Assignment 1 –

1) Prerequisite – We would require a Kubernetes cluster (in this case we are using minikube), Docker engine, and python installed on the VM.

```
jalaj_malhotra@LAPTOP-P00CD620:~$ minikube start
😄  minikube v1.33.1 on Ubuntu 22.04
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.44 ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
🔎  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
jalaj_malhotra@LAPTOP-P00CD620:~$
```

2) Python Script - Then we would need to write a python script which would run to clean the pod that are in not running. In this case we are using Kubernetes module to interact with the kubenetes cluster. Below is the script let see it in a bit detail.

a)

```
import time

import sys

from kubernetes import client, config

from datetime import datetime, timezone, timedelta
```

Here we are importing all the required modules. Kubernetes module would be used to interact with the cluster.

b)

```
def delete_old_failed_pods(namespace):

    # Load the kube config from within the cluster

    config.load_incluster_config()


    # Initialize the Kubernetes API client

    v1 = client.CoreV1Api()


    # Get all pods in the specified namespace

    pods = v1.list_namespaced_pod(namespace)


    # Define the time threshold (pods older than 5 minutes)

    threshold_time = datetime.now(timezone.utc) - timedelta(minutes=5)
```

We have defined a function called ‘delete_old_failed_pods’ . In this function we are taking as the namespace which needs to be cleaned as an argument. In the function first we are loading the kubeconfig. Then we are initializing the Kubernetes API. Then using we retrieve all the pod running in the namespace. Also we are calculating time 5 mins before and saving it in a variable threshold_time.

c)

```
for pod in pods.items:
        pod_creation_time = pod.metadata.creation_timestamp


        # Check if the pod is older than 5 minutes
        if pod_creation_time < threshold_time:
            containers = pod.status.container_statuses
            if containers:
                for container in containers:
                    if container.state.terminated or container.state.waiting:
                        print(f"Deleting pod {pod.metadata.name} in namespace {namespace}
(Status: {pod.status.phase})")
                        v1.delete_namespaced_pod(pod.metadata.name, namespace)
                        sys.stdout.flush()
                        break
        else:
            print(f"Pod {pod.metadata.name} in namespace {namespace} is not older than
5 minutes, skipping it.")
            sys.stdout.flush()
```

Now we are iterating on the all the pods in the namespace and checking if these were created before the threshold time. If their creation time is after the threshold time we skip the pod, if not then we retrieve the status of the container running in the pod. Once we have the list we iterate over the containers in the pod and if any of the container is in terminated state (like they have run successfully or terminated due to an error) or they are in the waiting state (like due to crashloopback or imagepull error) we terminate that pod and come out of the loop.

d)

```
# Define the namespaces
second_namespace = "second-namespace"  # The namespace where the cleanup is
performed


# Run the cleanup process only once
delete_old_failed_pods(second_namespace)
```

We are saving the name of the target namespace where the script needs to cleans the pod in the variable ‘second_namespace’ and passing it to the function ‘delete_old_failed_pods’.

3) Docker - Now we would need to create a docker image which would be used in the creation of pod and we would copy this script in the docker image. Below is the Dockerfile

Code Blame 13 lines (9 loc) · 274 Bytes Code 55% faster with GitHub Copilot Raw

```
FROM python:3.9-slim

# Install Kubernetes client library
RUN pip install kubernetes

# Copy the script into the Docker image
COPY clean_final.py  /app/cleanup.py

# Set the working directory
WORKDIR /app

# Run the cleanup script
CMD ["python3", "cleanup.py"]
```

We are using python3.9-slim as a base image and installing the Kubernetes module using Pip. Then we are copying the python script to the /app folder. At the end using the CMD command we are running that script.

Now we will build the image using ‘docker image build -t pod-cleanup:latest . ‘.

```
jalaj_malhotra@LAPTOP-P00CD620:~$  docker build -t pod-cleanup:latest .
[+] Building 1.2s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                   0.0s
 => => transferring dockerfile: 301B                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                     1.0s
 => [internal] load .dockerignore                                                                      0.0s
 => => transferring context: 2B                                                                        0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:85c7a2a383a01e0b77b5f9c97d8b1eef70409a99552fde03c518a98dfa196  0.0s
 => [internal] load build context                                                                      0.0s
 => => transferring context: 36B                                                                       0.0s
 => CACHED [2/4] RUN pip install kubernetes                                                            0.0s
 => CACHED [3/4] COPY clean_final.py  /app/cleanup.py                                                  0.0s
 => CACHED [4/4] WORKDIR /app                                                                          0.0s
 => exporting to image                                                                                 0.0s
 => => exporting layers                                                                                0.0s
 => => writing image sha256:774400bbfd7e9da905d3b1d9f9c50004892f5991335ed89b4c8bf5666fd05a44           0.0s
 => => naming to docker.io/library/pod-cleanup:latest                                                  0.0s
```

4) Creation of Namespace – Now we would create a namespace using ‘ kubectl create namespace second-namespace ’ . We would use this namespace to run the pod and our script would run in the default namespace.

5) Creation of service account - Now we will create a service account which would used by the pod as its identity. We will attach role to this service account which would help in list and deletion of pods in the second namespace where we want to clear pod.

```
Code  Blame  5 lines (5 loc) · 95 Bytes  Code 55% faster with GitHub Copilot  Raw

apiVersion: v1
kind: ServiceAccount
metadata:
  name: pod-cleanup-sa
  namespace: default
```

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl create -f sa.yml
serviceaccount/pod-cleanup-sa created
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
```

6) Creation of role and rolebinding to service-account –

a) We would create a role in the second-namespace which would have the rules to list and delete the pods.

```
# role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-cleanup-role
  namespace: second-namespace
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list", "delete"]
```

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl apply -f role.yml
role.rbac.authorization.k8s.io/pod-cleanup-role created
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
```

b) Now we will bind this role to the service account we created in step5.

```
Code  Blame  14 lines (14 loc) · 331 Bytes  Code 55% faster with GitHub Copilot  Raw

# rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-cleanup-rolebinding
  namespace:  second-namespace
subjects:
- kind: ServiceAccount
  name: pod-cleanup-sa
  namespace: default
roleRef:
  kind: Role
  name: pod-cleanup-role
  apiGroup: rbac.authorization.k8s.io
```

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl create -f rbinding.yml
rolebinding.rbac.authorization.k8s.io/pod-cleanup-rolebinding created
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
```

7) Creation of CronJob - We will create a cron job so that it will create pods to run the python script using the docker image at the give time interval. We have given the schedule to run it in every 5 minutes. Also we will attach the pod-cleanup-sa service account to the pod so that has the access to the second-namespace.

Code Blame 19 lines (18 loc) · 490 Bytes Code 55% faster with GitHub Copilot Raw

```yaml
#cleanup_cron.yml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: pod-cleanup-test-cronjob
  namespace: default
spec:
  schedule: "*/5 * * * *"  # This sets the schedule to run every 5 minutes
  jobTemplate:
    spec:
      template:
        spec:
          serviceAccountName: pod-cleanup-sa
          containers:
            - name: cleanup
              image: pod-cleanup:latest
              imagePullPolicy: Never
          restartPolicy: OnFailure

```

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl create -f cleanup_cron.yml
cronjob.batch/pod-cleanup-test-cronjob created
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$
```

8) Creation of failing pods – Now we will create some failing pods in our second-namespace so that the cron jobs runs and clean the pods.

```
jalaj_malhotra@LAPTOP-P00CD620:~$
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl apply -f mini.yml  -n second-namespace
pod/nginx-pod created
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl apply -f pod.yml  -n second-namespace
pod/fail-pod created
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl apply -f x.yml  -n second-namespace
pod/testpod created
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl get pods -n second-namespace
NAME        READY   STATUS             RESTARTS      AGE
fail-pod    0/1     CrashLoopBackOff   1 (12s ago)   19s
nginx-pod   0/1     ImagePullBackOff   0             25s
testpod     1/1     Running            0             12s
jalaj_malhotra@LAPTOP-P00CD620:~$
```

9) Now we will wait for the run of the cronjob so that it clears all the pods in not running state in the second-namespace. When the cronjob would create the pod it and run the script which would delete the pods which were not older than 5 mins.

a) First run – In the first run given they were not older than 5 minutes, it has skipped both the failing pods.

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
pod-cleanup-test-cronjob-28722915-lg69g   0/1     Completed   0          12s
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl logs pod-cleanup-test-cronjob-28722915-lg69g
Pod fail-pod in namespace second-namespace is not older than 5 minutes, skipping it.
Pod nginx-pod in namespace second-namespace is not older than 5 minutes, skipping it.
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl get pods -n second-namespace
NAME        READY   STATUS             RESTARTS      AGE
fail-pod    0/1     CrashLoopBackOff   3 (35s ago)   89s
nginx-pod   0/1     ImagePullBackOff   0             96s
testpod     1/1     Running            0             26m
jalaj_malhotra@LAPTOP-P00CD620:~$
```

b) Second run – In the second run all the failing pods were older than 5 minutes, hence they are got cleared up.

```
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
pod-cleanup-test-cronjob-28722915-lg69g   0/1     Completed   0          5m34s
pod-cleanup-test-cronjob-28722920-8lrwd   0/1     Completed   0          34s
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl logs pod-cleanup-test-cronjob-28722920-8lrwd
Deleting pod fail-pod in namespace second-namespace (Status: Running)
Deleting pod nginx-pod in namespace second-namespace (Status: Pending)
jalaj_malhotra@LAPTOP-P00CD620:~$ kubectl get pods -n second-namespace
NAME      READY   STATUS    RESTARTS   AGE
testpod   1/1     Running   0          31m
jalaj_malhotra@LAPTOP-P00CD620:~$
```

Assignment 2 –

1) Prerequisite – We would require Terraform cli, aws iam secrets, aws account and aws cli to work on this assignment.
2) AWS configure - First, we will pass our credentials using aws cli. We will use aws configure to pass the secrets.

```
16    }

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    powershell + ∨ ◫ 🗑 ··· ^ ×

PS C:\Users\Jalaj Malhotra\Documents\time to grow\bebo> aws configure
AWS Access Key ID [****************MMOR]: AKIAVRUVUHDYB[redacted]
AWS Secret Access Key [****************dEAO]: WWiGxg/0TP39Hy5rMRC1RGOZl3Pe[redacted]
Default region name [ap-south-1]: us-east-1
Default output format [json]:
PS C:\Users\Jalaj Malhotra\Documents\time to grow\bebo> 
```

3) Module main.tf - Now we will write our main.tf inside a module folder, this is the main tf file in which we write our logic to create vpc, subnets, gateways. And this would be sourced later by the our terraform code as a module. The block in the main.tf are as follows –
   a) In the below block we are creating a vpc and passing the cidr using a variable. And then we are creating a internet gateway and we will pass the vpc id so the internet gateway is created in this vpc only.

```
resource "aws_vpc" "vpc_assignment" {
  cidr_block = var.vpc_cidr
}

resource "aws_internet_gateway" "internet_assignment" {
  vpc_id = aws_vpc.vpc_assignment.id

}
```

   b) In the below code we are creating the subnets the attributes in the block means as below –
      i) Count – It tells the count of resources if it is 2 then it means that only 2 resource would be created.
      ii) vpc_id – We are passing the vpc_id, so that subnets get created in this vpc only.
      iii) Cidr_block – In this we are passing the cidr range of the subnet that needs to be created. In this we are using the <u>cidrsubnet</u> function(cidrsubnet(prefix, newbits, netnum)) which would be used to calculate CIDR blocks within a cidr block. In this case we are passing the vpc cidr (for example we can take 10.0.0.0/16) and the new bit is 4. So it will add four additional bits so it give 16 subnets of size of '10.0.0.0/20'.
      The we are using netnum to tell which subnet to select from the set of possible subnets.
      iv) AZ – In this we are passing in which az the subnet should get created. In this we are using the element function so that the az get select on the basis of index of the count.
      v) Map_public_ip_on_launch – This is a attribute which is only passed in public subnet so that public ip gets allocated automatically if instance is launched in this subnet.

```
resource "aws_subnet" "public_subnets" {
  count                   = var.public_subnets
  vpc_id                  = aws_vpc.vpc_assignment.id
  cidr_block              = cidrsubnet(var.vpc_cidr, 4, count.index)
  availability_zone       = element(var.azs, count.index)
  map_public_ip_on_launch = true
  tags = {
    Name = "Public_subnet"
  }

}

resource "aws_subnet" "private_subnets" {
  count             = var.private_subnets
  vpc_id            = aws_vpc.vpc_assignment.id
  cidr_block        = cidrsubnet(var.vpc_cidr, 4, count.index + var.public_subnets)
  availability_zone = element(var.azs, count.index)
  tags = {
    Name = "Private Subnet"
  }
}
```

```

#creating subnets for database
resource "aws_subnet" "private_subnets_for_db" {
  count             = var.private_subnets
  vpc_id            = aws_vpc.vpc_assignment.id
  cidr_block        = cidrsubnet(var.vpc_cidr, 4, count.index + var.public_subnets + var.private_subnets)
  availability_zone = element(var.azs, count.index)
  tags = {
    Name = "Private Subnet_for_db"
  }
}
```

c) Now we are creating a public route table in which there is a route which routes the request to internet gateway. In the route table except of the route we also passes the vpc_id. And then we associate this route table to the public subnet.

```
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.vpc_assignment.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.internet_assignment.id
  }

}

resource "aws_route_table_association" "public_rt_assoc" {
  count          = var.public_subnets
  subnet_id      = element(aws_subnet.public_subnets[*].id, count.index)
  route_table_id = aws_route_table.public_rt.id

}
```

d) In the below code we are creating an elastic ip and creating a NAT gateway. We are the elastic ip in the NAT gateway. Similarly, as did earlier we are suing the element function to allocate public subnet to NAT gateway. Also, we are using element to make sure every public subnet has NAT gateway in it.

```
resource "aws_eip" "nat" {
  count  = var.public_subnets
  domain = "vpc"
}


resource "aws_nat_gateway" "nat_assignment" {
  count         = var.public_subnets
  allocation_id = element(aws_eip.nat[*].id, count.index)
  subnet_id     = element(aws_subnet.public_subnets[*].id, count.index)
}
```

e) In the below code we are creating route for the private subnet and creating route which will route any request going towards the internet to NAT gateway. And at the end we are attaching this route table to the subnets. We are using element function for the dynamic allocation of the NAT gateway in the route table. And also using it to so for the dynamic allocation of route table to private subnet. Similarly we will do for subnets_db but they won't have the NAT gateway in their route.

```
resource "aws_route_table" "private_rt" {
  count  = var.private_subnets
  vpc_id = aws_vpc.vpc_assignment.id
  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = element(aws_nat_gateway.nat_assignment[*].id, count.index)
  }
}

resource "aws_route_table_association" "private_rt_assoc" {
  count          = var.private_subnets
  subnet_id      = element(aws_subnet.private_subnets[*].id, count.index)
  route_table_id = element(aws_route_table.private_rt[*].id, count.index)
}
```

f) At last, we are creating a transit gateway but we are using a Boolean expression If the defined variable is true then only the transit gateway would be created else terraform won't create any transit gateway.

```
resource "aws_ec2_transit_gateway" "transit_assignment" {
  count       = var.enable_transit_gateway ? 1 : 0
  description = "My Transit Gateway"
  tags = {
    Name = "My transit gateway"
  }

}
```

4) Module Variables.tf – In the variable.tf we have defined the variables which are required to pass when we are calling the module. There are some variable whose default value have been set, so

incase those are not defined in the module the default one would be taken.

```
variable "vpc_cidr" {
  description = "CIDR block for the VPC"
  type        = string
}

variable "private_subnets" {
  description = "Number of private subnets to create"
  type        = number
  default     = 3
}

variable "public_subnets" {
  description = "Number of public subnets to create"
  type        = number
  default     = 3
}

variable "azs" {
  description = "List of availability zones"
  type        = list(string)
  default     = ["us-east-1a", "us-east-1b", "us-east-1c"]
}

variable "enable_transit_gateway" {
  description = "Enable integration with a Transit Gateway"
  type        = bool
  default     = false
}
```

5) Module Output.tf – There is an output.tf which can give various output like id etc on the creation of the resource.

Code Blame 15 lines (12 loc) · 282 Bytes Code 55% faster with GitHub Copilot Raw

```
output "vpc_id" {
  value = aws_vpc.vpc_assignment.id
}

output "public_subnet_ids" {
  value = aws_subnet.public_subnets[*].id
}

output "private_subnet_ids" {
  value = aws_subnet.private_subnets[*].id
}

output "nat_gateway_id" {
  value = aws_nat_gateway.nat_assignment[*].id
}
```

6) Main code:

a) Provider.tf – In this we are passing the basic terraform configuration and specifying the provider to use with the default region.

Code Blame 13 lines (10 loc) · 146 Bytes Code 55% faster with GitHub Copilot Raw

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
  }
}

provider "aws" {
  region = "us-east-1"


}
```

b) Main.tf – This is the main file in which we are calling our modeule and passing all the variable, in this we have given the following attributes –

i) Source – In this we are passing the location of module.
ii) Vpc_cidr – Passing the cidr range of the vpc
iii) Private Subnets – Number of private subnets needed.
iv) Public Subnets – Number of public subnets needed.
v) Azs – The list of AZ's in which we want to create our infra.

Bebo_assignment > assignment-2 > main.tf > ...

```
module "vpc" {
  source          = "./module"
  vpc_cidr        = var.vpc_cidr
  private_subnets = var.private_subnets
  public_subnets  = var.public_subnets
  azs             = var.azs
}
```

7) Terraform init – We will run terraform init to install all the dependencies and it stores in .terraform folders.

```
PS C:\Users\Jalaj Malhotra\Documents\time to grow\bebo\Bebo_assignment\assignment-2> terraform init
Initializing the backend...
Initializing modules...
- vpc in module
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v5.62.0...
- Installed hashicorp/aws v5.62.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

8) Terraform plan – Now we will run terraform plan to verify our resources which will be created . (Complete output attached in the git repo)

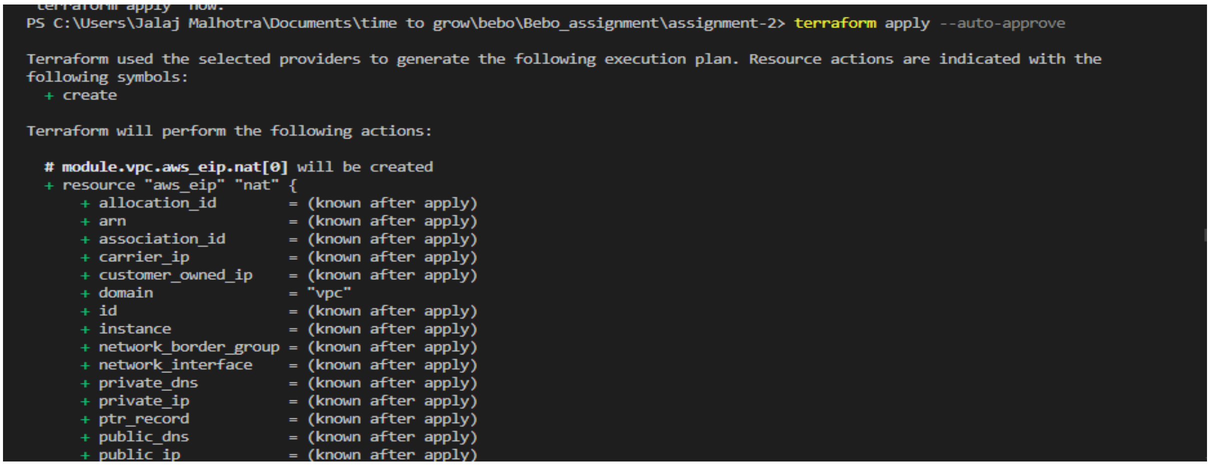
```
PS C:\Users\Jalaj Malhotra\Documents\time to grow\bebo\Bebo_assignment\assignment-2> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # module.vpc.aws_eip.nat[0] will be created
  + resource "aws_eip" "nat" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = "vpc"
      + id                   = (known after apply)
      + instance             = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
```

9) Terraform apply – Once we have verified our plan, we will run terraform apply –auto-approve to create the resources. (Complete output attached in the git repo)



```
PS C:\Users\Jalaj Malhotra\Documents\time to grow\bebo\Bebo_assignment\assignment-2> terraform apply --auto-approve

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # module.vpc.aws_eip.nat[0] will be created
  + resource "aws_eip" "nat" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = "vpc"
      + id                   = (known after apply)
      + instance             = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + ptr_record           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
```

10) AWS account – Now go to the aws console and you can verify your rersources that have been created.