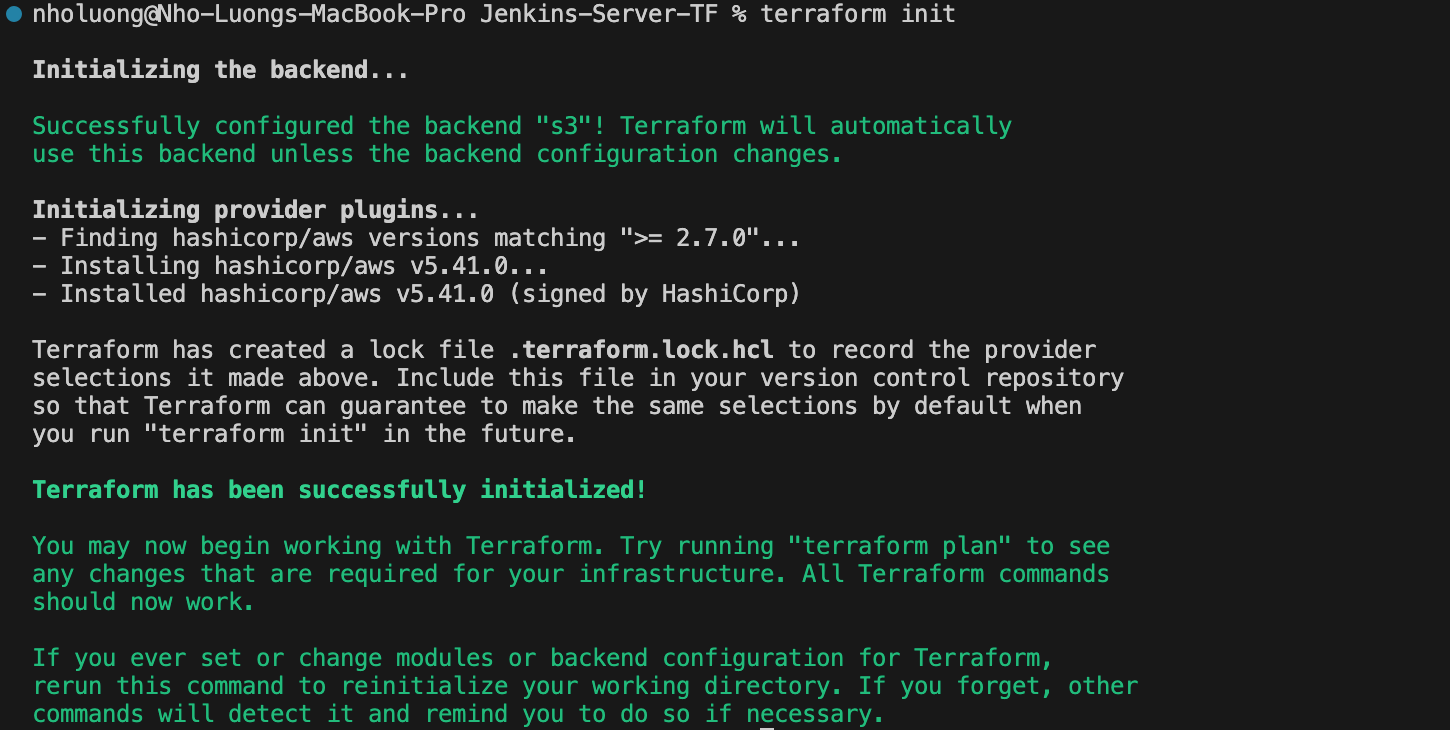
Do some modifications to the backend.tffile such as changing the **bucket** name and **dynamodb** table(make sure you have created both manually on AWS Cloud).





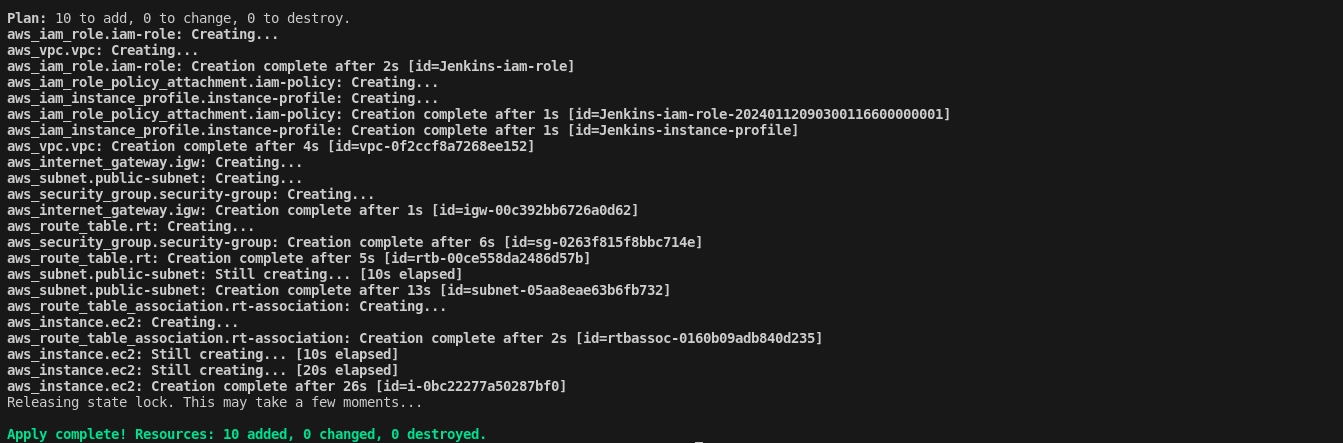
Initialize the backend by running the below command

terraform init



Now, run the below command to create the infrastructure on AWS Cloud which will take 3 to 4 minutes maximum

terraform apply -var-file=variables.tfvars --auto-approve -lock=false



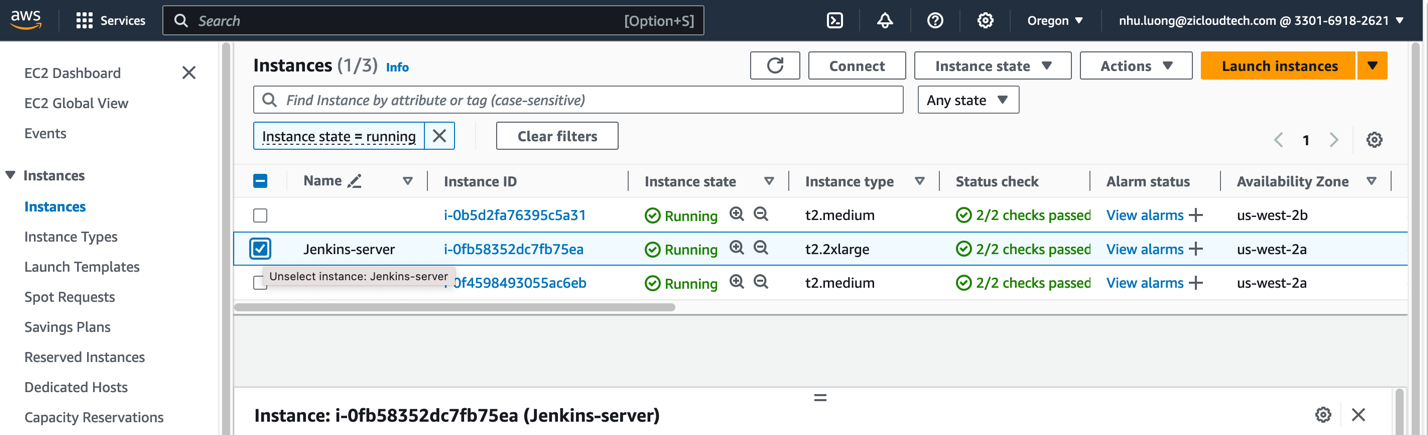
Now, connect to your Jenkins-Server by clicking on Connect.

cd Jenkins-Pipeline-Code

chmod 400 jenkins-key.pe

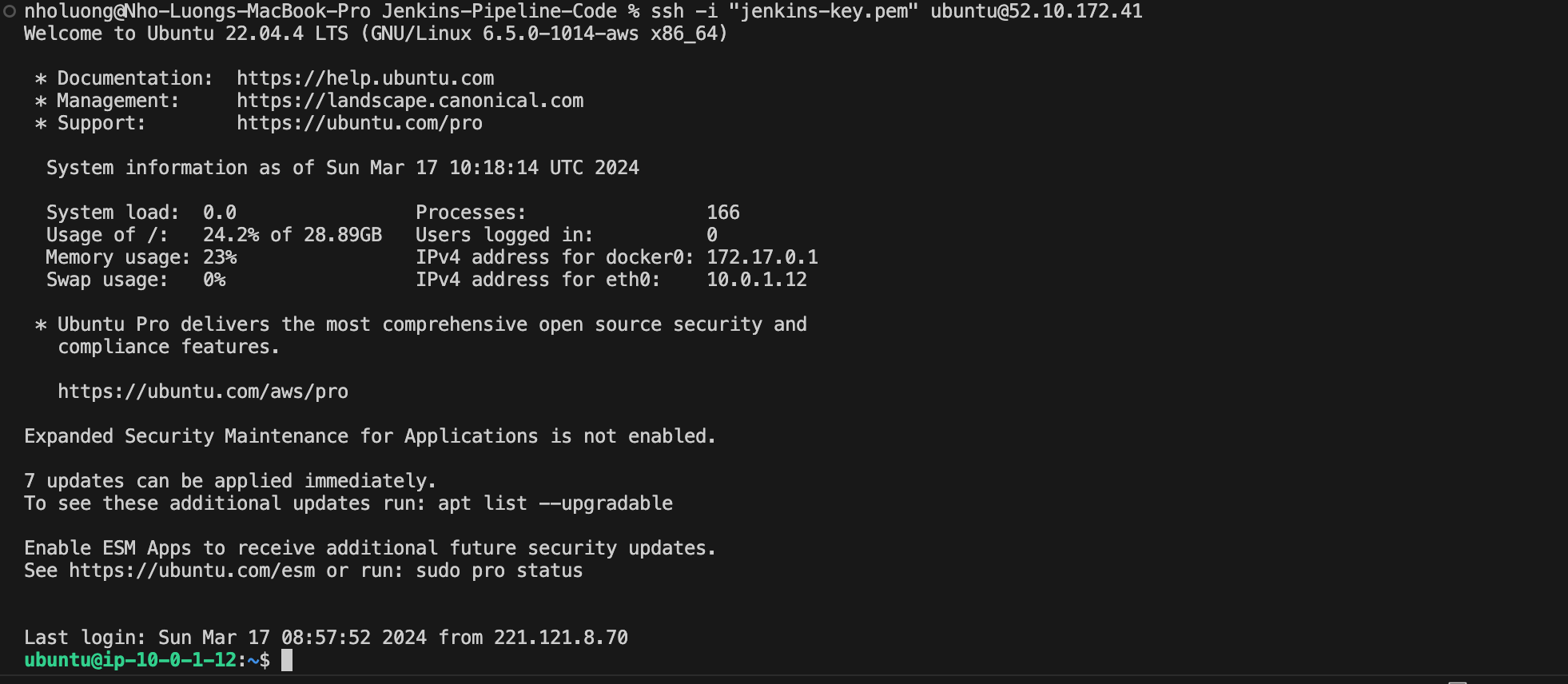
ssh -i "jenkins-key.pem" ubuntu@your public ip

sudo su



# ****Configure the Jenkins****

Now, we logged into our **Jenkins server.**



We have installed some services such as Jenkins, Docker, Sonarqube, Terraform, Kubectl, AWS CLI, and Trivy.

**Install helm:**

curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null

sudo apt-get install apt-transport-https --yes

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update

sudo apt-get install helm -y

Let’s validate whether all our installed or not.

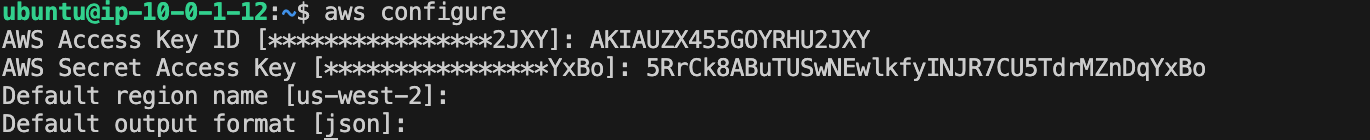
helm version

jenkins --version  
docker --version  
docker ps  
terraform --version  
kubectl version  
aws --version

# ****We will deploy the EKS Cluster using Jenkins****

Now, go back to your Jenkins Server **terminal** and configure the AWS.

(Input your aws access key, secrect key)



Click on **Manage Jenkins**

Select the **Available plugins** and install the following plugins and click on **Install**

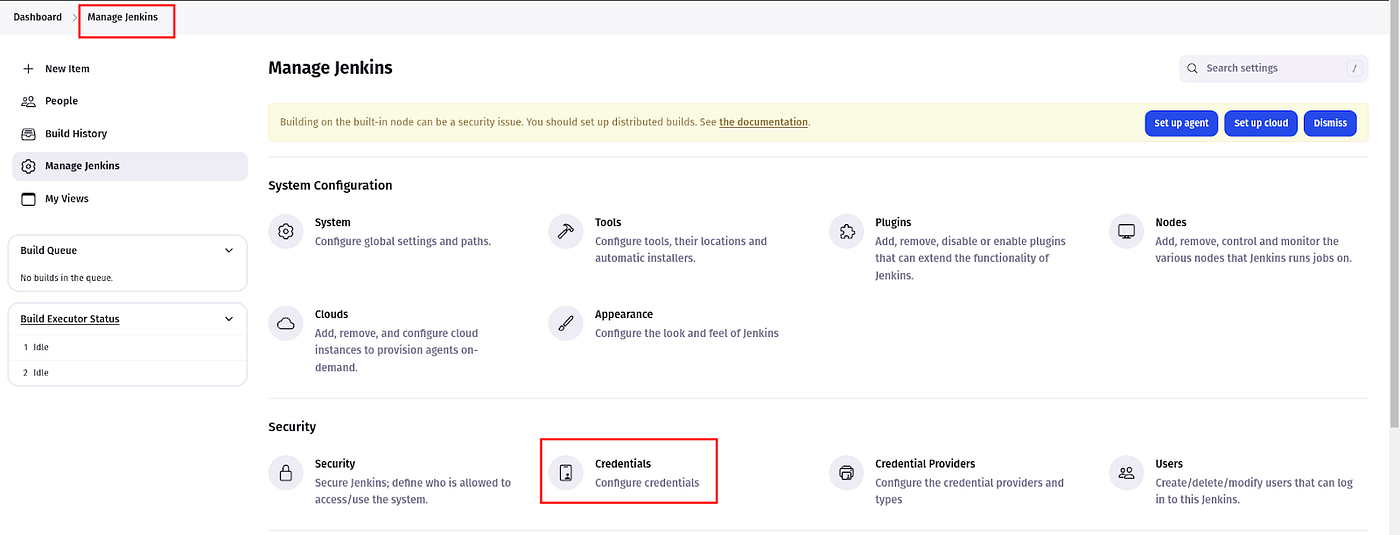
AWS Credentials  
Pipeline: AWS Steps

Docker  
Docker Commons  
Docker Pipeline  
Docker API  
docker-build-step  
Eclipse Temurin installer  
NodeJS  
OWASP Dependency-Check  
SonarQube Scanner

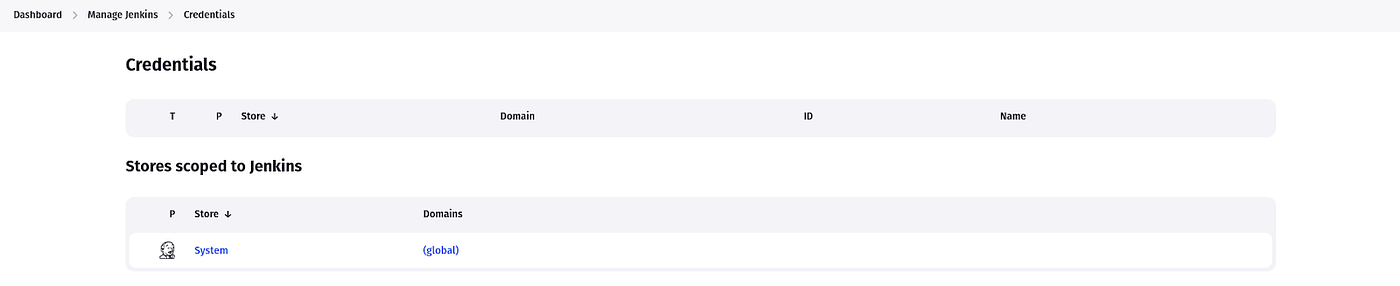
Once, both the plugins are installed, restart your Jenkins service.

Now, we have to set our AWS credentials on Jenkins

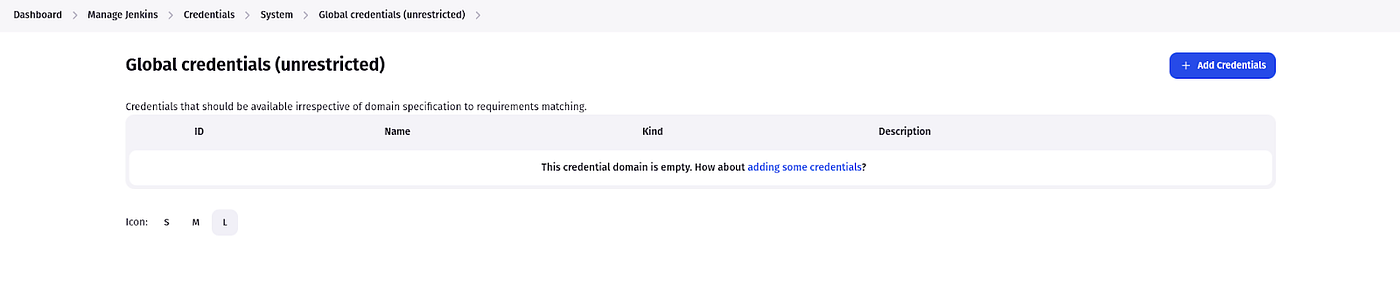
Go to **Manage Plugins** and click on **Credentials**



Click on **global.**



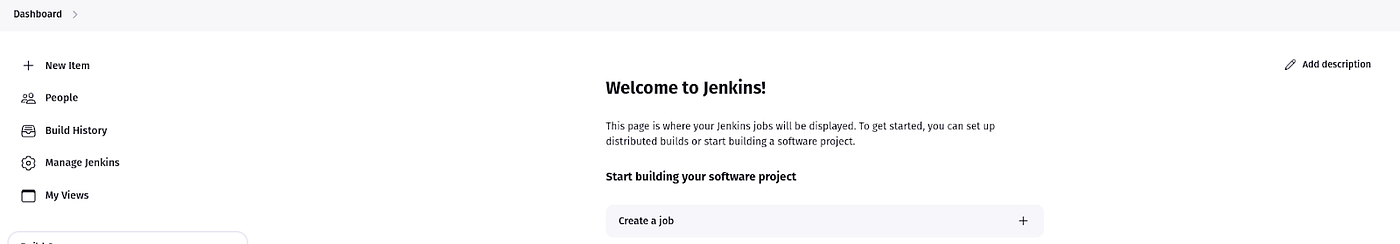
Click on **Add credentials**



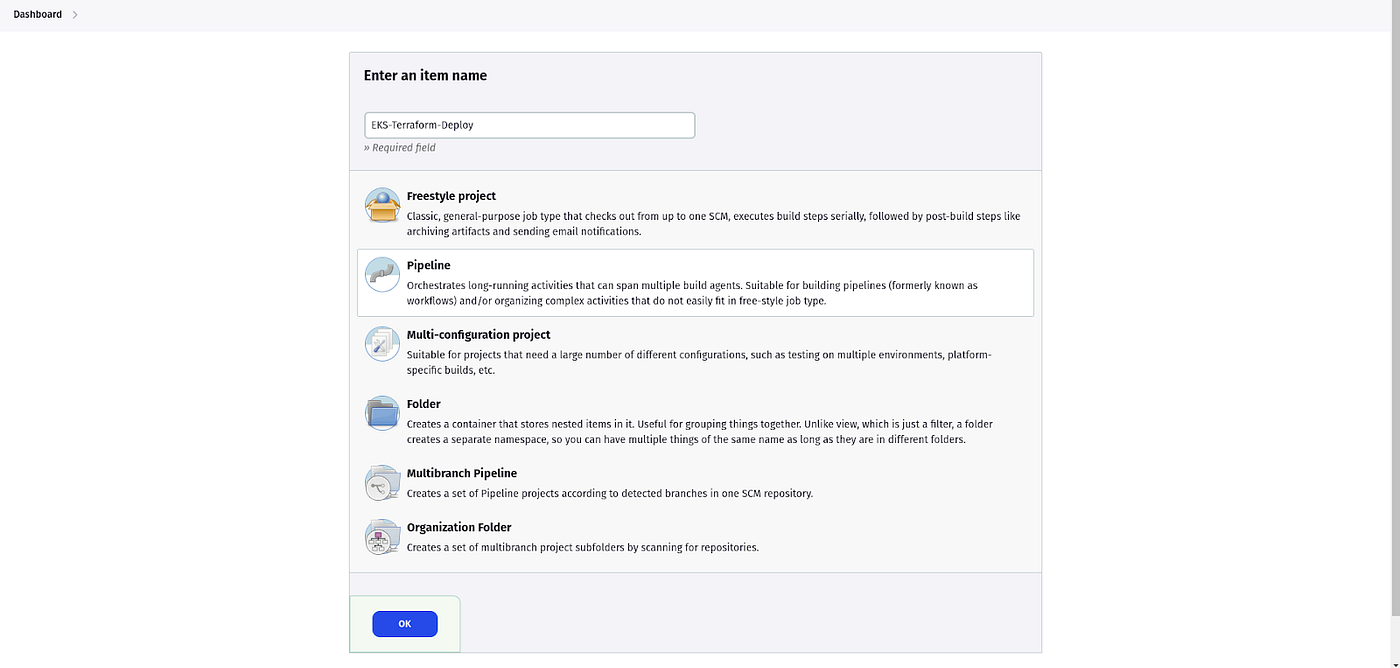
Select **AWS Credentials** as **Kind** and add **the ID** same as shown in the below snippet except for your AWS Access Key & Secret Access key and click on **Create.**



Now, Go to the **Dashboard** and click **Create a job**



Select the **Pipeline** and provide the name to your **Jenkins Pipeline** then click on **OK.**



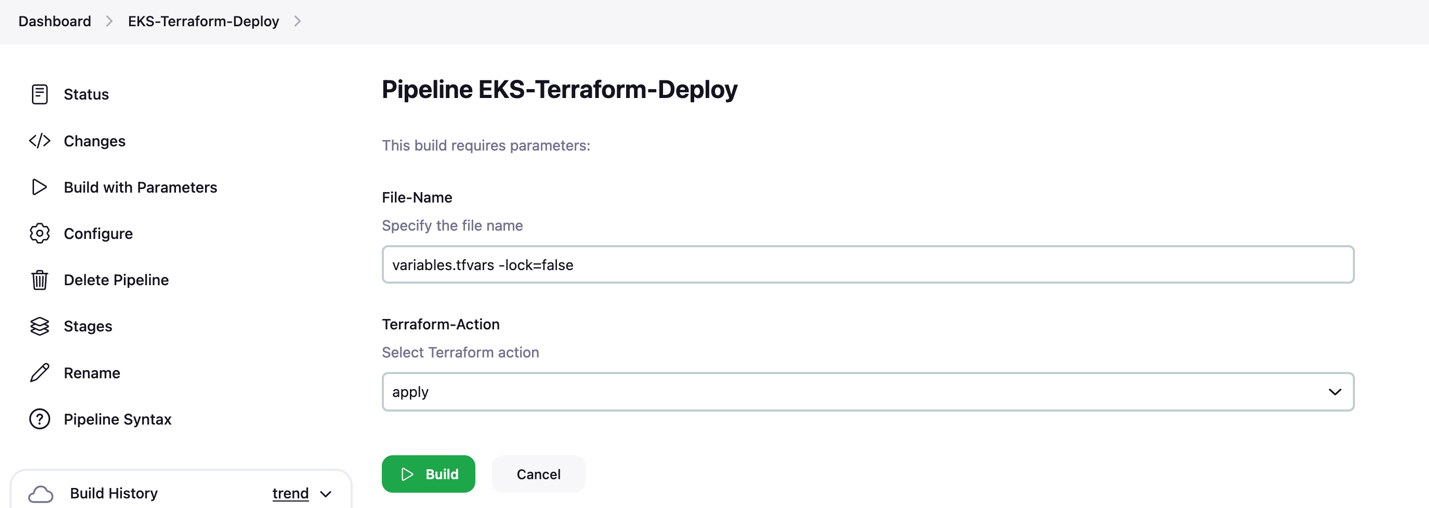
Now, Go to the GitHub Repository in which the Jenkins Pipeline code is located to deploy the EKS service using Terraform.

After pasting the Jenkinsfile code, click on **Save** & **Apply.**



Click on **Build**

**Notes: if failure then click** [Build with Parameters](http://54.214.142.157:8080/job/EKS-Terraform-Deploy/build?delay=0sec) and input File-Name and build again: variables.tfvars -lock=false



You can see our **Pipeline**

Now, we will configure the **EKS** **Cluster** on the **Jenkins Server**

Run the below command to configure the EKS Cluster on jenkins server

Now, connect to your Jenkins-Server by clicking on Connect.

cd Jenkins-Pipeline-Code

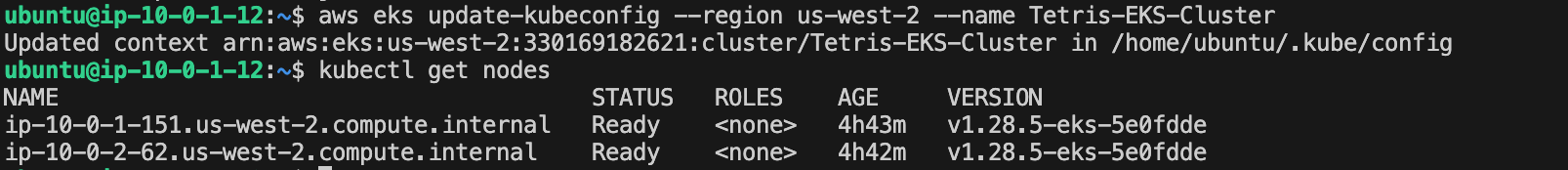
chmod 400 jenkins-key.pe

ssh -i "jenkins-key.pem" ubuntu@your public ip

aws eks update-kubeconfig --region us-west-2 --name Tetris-EKS-Cluster

To validate whether the EKS Cluster was successfully configured or not. Run the below command

kubectl get nodes

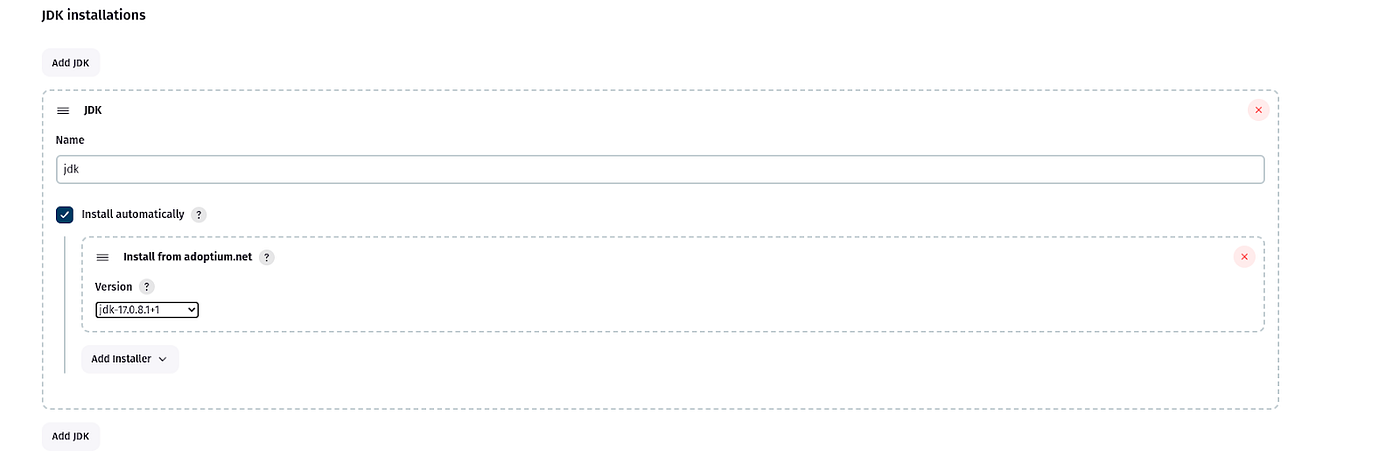


Now, we have to configure the installed plugins.

Go to **Dashboard -> Manage Jenkins -> Tools**

We are configuring jdk

Search for jdk and provide the configuration like below snippet.



Now, we will configure nodejs

Search for node and provide the configuration like the below snippet.





Now, we will configure Scanner installations

Search for Scanner and provide the configuration like the below snippet.

SonarQube Scanner: sonar-scanner

SONAR\_RUNNER\_HOME: /var/lib/jenkins/plugins/sonar/META-INF/maven/org.jenkins-ci.plugins/sonar



Go to **Dashboard -> Manage Jenkins -> Credentials**

Add your docker hub username and password in the respective field with ID **docker.**

Click on **Create**



Add GitHub credentials (nhuluongzicloudtec/ Abcd@123!)

Select the kind as **Secret text** and paste your GitHub Personal access token (ghp\_g6Q7sIacVasvFdzXa3wcAiQLnzTZES08M5wl) in **Secret** and keep other things as it is.

Click on **Create**

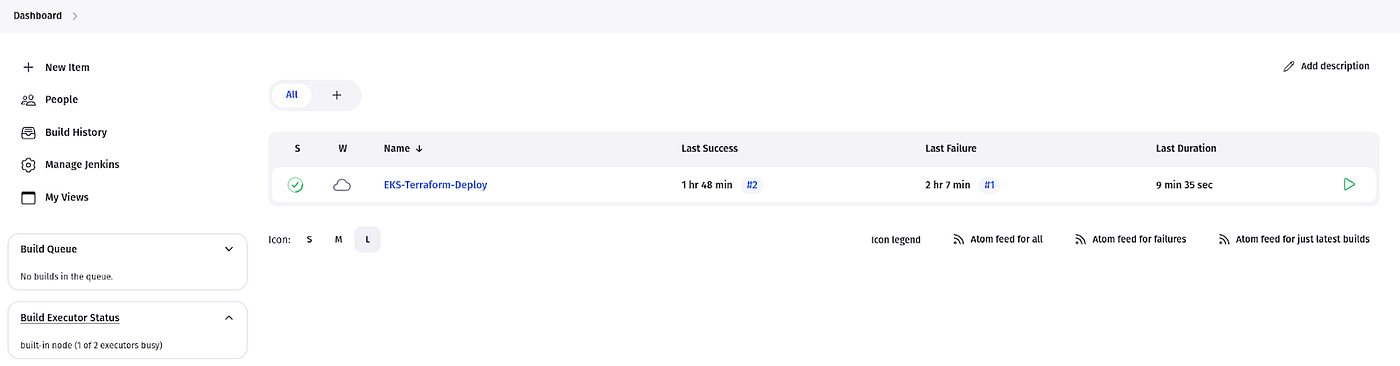
**Note:** If you haven’t generated your token then, you have it generated first then paste it in the Jenkins



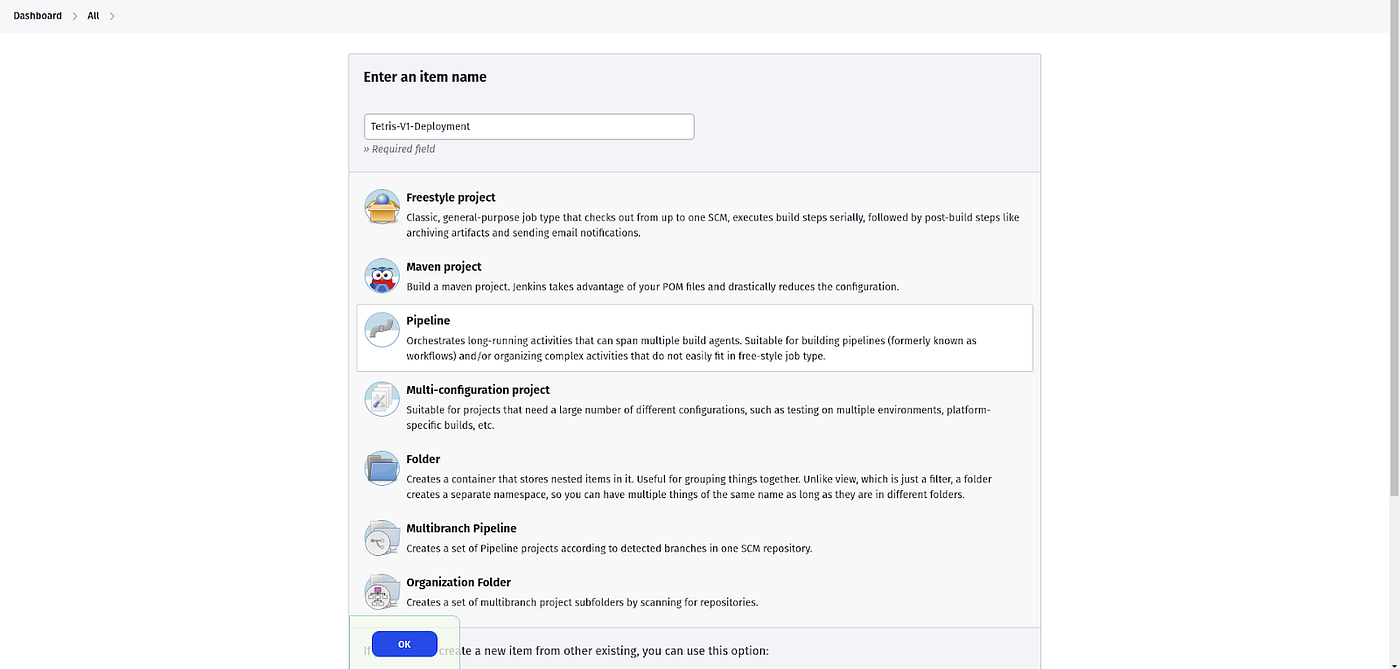
Now, we are ready to create our Jenkins Pipeline to deploy our Tetris Application.

Go to **Jenkins Dashboard**

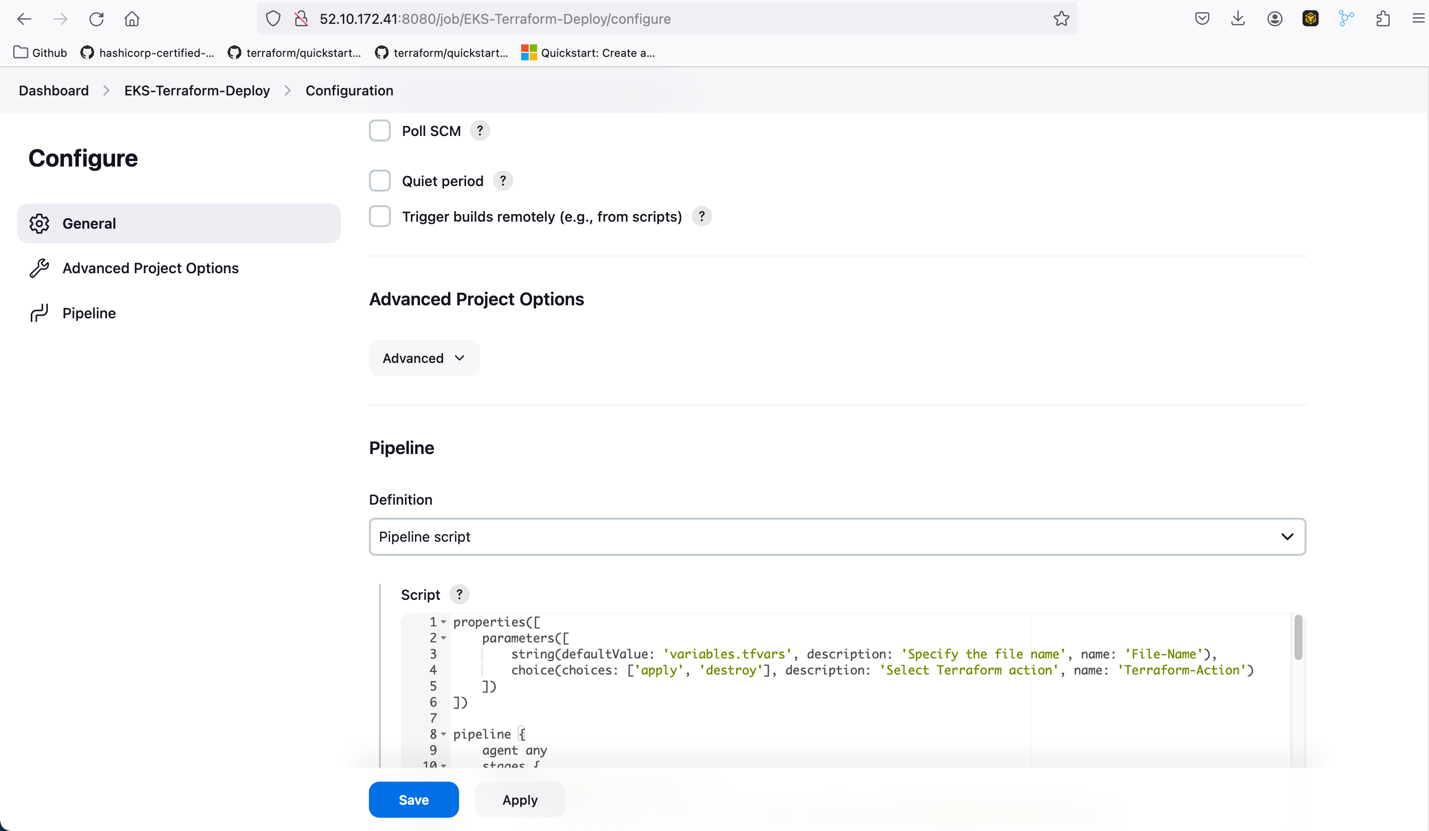
Click on **New Item**



Provide the name of your **Pipeline** and click on **OK.**

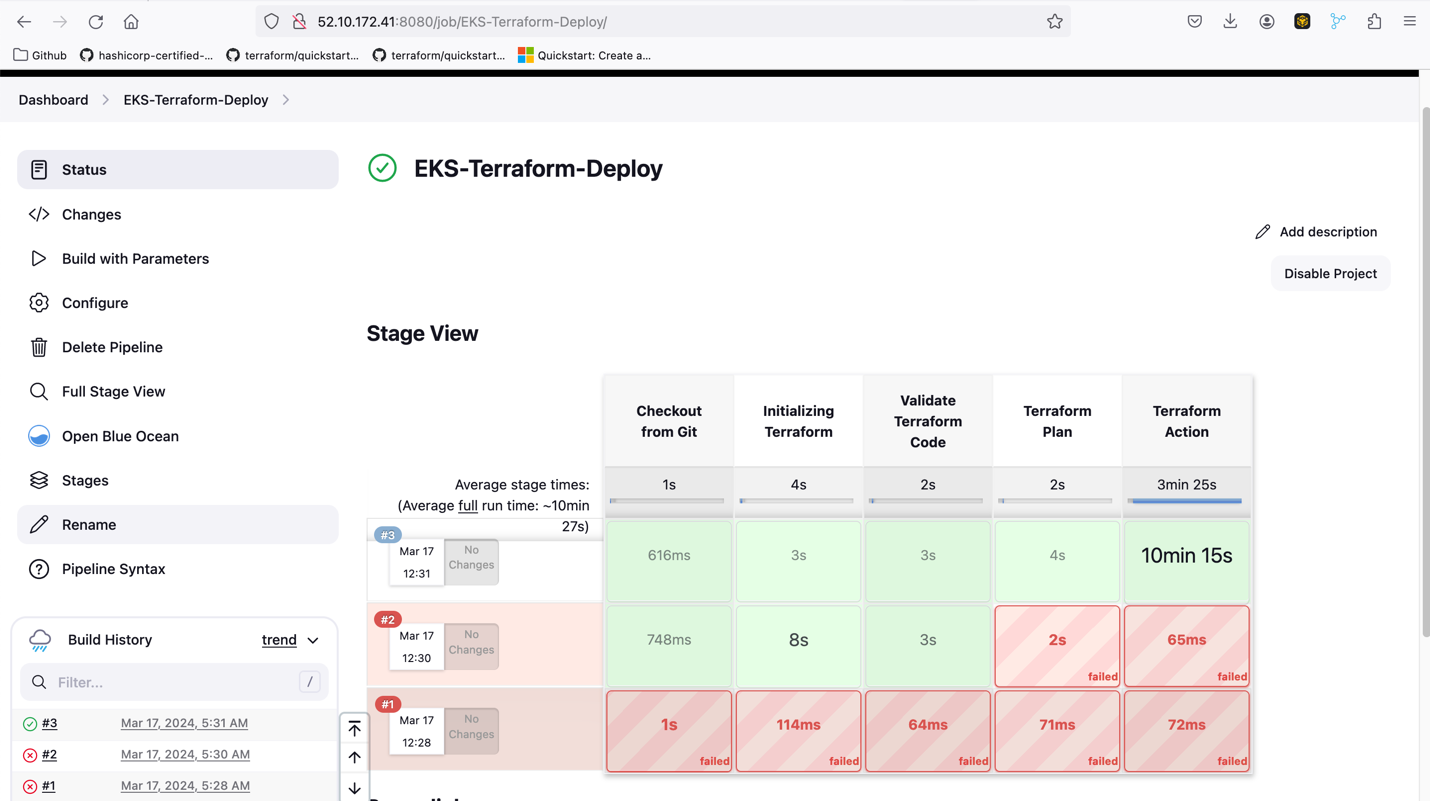


This is the Jenkinsfile to deploy Tetris Application Version 1 on EKS.



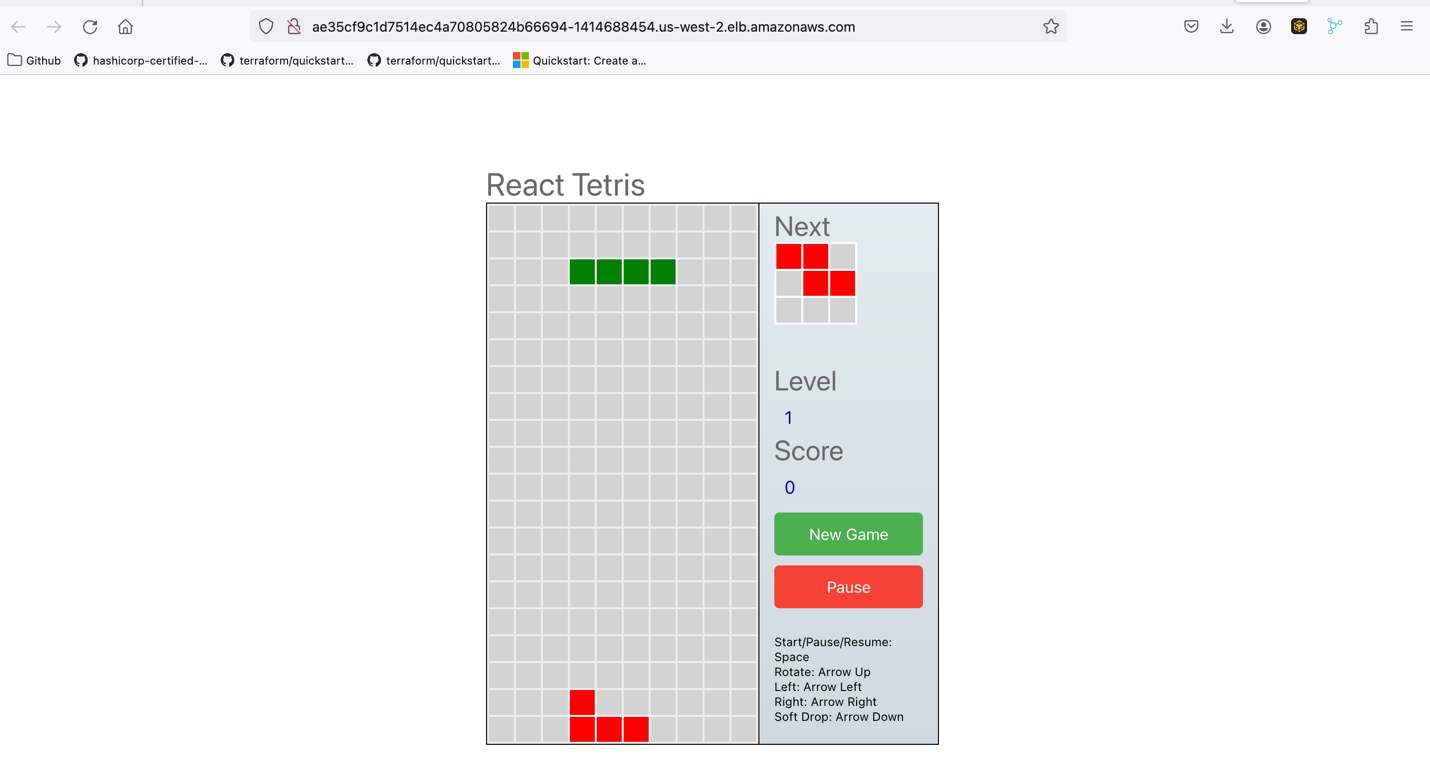
Now, click on the build.

Our pipeline was successful.



Copy the DNS name of your load balancer from ArgoCD Console or you can go to AWS Console and copy the Load Balancer and hit the DNS on your favorite browser to enjoy the **Tetris Game**.

<http://ae35cf9c1d7514ec4a70805824b66694-1414688454.us-west-2.elb.amazonaws.com/>



# ****We will deploy Tetris Application Version 2****

Now, suppose we have done some modifications to our previous version to make it more good in the sense of GUI or anything else. Then, we will have to deploy our **Version** **2** of our same application.

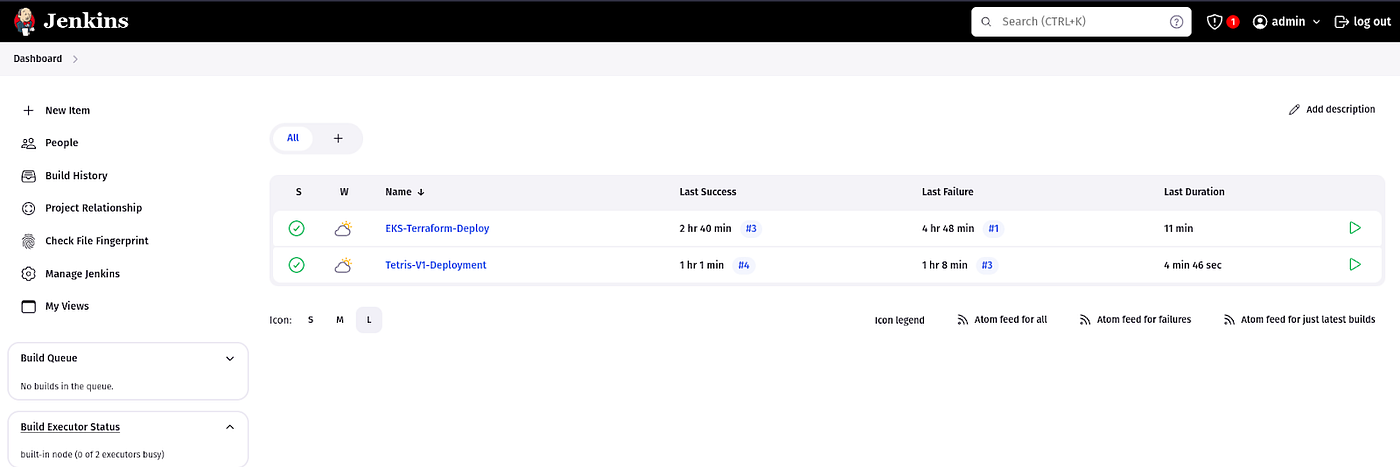
To do that, we will create a new pipeline. We can do it in the existing pipeline as well but this way you will be able to understand clearly.

We have a separate code for our Tetris Version 2. In which Dockerfile is present, so we will build the image and push it on docker and then update the same manifest file instead of v1 we will replace it with v2 manually first.

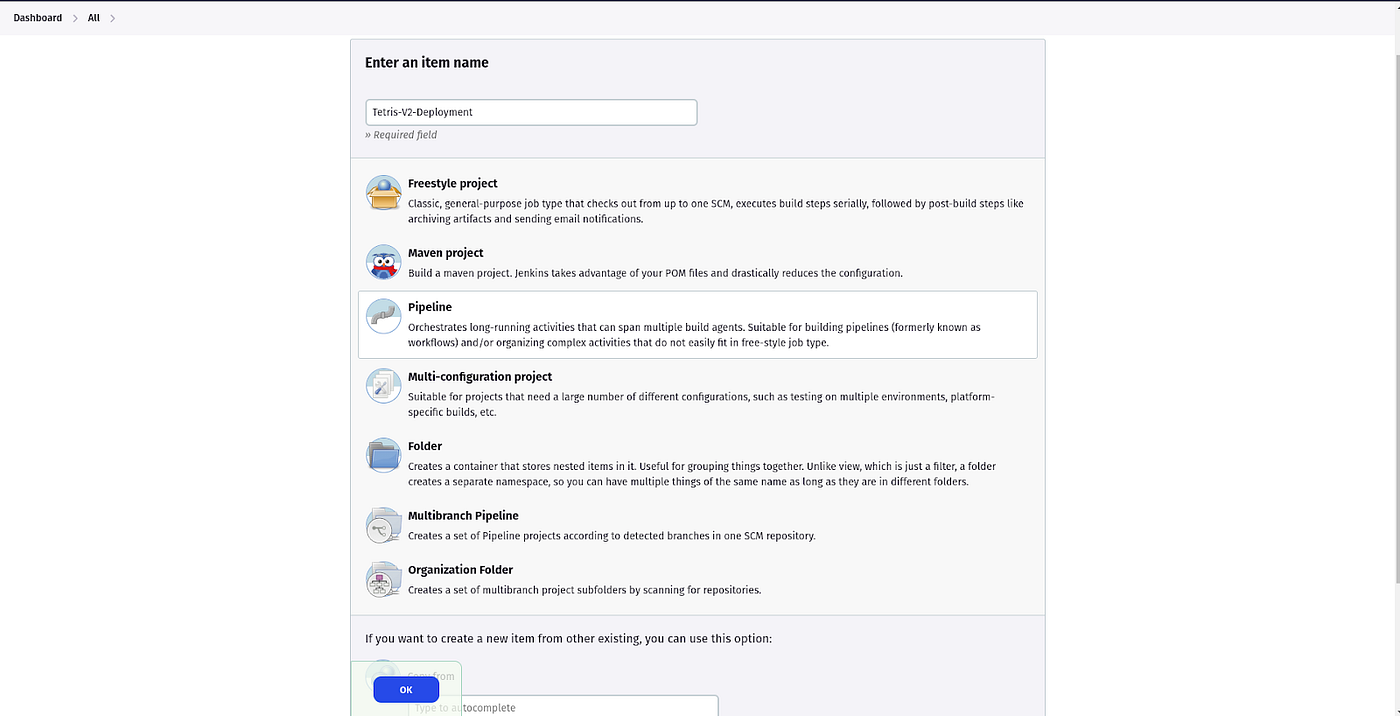
**Hope you get the high overview, what are we going to do next?**

Let’s make it and finish our project.

Go to **Jenkins** -> **Dashboard** and click on **New item**

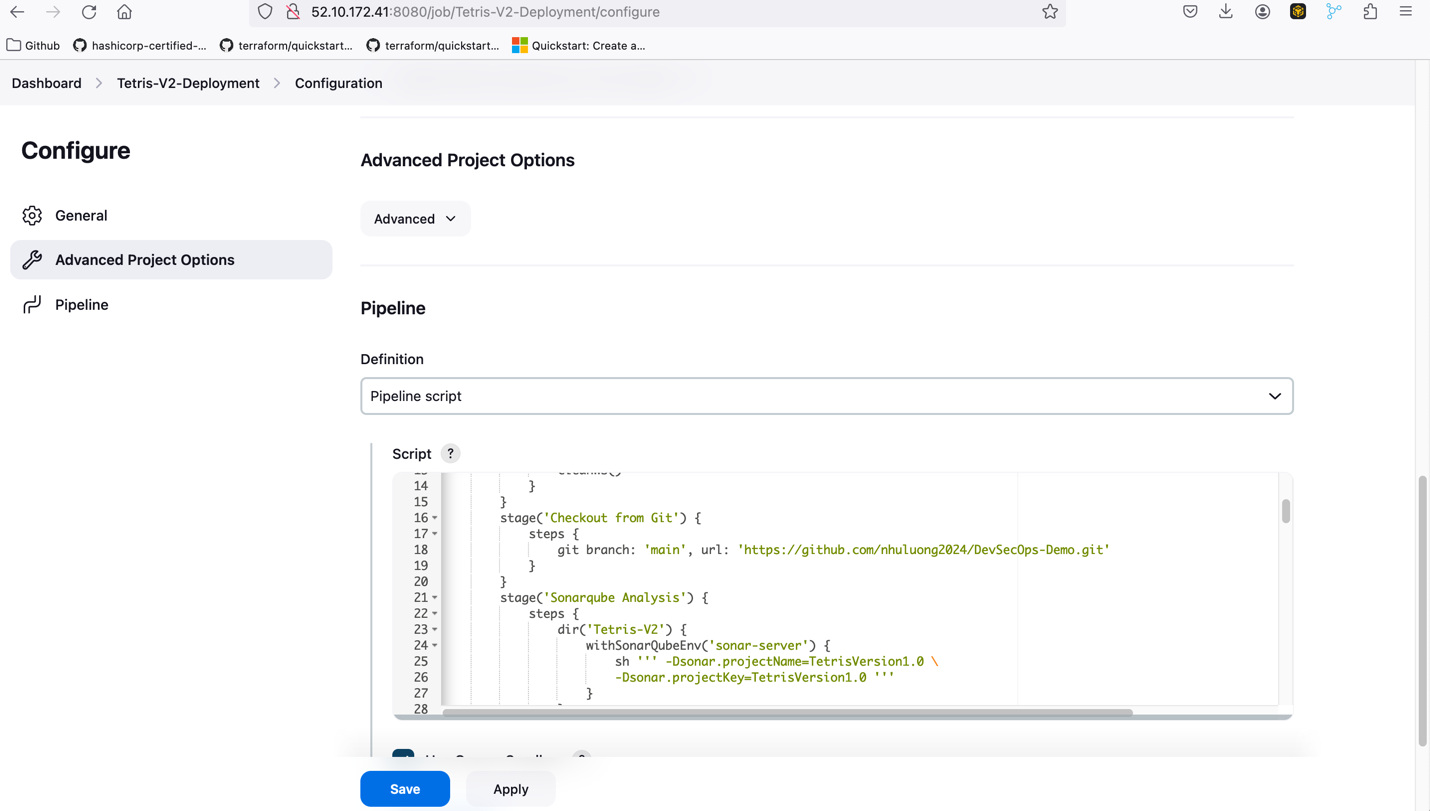


Provide the name to your **Pipeline name** and click on **OK.**



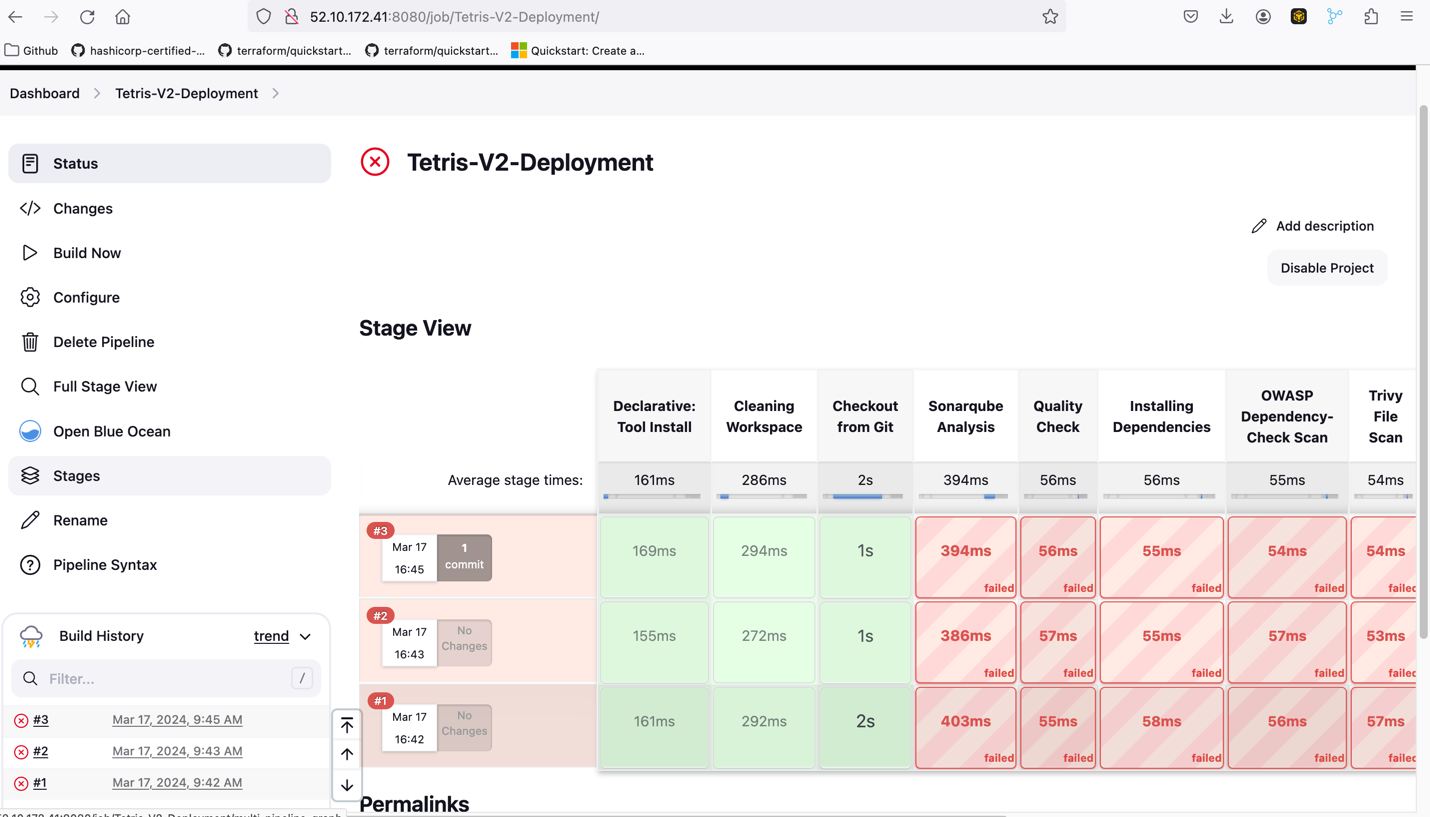
This is the Jenkinsfile to deploy Tetris Application Version 2 on EKS.

Click **Apply & Save.**



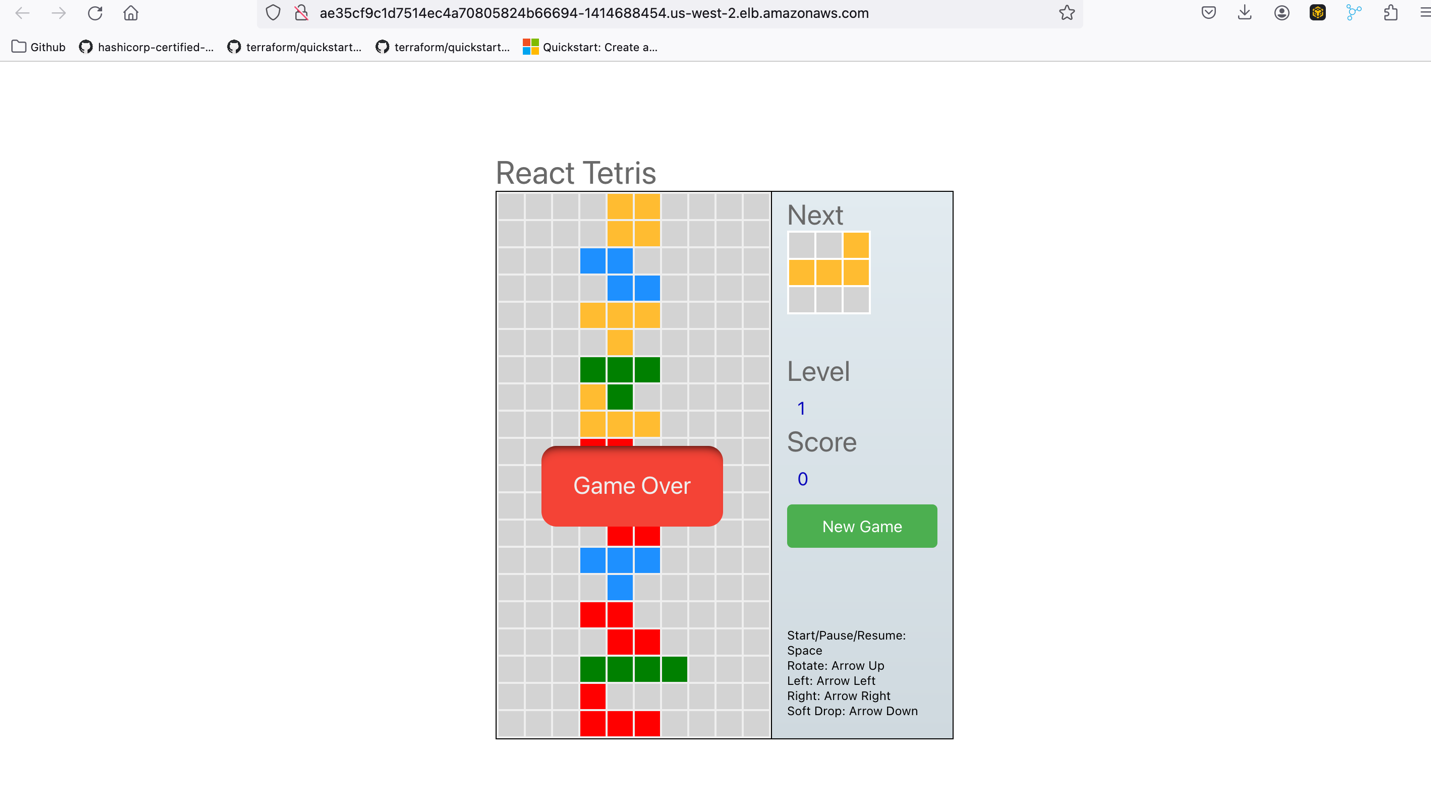
Before going to **build** the pipeline, update the manifest file.

Now, Once you click on the build to deploy our Tetris Application Version 2.   
You will see our **pipeline**



<https://ade96756e99b341c882da8e0342f560c-902802644.us-west-2.elb.amazonaws.com/applications/tetris?view=tree&resource=>

Now, you can enjoy the game.



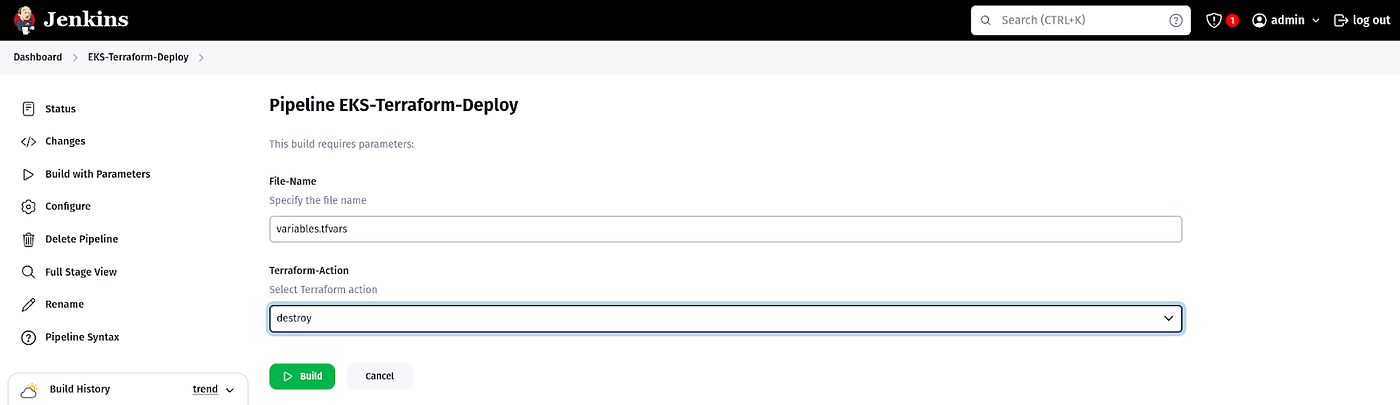
# ****Cleanup****

# ****Step 9: We will destroy the created AWS Resources****

Delete both the created **LoadBalancer** manually.

Select the **EKS-Terraform-Deploy** Pipeline.

Click on **Build with Parameters** and select the **destroy** and click on **Build.**



The Pipeline ran successfully which means the EKS Cluster has been deleted.

Now, we have to delete our **Jenkins Server**.

To do that, just run the below command on your local machine from where you create Jenkins Server.

terraform destroy -var-file=variables.tfvars --auto-approve -lock=false

Happy Learning