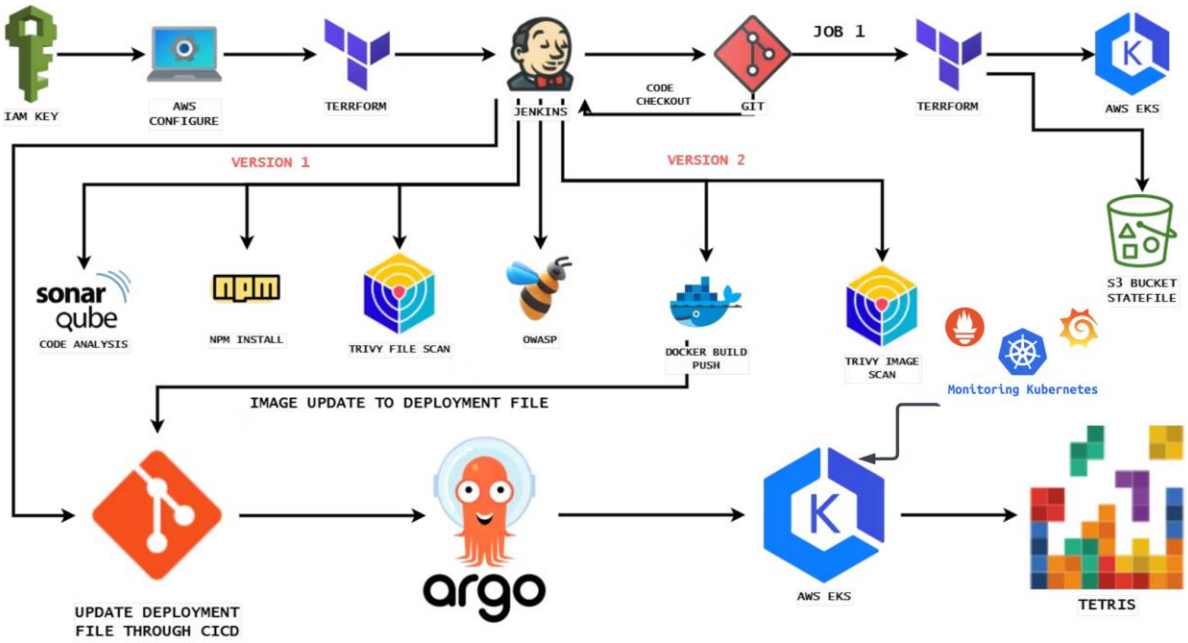
Automating Tetris Deployments Using DevSecOps



GitHub REPOSITORIES :

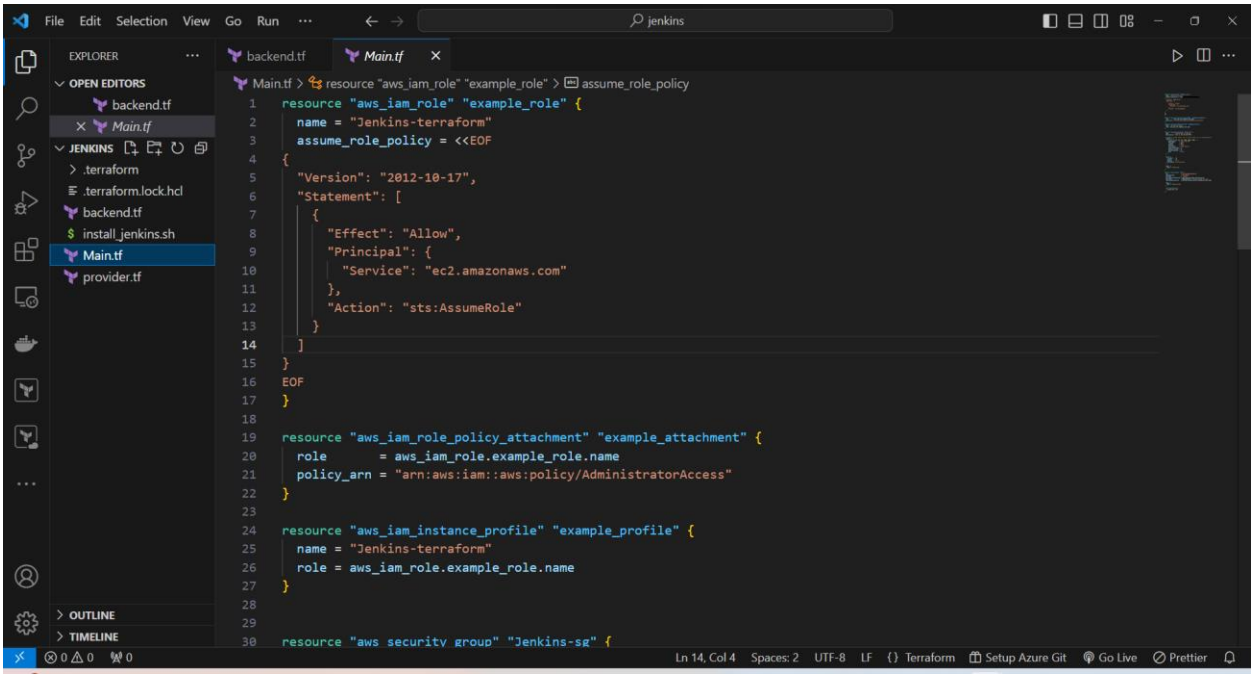
TETRIS-VERSION1 : https://github.com/Narasimha76/Tetrisv1

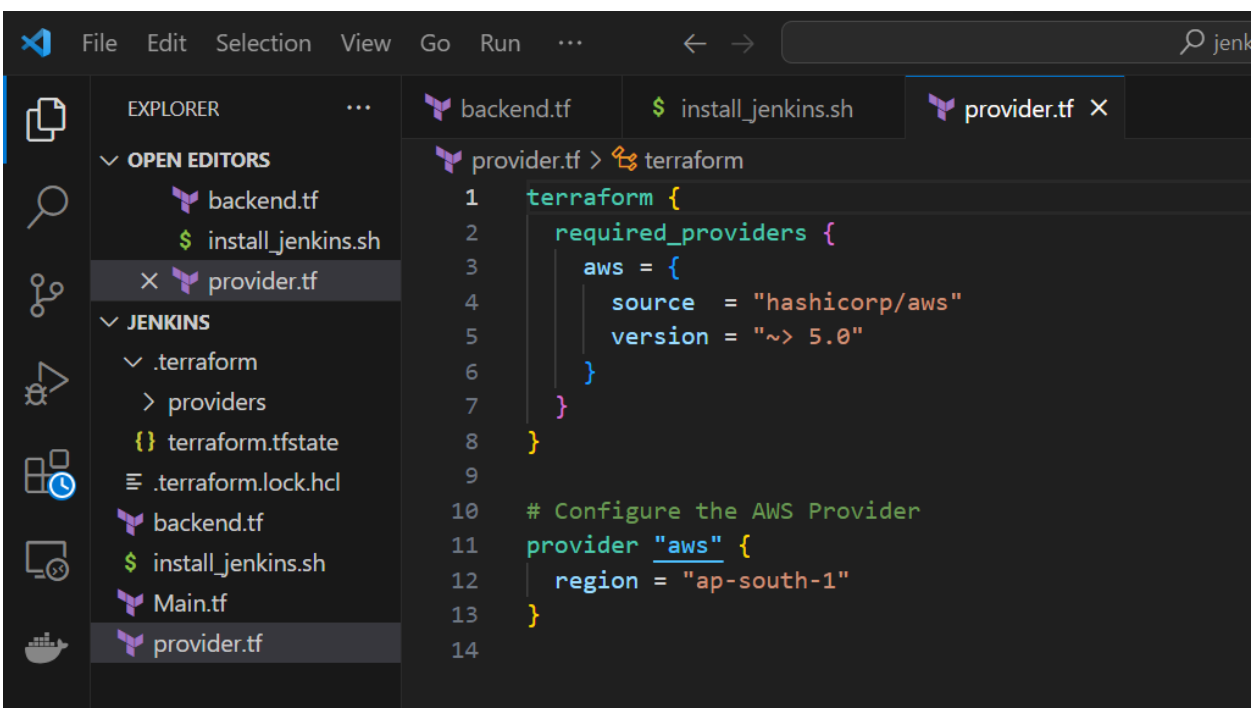
TETRIS-VERSION2 : https://github.com/Narasimha76/TetrisV2

TETRIS_MANIFEST : https://github.com/Narasimha76/Tetris-manifest

Step1: Terraform Provisioning

- Provision an AWS EC2 instance using Terraform, incorporating a bash script. This script orchestrates the installation of Docker, SonarQube, Trivy, OWASP, Terraform, AWS CLI, Jenkins with JDK, and Kubectl on the EC2 instance.





backend.tf

```
terraform {
  backend "s3" {
    bucket = "narasimha-v1" # Replace with your actual S3 bucket name
    key    = "JENKINS/terraform.tfstate"
    region = "ap-south-1"
  }
}
```

install_jenkins.sh

```
#!/bin/bash
sudo apt update -y
wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | tee /etc/apt/keyrings/adoptium.asc
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc] https://packages.adoptium.net/artifactory/deb $(awk -F= '/^VERSI
sudo apt update -y
sudo apt install temurin-17-jdk -y
/usr/bin/java --version
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.as
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee
sudo apt-get update -y
sudo apt-get install jenkins -y
sudo systemctl start jenkins
sudo systemctl status jenkins

#install docker
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

# install trivy
sudo apt-get install wget apt-transport-https gnupg lsb-release -y
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/tr
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc)
sudo apt-get update
sudo apt-get install trivy -y

#install terraform
```

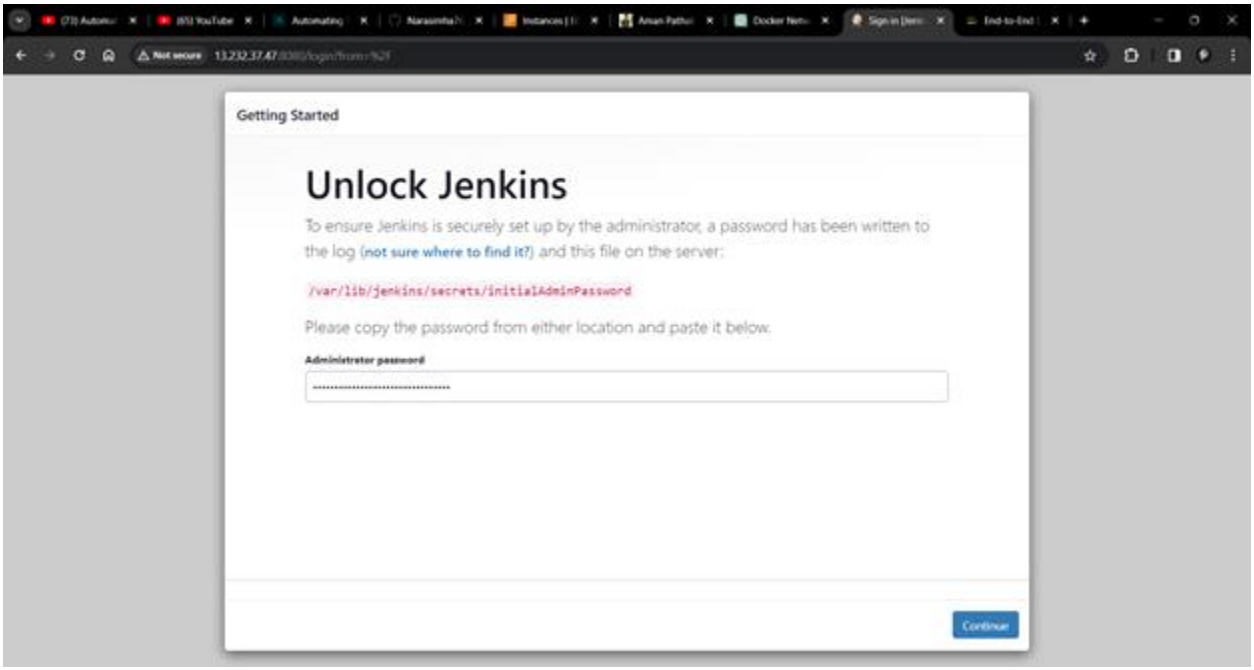
- Apply commands terraform init and terrafom apply –auto-approve
- Then the resource is creating in the aws.

Step2: Jenkins Pipeline for EKS

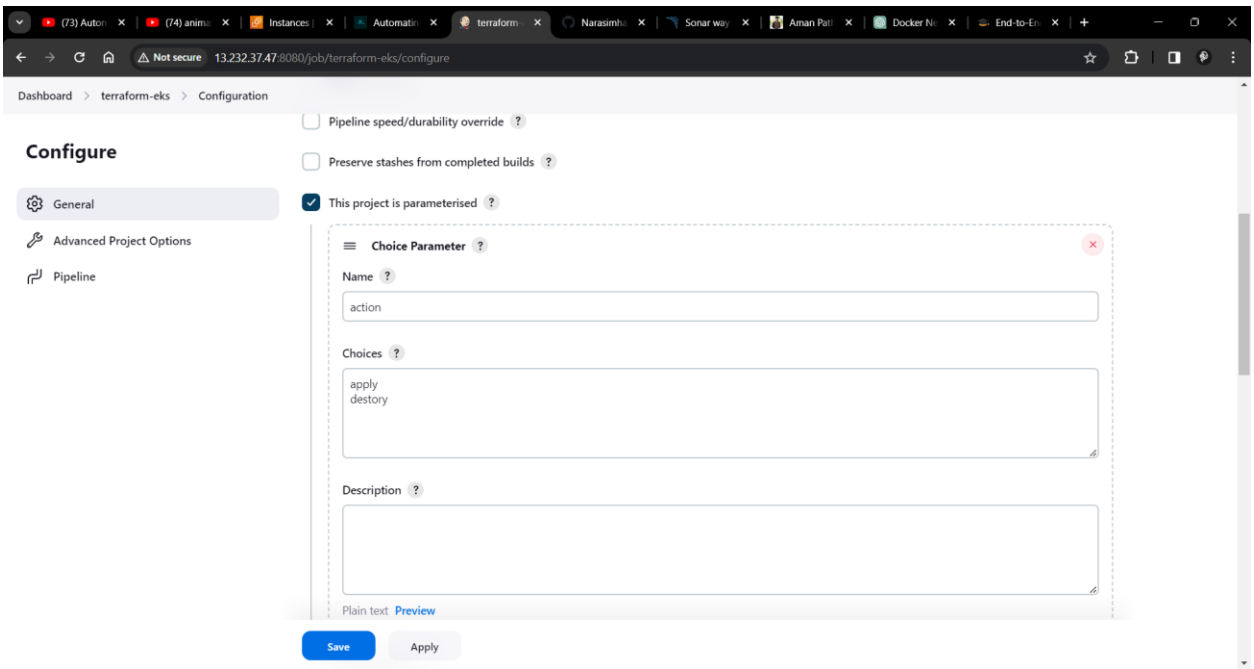
- Create a pipeline to provision an EKS cluster and node group using Terraform.
- Check all the scripts that are installed in the ec2 Server

```
ubuntu@ip-172-31-40-200:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
ubuntu@ip-172-31-40-200:~$ jenkins --version
2.426.3
ubuntu@ip-172-31-40-200:~$ terraform --version
Terraform v1.7.1
on linux_amd64
ubuntu@ip-172-31-40-200:~$ trivy --version
Version: 0.48.3
ubuntu@ip-172-31-40-200:~$ aws --version
aws-cli/2.15.15 Python/3.11.6 Linux/6.2.0-1017-aws exe/x86_64.ubuntu.22 prompt/off
ubuntu@ip-172-31-40-200:~$ kubectl
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
  create          Create a resource from a file or from stdin
  expose          Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service
  run             Run a particular image on the cluster
  set             Set specific features on objects

Basic Commands (Intermediate):
  explain         Get documentation for a resource
  get             Display one or many resources
  edit            Edit a resource on the server
  delete          Delete resources by file names, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout         Manage the rollout of a resource
  scale           Set a new size for a deployment, replica set, or replication controller
  autoscale       Auto-scale a deployment, replica set, stateful set, or replication controller

Cluster Management Commands:
  certificate     Modify certificate resources
```

- Configure the Jenkins



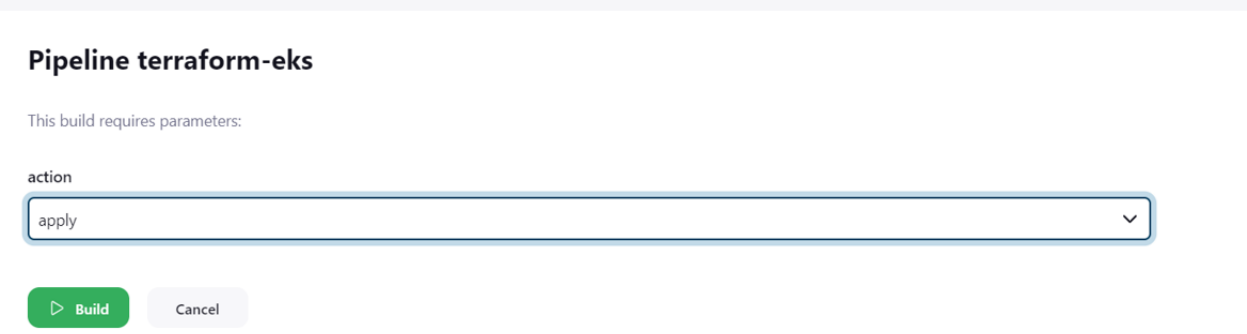
- Create a pipeline to provision an EKS cluster and node group using Terraform.



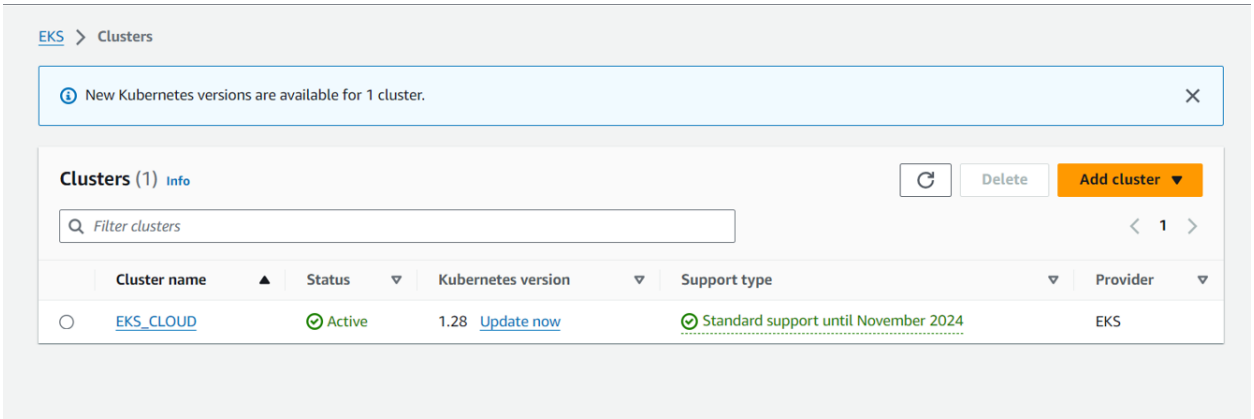
- Eks provision terraform code is present in git-hub

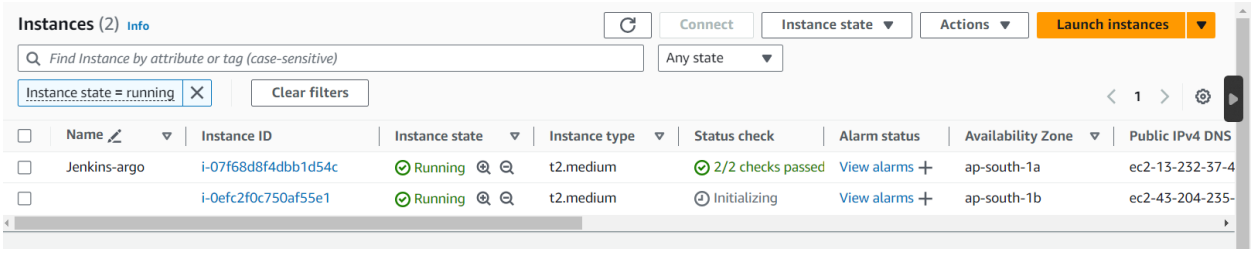
```
pipeline{
    agent any
    stages {
        stage('Checkout from Git'){
            steps{
                git branch: 'main', url: 'https://github.com/Narasimha76/Tetrisv1.git'
            }
        }
        stage('Terraform version'){
            steps{
                sh 'terraform --version'
            }
        }
        stage('Terraform init'){
            steps{
                dir('Eks-terraform') {
                    sh 'terraform init'
                }
            }
        }
        stage('Terraform validate'){
            steps{
                dir('Eks-terraform') {
                    sh 'terraform validate'
                }
            }
        }
        stage('Terraform plan'){
            steps{
                dir('Eks-terraform') {
                    sh 'terraform plan'
                }
            }
        }
        stage('Terraform apply/destroy'){
            steps{
                dir('Eks-terraform') {
                    sh 'terraform ${action} --auto-approve'
                }
            }
        }
    }
}
```

Save Apply



- Eks cluster has been created in AWS





Step3: Creating Full DevSecOps Pipeline

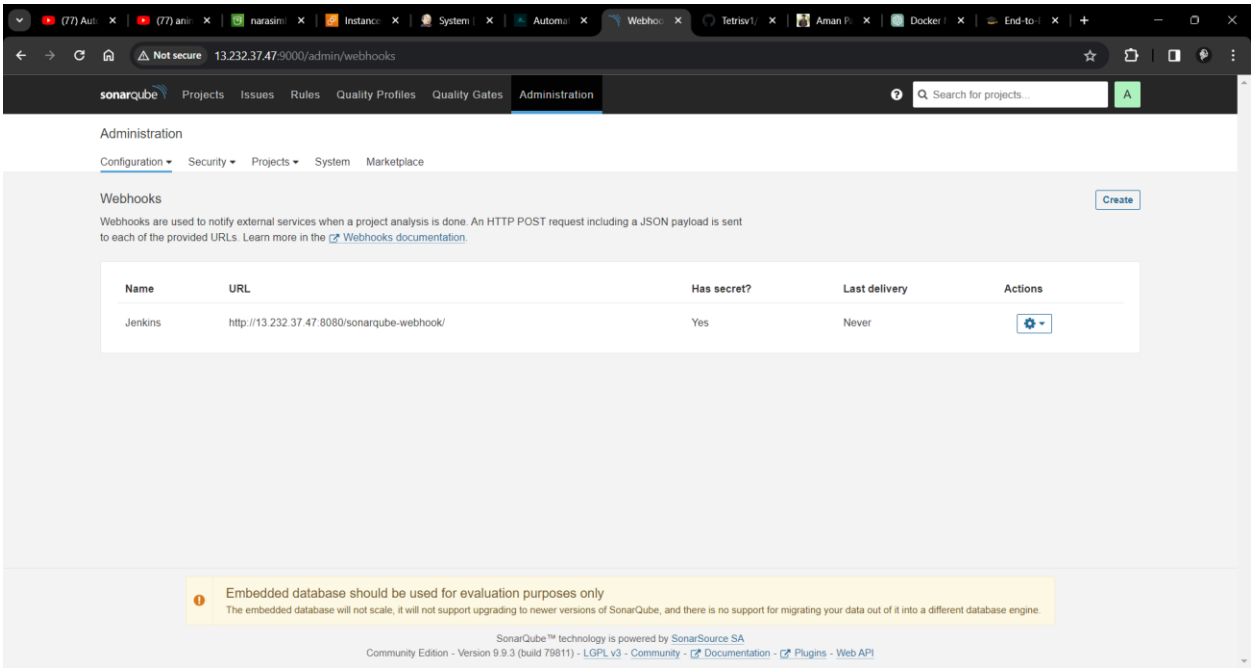
1. Checkout the repository.
2. SonarQube analysis, including quality gate.
3. Install dependencies using npm.
4. OWASP Scan with Dependency Check.
5. Trivy Scan for file analysis.
6. Build Docker image and push to Docker Hub.
7. Trivy scan Docker image.
8. Trigger another pipeline (Manifest Pipeline).

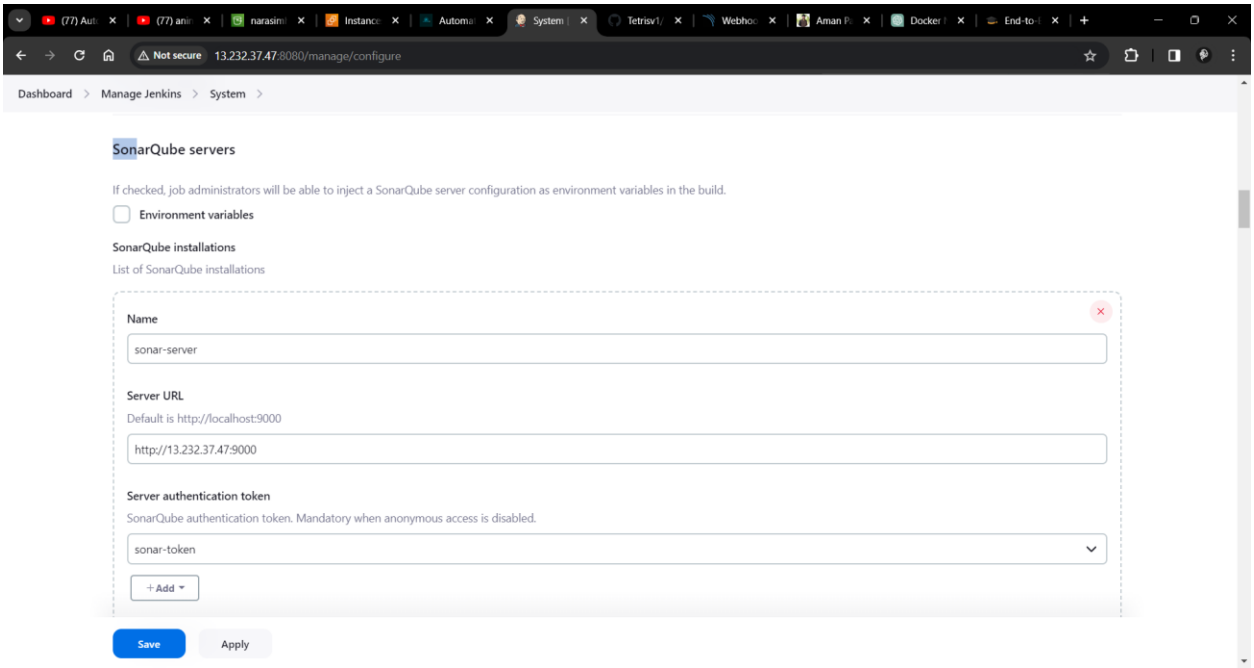
We need some plugins to complete this process

1. Eclipse Temurin installer
2. Sonarqube Scanner
3. NodeJs
4. Owasp Dependency-Check
5. Docker, Docker Commons, Docker Pipeline, Docker API, Docker-build-step

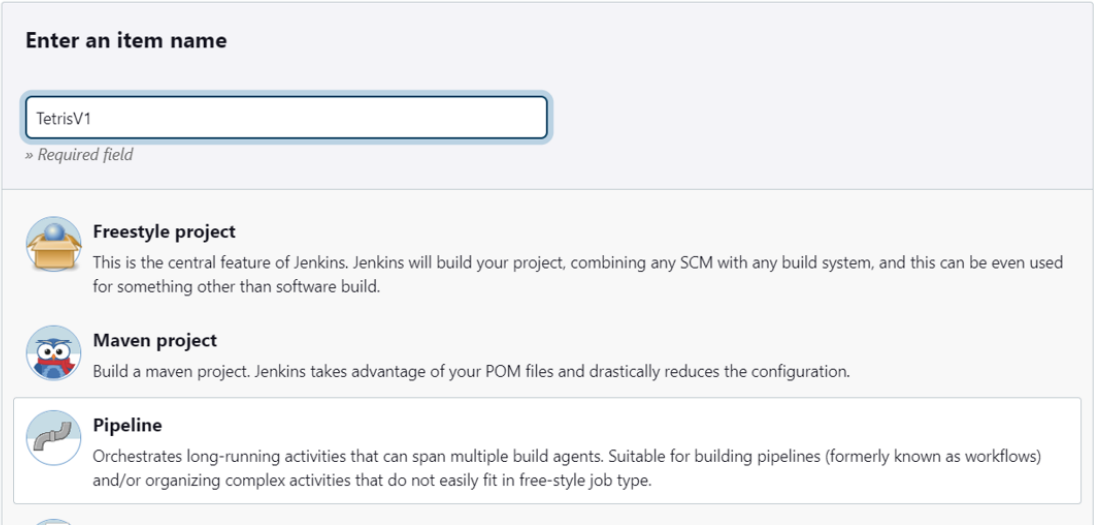
Add the tools in the Jenkins UI for JDK, Docker, NodeJS, Sonarqube, OWSAP(DependencyCheck)

- Configure the Sonarqube by creating user token and that token to the Jenkins credentials and taken the credentials and create the system for the Jenkins in the Jenkins UI.

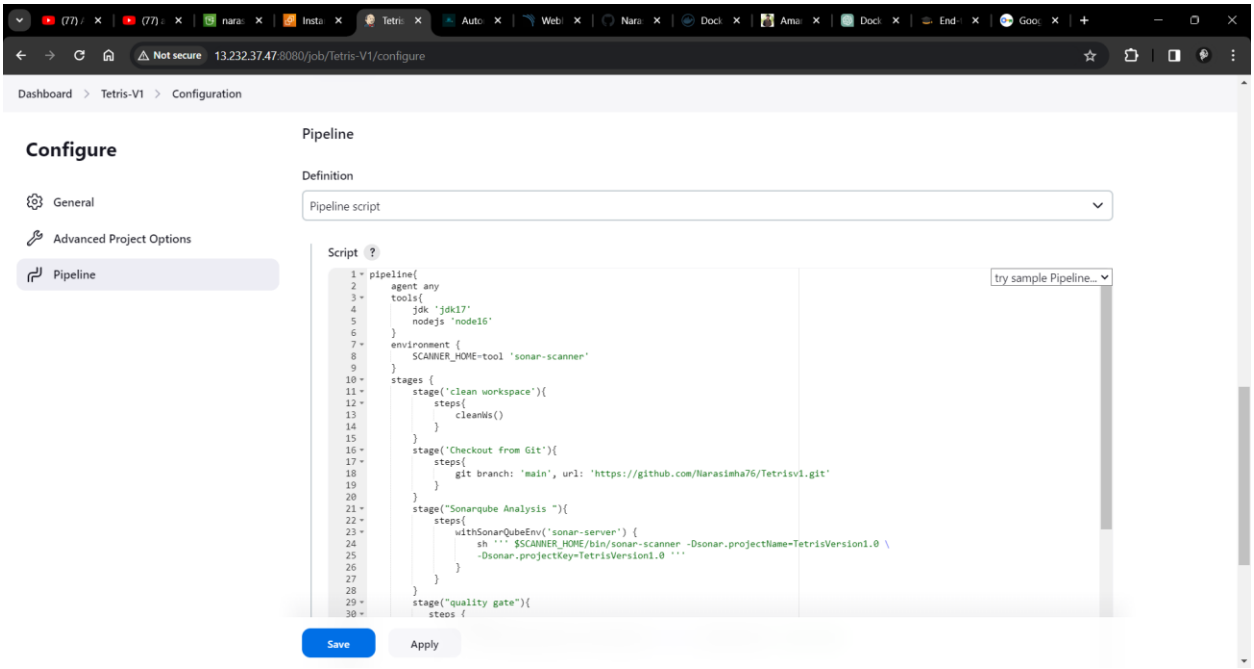




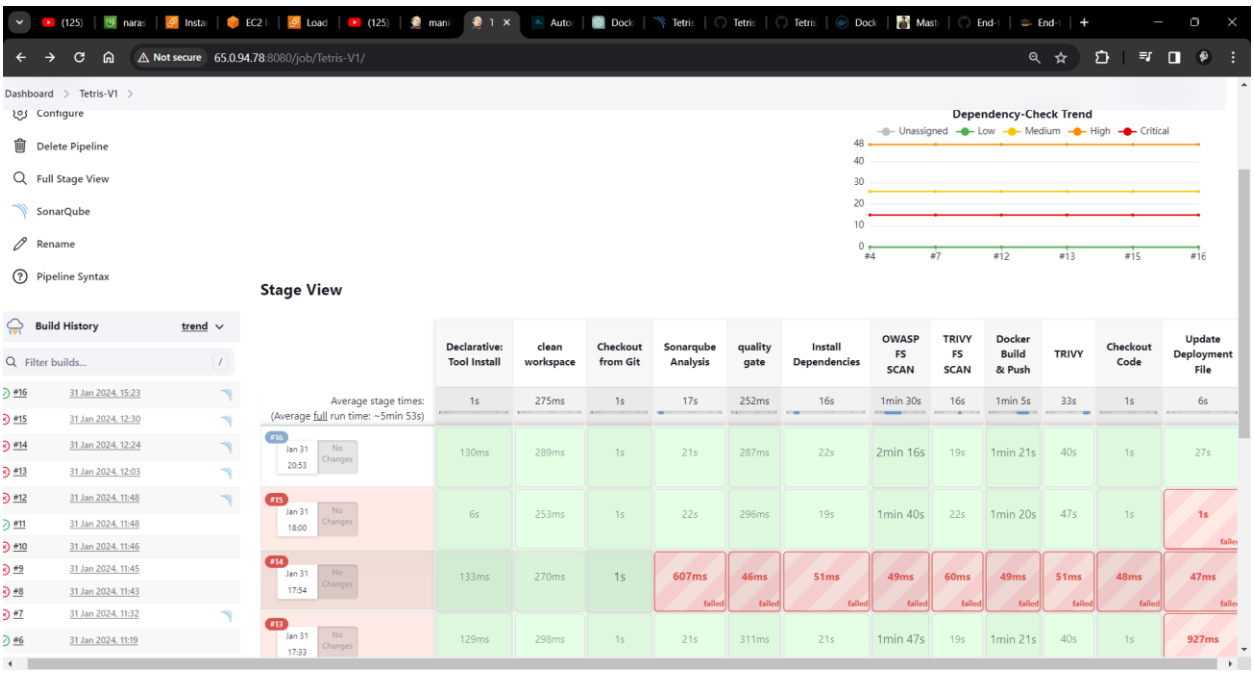
- Now let’s create an pipeline



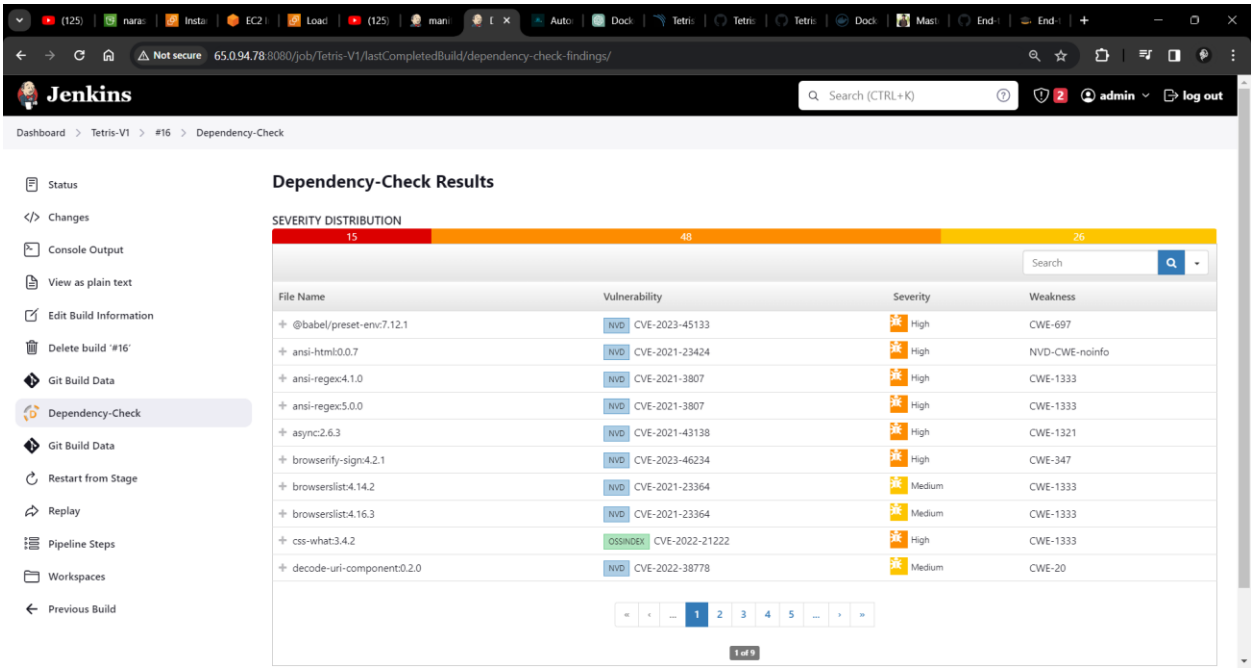
- Add the Jenkins Script in the pipeline which is present in my git repo



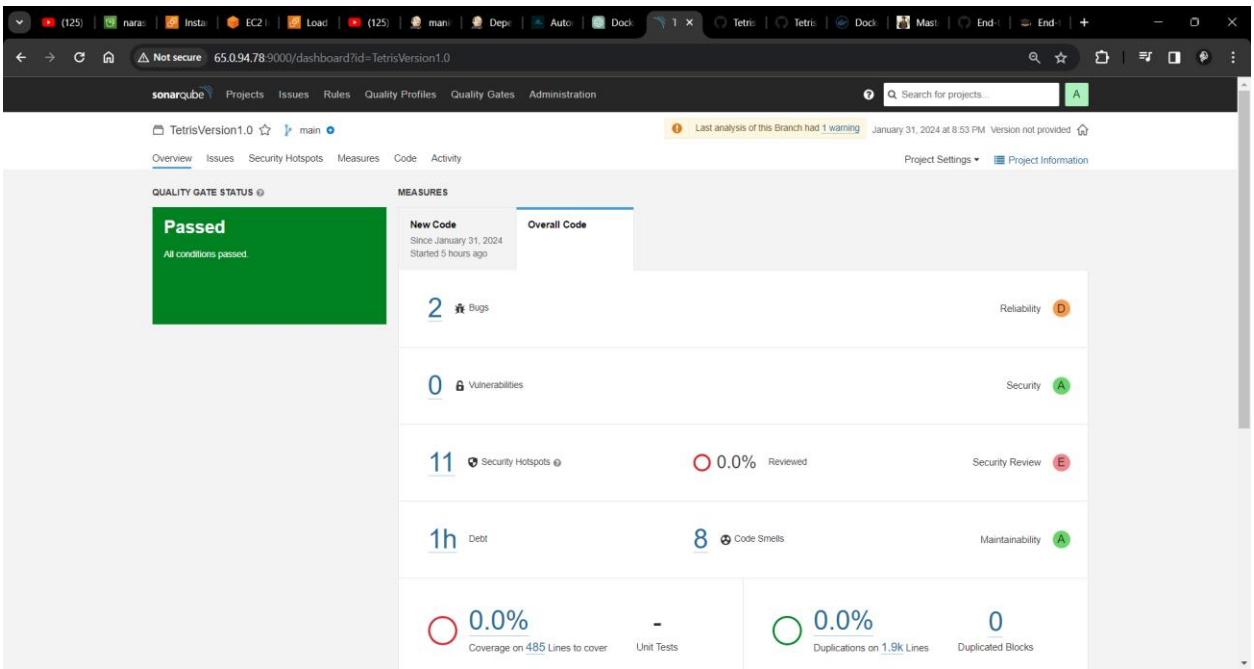
- Now run the pipeline



- Dependency-Check Results for the pipeline



- Sonarqube check for the pipeline



Step4: Create a Manifest Pipeline

- Add this code to the Jenkins pipeline

Narasimha76 Create Jenkinsfile

Code Blame 27 lines (27 loc) · 1013 Bytes Code 55% faster with GitHub Copilot

```
pipeline{
    agent any
    environment {
        GIT_REPO_NAME = "Tetris-manifest"
        GIT_USER_NAME = "Narasimha76"
    }
    stages {
        stage('Checkout Code') {
            steps {
                git branch: 'main', url: 'https://github.com/Narasimha76/Tetris-manifest.git'
            }
        }
        stage('Update Deployment File') {
            steps {
                script {
                    withCredentials([string(credentialsId: 'github', variable: 'GITHUB_TOKEN')]) {
                       NEW_IMAGE_NAME = "narasimhaswamy76/tetrisv1:latest"   #update your image here
                       sh "sed -i 's|image: .*|image: $NEW_IMAGE_NAME|' deployment.yml"
                       sh 'git add deployment.yml'
                       sh "git commit -m 'Update deployment image to $NEW_IMAGE_NAME'"
                       sh "git push https://${GITHUB_TOKEN}@github.com/${GIT_USER_NAME}/${GIT_REPO_NAME} HEAD:main"
                    }
                }
            }
        }
    }
}
```

Step5: Install ArgoCD on AWS EKS and configure it

- Before installation add the eks cluster to your CLI

```
</body></html>
ubuntu@ip-172-31-40-200:~$ aws eks update-kubeconfig --name EKS_CLOUD --region ap-south-1
Added new context arn:aws:eks:ap-south-1:862547479026:cluster/EKS_CLOUD to /home/ubuntu/.kube/
config
ubuntu@ip-172-31-40-200:~$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
ip-172-31-14-253.ap-south-1.compute.internal   Ready    <none>   5h52m   v1.28.5-eks-5e0fdde
ubuntu@ip-172-31-40-200:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   5h57m
kube-node-lease   Active   5h57m
kube-public       Active   5h57m
kube-system       Active   5h57m
ubuntu@ip-172-31-40-200:~$
```

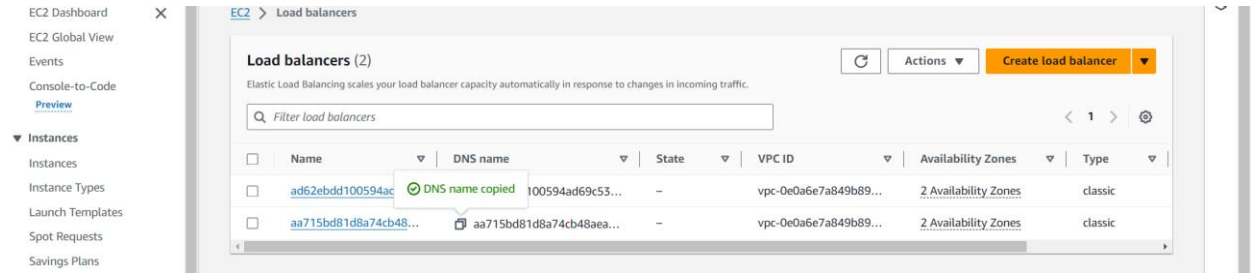
- Install the ArgoCD

```
ubuntu@ip-172-31-40-200:~$ kubectl create namespace argocd
namespace/argocd created
ubuntu@ip-172-31-40-200:~$ kubectl get namespaces
NAME              STATUS   AGE
argocd            Active   2s
default           Active   5h58m
kube-node-lease   Active   5h58m
kube-public       Active   5h58m
kube-system       Active   5h58m
ubuntu@ip-172-31-40-200:~$ kubectl create namespace argocd
Error from server (AlreadyExists): namespaces "argocd" already exists
ubuntu@ip-172-31-40-200:~$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
```

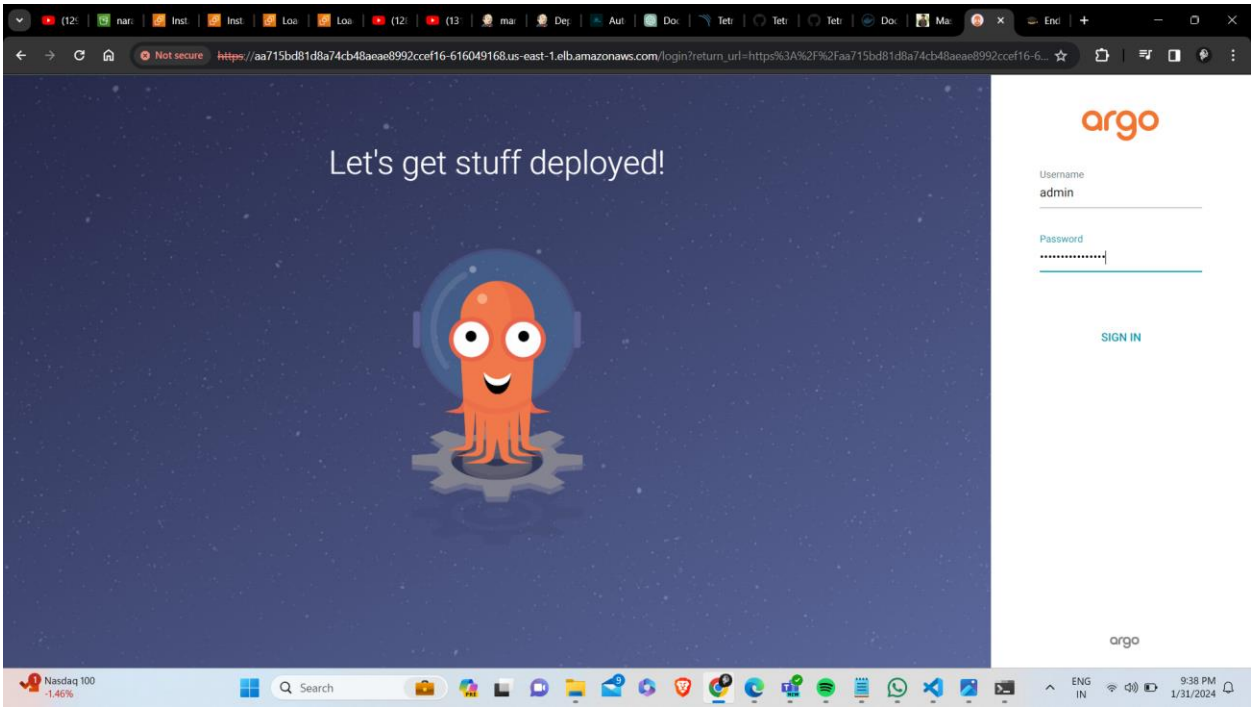
```
ubuntu@ip-172-31-40-200:~$ kubectl get all -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
pod/argocd-application-controller-0                 1/1     Running   0          42s
pod/argocd-applicationset-controller-749957bcc9-7f4tr   1/1     Running   0          42s
pod/argocd-dex-server-b6c8d496b-jmdj6               1/1     Running   0          42s
pod/argocd-notifications-controller-5fff689764-mf6x7   1/1     Running   0          42s
pod/argocd-redis-55c6985b84-q45k5                   1/1     Running   0          42s
pod/argocd-repo-server-d7666859-cbw9r               1/1     Running   0          42s
pod/argocd-server-6984446998-8zlmp                  1/1     Running   0          42s

NAME                                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
service/argocd-applicationset-controller      ClusterIP   10.100.106.144   <none>        7000/TCP,8080/TCP            42s
service/argocd-dex-server                     ClusterIP   10.100.1.181     <none>        5556/TCP,5557/TCP,5558/TCP   42s
service/argocd-metrics                        ClusterIP   10.100.5.23      <none>        8082/TCP                     42s
service/argocd-notifications-controller-metrics   ClusterIP   10.100.201.70    <none>        9001/TCP                     42s
service/argocd-redis                          ClusterIP   10.100.97.8      <none>        6379/TCP                     42s
service/argocd-repo-server                    ClusterIP   10.100.207.69    <none>        8081/TCP,8084/TCP            42s
service/argocd-server                         ClusterIP   10.100.141.29    <none>        80/TCP,443/TCP               42s
service/argocd-server-metrics                 ClusterIP   10.100.223.40    <none>        8083/TCP                     42s

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/argocd-applicationset-controller   1/1     1            1           42s
deployment.apps/argocd-dex-server              1/1     1            1           42s
deployment.apps/argocd-notifications-controller    1/1     1            1           42s
deployment.apps/argocd-redis                   1/1     1            1           42s
deployment.apps/argocd-repo-server             1/1     1            1           42s
deployment.apps/argocd-server                  1/1     1            1           42s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/argocd-applicationset-controller-749957bcc9   1         1         1       42s
replicaset.apps/argocd-dex-server-b6c8d496b           1         1         1       42s
replicaset.apps/argocd-notifications-controller-5fff689764    1         1         1       42s
replicaset.apps/argocd-redis-55c6985b84               1         1         1       42s
replicaset.apps/argocd-repo-server-d7666859           1         1         1       42s
replicaset.apps/argocd-server-6984446998              1         1         1       42s

NAME                                          READY   AGE
statefulset.apps/argocd-application-controller   1/1     42s
```

- Now modify the argocd service to loadbalancer

```
ubuntu@ip-172-31-40-200:~$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
ubuntu@ip-172-31-40-200:~$
ubuntu@ip-172-31-40-200:~$
```



- Now copy the dns of the loadbalancer and paste in the web



- Take the password from the secrets and decode it into bas64

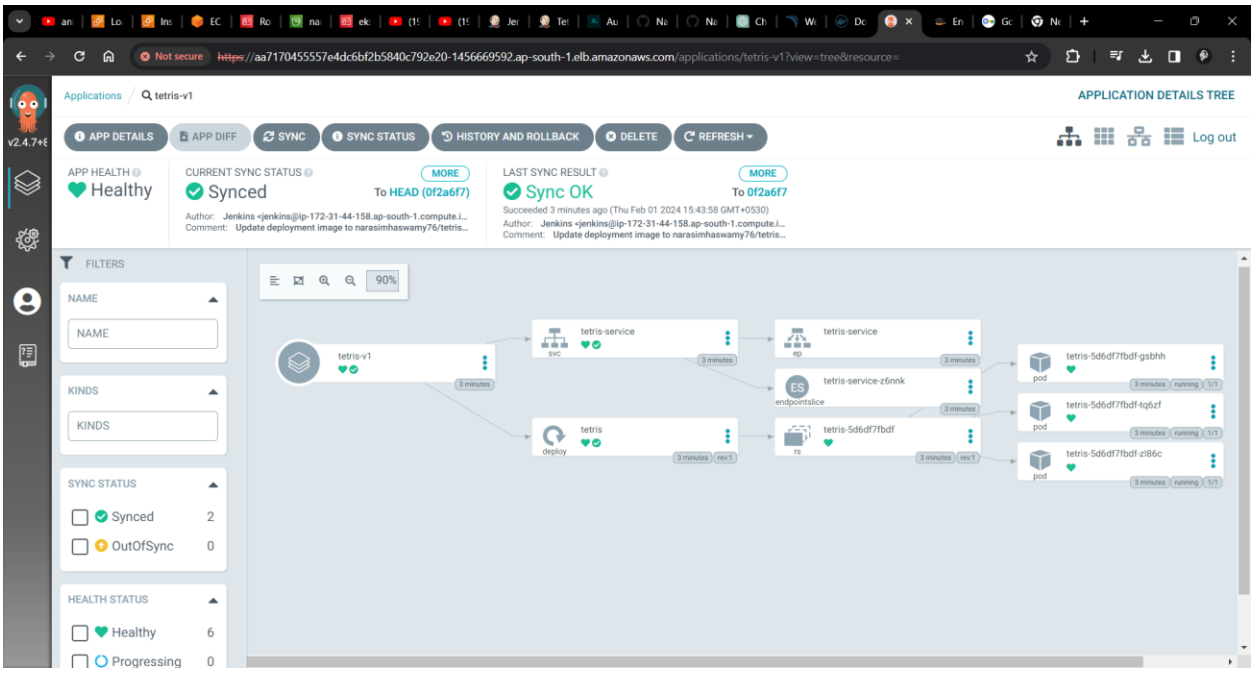
```
Command Prompt    ubuntu@ip-172-31-40-200: ~
ubuntu@ip-172-31-40-200:~$ kubectl get secrets -n argocd
NAME                          TYPE     DATA   AGE
argocd-initial-admin-secret   Opaque   1      17m
argocd-notifications-secret   Opaque   0      17m
argocd-secret                 Opaque   5      17m
ubuntu@ip-172-31-40-200:~$ kubectl edit secret argocd-initial-admin-secret -n argocd
```

```
Command Prompt    ubuntu@ip-172-31-40-200: ~
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  password: VEhDQk11ampoT0pqT1IxWA==
kind: Secret
metadata:
  creationTimestamp: "2024-01-31T15:48:55Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "52179"
  uid: feb5149d-3255-4a1b-9058-2d77802f879c
type: Opaque
~
~
~
~
```

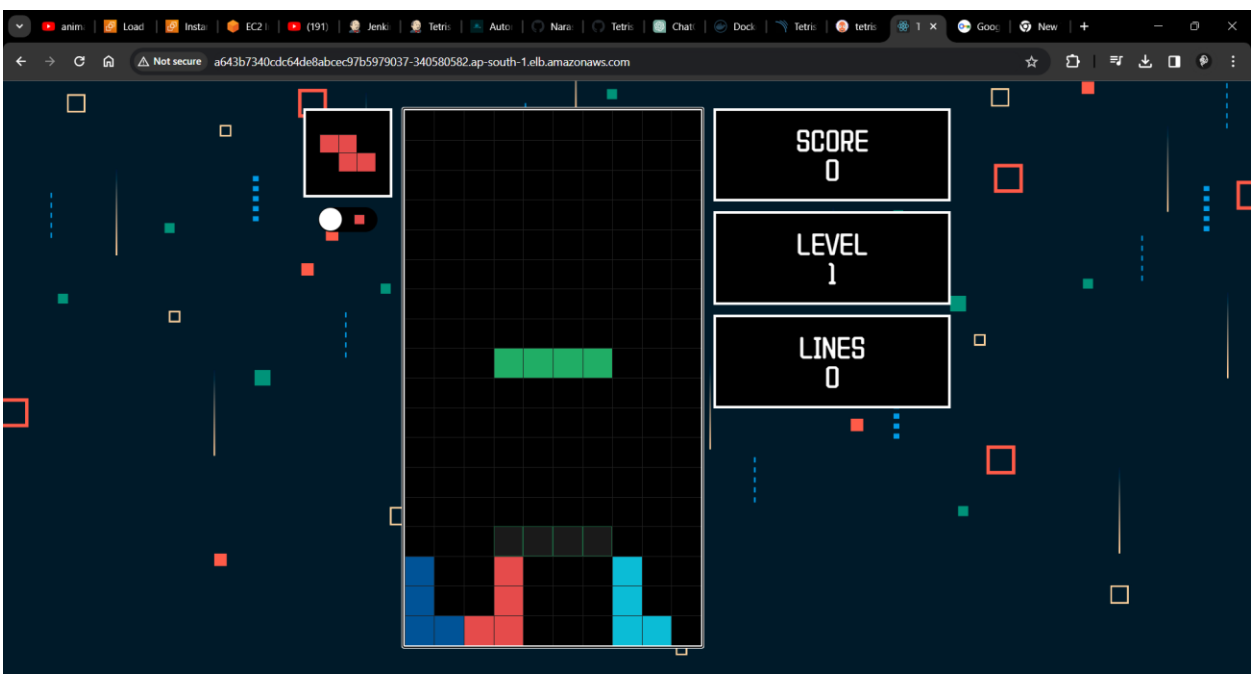
```
Command Prompt    ubuntu@ip-172-31-40-200: ~
ubuntu@ip-172-31-40-200:~$ kubectl get secrets -n argocd
NAME                          TYPE     DATA   AGE
argocd-initial-admin-secret   Opaque   1      17m
argocd-notifications-secret   Opaque   0      17m
argocd-secret                 Opaque   5      17m
ubuntu@ip-172-31-40-200:~$ kubectl edit secret argocd-initial-admin-secret -n argocd
Edit cancelled, no changes made.
ubuntu@ip-172-31-40-200:~$ echo  VEhDQk11ampoT0pqT1IxWA== | base64 --decode
THCBMujjhOJjOR1Xubuntu@ip-172-31-40-200:~$
```

Step6: Use ArgoCD to synchronize AWS EKS deployments with manifest changes in git repo

- Create a project in the argocd with the connection of git-hub manifest repo and define the destination eks cluster in the aws eks.

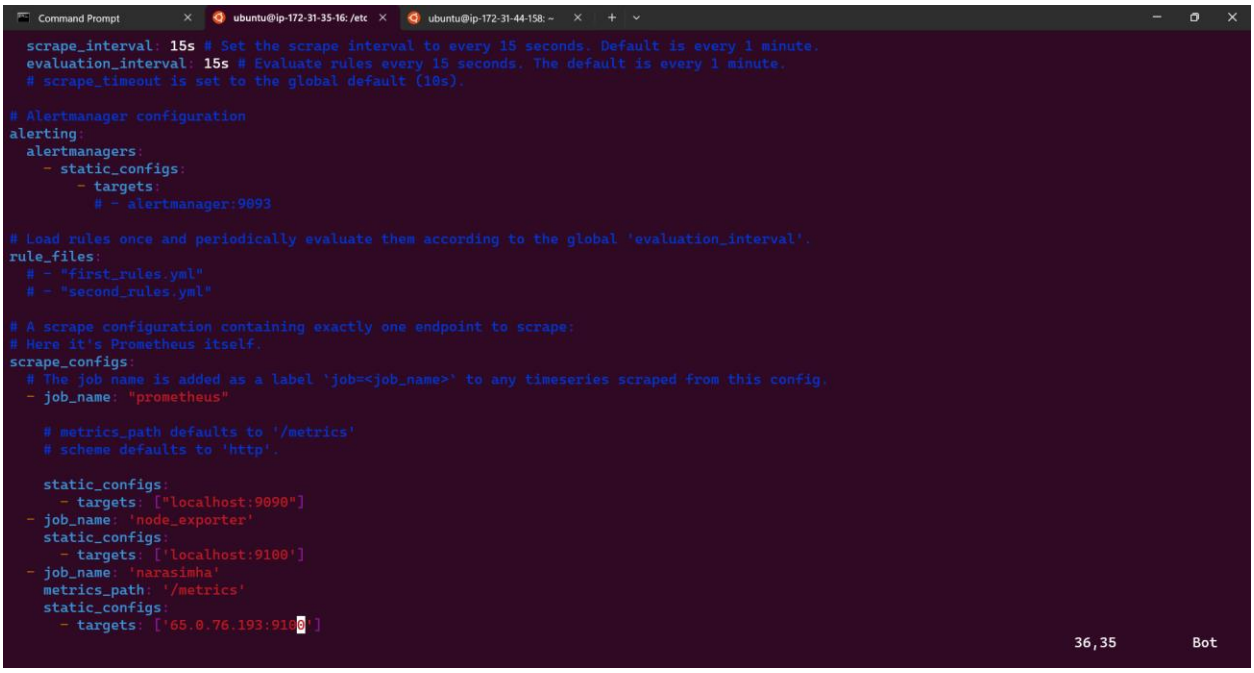


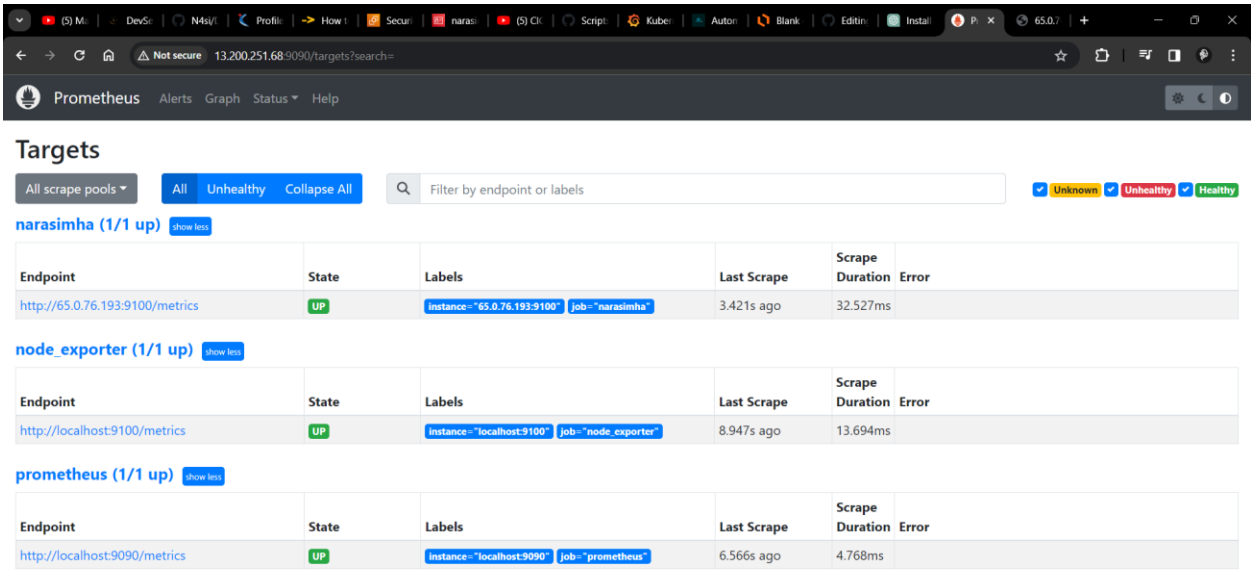
- It will monitors the manifest repo when ever the change occur in the repo it will keep the eks cluster in the desired state
- After deploying the pods and load balancer service in the eks cluster then we can access the game through the service endpoint



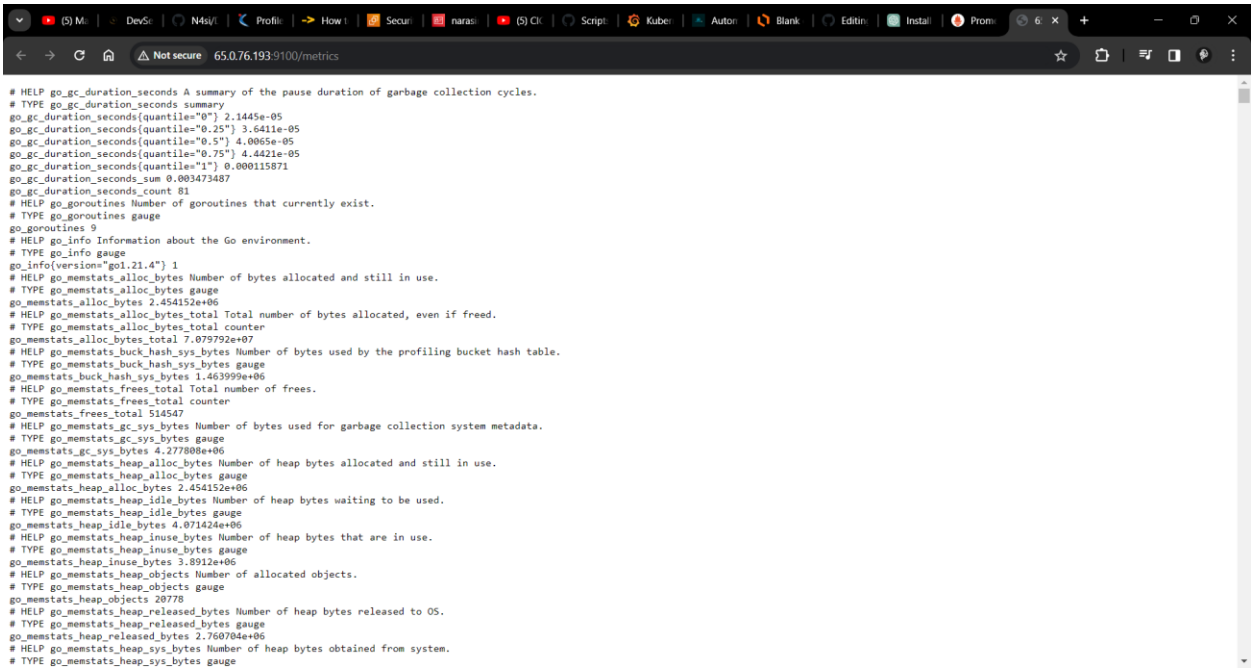
Step7: Install Prometheus, Grafana, and Node Exporter using Helm on AWS EKS.

- First install the helm in the eks cluster.
- Install the grafana and node exporter and grafana using helm commands in the eks cluster.
- After installation edit the Prometheus file to add the targets of node exporter

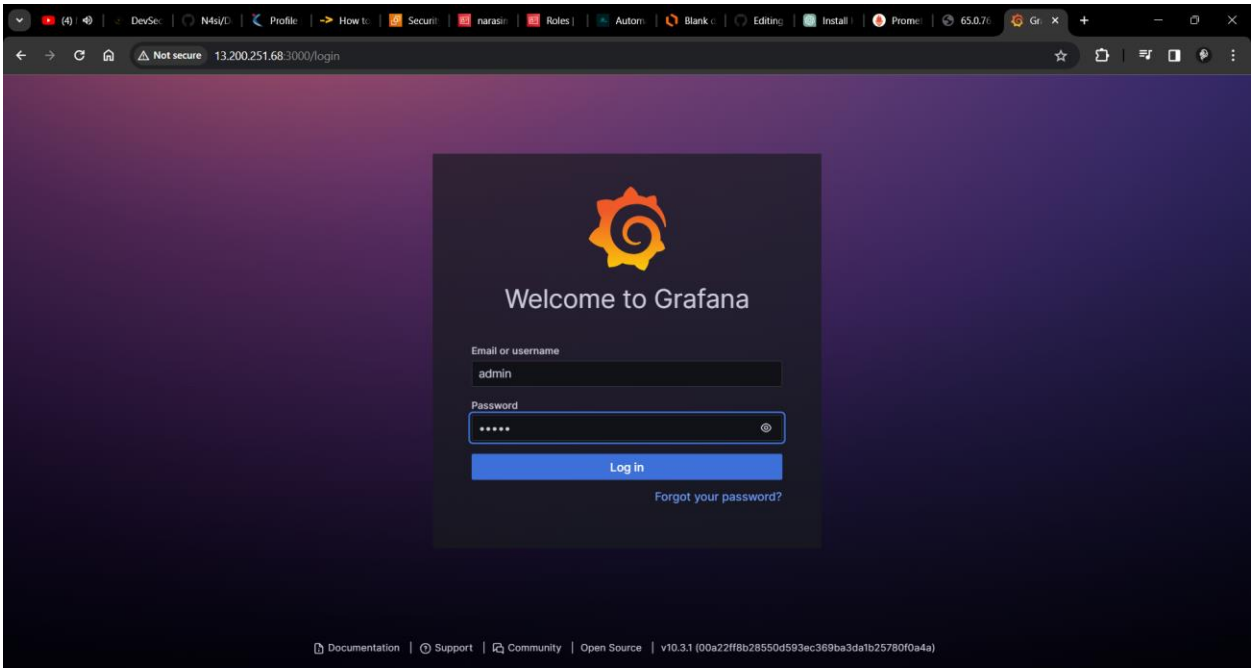




- Node Exporter metrics :



Grafana :



- Login with grafana and add the the datasource as the Prometheus and create a dashboard with that Prometheus datasource
- Then the dashboard of the eks cluster will be displayed