Task-2(JENKINS)

14 April 2023 16:57

Next Set of Tasks [Part-2]:
1. Install Jenkins on local
2. Create a Jenkins Job to Run a react-app from GitHub : https://github.com/pratik-choudhari/react-todo-list
3. Build Docker Image using Docker file in above repo (below commands should be in windows batch command)
   docker build -t react-todo:latest .
   docker images
   docker run -d -p 3000:3000 --network host react-todo:latest

The To-DO app should be accessible on your local browser at http://localhost:3000

PART 1

1. Install Jenkins on local:
2. Downlaod the jenkins from https://www.jenkins.io/download/ .
3. Download and install jdk supported version and set it path.
4. Configured it with the port(8090).
5. Install the plugins (github, git , docker for this task) (disable VPN while installing plugin).
6. Create a account on the jenkins.
7. Create a new job using freestyle project.
8. Go to configure -> source code management. Select git to clone the git repo, add repo URL and credentials if private repo, else don't.
9. Go to configure ->build steps, select window batch commands and add the following commands

docker build -t react-todo:latest .
---------- This command build the image named react-todo using dockerfile in workspace.
docker images
---------- shows the images on the local.
docker run -d -p 3000:3000 react-todo:latest
---------- run the container at 3000 localhost and 3000 container port.

Go to webaddress localhost:3000, to do list should run on this page

Refered Video: How To Install Jenkins On Windows 10

Task-3/4 Declarative Pipeline

18 April 2023 10:27

Task:
1.Create similar job like in PART-2 , but now pipeline-as-a-code using Declarative Pipeline.
2. Learn more docker commands

1. Create a new job using pipeline project.
2. Install pipeline pluging for this task.
3. New Item-> selec Pipeline.
4. Go to configure ->Pipeline , select pipeline script.
5. Use pipeline syntax to generate snippet for pipeline.

```
pipeline {
   agent any
   stages{
      stage("git"){
         steps{
            git branch: 'main', url: 'https://github.com/pratik-choudhari/react-todo-list.git'
         }
      }
      stage("docker-build"){
         steps{
            bat 'docker images'
            bat 'docker build -t react-todo:latest .'
         }
      }
      stage("docker-image-check"){
         steps{
             bat 'docker images'
         }
      }

      stage("docker -container run"){
         steps{
           bat 'docker run -d -p 3000:3000 react-todo:latest'
         }
      }
      }
   }
```

And save.
Build the job.
Output -> to-do app accessible from localhost:3000.

Refered Video: https://www.youtube.com/watch?v=yp_PP_YcqLc

Next Set of Tasks [Part-4]:

1. Parameterize the pipeline job for giving custom Port Number
2. Install MiniKube on your local laptops

Part 4

1. Go to configure -> General , select
2. This project is parameterized
3. And add string parameter to make visible the UI to the user to enter the port.
4. In pipeline script add

```
parameters {
    string defaultValue: '3000', description: 'Parameterize the pipeline job for giving custom Port Number', name: 'PORT'
}
```

5. and bat "docker run -p ${PORT}:3000 react-todo:latest" command to run app at port given by user.
6. Use " " for docker while using dynamic values.
7. Click Build with paramteres enter the port number and check console output.

To-do list should be accesible at localhost:port given by user.

Task -5 (minikube kind:pod)

20 April 2023 10:31

Task

- **Now that minikube is installed, run the "To-Do" App via a POD and not docker container**
- **Expectation out of this task - To-Do app should be accessible inspite of deleting the running docker container**

Minikube : single node kubernetes cluster to run kubernetes on your local machine.

Kubectl: kubernetes CLI .

1. Install kubectl and minikube on your local machine (no need of VM, but docker should be present) . Check virtualisation is enabled on your machine (check task manager).
2. Run command minikube start , to start minikube on your machine.*(everytime we need to run minikube start, since it get delete when we close cmd or lock pc)*
3. Run command kubectl get node.
4. Some basic command
   a. *minikube start*
   b. *minikube stop*
   c. *minikube delete*
   d. *minikube service <service_name>*
   e. *minikube service <service_name> --url*
   f. *kubectl apply -f <file_name>*
   g. *kubectl get nodes*
   h. *kubectl get pods*
   i. *kubectl get all*
   j. *Kubectl describe resource <resource_name>*
5. To deploy the application using minikube, using docker image stored on ur local machine
   Build a docker a image in minikube docker daemon, since minikube will fetch the image either from public repo our its own docker env.

6. To build a image :
   a. Run following command (on windows)
      i. *minikube docker-env* ---sets the docker env
      ii. Run command recommended in cmd

```
C:\Users\firktan>minikube start
* minikube v1.30.1 on Microsoft Windows 10 Enterprise 10.0.19045.2728 Build 19045.2728
* Using the docker driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Restarting existing docker container for "minikube" ...
! This container is having trouble accessing https://registry.k8s.io
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\firktan>minikube docker-env
SET DOCKER_TLS_VERIFY=1
SET DOCKER_HOST=tcp://127.0.0.1:63887
SET DOCKER_CERT_PATH=C:\Users\firktan\.minikube\certs
SET MINIKUBE_ACTIVE_DOCKERD=minikube
REM To point your shell to minikube's docker-daemon, run:
REM @FOR /f "tokens=*" %i IN ('minikube -p minikube docker-env --shell cmd') DO @%i

C:\Users\firktan>@FOR /f "tokens=*" %i IN ('minikube -p minikube docker-env --shell cmd') DO @%i

C:\Users\firktan>^SS
```

      iii. *docker build -t react-todo .* (specify path if dockerfile is located in other folder/ open cmd in that folder and set minikube docker-env)
      iv. Check docker image in minikube docker
      v. Run *minikube ssh*
      vi. *docker images*
7. Create just pod.yaml file or (service.yaml we need to expose to other network) file with required configuration (imagePullPolicy :never {else it will give imagepullback error}
)
8. Build the file
   kubectl apply -f pod.yaml

   kubectl apply -f service.yaml
9. Check if pod is running
   kubectl get pod
   Or
   kubectl get all
10. Run *minikube service <service_name>* (in case of service)
11. Or run *kubectl port-forward pod-name local_port:container_port*
12. Delete the container , minikube will again generate a contianer.
13. And run *minikube service <service_name>*

Reference : https://medium.com/bb-tutorials-and-thoughts/how-to-use-own-local-doker-images-with-minikube-2c1ed0b0968

Task –6 deployment /ingress

Wednesday, April 26, 2023 11:24 AM

Follow same steps in task 5
With new deployment kind file with container port 3000 (since the react default port is 3000)and imagePullPolicy :never {else it will give imagepullback error}
Delete the pod
New pod will be generate
Access the application using the newly generated pod.

(if error occurs for image building : for npm install –silent error , then remove the –silent)

Task –7

Friday, April 28, 2023 9:37 AM

PART1:

- Add replicaset : n
- In your deployment kind of yaml file under spec and build the file.
- It will create n number of pods.

PART 2:

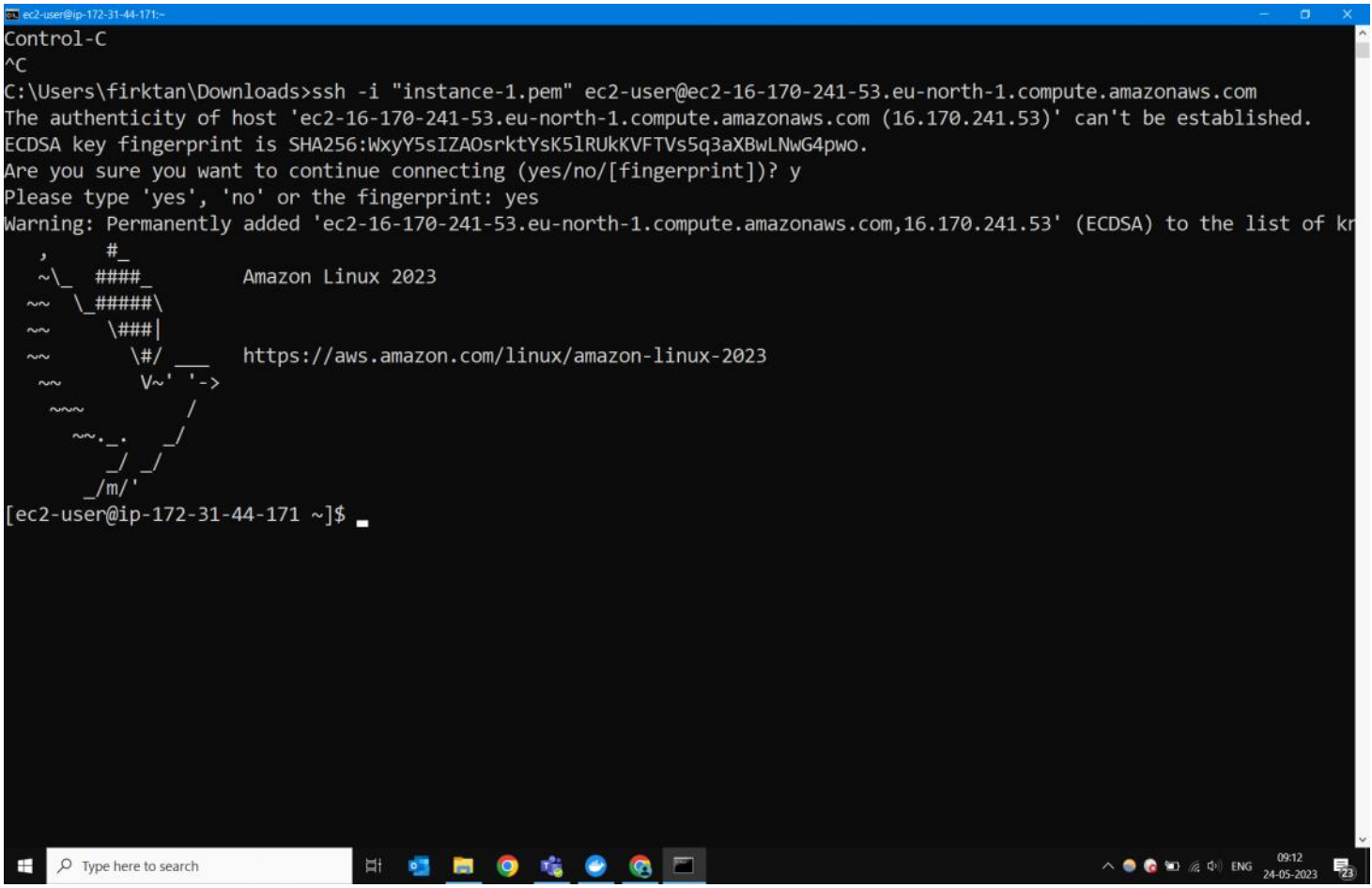
AWS CLOUD

1. Go to amazon cloud service click on complete sign up
2. Enter email address in root user set password and complete the procedure, require payment of 2 Rs, enter debit card details and complete the transaction.

PART 3:

EC2 INSTANCE

1. GO to Ec2 –dashboard
2. Click on launch an instance
3. Select the setting and launch the instance.



Task-8

24 May 2023 09:53

TASK:

- After SSH into EC2 , install Nginx in it
- Once nginx installed, make sure you are able to access nginx page via browser

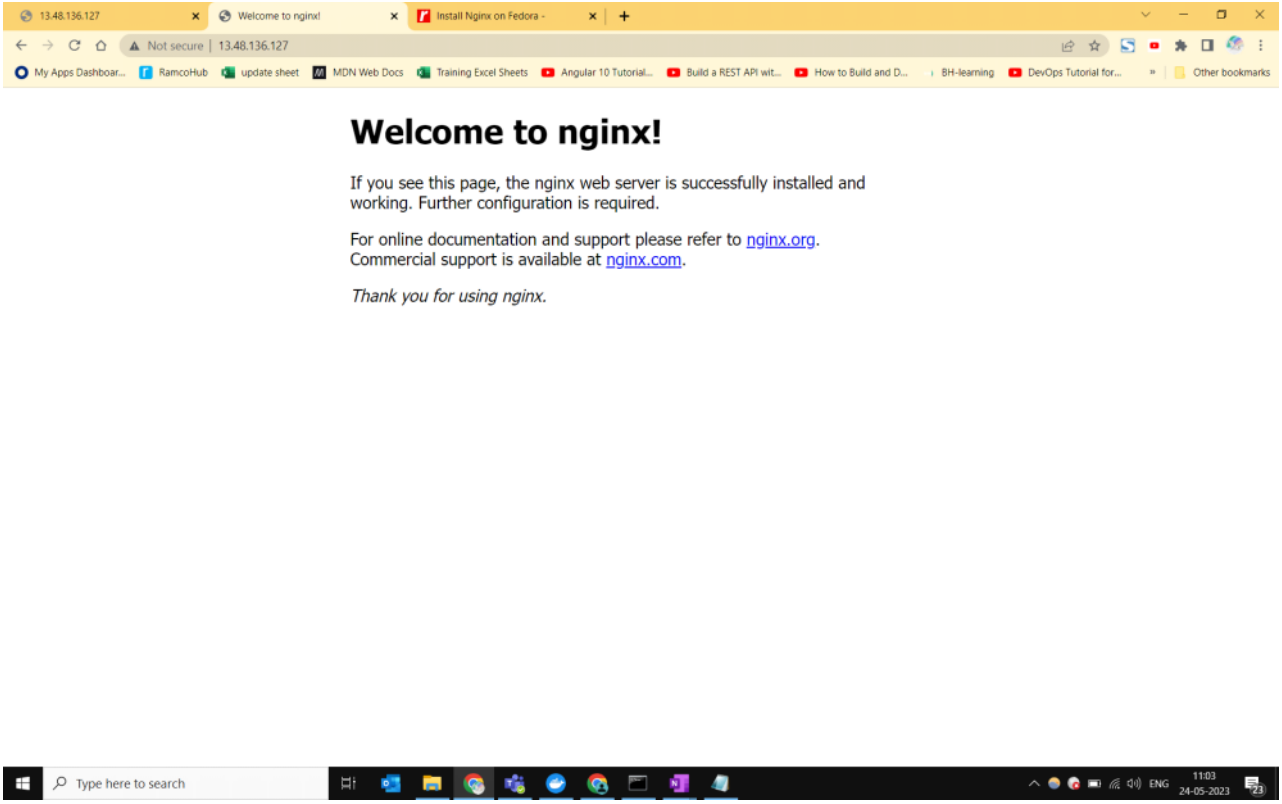
1. In previous task we launch the instance of amazon linux ,
2. Now to install Nginx in it, check the distro of your Amazon image in EC2 instance.

To troubleshoot:

Check the Amazon image version of your EC2 instance.
Command: cat /etc/image-id
If it is Amazon Linux 2023, it does not have extras.
However, most of the packages are already included in the distro.
You can run sudo yum install nginx which will install version 1.22

4. Install nginx in it
5. Enable the nginx service and check if it is accessible on browser using localhost.

From <https://stackoverflow.com/questions/75966794/sudo-amazon-linux-extras-command-not-found>



Task-9 Cloud formation

25 May 2023 10:30

AWS Cloud Formation is like template of instances which helps if we want to create same instances with same resources.
Stack keeps the all the resources of instance.
Aws CF is the template in json or yaml format which contains all configuration, metadata to create an instance.
Which keeps the information on the stack on the cloud , that can be updated, deleted which is called change set.

Task:

- **Now, create an EC2 instance via CloudFormation**
- **Outcome of this task should be EC2 created with CloudFormation Only**

Steps:

1. Create a yaml file of the instance template
2. Select the CloudFormation service from aws services
3. Click on *create stack*
4. Select *Upload a template file* and choose the yaml file created in step 1, aws itself creates and uploads the file into a S3 bucket.
5. Click on *next* and enter a stack name and click on *next* and keep everything as it is and click *next*
6. Here you can see the template url and click on *Create stack.*
7. Stack creation will start and after sometime the stack will be created and the EC2 Instance will be launched which you can view by clicking on the Physical ID of the instance.

Reference :
AWS Basics: Create EC2 instance using AWS CLoudFormation


```yaml
AWSTemplateFormatVersion: "2010-09-09"
Parameters:
  InstanceType:
    Description: EC2 Instance Type
    Type: String
    Default: t3.micro
    AllowedValues:
      - t3.micro
      - t3.small
      - m4.large
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      ImageId: ami-01c1e791e12290220
      SecurityGroups:
        - !Ref InstanceSecurityGroup

  InstanceSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp:  0.0.0.0/0
Outputs:
  InstanceId:
    Description: InstanceId of the EC2 instance
    Value: !Ref EC2Instance
  PublicDNS:
    Description: Public DNS of the EC2 instance
    Value: !GetAtt
            - EC2Instance
            - PublicDnsName
  PublicIP:
    Description: Public IP of the EC2 instance
    Value: !GetAtt
            - EC2Instance
            - PublicIp
```

Task - 10 EKS Cluster & Task 11

30 May 2023 12:12

Task 10 :

- **Create an EKS Cluster with 2 Nodes**
- **Create an EC2 and connect the cluster from that EC2 using kubectl command.**
- **Outcome of this should be, EKS Cluster with 2 nodes should be created in READY state.**

Steps:

1. Using eksctl -
2. Install eksctl and aws cli on your machine.
    a. Downlaod direct eksctl from this link https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_windows_amd64.zip

    a. Or refer for isntalling eksctl:
    b. https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html

    a. Check the version of eksctl after installing and setting the path.
3. Install AWS CLI
    a. Download the AWS CLI installer for windows by clicking on below link.
    b. https://awscli.amazonaws.com/AWSCLIV2.msi
    c. Check the aws version by *aws version*

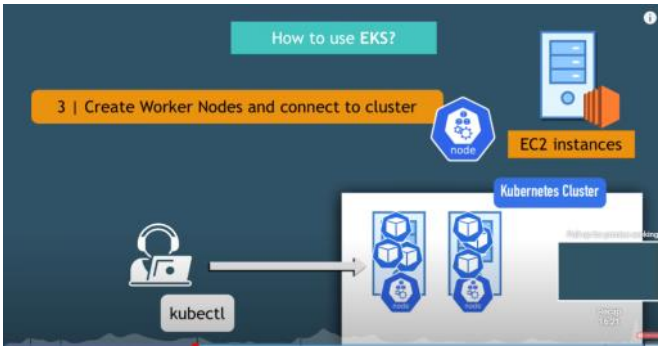
4. Create a IAM USER
    a. If you are instance to run the cluster create a new role with all required permissions and modify the instance by attaching the role else.
    b. For using AWS CLI create a IAM user
    c. Go to IAM service, select user and add new user. Add permissions.
    d. Create secret key and access id.
5. Go to CMD
    a. Type aws configure
    b. Enter the access id and secret key of newly created user.
    c. Now the cluster created will be associated with this user hence nodes will be visible.

6. To create a Cluster using eksctl

    a. Command:
    b. eksctl create cluster --name <cluster_name> --region <region_name> --node-type <node_type>

7. Check the stack it will show creation of cluster also 2 instances will be created as working node.
8. Go to cluster check the cluster , cluster with cluster_name will be created there, go to compute and check the nodes created using eksctl.
9. Open CMD
10. Give command: kubectl get node
11. It should show the two working node created in cluster in ready state.

EKS manages master node for you, user only need to manage worker nodes.
Worker nodes are here the EC2 instances.
How to use EKS:



Task 11:

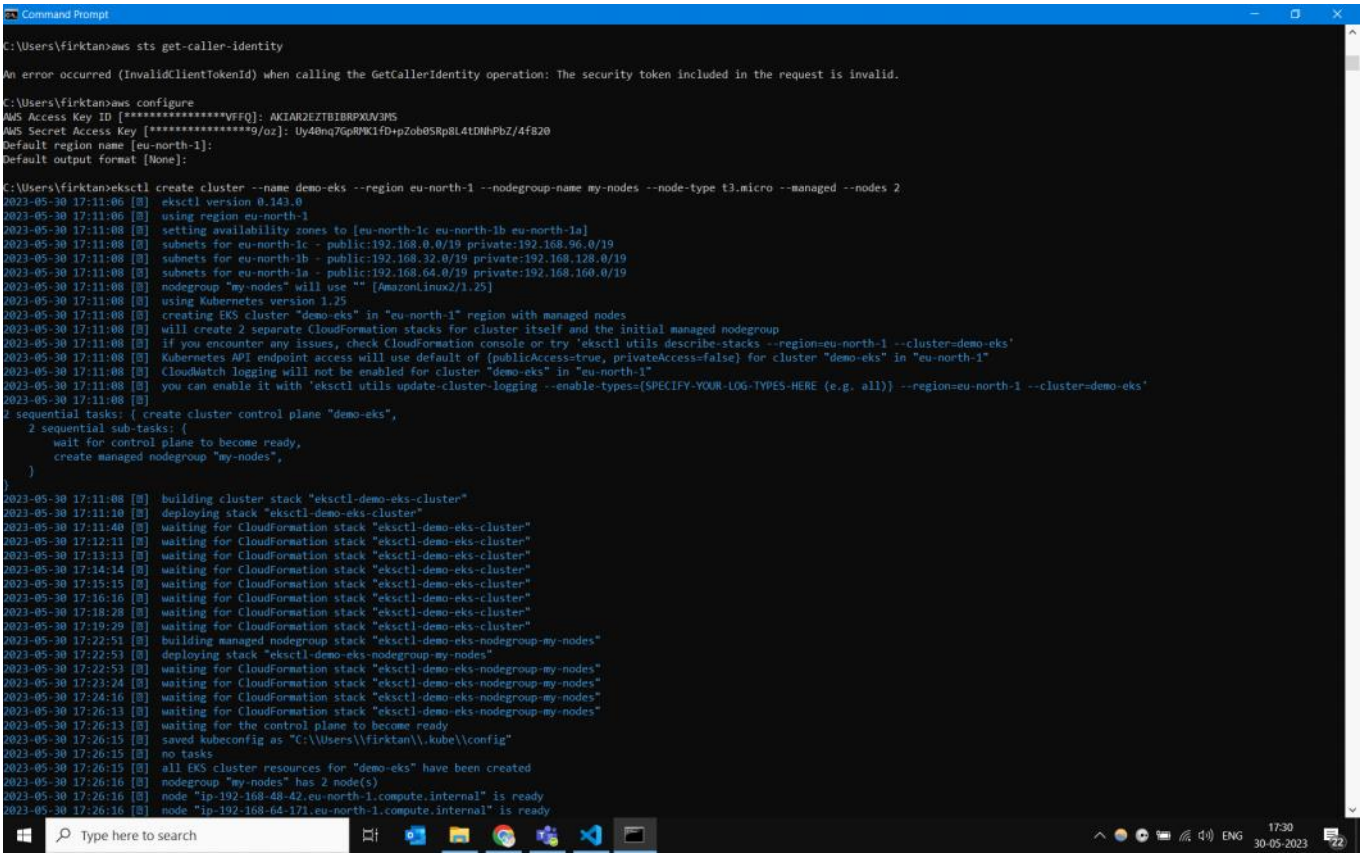
1. Create a deployment for the app.
2. Expose the deployement using Loadbalancer.
    a. Either specify in yaml file in services or give command on cmd to expose with --type = LoadBalancer
3. Go to AWS EC2 dashboard to check the loadbalancer. There should be instance of Load Balancer present there.
4. Kubectl get all or kubectl get services
5. Will show the services with loadbalancer copy the external IP and run it in the web-browser

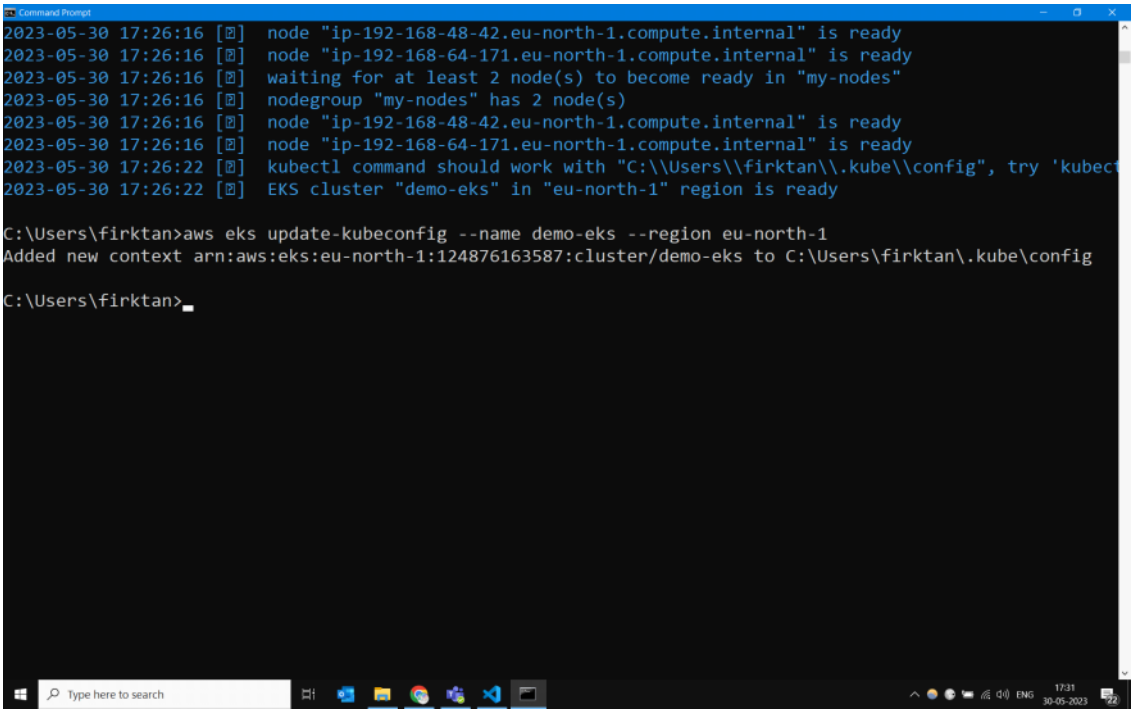
References:
https://www.youtube.com/watch?v=1O-I7NtwCeE

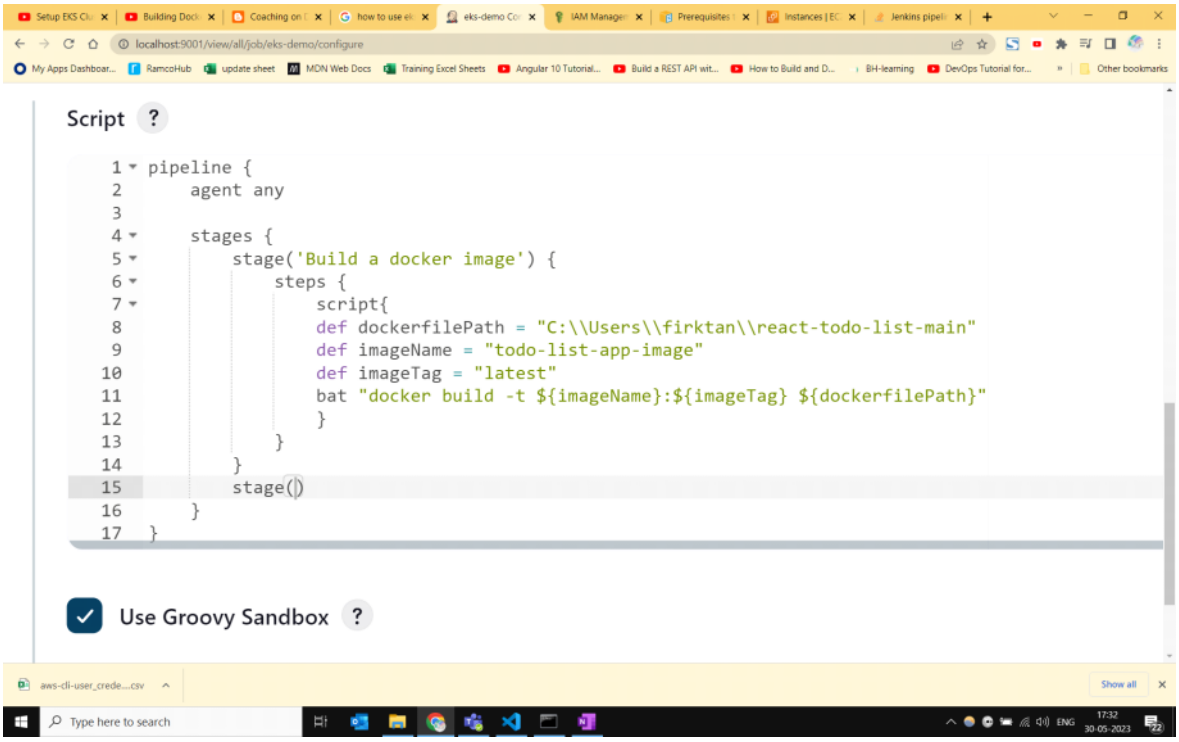
https://www.stacksimplify.com/aws-eks/eks-cluster/install-aws-eksctl-kubectl-cli/

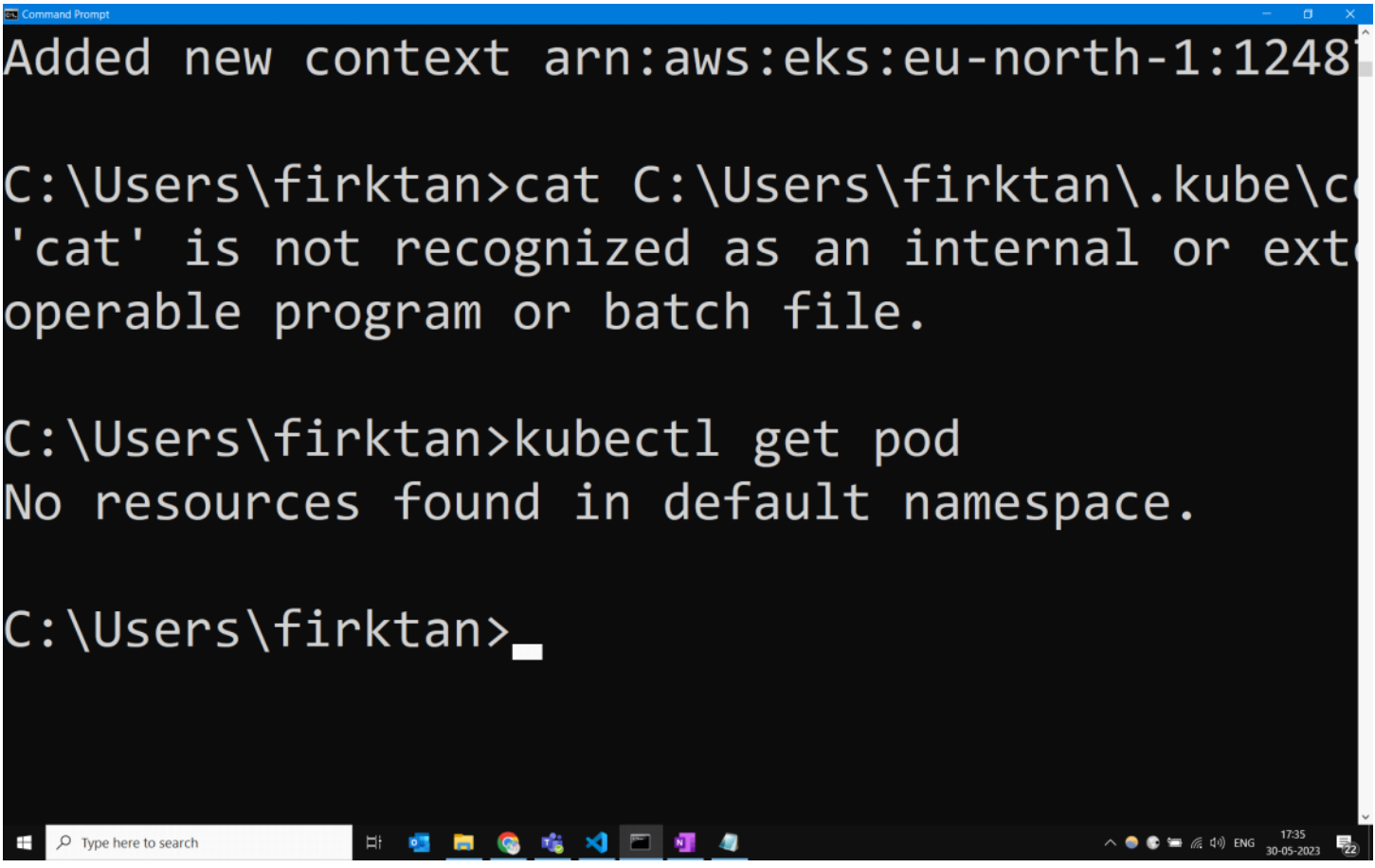
Task 10-11

30 May 2023 17:31






Script
pipeline {
agent any
stages {
stage('Build a docker image') {
steps {
script{
def dockerfilePath = "C:\\Users\\firktan\\react-todo-list-main"
def imageName = "todo-list-app-image"
def imageTag = "latest"
bat "docker build -t ${imageName}:${imageTag} ${dockerfilePath}"
}
}
}
stage()
}
}
Use Groovy Sandbox


Command Prompt
Added new context arn:aws:eks:eu-north-1:1248
C:\Users\firktan>cat C:\Users\firktan\.kube\c
'cat' is not recognized as an internal or ext
operable program or batch file.
C:\Users\firktan>kubectl get pod
No resources found in default namespace.
C:\Users\firktan>

Task-14

29 June 2023 18:39

Assignment :

1. **Create a Linux EC2 Instance using Terraform.**
2. **In the EC2 created in Step-1, write below two shell scripts:**

   a. write a Shell Bash Script to check if a number input from standard input is odd or even
   b. Print Fibonacci Series for 10 number.

Scripts:

1. **Write a Shell Bash Script for check if a number input from standard input is odd or even**
2. **Print Fibonacci Series for 10 numbers**

Steps:

1. Install terraform if window run choco intall terraform in powershell as a administrator(if chocolatey package is installed on windows)
2. Go through https://developer.hashicorp.com/terraform/downloads to install terraform on your machine.
3. Write a terraform file save as .tf file

```
provider "aws" {
  region = "us-west-2"
}
resource "aws_security_group" "shell_instance_sg" {
  name        = "allow-ssh"
  description = "Allow all traffic for SSH"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
resource "aws_instance" "shell_instance" {
  ami           = "ami-07dfed28fcf95241c"
  instance_type = "t2.micro"
  key_name      = "key_for_shell_instance"
  vpc_security_group_ids = [aws_security_group.shell_instance_sg.id]
  tags = {
    "Name" = "ExampleShellInstance"
  }
}
```

Here provider specifies the service that we will be using to create our infrastructure.
And declare resource using resource

4. Go to https://registry.terraform.io/providers/hashicorp/aws/latest/docs to read and get the syntax for AWS resources.

5. Save this file and run terraform init to initiate the registry and pull the provider.
   a. terraform fmt ----> to correct the indentation or alignment
   b. terraform validate ----> to check the syntax
   c. terraform plan -------> to see what resources will be made and what changes were implemented if any
   d. terrraform apply ------> it will create the resources after confirming from user.

6. Go to AWS console check the EC2 instances
7. Connect to the EC2 instance using ssh
8. Go to cmd run the command

9. After the instance is running check the machine type using command
   cat /etc/os-release
   To use corrrect command syntax(due to slight change in syntax)

10. To edit the file we can use vim editor or nano
    To install them, run following command (can use either of them):
    a. apt install vim

b. apt install nano

Check some basic command for vim editor --> https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

11. Next , run touch file_name.sh to create a empty file
12. Check the file using ls
    (vim file_name.sh will create empty file also and open it.)
    Command to edit the file
    a. nano filename.sh
    b. vim filename.sh

13. Add the shell script and save the file
14. To check the content of the file on console run cat filename.sh
15. Next change the permission of the file using chmod
    a. chmod 700
    b. or
    c. chmod 777

16. Next run ./filename.sh to execute the file.

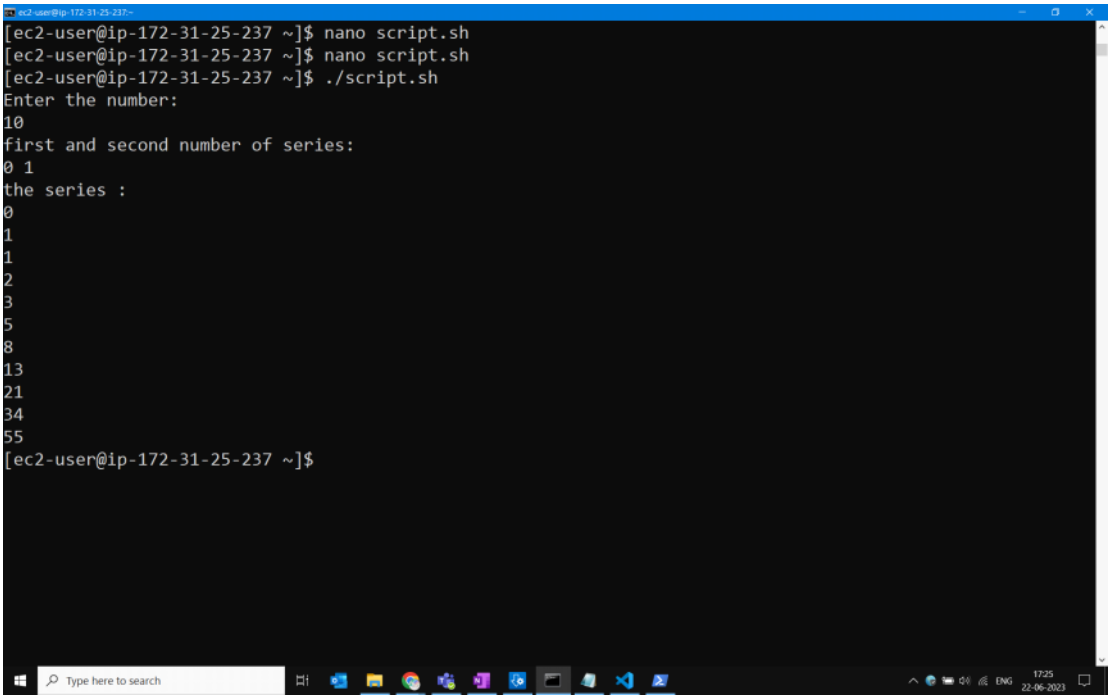
Write a Shell Bash Script for check if a number input from standard input is odd or even

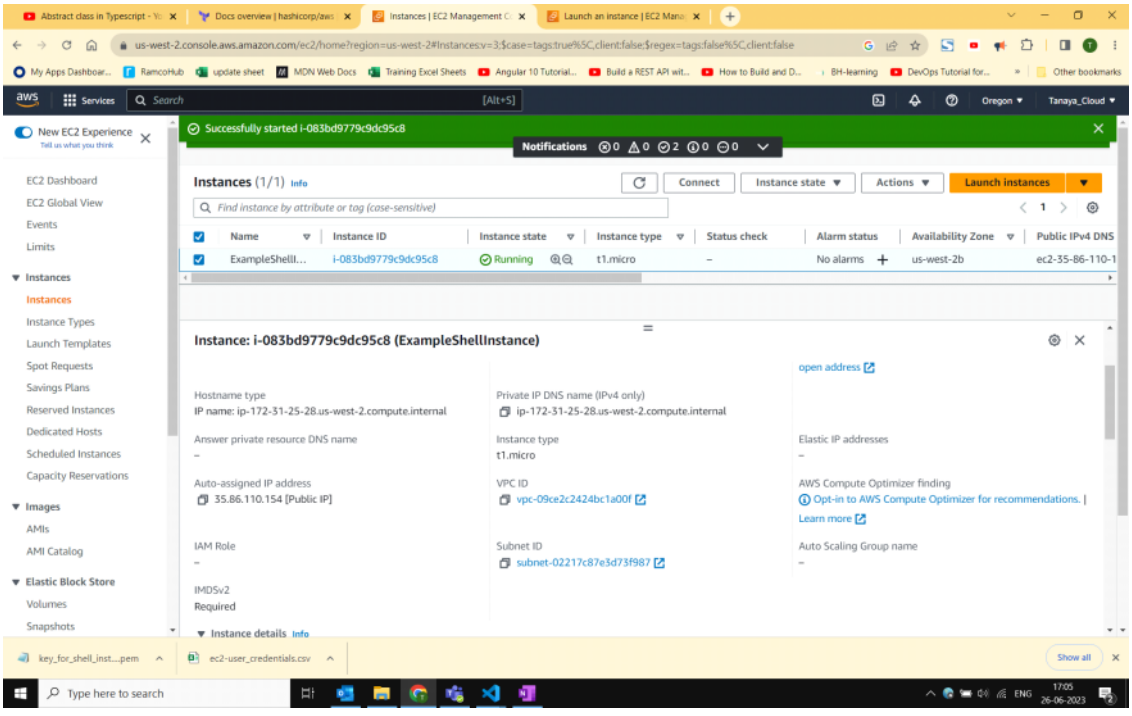
```
#!/bin/Bash
#take a input from  a user
echo "Enter the number: "
read number

#check if number is even or odd
echo "The given number is "
if [ $((number%2)) == 0 ] ;then
echo "even"
else
   echo "odd"
fi
```

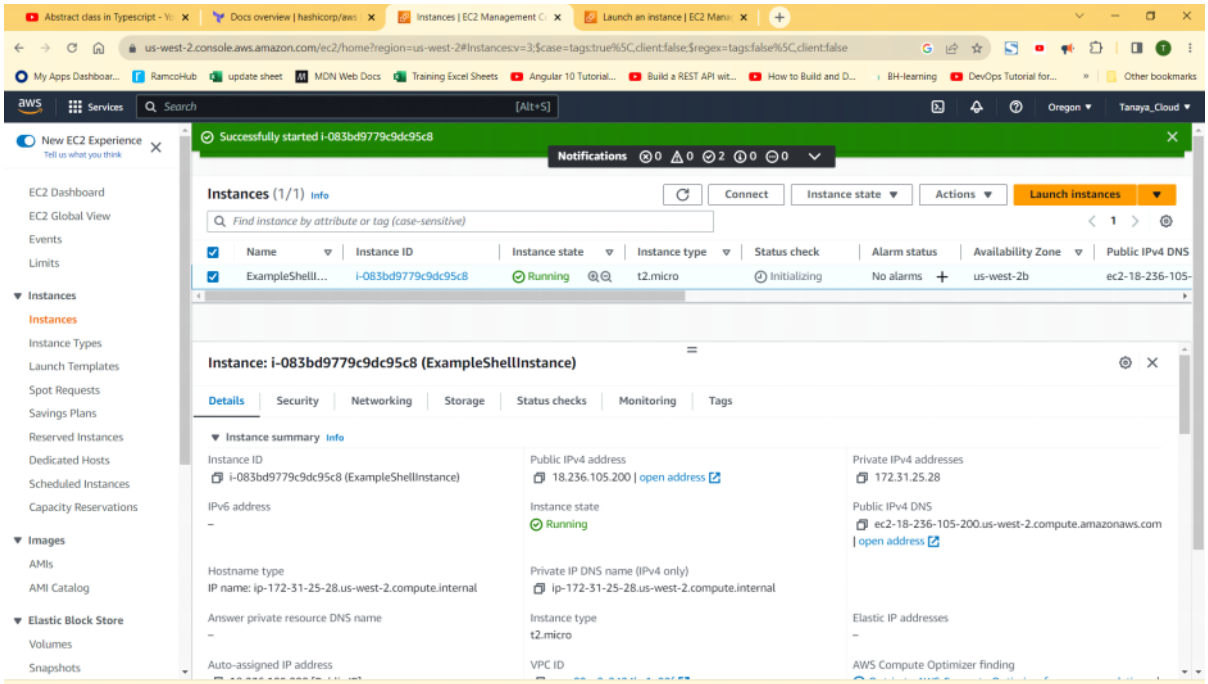
Print Fibonacci Series for 10 numbers

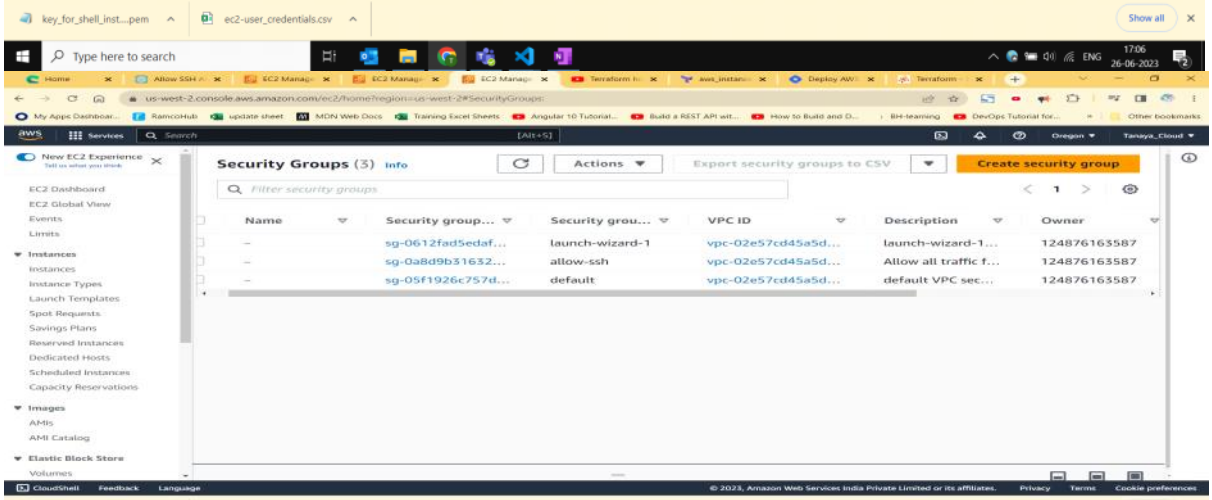
```
#!/bin/bash
#take a input from  a user
echo "Enter the number: "
read number
i=2
echo "first and second number of series:"
read f s

my_array=(
   [0]=0
   [1]=1)

while [ $i -le $number ]
do
   n=$(($f + $s))
   my_array[$i]="$n"
   f="$s"
   s="$n"
   ((i++))

done
echo "the series : "
for element in "${my_array[@]}"; do
  echo "$element"
done
```

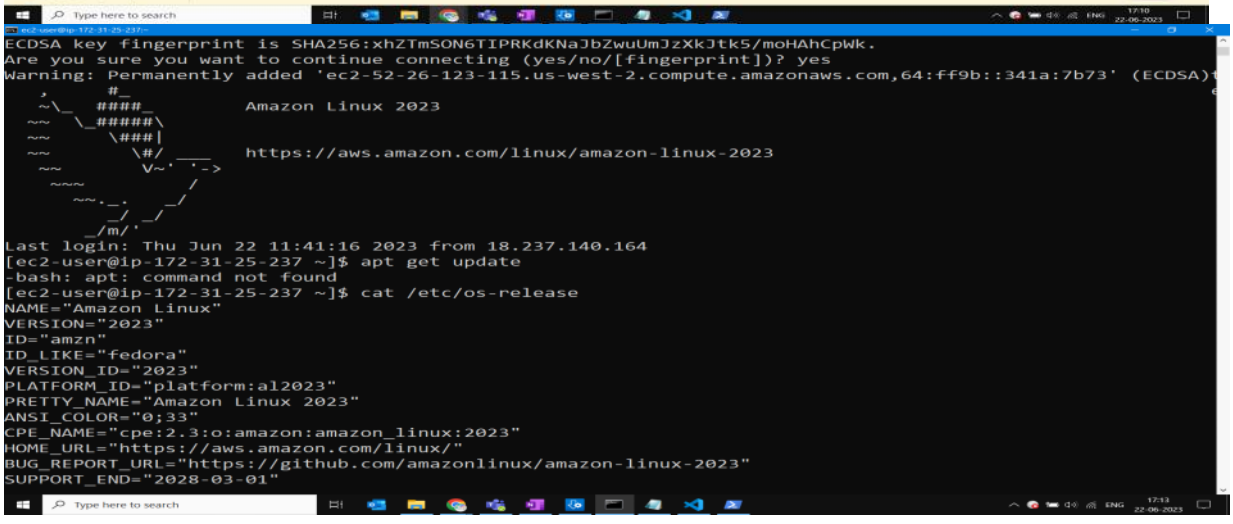
Reference:
https://www.makeuseof.com/run-ubuntu-as-docker-container/

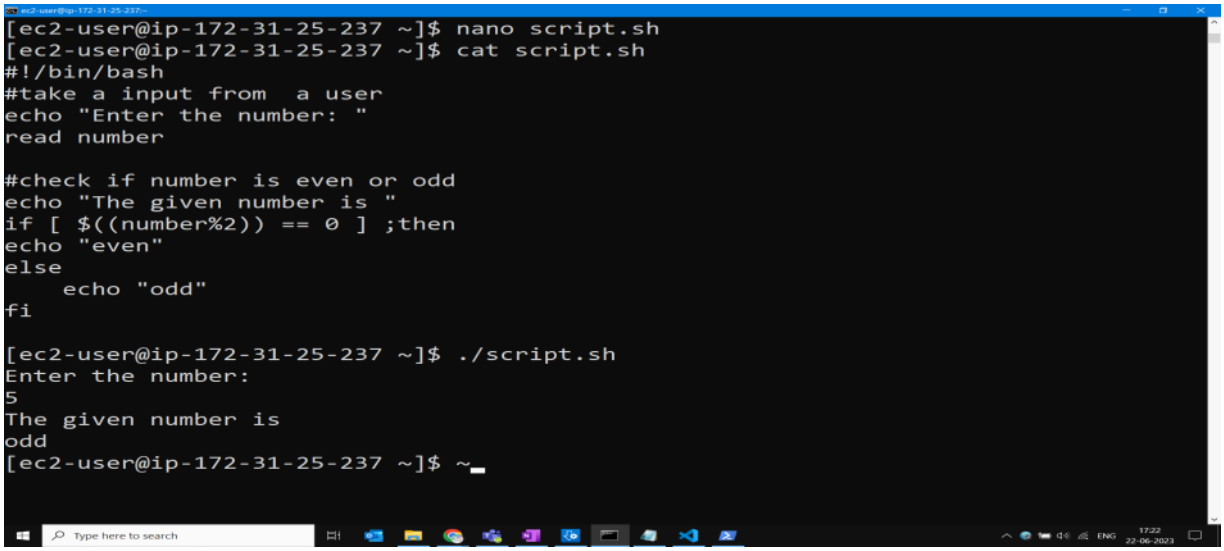

[ec2-user@ip-172-31-25-237 ~]$ nano script.sh
[ec2-user@ip-172-31-25-237 ~]$ nano script.sh
[ec2-user@ip-172-31-25-237 ~]$ ./script.sh
Enter the number:
10
first and second number of series:
0 1
the series :
0
1
1
2
3
5
8
13
21
34
55
[ec2-user@ip-172-31-25-237 ~]$


Successfully started i-083bd9779c9dc95c8
Instances (1/1)
Launch instances
Instance: i-083bd9779c9dc95c8 (ExampleShellInstance)


Successfully started i-083bd9779c9dc95c8
Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
ExampleShell...
i-083bd9779c9dc95c8
Running
t2.micro
Initializing
No alarms
us-west-2b
Instance: i-083bd9779c9dc95c8 (ExampleShellInstance)
Details
Security
Networking
Storage
Status checks
Monitoring
Tags
Instance summary Info
Instance ID
i-083bd9779c9dc95c8 (ExampleShellInstance)
Public IPv4 address
18.236.105.200 | open address
Private IPv4 addresses
172.31.25.28
IPv6 address
Instance state
Running
Public IPv4 DNS
ec2-18-236-105-200.us-west-2.compute.amazonaws.com | open address
Hostname type
IP name: ip-172-31-25-28.us-west-2.compute.internal
Private IP DNS name (IPv4 only)
ip-172-31-25-28.us-west-2.compute.internal
Answer private resource DNS name
Instance type
t2.micro
Elastic IP addresses
Auto-assigned IP address
VPC ID
AWS Compute Optimizer finding


Security Groups (3) Info
Actions
Export security groups to CSV
Create security group
Filter security groups
launch-wizard-1
allow-ssh
default
Allow all traffic f...
default VPC sec...
124876163587


ECDSA key fingerprint is SHA256:xhZTmSON6TIPRKdKNaJbZwuUmJzXkJtk5/moHAhCpWk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-26-123-115.us-west-2.compute.amazonaws.com,64:ff9b::341a:7b73' (ECDSA)
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Thu Jun 22 11:41:16 2023 from 18.237.140.164
[ec2-user@ip-172-31-25-237 ~]$ apt get update
-bash: apt: command not found
[ec2-user@ip-172-31-25-237 ~]$ cat /etc/os-release
NAME="Amazon Linux"
VERSION="2023"
ID="amzn"
ID_LIKE="fedora"
VERSION_ID="2023"
PLATFORM_ID="platform:al2023"
PRETTY_NAME="Amazon Linux 2023"
ANSI_COLOR="0;33"
CPE_NAME="cpe:2.3:o:amazon:amazon_linux:2023"
HOME_URL="https://aws.amazon.com/linux/"
BUG_REPORT_URL="https://github.com/amazonlinux/amazon-linux-2023"
SUPPORT_END="2028-03-01"



FOR VIM:

Esc+ __then_required_key:

To insert: esc+ I

To save: esc + wq!

To exit : esc + q!

Needs to change the permission

Chmod

4:read

2: write

1: execute

First digit: user

Second digit: groups

Third digit: others

Task 15- terraform backend

29 June 2023 18:24

Assingment :

1. **Modify instance created using TF manually and check what happens when you run terraform apply again an share the observations**

===>

1. On modifying the type of instance using aws console and then again run terraform apply command , it changes back the type of instance of ec2 to the original as mentioned in the terraform file.
2. Terraform uses the tfstate file or backend to store the state of the file/configuration/ infrastructure. Backend can be stored on local or on cloud.

Terminal logs:

```
aws_instance.shell_instance: Creating...
aws_instance.shell_instance: Still creating... [10s elapsed]
aws_instance.shell_instance: Still creating... [20s elapsed]
aws_instance.shell_instance: Creation complete after 25s [id=i-083bd9779c9dc95c8]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
After applying changes:
aws_instance.shell_instance: Modifying... [id=i-083bd9779c9dc95c8]
aws_instance.shell_instance: Still modifying... [id=i-083bd9779c9dc95c8, 10s elapsed]
aws_instance.shell_instance: Still modifying... [id=i-083bd9779c9dc95c8, 20s elapsed]
aws_instance.shell_instance: Still modifying... [id=i-083bd9779c9dc95c8, 30s elapsed]
aws_instance.shell_instance: Still modifying... [id=i-083bd9779c9dc95c8, 41s elapsed]
aws_instance.shell_instance: Still modifying... [id=i-083bd9779c9dc95c8, 51s elapsed]
aws_instance.shell_instance: Modifications complete after 56s [id=i-083bd9779c9dc95c8]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

2. **Read about terraform backend as s3 bucket and implement the same using ec2 terraform.**

====>

Implementing the S3 backend of keypair created using terraform.

Steps:

1. Create a bucket on aws S3, enable the public access to the object
   To create using terraform:

```
provider "aws" {
  region = "us-west-2"
}
resource "aws_s3_bucket" "example" {
  bucket = "my-trial-bucket-by-tanaya"
  tags = {
    Name        = "My bucket"
    Environment = "DevOps"
  }
}
resource "aws_s3_bucket_public_access_block" "example" {
  bucket = aws_s3_bucket.example.id
  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}
```

2. Add a terraform backend block in the file for which u want to create a S3 backend file.

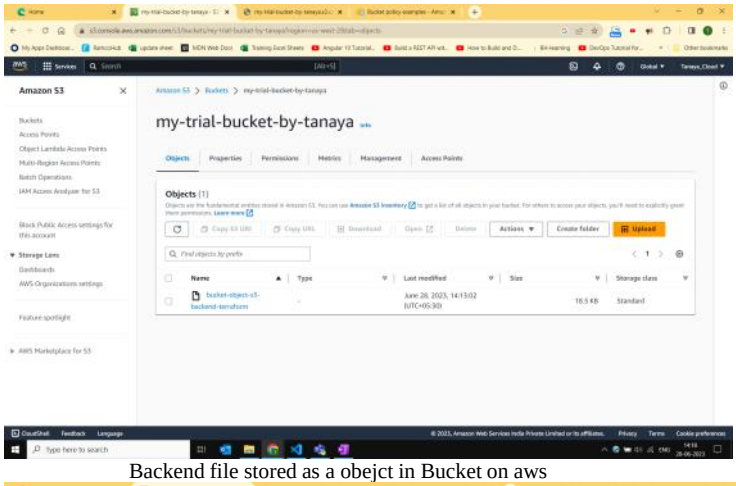
```
terraform {
  backend "s3" {
    bucket = "my-trial-bucket-by-tanaya"
    key    = "bucket-object-s3-backend-terraform"
    region = "us-west-2"

  }
}
```

always use a unique name for a bucket .

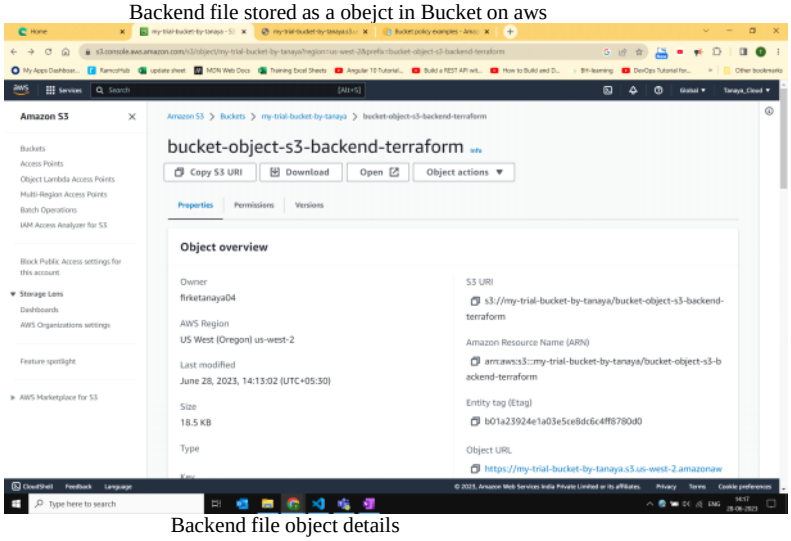
3. Run terraform init after changing the backend configuration.

Always check the region are same for all the depended resources.

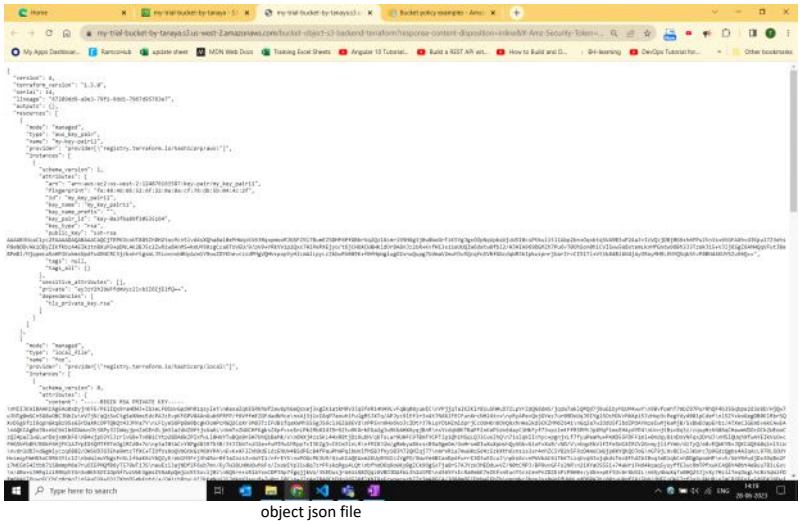
Screenshot:



Backend file stored as a obejct in Bucket on aws



Backend file object details



object json file

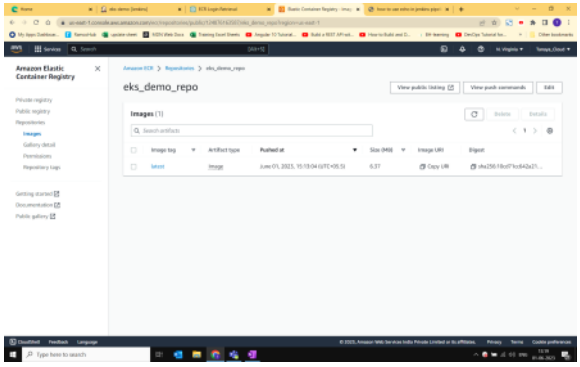
- Some points to be pointed:
  Backend stores the state of the file.
  Terraform stores local by default
  In .tfstate
  It can be stored on aws S3 Bucket.

Task-10 and 11

25 May 2023 14:20

1. Create an EKS Cluster with 2 Nodes
2. Create an EC2 and connect the cluster from that EC2 using kubectl command.
3. Outcome of this should be, EKS Cluster with 2 nodes should be created in READY state.

4. Deploy the To-Do app earlier deployed on MiniKube to EKS Cluster created in Part-10
5. Create a LoadBalancer to access the To-Do app deployed in EKS Cluster
6. Deployment of to-do app can be done via Jenkins Job or Direct kubectl command (Preferred Jenkins)
7. Outcome of this should be, To-Do app deployed on EKS Cluster should be accessible from LoadBalancer URL



Task can be divided into following parts:

- ❖ Creating a cluster
- ❖ Creating an EC2 Instance
- ❖ Creating an ECR repo
- ❖ Writing a Jenkins pipeline to push the image of application to ECR and deploying application on eks cluster

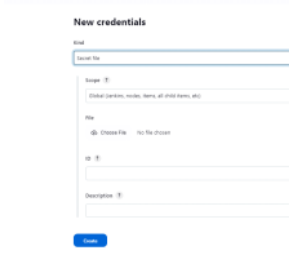
➢ Creating an ECR repo
   a. Go to aws console, search ECR
   b. Select public registry put the name and create the repo.
   c. Click on view push command to see push image to repo.

➢ Jenkins Pipeline to push the ECR and deploying the application:

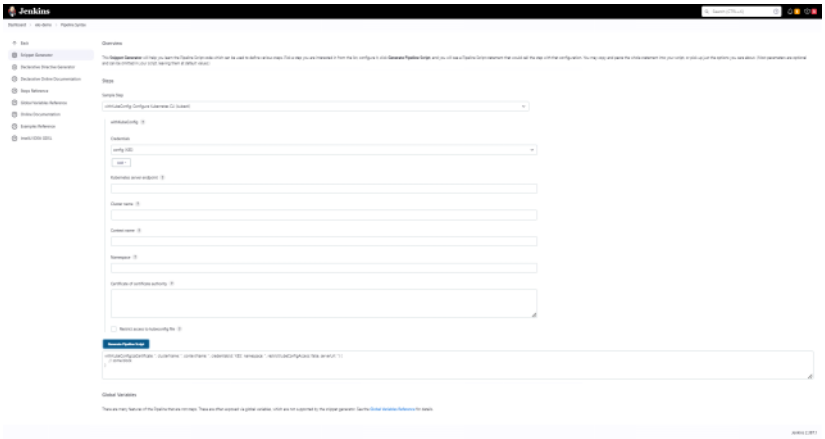
- Credentials to access AWS ECR
   a. Install AWS pipeline, AWS credentials and required plugins
   b. Go to manage jenkins-> configure systems-> global variables
   c. Add the environment variables
   d. Add AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY AND DEFAULT_REGION of your IAM user or root user(not recommeded)
      i. 

- Create credentials to connect EKS Cluster using kubectl command.
   a. Go to global credentials click on Add Credentials

b. secret file and upload config file enter ID and description and create:
(config file is stored in .kube folder when eks cluster is created)



- To generate script of deployment -->
Go to snippet generator, search for withKubeConfig , filled the parameter it will generate the snippet.



Pipeline script:

```
pipeline {
    agent any
    stages {
        stage('Build a docker image') {
            steps {
                script{
                def dockerfileP ath = "path\\to\\dockerfile"
                def imageName = "image_name"
                def imageTag = "latest"
                bat "docker build -t ${imageName}:${imageTag} ${dockerfilePath}"
                }
            }
        }
        stage('Push to ECR'){
            steps{
                script{
                withEnv (["AWS_ACCESS_KEY_ID=${env.AWS_ACCESS_KEY_ID}", "AWS_SECRET_ACCESS_KEY=${env.AWS_SECRET_ACCESS_KEY}",
"AWS_DEFAULT_REGION=${env.AWS_DEFAULT_REGION}"])
                {
                        def ecrLoginCmd = bat(script: 'aws ecr-public get-login-password --region us-east-1', returnStdout: true).trim()
                        def password = ecrLoginCmd.tokenize()[5]
                        bat "docker login -u AWS -p ${password} public.ecr.aws/k0a9k3j3"
                        def imageName = "demo_image"
                        def imageTag = "latest"
                        bat "docker tag ${imageName}:${imageTag} public.ecr.aws/k0a9k3j3/eks_demo_repo:latest"
                        bat "docker push public.ecr.aws/k0a9k3j3/eks_demo_repo:latest
                }
             }
            }
      }
        stage('Deploying the application'){
              Steps{
                    script {
                                withKubeConfig([credentialsId: 'K8S', serverUrl: '']) {
                                bat ('kubectl apply -f  eks-deploy-k8s.yaml')
                                }
                            }
              }
        }

    }
}
```

Task-13 AWS CLOUDFORMATION

07 June 2023 17:02

Steps:

❖ For creating eks cluster with nodegroup

- Resources needed eks cluster and worker nodegroup
- For eks cluster securitygroup and subnets are required
- For subnets create a vpc
- Create resources in reverse order managing the dependency of resources on other resource

1. Vpc
2. Subnets
3. Security group
4. Cluster
5. Workernodes : autoscaling group

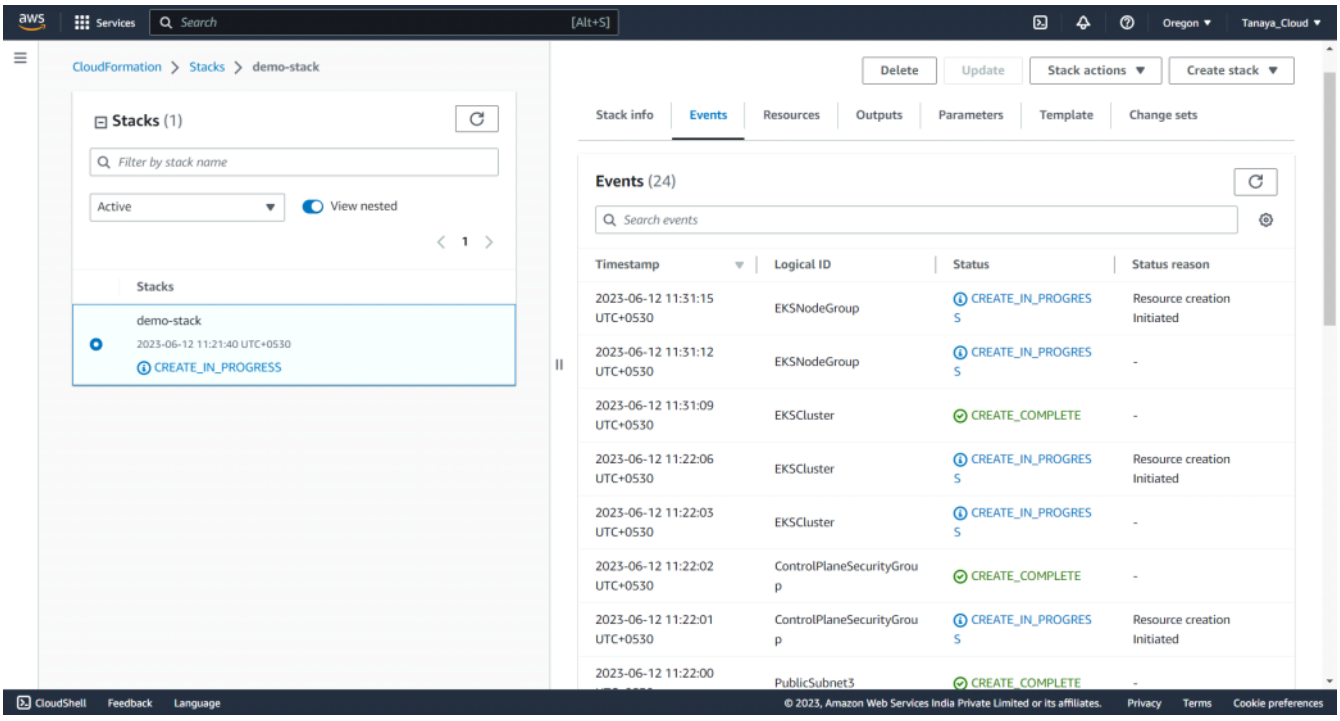
Upload the template file on aws cloudformation.

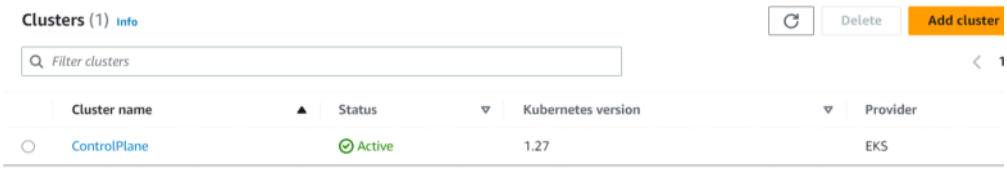
https://cloudkatha.com/how-to-create-security-group-in-aws-using-cloudformation/

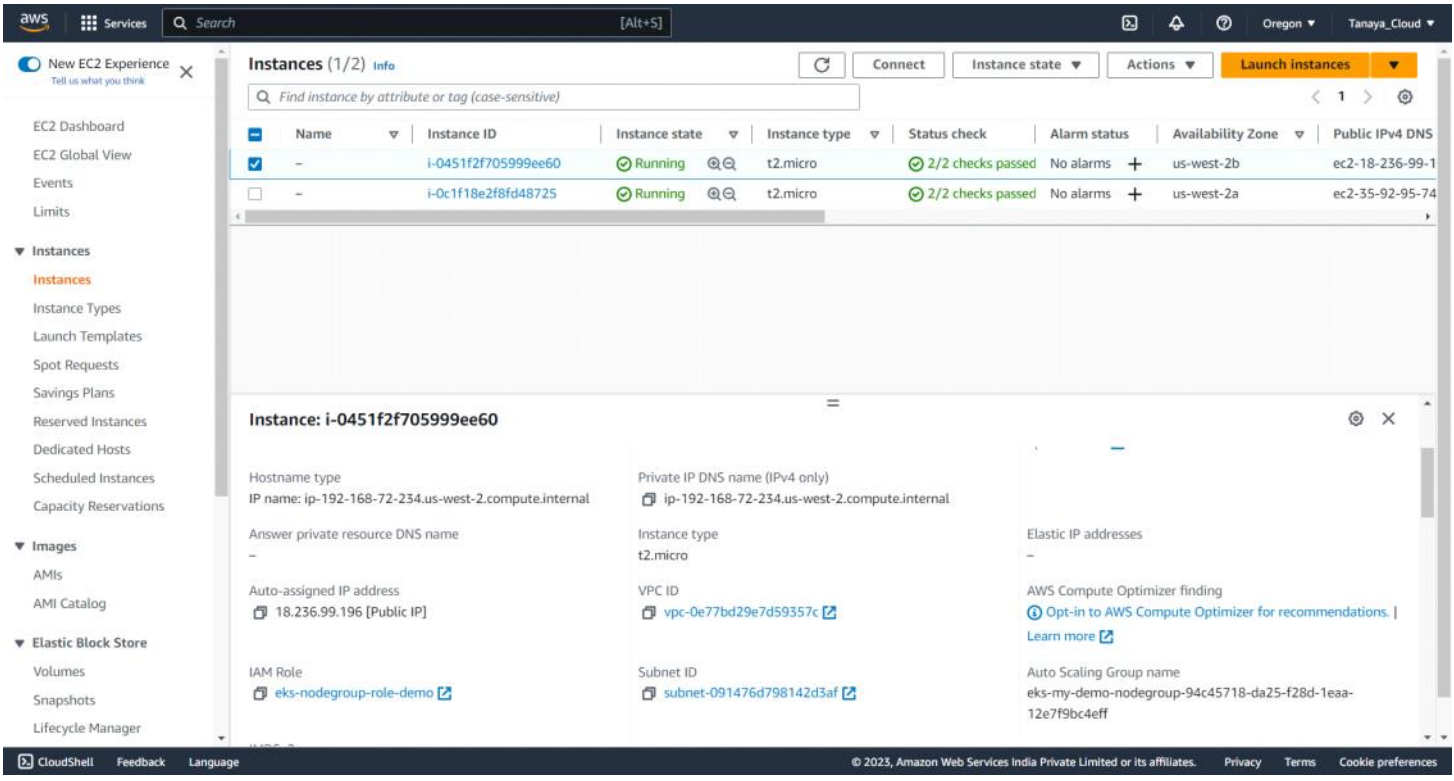
```
AWSTemplateFormatVersion: '2010-09-09'
Description: EKS cluster with nodegroups
Parameters:
  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 192.168.0.0/16
  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public
subnet in the first Availability Zone
    Type: String
    Default: 192.168.0.0/18
  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public
subnet in the second Availability Zone
    Type: String
    Default: 192.168.64.0/18
  PublicSubnet3CIDR:
    Description: Please enter the IP range (CIDR notation) for the public
subnet in the first Availability Zone
    Type: String
    Default: 192.168.128.0/18
  PublicSubnet4CIDR:
    Description: Please enter the IP range (CIDR notation) for the public
subnet in the second Availability Zone
    Type: String
    Default: 192.168.192.0/18
Resources:
#============================================================================#
  # VPC
      # CidrBlock: The IPv4 network range for the VPC, in CIDR notation.
      # EnableDnsHostnames: Indicates whether the instances launched in the
VPC get DNS hostnames.
      # EnableDnsSupport: Indicates whether the DNS resolution is supported
for the VPC.
#============================================================================#
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
#============================================================================#
  # Subnets
    #  CidrBlock: The IPv4 CIDR block assigned to the subnet.
    # VpcId:The ID of the VPC the subnet is in.
    #  MapPublicIpOnLaunch: Indicates whether instances launched in this
```

```
subnet receive a public IPv4 address. The default value is false.
#============================================================================#
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
        - '0'
        - Fn::GetAZs:
            Ref: AWS::Region
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
        - '1'
        - Fn::GetAZs:
            Ref: AWS::Region
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true
  PublicSubnet3:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
        - '0'
        - Fn::GetAZs:
            Ref: AWS::Region
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnet3CIDR
      MapPublicIpOnLaunch: true
  PublicSubnet4:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
        Fn::Select:
        - '1'
        - Fn::GetAZs:
            Ref: AWS::Region
      VpcId: !Ref VPC
      CidrBlock: !Ref PublicSubnet4CIDR
      MapPublicIpOnLaunch: true
#============================================================================#
  # Security Group
#============================================================================#
  ControlPlaneSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: A security group for a control plane of a cluster to
enable communication between master node and worke nodes.
      GroupName: ControlPlaneSG
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
      SecurityGroupIngress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
      VpcId: !Ref VPC

#============================================================================#
  # Control plane i.e. CLuster
#============================================================================#
  EKSCluster:
    Type: AWS::EKS::Cluster
    Properties:
      Name: "ControlPlane"
      Version: "1.27"
      RoleArn: "arn:aws:iam::124876163587:role/eks-cluster-role-demo"
      ResourcesVpcConfig:
        SecurityGroupIds:
          - !GetAtt ControlPlaneSecurityGroup.GroupId
        SubnetIds:
          - !Ref PublicSubnet1
          - !Ref PublicSubnet2
          - !Ref PublicSubnet3
          - !Ref PublicSubnet4
        EndpointPublicAccess: true
        EndpointPrivateAccess: false
        PublicAccessCidrs: [ "0.0.0.0/0" ]

#============================================================================#
  # NodeGroup
#============================================================================#
  EKSNodeGroup:
```

```
Type: AWS::EKS::Nodegroup
DependsOn: EKSCluster
Properties:
  AmiType: AL2_x86_64
  CapacityType: ON_DEMAND
  ClusterName: ControlPlane
  DiskSize: 5
  InstanceTypes:
    - "t2.micro"
  NodegroupName: "my-demo-nodegroup"
  NodeRole: "arn:aws:iam::124876163587:role/eks-nodegroup-role-demo"
  ScalingConfig:
    DesiredSize: 2
    MaxSize: 2
    MinSize: 2
  Subnets:
    - !GetAtt PublicSubnet1.SubnetId
    - !GetAtt PublicSubnet2.SubnetId
    - !GetAtt PublicSubnet3.SubnetId
    - !GetAtt PublicSubnet4.SubnetId
```







Some points to learn:

AWS IAM: to provide the specific permission to the particular user

AWS EBS : storage to store the data

Uses SAN to store the data

Snapshot : to back-up the data

Cloudformation uses templates to generate the stack
Can use designer to generate template

AWS services call the API of resources to generate that resource specified in the template.
Resources are declared under with the logical names and type
Resources:
Some resources requires properties to be specified like EC2 instances while EC2 S3 can be created with default properties.
{
Specifying a value for such properties is entirely optional and based on your needs. In the example above, because the AWS::S3::Bucket resource has only optional properties and we didn't need any of the optional features, we could accept the defaults and omit the Properties attribute.
}

You use literal strings to refer to existing AWS resources.

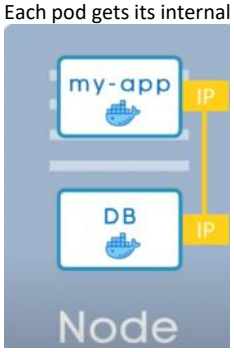
subnets to allow communication between your nodes and the Kubernetes control plane.

You can't use the same role that is used to create any clusters. Needs different roles for cluster and nodegroups.

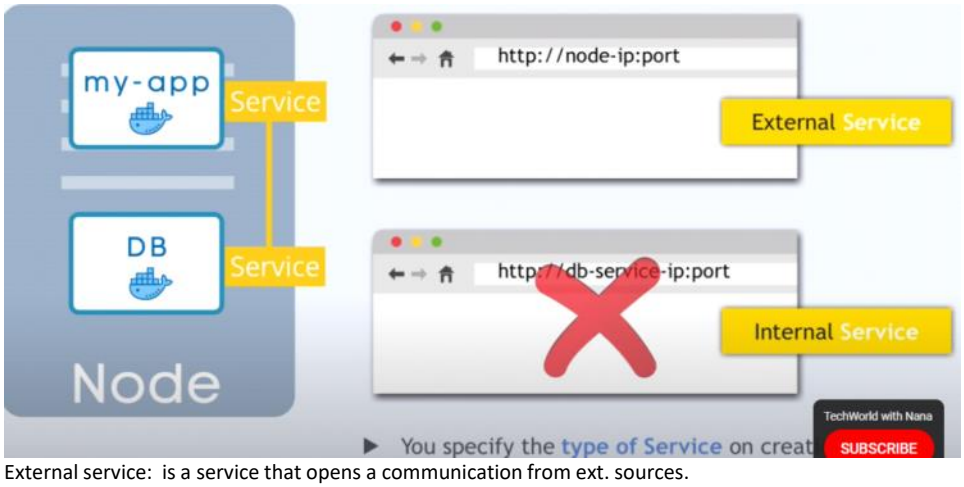
KUBERNETES

17 April 2023 10:44

Usually run 1 application per pod. Though we can run more container in single pod.
Other components are in another pod as a helper pod.
Each pod gets its internal own IP ADDRESS, using it communicate with each other.

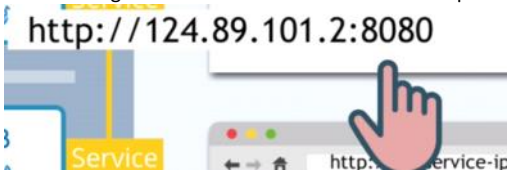


SERVICES are like endpoint of pods used to communicate pods to each other, since pod are get relocated, get new ID add.
SERVICES ARE THE PERSISTENT STATIC IP ADDRESS.

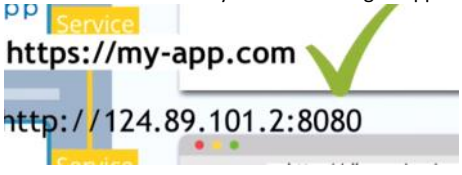


External service: is a service that opens a communication from ext. sources.
U don't want to open DB for public hence , creates internal service for database.

The service gives IP address like this : it container ip of node: port of container



But we want like this ,
More secure and directly communicating to application



Hence here comes INGRESS
First request to ingress then to service

Deployment is like a blue-print of the pod.
For pod labels are essential , but for deployment label is optional.

The nodeport service is available at cluster node Ip i.e. workers node IP, in case of minikube , it is only single node.

Configmap : Mongoab endpoint

Secret: mongodb user and pwd

Deployment : mongodb application with internal service

Deployement service: our webapp with external service

Label : uniquely identify the compnent.
Match label: how does deployment knows which pod belongs to it
Using matchlabel its matches the label of pods and deployment.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demoapp-deployment
  labels:
    app: demoapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demoapp
  template:
    metadata:
      labels:
        app: demoapp
    spec:
```

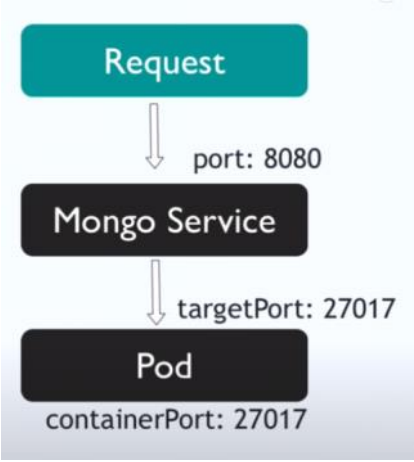
This can be different .

This both should be same , so that deployment can be applied to the pods of same app labels.

In services, selector app: <> works as a matchlabels there.

t

PORTS IN SERVICE:



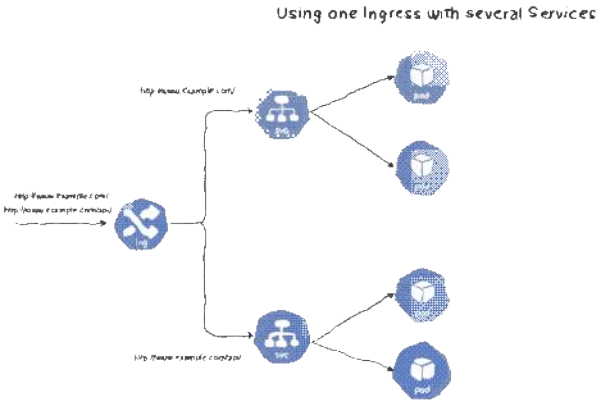
Port: port at which the services will listen , will receive the traffic
Targetport: where the service will send the traffic that is port of the pods
nodePort: the port at which we can access the service at a external client.

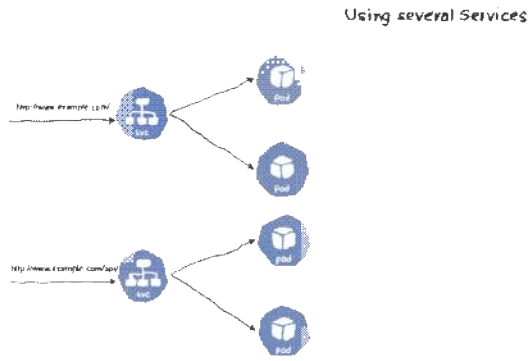
Env var: to expose the variable to that component.

What's The Difference Between A Service And Ingress?

Before we go ahead with creating the resources let's address this question. A Service is a Kubernetes object that receives HTTP requests and load balances them among the Pods under its control. There are several types of Services that you can use depending on your specific case. So, in our scenario, you may be thinking of using a LoadBalancer type of Service.

You're correct, a LoadBalancer service will create a load balancer that is publicly exposed and has an external IP address. Any traffic arriving at this IP will be routed to one of the backend pods. However, the load balancer is tied to only one service. If we need more than one service for our application (which is a typical use case), we'll need to create more load balancers and more IP addresses, which is impractical and cannot scale. The Ingress controller, on the other hand, allows you to tie it to more than one service and choose which requests go to which service depending on the condition you specify (in our case, it'll be the URL path). You can refer to the following diagram for a better understanding of how Ingress differs from a Service:





Services, Ingress, Ingress Controller

14 June 2023 12:05

One cluster can have multiple hosts or nodes.
One node can have multiple pods
One pod can have multiple container

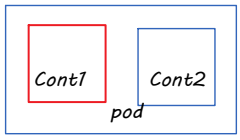
In one cluster pods on different nodes can also communicate with each other without any explicit network establishment.

{ When a pod needs to communicate with another pod, it can use the IP address of the target pod, regardless of the host it is running on. Kubernetes networking takes care of routing the traffic between the appropriate hosts and delivering it to the target pod. }

Communication:

https://www.tutorialworks.com/kubernetes-pod-communication/#:~:text=They%20can%20communicate%20with%20each,use%20the%20address%20localhost%3A8080%20.

https://platform9.com/blog/ultimate-guide-to-kubernetes-ingress-controllers/



Each pod has unique ip address
And containers in the pod share the same namespace and ip address.
The containers in the same pod communicate with each other using localhost:port, localhost ip is same but the port on which the container is running is different. {
this means that two containers can't share the same port, if they're in the same Pod.

From <https://www.tutorialworks.com/kubernetes-pod-communication/>

}

Ingress isn't a service type like NodePort, ClusterIP, or LoadBalancer. Ingress actually acts as a proxy to bring traffic into the cluster, then uses internal service routing to get the traffic where it is going. Under the hood, Ingress will use a NodePort or LoadBalancer service to expose itself to the world so it can act as that proxy.

From <https://platform9.com/blog/understanding-kubernetes-loadbalancer-vs-nodeport-vs-ingress/>

Within the metadata we are selecting which ingress class we want to use, in our case ingress-nginx, which will let the Nginx Ingress controller know that it needs to monitor and make updates. The ingress.class option is also used to specify which Ingress Controller should monitor the resource if we are using multiple controllers.

From <https://platform9.com/blog/ultimate-guide-to-kubernetes-ingress-controllers/>

If single controller ---> classname
If multiple controller ----> ingress class resource

Application is built now infrastructure is required to deploy and run the application.
Infrastructure requires servers, network set up , routes, database, memory, computing system.
These all needs to be set up initially and then needs maintenance after deployment.
Infrastructure as a Code is tool to create / provision the required infrastructure using the script , using template replicate the environment rather than going through console setting each time.
Terraform is a IaC tool that uses declarative approach that is user gives the end-state of infrastructure required while in imperative approach user needs to give commands to acquire the desired infrastructure.

Load Balancer is kind of a traffic manager that routes the traffic to the targeted through the listener which lists the rules for routing of the traffic.
Application load traffic routes the application traffic that is the any activity /request/response on application on web-browser by the user/client.
It operates at the 7-layer i.e. application layer and uses higher criteria through routing such as https/ header, urls, paths.
Network LB routes the traffic related to network it operates at the 4-Layer i.e. transport layer it routes depends on info of ip address and ports.
GatewayLB manages traffic between the networks.
Classic LB can handle both application and network level but is

Terraform is a declarative approach while is user gives the end-state of infrastructure required while in imperative approach user needs to give commands to acquire the desired infrastructure.

of ip address and ports.
GatewayLB manages traffic between the networks.
Classic LB can handle both application and network level but is not advanced.

Some Basics

20 June 2023 18:04

Virtual machines only gives hardware.

In virtualization it creates a new virtual machine just like actual computer with its own Operating system hence takes more time to create.

In docker each virtual machine has divided OS
Docker divides hardware and OS, it kind of virtualize the operating system and each container gets each unique IP. All container are running under same OS just gives feel of different dedicated OS to each container.

Dev	Ops
Clone	Maintaining the application
Compile	Keeping the application running
Integrate with QA system	Setting infrastructure required to run the application
deploy	Upscaling/downscaling infrastructure

Devops work is automate this work i.e. CI/CD