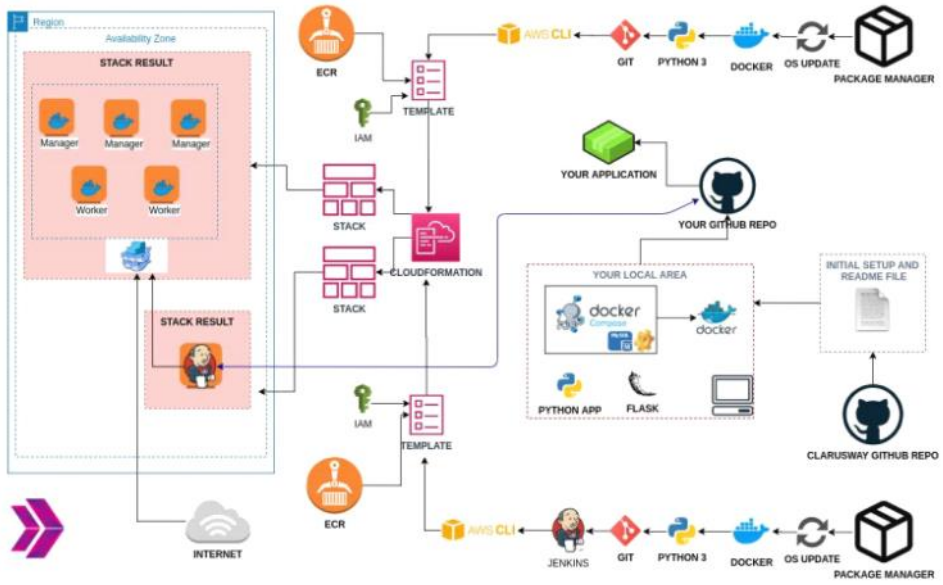
5 - Project 205 - Jenkins Pipline Phone book

Saturday, 9 October 2021 15.06

Project 205-JenkinsPipelinePhoneBook

Saturday, 9 October 2021 0.27



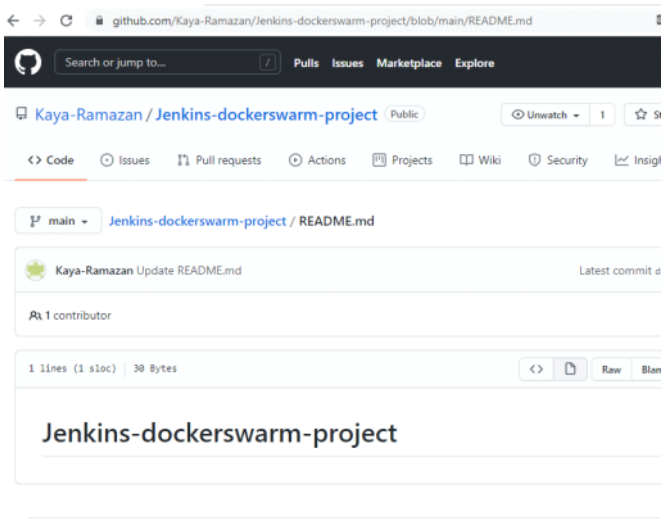
Yeni repo olusturup onu clonleadik public repo

Daha önceki derste terraform kurrarak 3 manager 2 worker in oldugu containnerlarin oldugu bir stek olusturmustuk
Simdide bunu Jenkins bizim yerimize yapacak

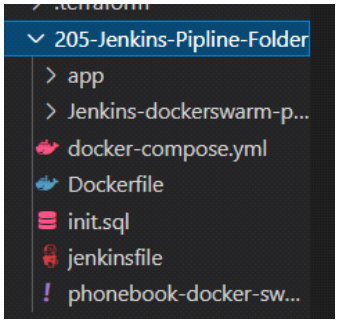
Jengins File in bunu yapabilmesi icin icerisinde;

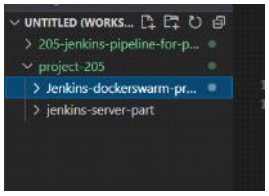
- **Git**
- **Python 3**
- **Docker**
- **Jenkins in yuklu olmasi gerekiyor**

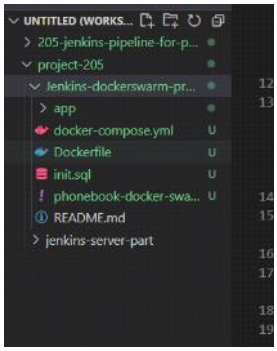
Amacimiz Jenkins pipline ile daha önce yaptigimiz islemi yapabilmek



Ve olusturdugumuzu dosyayi kendi publigimizde clonaladik







Bu sekilde bir dosya yapisi olusturduk kendi lokalimizde olusturdugumuz dosya da yani Jenkins-docker-Swarm -Project de;

App dosyamiz
Docker compose yaml dosyamiz
Docker file dosyamiz
Init.sql dosyamiz
Ve phone book docker swarm yaml imiz oldu digerlerini git hub dosyamizda cok yer kaplamasin diye ust klasörler koyduk

Docker server part i ile de jenkinsi kolayca ayaga kaldiracagiz

Simdi Tf dosyamiza göz atalim

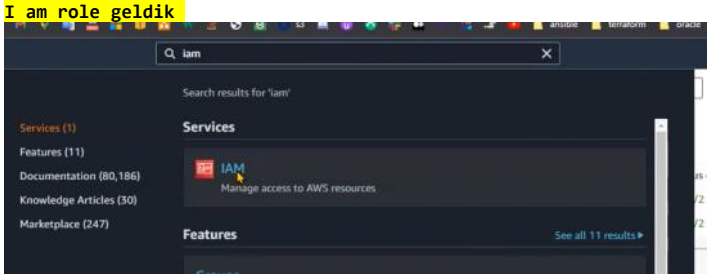
```
provider "aws" {     # aws installa Terraform dedigimizde bu kismi direk
```

```
alabiliyorduk
  region = "us-east-1"
  \\\profile = "tyler"
  access_key = ""
  secret_key = ""      #c bu sekilde aws e kolayca baglanabiliyoruz
```

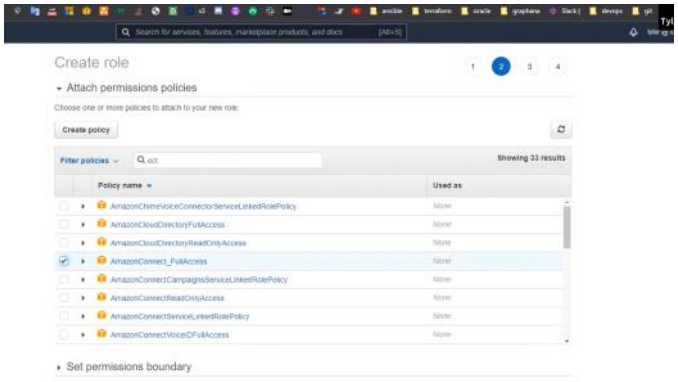
From <https://app.slack.com/client/T0227UVRJU8/D02DU006A00/thread/C021WSETWBB-1633805656.264700>

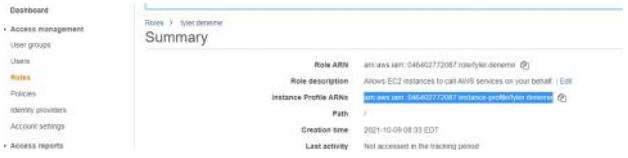
```
}
data "aws_ami" "tf-ami" {                    # ami yi resorce kullanarak cekiyoruz  elle de yazabilirdik
  owners      = ["amazon"]
  most_recent = true
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm*"]          #filter kullanarak  wáws den ami cekecek
  }
}
data "aws_caller_identity" "current" {}    # bir digetr data resource muz bu da bizim account numaramizi cekiyor


data "aws_region" "current" {
  name = "us-east-1"
}
```

I am role geldik



Rolde ec2-full-access sectik ve bir role olusturdugumuzu farz edersek

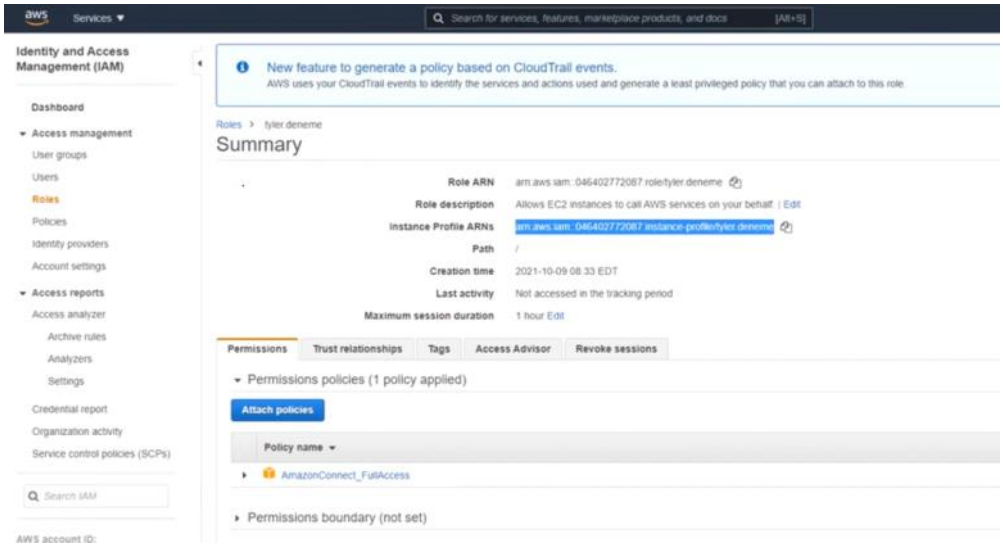




Role de kullandigimiz alan burasidir

```
resource "aws_iam_role" "aws_access" {     # bu rdaki önemli konulardan biri rol olustirmak roller bizim adimiza istedigimiz islemleri yaoacak  bunun iki yolu var biri iam role olusturup ec2 ya attach etmek digeride tf dosyasi ile bunu yapmak
  name = "awsrole"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      },
    ]
```

```
  })
  inline_policy {     # normalde instancelarin msh komutu ile git kullanmadan
    name = "my_inline_policy"instancelaruin briur biryle irtibat kurmasini
sagliyor
    policy = jsonencode({
      Version = "2012-10-17"
      Statement = [
        {
          "Effect" : "Allow",
          "Action" : "ec2-instance-connect:SendSSHPublicKey",
          "Resource" : "arn:aws:ec2:${data.aws_region.current.name}:
${data.aws_caller_identity.current.account_id}:instance/*",
          "Condition" : {
            "StringEquals" : {
              "ec2:osuser" : "ec2-user"
            }
          }
        },
        {
          "Effect" : "Allow",
          "Action" : "ec2:DescribeInstances",
          "Resource" : "*"
        }
      ]
    })
  }
        managed_policy_arns =
["arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryFullAccess",
"arn:aws:iam::aws:policy/AmazonEC2FullAccess",
"arn:aws:iam::aws:policy/IAMFullAccess",
"arn:aws:iam::aws:policy/AWSCloudFormationFullAccess"]
}
```



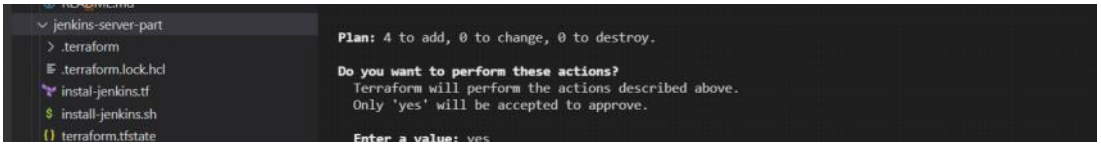

```
Aws d bir role olusturdugumuz farz edersek instance profile ARN sini kopyalayip
nmanagepolisy kismina yapistiriyoruz diyebiliriz  devaminda da Ecr full access ac2
full access

Bunlari ec2 ya direk attach edemiyoruz bunu n icin debir role profile olusturup
ona attach ediyoruz

resource "aws_iam_instance_profile" "ec2-profile" {    # bunlari ec2 lara attach
etmek icin kullaniyoruz

  name = "jenkins-profile"
  role = aws_iam_role.aws_access.name
}
resource "aws_instance" "tf-jenkins-server" {  # instance olusturuyoruz
  ami           = data.aws_ami.tf-ami.id
  instance_type = "t2.micro"
  key_name      = "tyler-team"
  //  Write your pem file name
  security_groups = ["jenkins-server-sec-gr"]
  iam_instance_profile = aws_iam_instance_profile.ec2-profile.name  # profile ile
rolu bagliyoruz
  tags = {
    Name = "Jenkins_Server"
  }
  user_data = file("install-jenkins.sh")  # user datayi sh dposyamizdan cekiyor
}
resource "aws_security_group" "tf-jenkins-sec-gr" {
  name = "jenkins-server-sec-gr"
  tags = {
    Name = "jenkins-server-sec-group"
  }
  ingress {
```

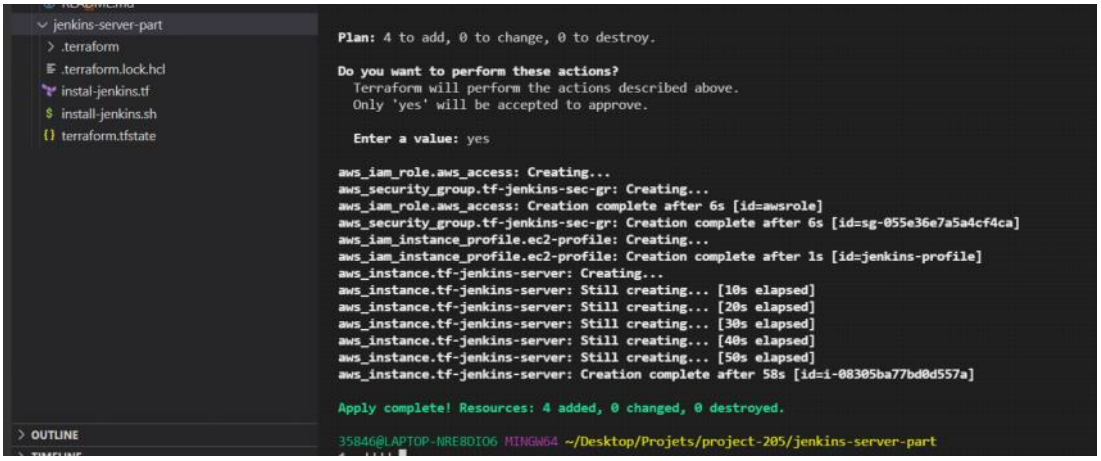
```
    from_port   = 80
    protocol    = "tcp"
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 22
    protocol    = "tcp"
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 8080
    protocol    = "tcp"
    to_port     = 8080
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    protocol    = -1
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Simdi de install.jenkins.sh dosyamiza göz atalim;

```
#! /bin/bash
sudo yum update -y
hostnamectl set-hostname jenkins-server  # burda isim verdik
yum install git -y   # git i yukledik
sudo amazon-linux-extras install java-openjdk11 -y    # jenkins yukleniyor
sudo wget -O /etc/yum.repos.d/jenkins.repo
https://pkg.jenkins.io/redhat/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key
sudo amazon-linux-extras install epel -y
sudo yum install jenkins -y
sudo systemctl start jenkins   # start enable -status yapiyoruz
sudo systemctl enable jenkins
sudo systemctl status jenkins
amazon-linux-extras install docker -y    # docker yukleniyor
systemctl start docker
systemctl enable docker
usermod -a -G docker ec2-user   # guruplari ekliyoruz
usermod -a -G docker jenkins
cp /lib/systemd/system/docker.service /lib/systemd/system/docker.service.bak   #
docker in server daki bir konfigurSYONU
sed -i 's/^ExecStart=.*/ExecStart=\/usr\/bin\/dockerd -H tcp:\/\/127.0.0.1:2375 -H
unix:\/\/\/var\/run\/docker.sock/g' /lib/systemd/system/docker.service
systemctl daemon-reload
systemctl restart docker
systemctl restart jenkins
curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-
compose-$(uname -s)-$(uname -m)" \   # docker compose install ediliyor
-o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
# uninstall aws cli version 1
rm -rf /bin/aws   # önce cli version 1 varsa siliyor version 2 yi install ediyoruz
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
./aws/install
yum install python3 -y
amazon-linux-extras install epel -y
yum install python-pip -y
pip install ec2instanceconnectcli
yum install amazon-ecr-credential-helper -y
mkdir -p /home/jenkins/.docker
cd /home/jenkins/.docker
echo '{"credsStore": "ecr-login"}' > config.json
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform      # terraform install eddiyoruz
```

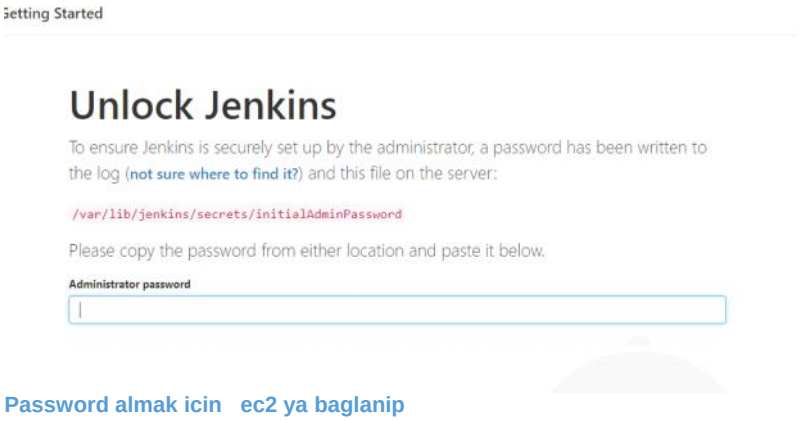
Jenkins server part dosyamizin icerisn de terraform dosyamizi init ediyoruz





Ve daha sonra acilan ec2 muzun public ip sinin : 8080 ile browser da acip jenkins e baglaniyoruz

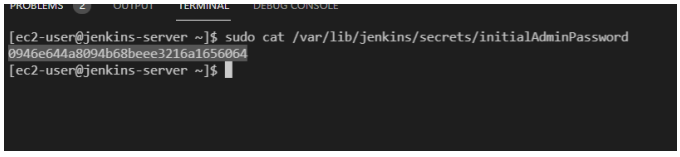
Ve

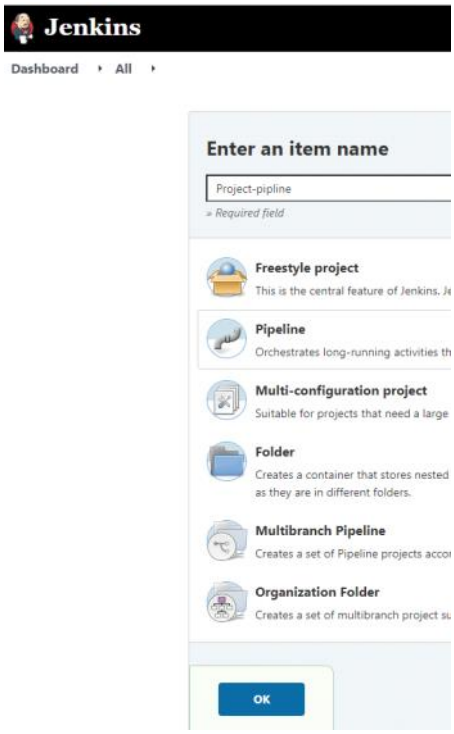


Password almak icin ec2 ya baglanip

`sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

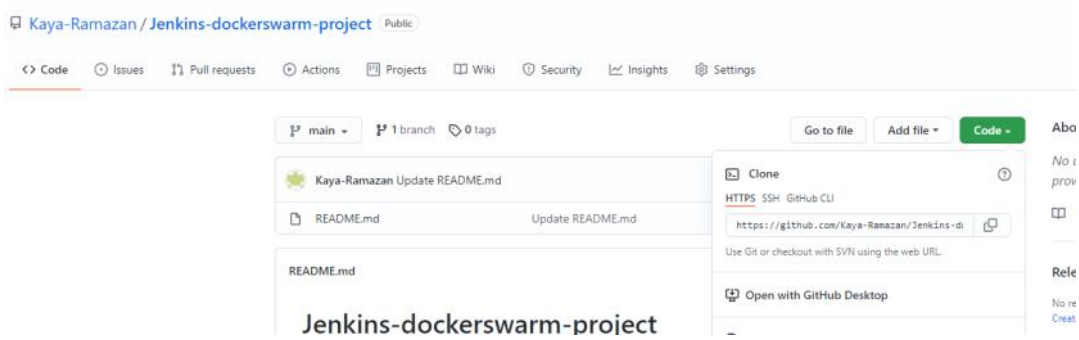
Komutunu uyguluyoruz



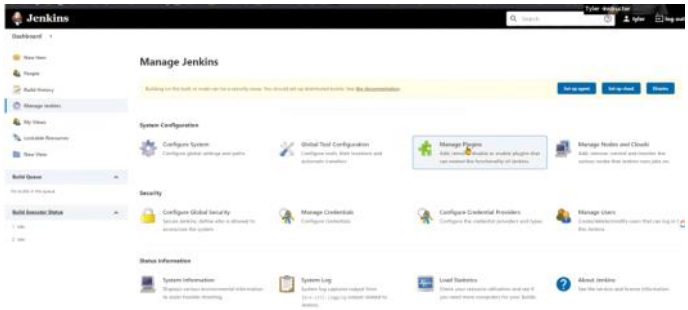


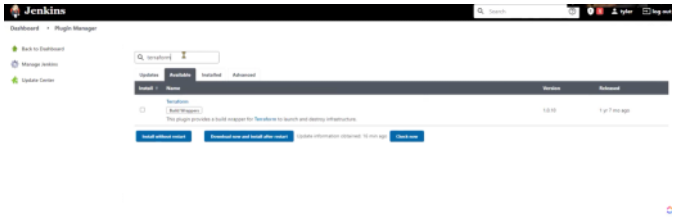
Ve ilk pipline olusturuyoruz

Github url miizi kopyaliyoruz



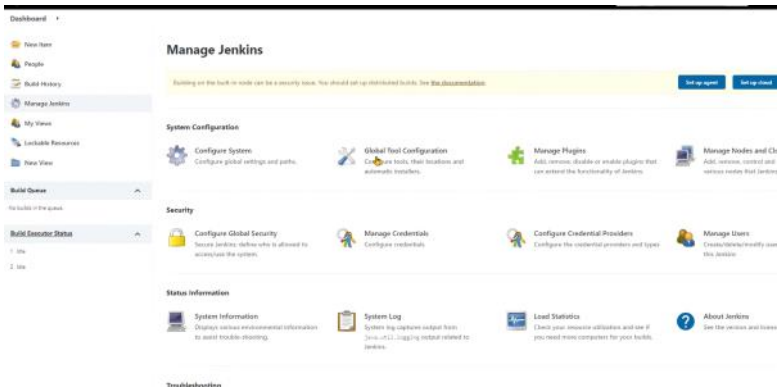
Jenkins de Terraform u kullanmak icin managementta dan Terraform un plagin inin yukluyoruz



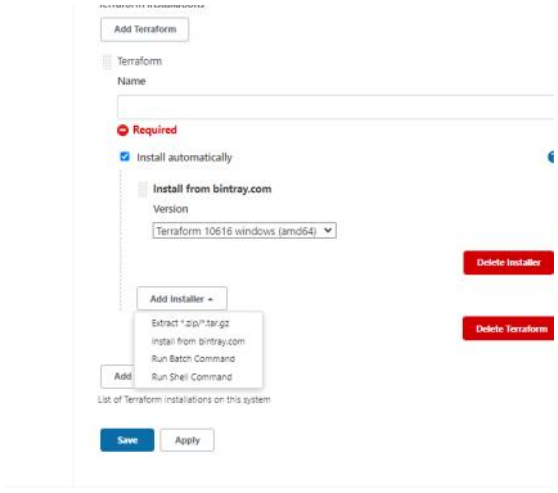


Bir sey adajha yapacagiz indirdigimiz Terraform n versiyonunu secmek
Bununn icin global tool managed,ent a godiyoruz ve eb altta terrafom u yaziyiruz

Vscoda dan terraform version dedigimizde cikan cversionu buraya ekliyoruz

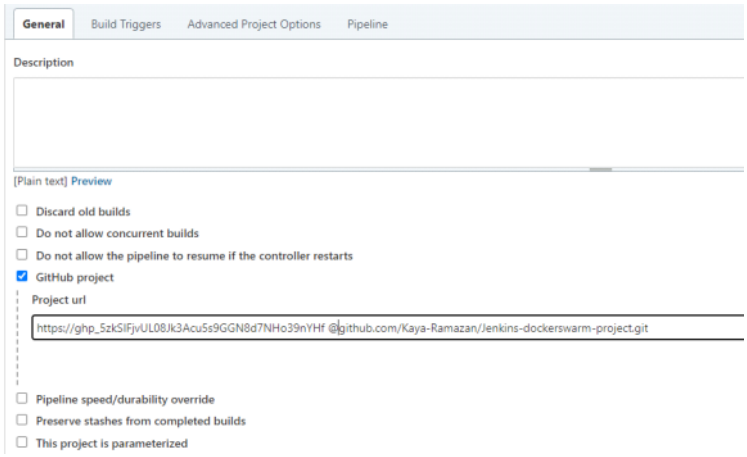



```
35846@LAPTOP-NREBDIO6 MINGW64 ~/Desktop/Projets/project-205/jenkins-server-part
$ terraform version
Terraform v1.0.6
on windows_amd64
+ provider registry.terraform.io/hashicorp/aws v3.62.0

Your version of Terraform is out of date! The latest version
is 1.0.8. You can update by downloading from https://www.terraform.io/downloads.html
```

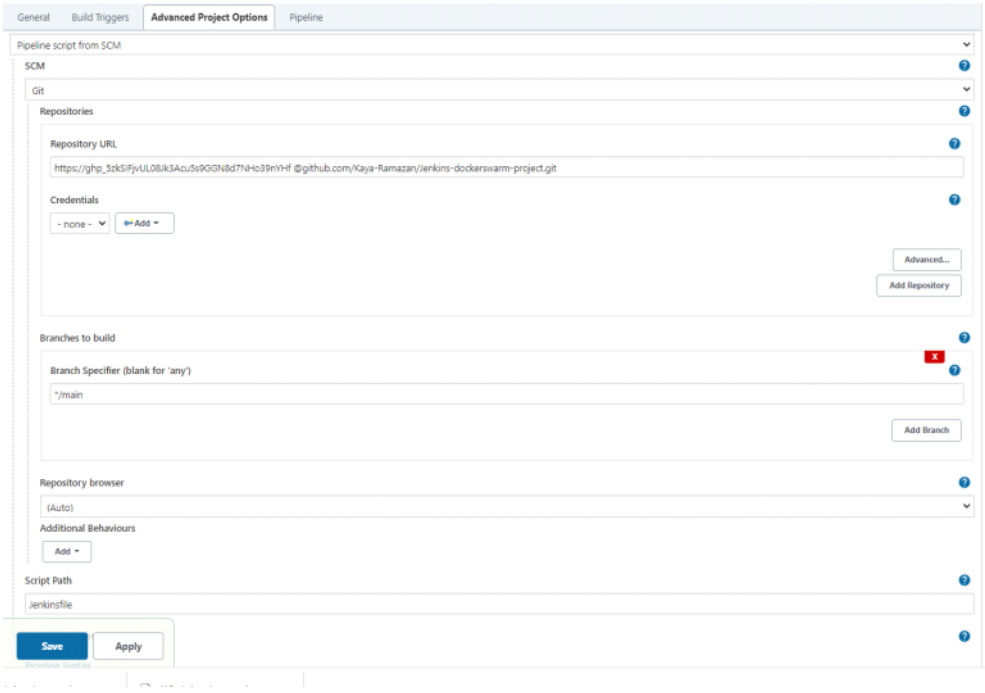


Artik jenkins server uzerinden Terraform calistirabiliriz

Ve bir job olusturuyoruz



Url adresimizi yazarken http// token adresinmiz @ github adresimiz seklinde yapiyoruz



Ve pipline olustu

Lokalde olustirdugumuz dosyaya bu filelarimizi kopyaliyoruz diger dosyalari bu folderin disinda olusturacagiz Github a daha buyuk boyutta dosya göndermemek icin bu yöntemi uyguluyoruz

Jenkins part isimli dosyada da gereksinimlerimiz var onlari kullnacagiz Bir onceki derste terrafrm ile 5 node dan olusan dcker swarm calisan bir inf kurmustuk aynisi cfn ile ve jenkins ekleyerek yapiyoruz

Jenkins server ayaga kaldiracagiz Jenkins de baya bir seyler olmali python docker vs Ecr a yetki vermemiz lazim Imaji gonderip imaj cekecegiz jenkins vasitasiyla Tf file sayesinde apply ile inf kuruldu

Ecr reposu create edildi imaj build edildi sonra docker swrm cluster ayaga kaldkti

Bunu tek bir dosya uzerinden yapmistim bunu burda pipeline ile farkli stagelere bolecegiz

Bu uygulamanin pipeline da nasil gorundugunu gormek amacimiz

Project-205: Jenkins Pipeline for Dockerized Phonebook Application (Python Flask & MySQL) Deployed on Docker Swarm

Description
This project aims to create a Jenkins pipeline to deploy the Phonebook Application web application with Docker Swarm on Elastic Compute Cloud (EC2) Instances by pulling the app images from the AWS Elastic Container Registry (ECR) repository.

Problem Statement

- Your company has recently started a project that aims to serve as phonebook web application. Your teammates have started to work on the project and developed the UI and backend part of the project and they need your help to deploy the app in development environment.
- You are, as a cloud engineer, requested to create a Jenkins pipeline to deploy the Phonebook Application in the development environment on Docker Swarm on AWS EC2 Instances using AWS Cloudformation Service to showcase the project.
- To prepare the application for deployment, you need to;
  - Create a new public repository for the project on GitHub.
  - Create docker image using the `Dockerfile` from the base image of `python:alpine`.
  - Deploy the app on swarm using `docker compose`. To do so on the `Compose` file;
    - Create a MySQL database service with one replica using the image of `mysql:5.7`;
      - attach a named volume to persist the data of database server.
      - attach `init.sql` file to initialize the database using `configs`.
    - Configure the app service to;
      - pull the image of the app from the AWS ECR repository.
      - deploy the one app for each swarm nodes using `global` mode.
      - run the app on `port 80`.
    - Use a custom network for the services.
- You are also requested; to use AWS ECR as image repository, to create Docker Swarm with 3 manager and 2 worker node instances, to automate the process of Docker Swarm initialization through Cloudformation in the development environment. So, to prepare the infrastructure, you can configure Cloudformation template using the followings;
  - The application stack should be created with new AWS resources.
  - The application should run on Amazon Linux 2 EC2 Instance
  - EC2 Instance type can be configured as `t2.micro`.
  - To use the AWS ECR as image repository;
    - Enable the swarm node instances with IAM Role allowing them to work with ECR repos using the instance profile.
    - Install AWS CLI `Version 2` on swarm node instances to use `aws ecr` commands.
    - Use Amazon ECR Credential Helper to allow Docker to interact with ECR easily.
  - To automate the process of Docker Swarm initialization;
    - Install the docker and docker-compose on all nodes (instances) using the `user-data` bash script.
    - Create a separate instance with `instance profile` to be first manager node of the swarm. Within the `user-data` script;
      - Set the first manager node hostname as `Grand-Master`.
      - Initialize Docker swarm.
      - Create a docker service named `viz` on the manager node on port `8080` using the `dockersamples/visualizer` image, to monitor the swarm nodes easily.
      - Download `docker-compose.yml` file from the repo and deploy application stack on Docker Swarm.
    - Create a launch template with `instance profile` for Manager Nodes. Within the `user-data` script;
      - Install the python `ec2instanceconnectcli` package for `mssh` command.
      - Connect from manager node to the `Grand-Master` to get the `join-token` and join the swarm as manager node using `mssh` command.
    - Create two manager node instances using the `Manager Launch Template`.
    - Create a launch template with `instance profile` for Worker Nodes. Within the `user-data` script;
      - Install the python `ec2instanceconnectcli` package to use `mssh` command.
      - Connect from worker node to the `Grand-Master` to get the `join-token` and join the swarm as worker node using `mssh` command.
    - Create two worker node instance using the `Worker Launch Template`.
  - Create a single security group for all swarm nodes and open necessary ports for the app and swarm services.
  - Tag the swarm node instances appropriately as `Docker Manager/Worker <Number> of <StackName>` to discern them from AWS Management Console.
  - The Web Application should be accessible via web browser from anywhere.
  - Phonebook App Website URL, Visualization App Website URL should be given as output by Cloudformation Service, after the stack created.
- To create a Jenkins Pipelines, you need to launch a Jenkins Server with security group allowing SSH (port 22) and HTTP (ports 80, 8080) connections. For this purpose, you can use pre-configured [*Cloudformation Template for Jenkins Server enabled with Git, Docker, Docker Compose and also configured to work with AWS ECR using IAM role*](./clarusway-jenkins-with-git-docker-ecr-cfn.yml).

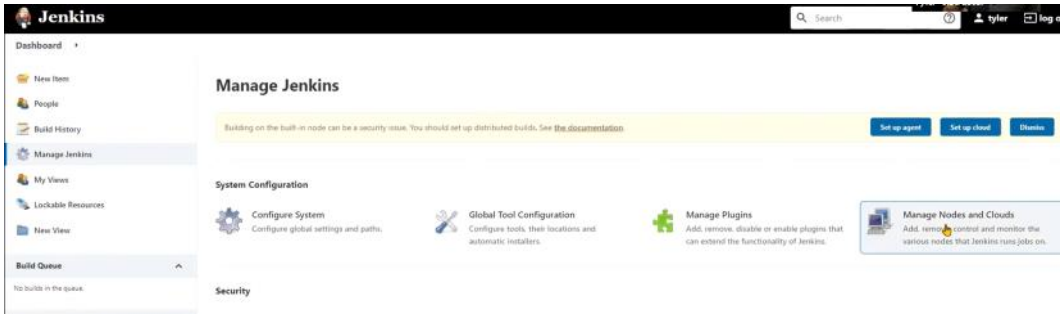
Istrelerimiz bunlardi

- Create a Jenkins Pipeline with following configuration;
  - Create an image repository on ECR for the app. # bir ECR reposu olusturracak
  - Build the application Docker image and push it to the ECR repository.
  - Build the infrastructure for the app on EC2 instances using Cloudformation template.
  - Deploy the application on Docker Swarm

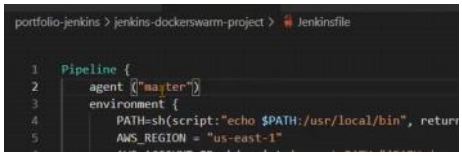
Project Skeleton

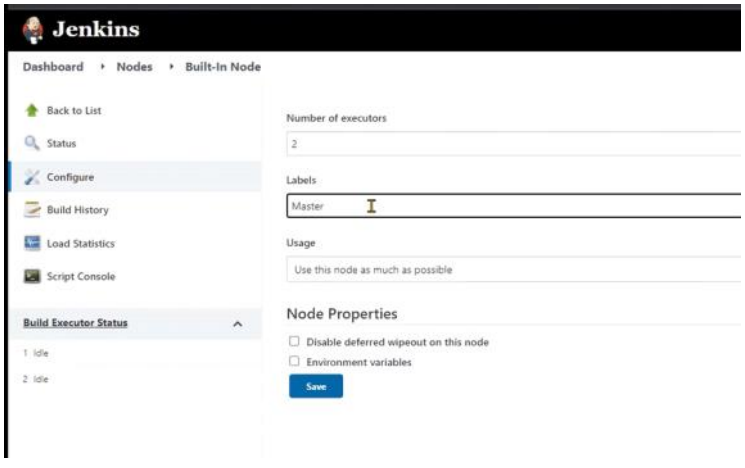
```text
205-jenkins-pipeline-for-phonebook-app-on-docker-swarm (folder)
|
|----readme.md                # Given to the students (Definition of the project)
|----phonebook-cfn-template.yml # To be delivered by students (Cloudformation template)
|----Dockerfile               # To be delivered by students
|----docker-compose.yml       # To be delivered by students
|----init.sql                 # Given to the students (SQL statements to initialize db)
|----jenkins-cfn-template.yml  # Given to the students (Cloudformation template for Jenkins Server)
|----Jenkinsfile              # To be delivered by students
|----app
     |----phonebook-app.py    # Given to the students (Python Flask Web Application)
     |----requirements.txt    # Given to the students (List of Flask Modules/Packages)
     |----templates
          |----index.html     # Given to the students (HTML template)
          |----add-update.html # Given to the students (HTML template)
          |----delete.html    # Given to the students (HTML template)
```

**** Simdi jenkins file olusturacagiz



Eger ki manage jenkins de manage nodes da label olarak master diye belirtirsek agent da master demmeiz gerekir ki byu su anlama gelir master nodes haricinde calsima

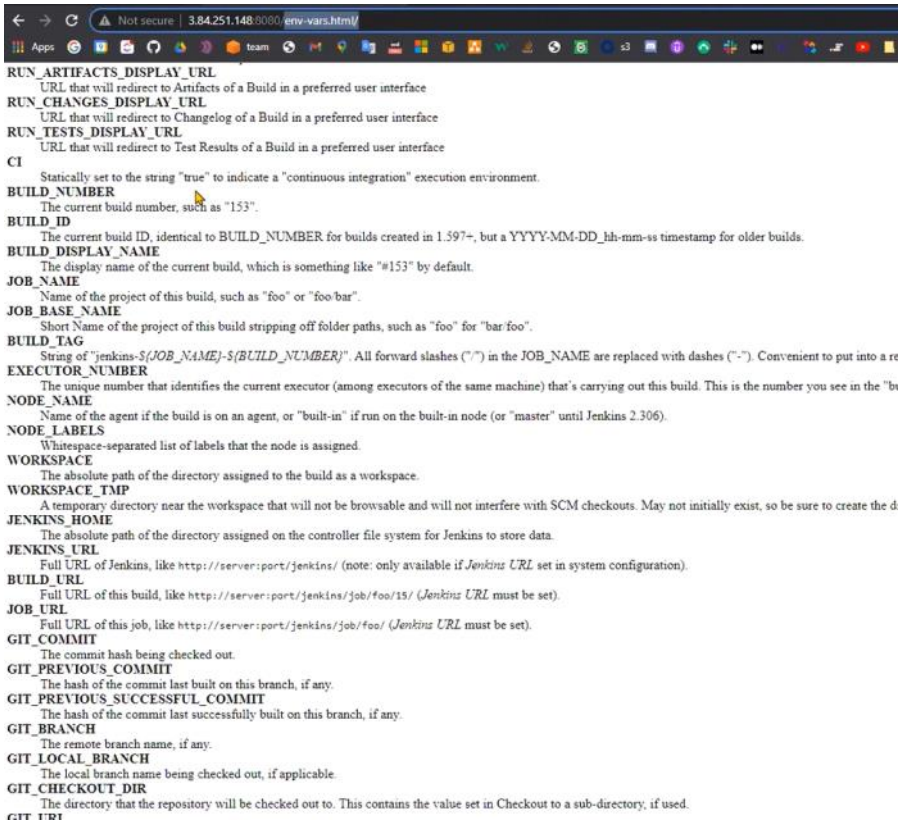




```
pipeline {      #
    agent any
    environment {
        PATH=sh(script:"echo $PATH:/usr/local/bin", returnStdout:true).trim()  #
AWS komutlarinin calismasi icin Jenkins in hangi ped e bakmasi gerektigini bilmesi
gerekiyor
        AWS_REGION = "us-east-1"
        AWS_ACCOUNT_ID=sh(script:'export PATH="$PATH:/usr/local/bin" && aws sts
get-caller-identity --query Account --output text', returnStdout:true).trim()  #
account id yi almaizi sagliyopr
        ECR_REGISTRY="${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com"
        // ECR_REGISTRY = "046402772087.dkr.ecr.us-east-1.amazonaws.com"
        APP_REPO_NAME = "clarusway-repo/phonebook-app"
        APP_NAME = "phonebook"
        AWS_STACK_NAME = "Call-Phonebook-App-${BUILD_NUMBER}"
        CFN_TEMPLATE="phonebook-docker-swarm-cfn-template.yml"
        CFN_KEYPAIR="tyler-team"
        HOME_FOLDER = "/home/ec2-user"
        GIT_FOLDER = sh(script:'echo ${GIT_URL} | sed "s/.*\\///;s/.git$//"',
returnStdout:true).trim()
    }
    stages {   # stagelerimizi bizden isterler göre yaziyoruz
        stage('Create ECR Repo') {   # ECR repo create deiyoruz
            steps {
                echo 'Creating ECR Repo for App'   # cli komutlari ile ecr i
jenkins e create ettirecegiz
                sh """
                aws ecr create-repository \
                  --repository-name ${APP_REPO_NAME} \
                  --image-scanning-configuration scanOnPush=false \
                  --image-tag-mutability MUTABLE \
                  --region ${AWS_REGION}
                """
            }
        }
        stage('Build App Docker Image') {
            steps {
                echo 'Building App Image'
                sh 'docker build --force-rm -t "$ECR_REGISTRY/
$APP_REPO_NAME:latest" .'
                sh 'docker image ls'
            }
        }
        stage('Push Image to ECR Repo') {  # ERC repoyu Push ediyrruz
            steps {
                echo 'Pushing App Image to ECR Repo'
                sh 'aws ecr get-login-password --region ${AWS_REGION} | docker
login --username AWS --password-stdin "$ECR_REGISTRY"'
                sh 'docker push "$ECR_REGISTRY/$APP_REPO_NAME:latest"'
            }
        }
        stage('Create Infrastructure for the App') {  # infrastructure cretae
eediyoruz
            steps {
                echo 'Creating Infrastructure for the App on AWS Cloud'
                sh "aws cloudformation create-stack --region ${AWS_REGION} --
stack-name ${AWS_STACK_NAME} --capabilities CAPABILITY_IAM --template-body file://
${CFN_TEMPLATE} --parameters ParameterKey=KeyPairName,ParameterValue=
${CFN_KEYPAIR}"
                script {

                    while(true) {   # while döngusu ile infstracture calisiyor mu
calismiyormu onu kont roledecegiz
                        s
                        echo "Docker Grand Master is not UP and running yet. Will
try to reach again after 10 seconds..."
                        sleep(10)
                        ip = sh(script:'aws ec2 describe-instances --region
${AWS_REGION} --filters Name=tag-value,Values=docker-grand-master Name=tag-
value,Values=${AWS_STACK_NAME} --query
Reservations[*].Instances[*].[PublicIpAddress] --output text | sed "s/\\s*None
\\s*//g"', returnStdout:true).trim()    # aws insstancelarinin calistigi region a
göre bulup bunlardan filter ile suzup aradigimiz valudegeri grand master olanlari
bulup getiriyor
                        if (ip.length() >= 7) {  # ip 7 karakterden fazladir ve
eger ki 7 karakterden fazla ise olustugu nu dusunebiliriz
                            echo "Docker Grand Master Public Ip Address Found:
$ip"

                            env.MASTER_INSTANCE_PUBLIC_IP = "$ip"
                            break
                        }
                    }
                }

            }
        }
        stage('Test the Infrastructure') {  # Infstracture i test ediyoruz
```

```
            steps {
                echo "Testing if the Docker Swarm is ready or not, by checking Viz
App on Grand Master with Public Ip Address: ${MASTER_INSTANCE_PUBLIC_IP}:8080"
                script {
                    while(true) {  # curl komutu kullanarak master instance public
ipsine baglanmaya calisiypruz     burda da amac docker swarm ayaga kalkti mi
vizulation calisiyor mu onu kontrol etmek
                      try {
                        sh "curl -s --connect-timeout 60
${MASTER_INSTANCE_PUBLIC_IP}:8080"   # 60 saniye icinde baglanamiyorsa 5 saniye
daha bekliyor ve
                        echo "Successfully connected to Viz App."
                        break
                      }
                      catch(Exception) {
                        echo 'Could not connect Viz App'
                        sleep(5)
                      }
                    }
                }

            }
        }
        stage('Deploy App on Docker Swarm'){   # uygulama yi daha sonra docker
swarm uzerinde n deploy edecegiz
            environment {
                MASTER_INSTANCE_ID=sh(script:'aws ec2 describe-instances --region
${AWS_REGION} --filters Name=tag-value,Values=docker-grand-master Name=tag-
value,Values=${AWS_STACK_NAME} --query
Reservations[*].Instances[*].[InstanceId] --output text',
returnStdout:true).trim()
            }
            steps {

                echo "Cloning and Deploying App on Swarm using Grand Master with
Instance Id: $MASTER_INSTANCE_ID"
                sh 'mssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no --region ${AWS_REGION} ${MASTER_INSTANCE_ID} git clone
${GIT_URL}'  # master instance baglan ve git clone komutunu calistir anlamina
geliyor repostory den dosyalariizi cekmeizi saglayacak
                sleep(10)
                sh 'mssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no --region ${AWS_REGION} ${MASTER_INSTANCE_ID} docker stack
deploy --with-registry-auth -c ${HOME_FOLDER}/${GIT_FOLDER}/docker-compose.yml
${APP_NAME}'
            }
        }

    }
    post {           # ECR calissada calismasa da lokaldeki imajlari
silebiliryoruz  bu kisim pipline in siónunda yapmak istedigimiz seyler icin
        always {
            echo 'Deleting all local images'
            sh 'docker image prune -af'
        }
        failure {  # eger failure cikarsa ECR reposunu siliyoruz cunku silmez isek
bize ayni isim de ECR oldugunu söykleyecek
            echo 'Delete the Image Repository on ECR due to the Failure'
            sh """
                aws ecr delete-repository \
                  --repository-name ${APP_REPO_NAME} \
                  --region ${AWS_REGION}\
                  --force
                """
            echo 'Deleting Cloudformation Stack due to the Failure'   #  CLI ile
resource lari tekrardan siliyoruz
                sh 'aws cloudformation delete-stack --region ${AWS_REGION} --
stack-name ${AWS_STACK_NAME}'
        }
    }
}
```

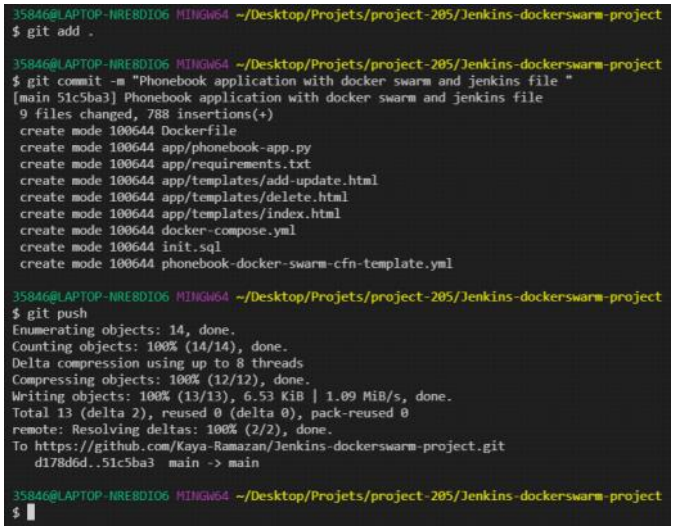
Ayrica jenkinse baglandigimiz konsola gidip : public ip : 8080 / env-vars.html/ de digimiz de



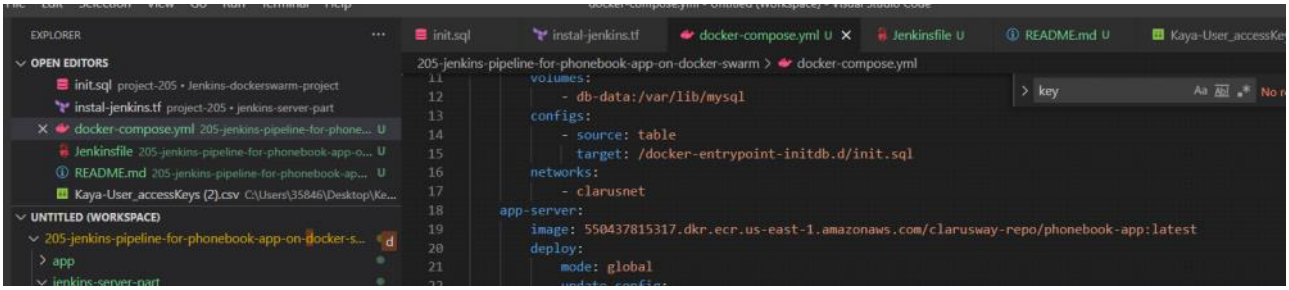
Eger ki

Jenkins file mizda hazir olduguna göre artik dosyalarimizi push eedbiliriz

Git hub a göndermek istedigimiz dosyalari docker swarm klasöru icerisinde bulunduruyorduk



Awx consolda ecr create ettigimizde bulunan image numberiimiz ile docker compose yaml da bulunan image numaralarimizi ayni olmasi gerekiyor



Expected Outcome
![Phonebook App Search Page](./search-snapshot.png)

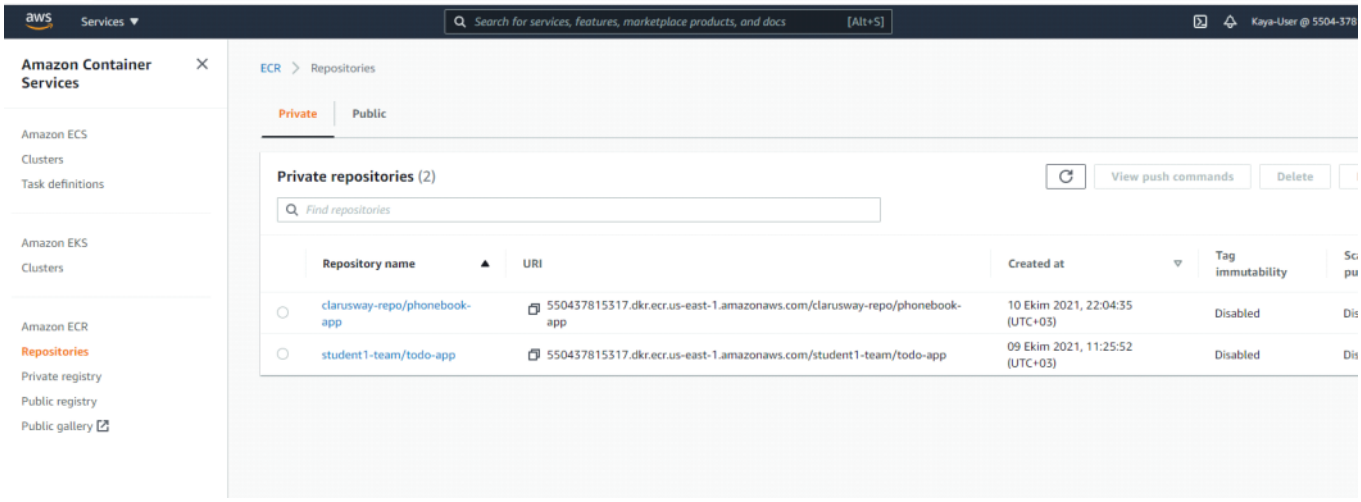
1-26

At the end of the project, following topics are to be covered;
- Jenkins Pipeline Configuration
- Docker Swarm Deployment
- Web App and MySQL Database Configuration in Docker Swarm
- Bash scripting
- AWS ECR as Image Repository
- AWS IAM Policy and Role Configuration
- AWS EC2 Launch Template Configuration
- AWS EC2 Configuration
- AWS EC2 Security Group Configuration
- AWS Cloudformation Service
- AWS Cloudformation Template Design
- Git & Github for Version Control System

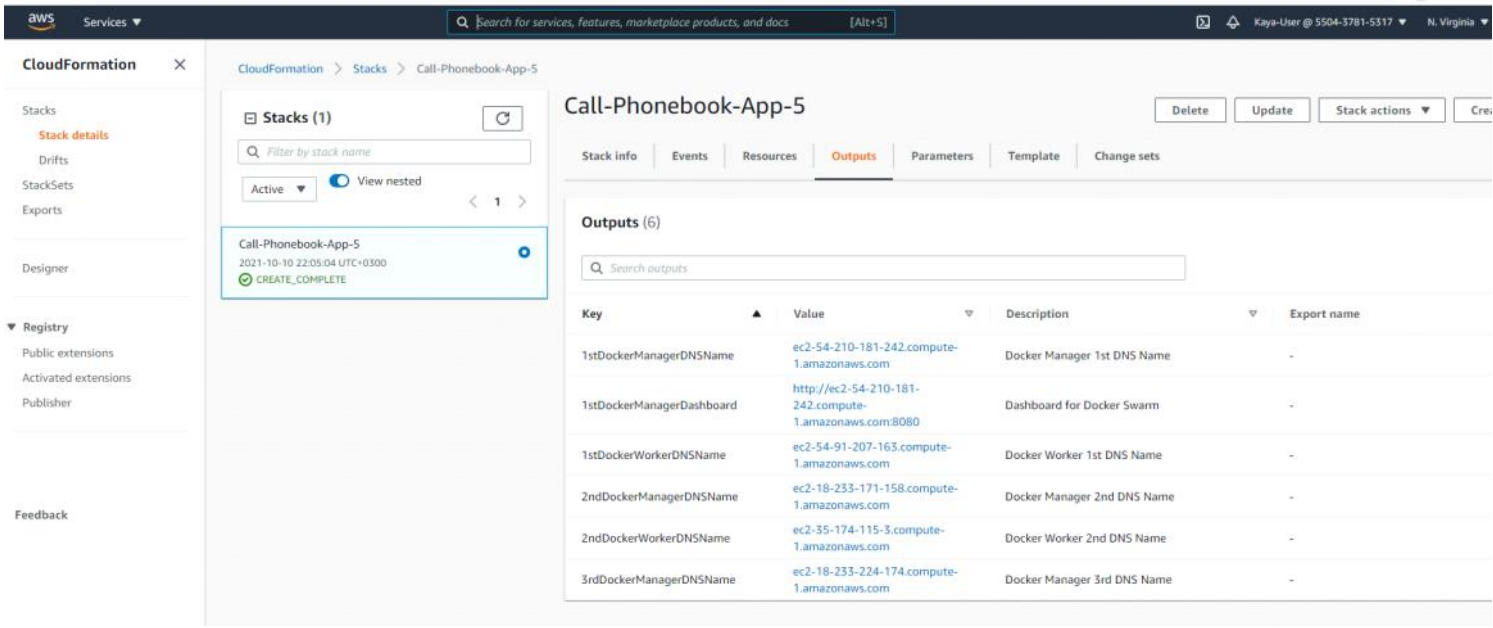
At the end of the project, students will be able to;
- demonstrate how to configure Jenkins pipeline to deploy app on Docker Swarm together with Cloudformation Template.
- demonstrate how to configure Dockerfile and docker-compose files.
- set up a Docker Swarm cluster to work with AWS ECR using Cloudformation.
- deploy an application stack on Docker Swarm.
- create and configure AWS ECR from the AWS CLI.
- use Docker commands effectively to tag, push, and pull images to/from ECR.
- demonstrate bash scripting skills using `user data` section in Cloudformation to install and setup environment for Docker Swarm on EC2 Instances.
- demonstrate their configuration skills of AWS EC2, Launch Templates, IAM Policy, Role, Instance Profile, and Security Group.
- configure Cloudformation template to use AWS Resources.
- show how to launch AWS Cloudformation Templates from AWS CLI.
- apply git commands (push, pull, commit, add etc.) and Github as Version Control System.

Resources
- [AWS Cloudformation User Guide](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html)
- [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/index.html)
- [AWS ECR Credential Helper](https://github.com/awslabs/amazon-ecr-credential-helper)
- [Authenticating Amazon ECR Repositories for Docker CLI with Credential Helper](https://aws.amazon.com/blogs/compute/authenticating-amazon-ecr-repositories-for-docker-cli-with-credential-helper/)
- [Docker Compose File Reference](https://docs.docker.com/compose/compose-file/)
- [Docker Reference Page](https://docs.docker.com/reference/)
- [EC2 Instance Connect](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Connect-using-EC2-Instance-Connect.html)
- [Jenkins Handbook](https://www.jenkins.io/doc/book/)

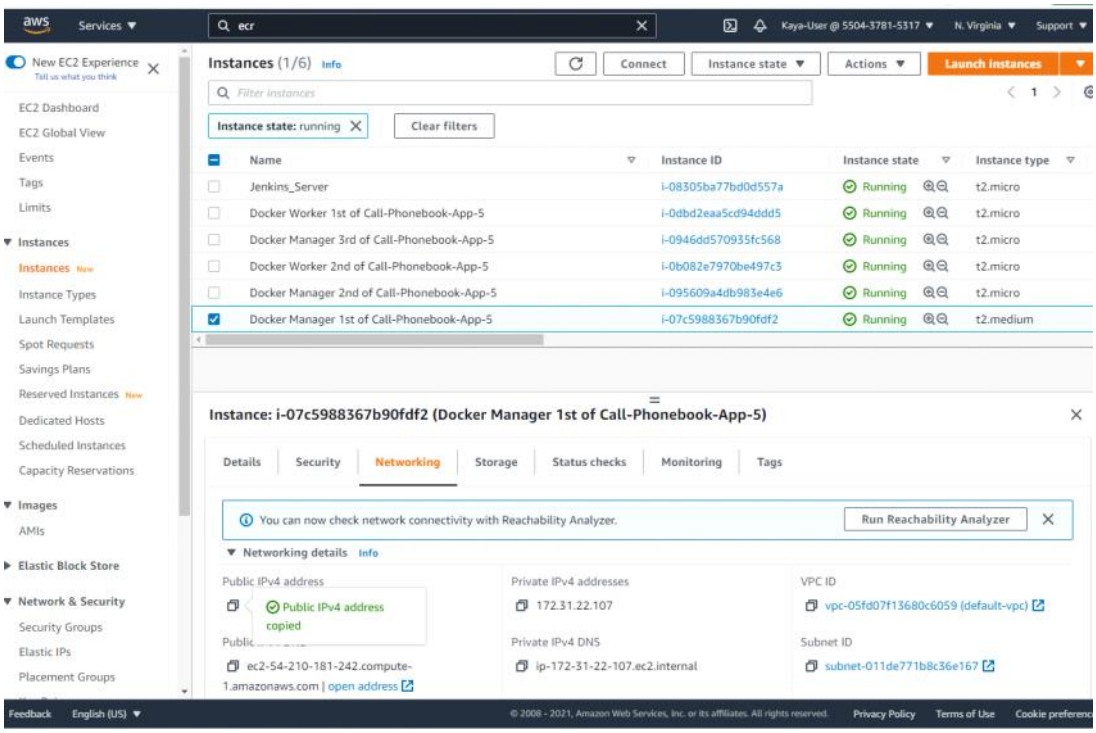
Sonucta ECR repomuzada stack olustu

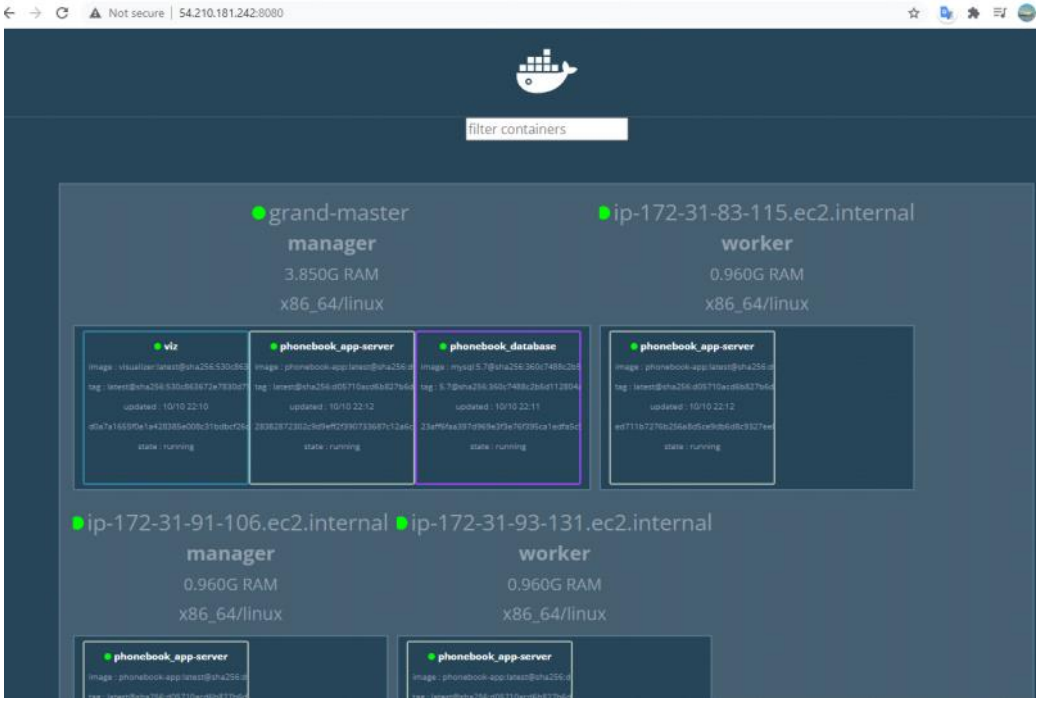


Cloudformation da stack de olusmus durumda burdaki outpud ile konsolda ciktimizi aliyoruz

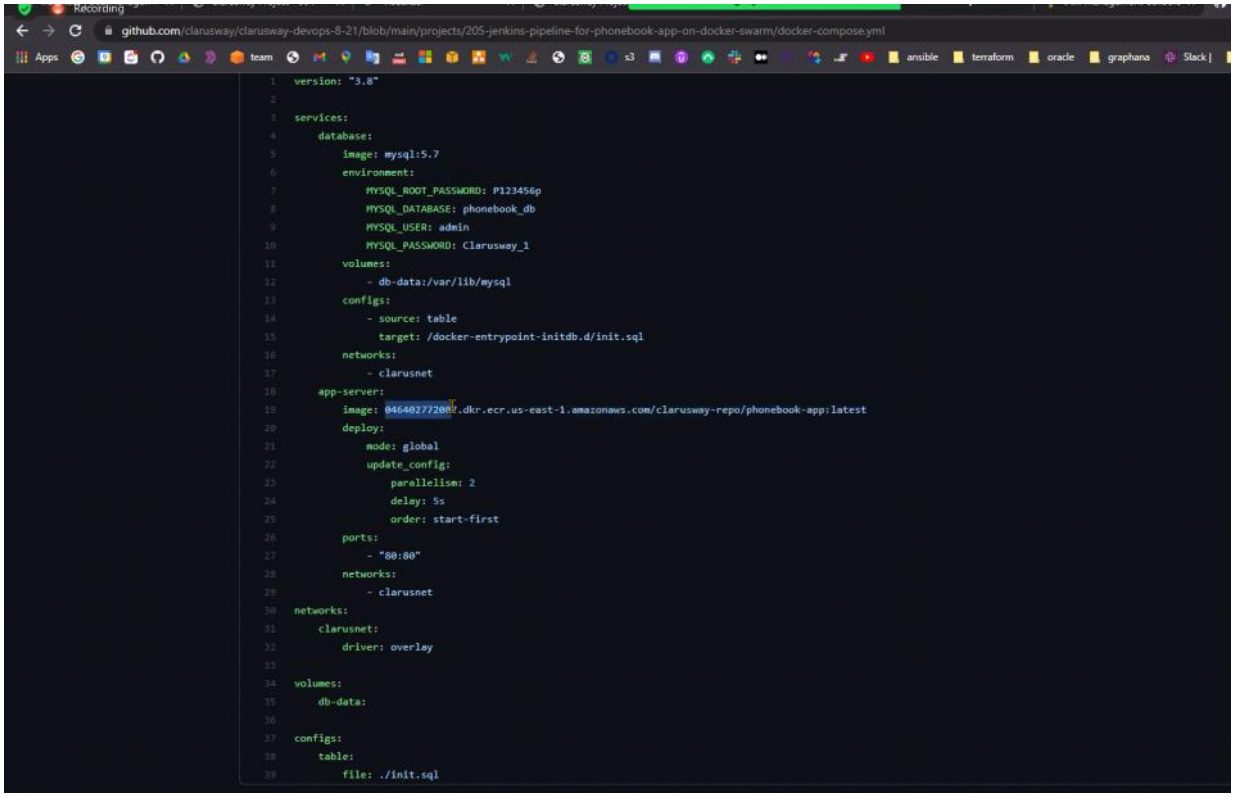


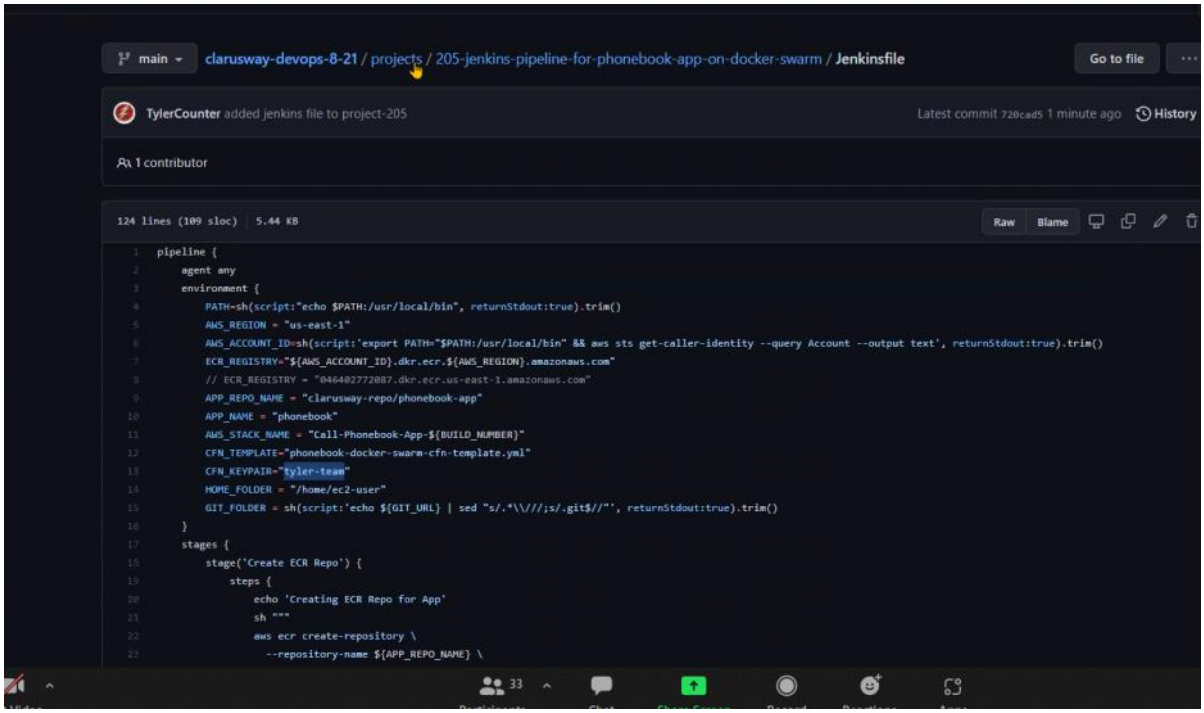
Ve son olarak Docker instanc larda olusan Docker Manager 1 in public ip ile : 8080 den baglandigimizda





File olusturuken degistirmeyi unutmamiz gereken husular


github.com/clarusway/clarusway-devops-8-21/blob/main/projects/205-jenkins-pipeline-for-phonebook-app-on-docker-swarm/docker-compose.yml
version: "3.8"
services:
database:
image: mysql:5.7
environment:
MYSQL_ROOT_PASSWORD: P123456p
MYSQL_DATABASE: phonebook_db
MYSQL_USER: admin
MYSQL_PASSWORD: Clarusway_1
volumes:
- db-data:/var/lib/mysql
configs:
- source: table
target: /docker-entrypoint-initdb.d/init.sql
networks:
- clarusnet
app-server:
image: 046402772087.dkr.ecr.us-east-1.amazonaws.com/clarusway-repo/phonebook-app:latest
deploy:
mode: global
update_config:
parallelism: 2
delay: 5s
order: start-first
ports:
- "80:80"
networks:
- clarusnet
networks:
clarusnet:
driver: overlay
volumes:
db-data:
configs:
table:
file: ./init.sql


main
clarusway-devops-8-21 / projects / 205-jenkins-pipeline-for-phonebook-app-on-docker-swarm / Jenkinsfile
Go to file
TylerCounter added jenkins file to project-205
Latest commit 720cad5 1 minute ago
History
1 contributor
124 lines (109 sloc) 5.44 KB
Raw
Blame
pipeline {
agent any
environment {
PATH=sh(script:"echo $PATH:/usr/local/bin", returnStdout:true).trim()
AWS_REGION = "us-east-1"
AWS_ACCOUNT_ID=sh(script:'export PATH="$PATH:/usr/local/bin" && aws sts get-caller-identity --query Account --output text', returnStdout:true).trim()
ECR_REGISTRY="${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com"
// ECR_REGISTRY = "046402772087.dkr.ecr.us-east-1.amazonaws.com"
APP_REPO_NAME = "clarusway-repo/phonebook-app"
APP_NAME = "phonebook"
AWS_STACK_NAME = "Call-Phonebook-App-${BUILD_NUMBER}"
CFN_TEMPLATE="phonebook-docker-swarm-cfn-template.yml"
CFN_KEYPAIR="tyler-team"
HOME_FOLDER = "/home/ec2-user"
GIT_FOLDER = sh(script:'echo ${GIT_URL} | sed "s/.*\\///;s/.git$//"', returnStdout:true).trim()
}
stages {
stage('Create ECR Repo') {
steps {
echo 'Creating ECR Repo for App'
sh """
aws ecr create-repository \
--repository-name ${APP_REPO_NAME} \