**Launch EC2 instance using Terraform code**

provider "aws" {

region = "us-east-1"

access\_key = "" // AWS access key should be updated

secret\_key = " " // Secret key should be updated

}

resource "aws\_vpc" "my\_vpc" {

cidr\_block = "172.16.0.0/16"

tags = {

Name = "tf-example"

}

}

resource "aws\_subnet" "my\_subnet" {

vpc\_id = aws\_vpc.my\_vpc.id

cidr\_block = "172.16.10.0/24"

availability\_zone = "us-west-2a"

tags = {

Name = "tf-example"

}

}

resource "aws\_network\_interface" "foo" {

subnet\_id = aws\_subnet.my\_subnet.id

private\_ips = ["172.16.10.100"]

tags = {

Name = "primary\_network\_interface"

}

}

resource "aws\_instance" "foo" {

ami = var.ami # us-west-2

instance\_type = var.instance\_type

}

##variables

variable "ami" {

description = "this is the ami of the ec2 machine"

type = string

default = ""

}

variable "instance\_type" {

type = string

default = "t2.micro"

}

**Connect to the EC2 instance**

Make sure SSH port 22 opened inbound for the machine

Open Putty or SSH connector / Terminal and pass “ ssh EC2admin@Public IP of the machine the connect

**ECS Cluster Creation**

data "template\_file" "container\_instance\_cloud\_config" {

template = "${file("cloud-config/container-instance.yml.tpl")}"

vars {

environment = "${var.environment}"

}

}

module "container\_service\_cluster" {

source = "github.com/azavea/terraform-aws-ecs-cluster?ref=3.0.0"

vpc\_id = "vpc-20f74844"

ami\_id = "ami-b2df2ca4"

instance\_type = "t2.micro"

key\_name = "hector"

cloud\_config\_content = "${data.template\_file.container\_instance\_cloud\_config.rendered}"

root\_block\_device\_type = "gp2"

root\_block\_device\_size = "10"

health\_check\_grace\_period = "600"

desired\_capacity = "1"

min\_size = "0"

max\_size = "1"

enabled\_metrics = [

"GroupMinSize",

"GroupMaxSize",

"GroupDesiredCapacity",

"GroupInServiceInstances",

"GroupPendingInstances",

"GroupStandbyInstances",

"GroupTerminatingInstances",

"GroupTotalInstances",

]

subnet\_ids = [...]

project = "Something"

environment = "Staging"

}

resource "aws\_security\_group\_rule" "container\_instance\_http\_egress" {

type = "egress"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

security\_group\_id = "${module.container\_service\_cluster.container\_instance\_security\_group\_id}"

}

resource "aws\_security\_group\_rule" "container\_instance\_https\_egress" {

type = "egress"

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

security\_group\_id = "${module.container\_service\_cluster.container\_instance\_security\_group\_id}"

}

**Creation of a Network load balancer with Elastic IP:**

|  |
| --- |
|  |
| resource "aws\_eip" "nlb" { |
|  | tags = merge( |
|  | { |
|  | Name = "${local.name\_prefix}\_nlb\_eip" |
|  | }, var.tags) |
|  | } |
|  |  |
|  | resource "aws\_lb" "this" { |
|  | name = "${replace(local.name\_prefix, "\_", "-")}-nlb" |
|  | internal = false |
|  | load\_balancer\_type = "network" |
|  |  |
|  | subnet\_mapping { |
|  | subnet\_id = aws\_subnet.public.id |
|  | allocation\_id = aws\_eip.nlb.id |
|  | } |
|  |  |
|  | tags = merge( |
|  | { |
|  | Name = "${local.name\_prefix}\_nlb" |
|  | }, var.tags |
|  | ) |
|  | } |
|  |  |
|  | resource "aws\_lb\_target\_group" "http" { |
|  | name = "${replace(local.name\_prefix, "\_", "-")}-http" |
|  | vpc\_id = aws\_vpc.this.id |
|  | port = var.webserver\_port |
|  | protocol = "TCP" |
|  | target\_type = "instance" |
|  |  |
|  | health\_check { |
|  | enabled = true |
|  | interval = 30 |
|  | port = var.webserver\_port |
|  | } |
|  |  |
|  | tags = var.tags |
|  | } |
|  |  |
|  | resource "aws\_alb\_listener" "http" { |
|  | load\_balancer\_arn = aws\_lb.this.arn |
|  | port = var.webserver\_port |
|  | protocol = "TCP" |
|  |  |
|  | default\_action { |
|  | type = "forward" |
|  | target\_group\_arn = aws\_lb\_target\_group.http.arn |
|  | } |
|  | } |
|  |  |

**Installation of Docker**

sudo apt-get update

sudo apt-get install \

ca-certificates \

curl \

gnupg

Add Docker’s official GPG key:

sudo mkdir -m 0755 -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

Use the following command to set up the repository:

echo \

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Install Docker Engine

sudo chmod a+r /etc/apt/keyrings/docker.gpg

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

**Installation of Kubernetes**

# **Verify Cluster Setup**

In the Master node,

kubectl get nodes

kubectl config view

kubectl config current-context

kubectl config get-contexts kubernetes-admin@kubernetes

# **Deep-dive into Master setup**

kubectl cluster-info

kubectl cluster-info dump > cluster-dump

kubectl get node worker-node-1.example.com

kubectl describe node worker-node1.example.com | less

# Look at Status(should be FALSE), Address, Capacity, and Events

kubectl get namespaces

kubectl get pods -A

kubectl get pods -n kube-system

# Look into /etc/kubernetes/ - Config, manifests & pki

kubectl get pods -n kube-system -o wide | grep proxy

service kubelet status

# **Registering Working Nodes**

kubectl get nodes

kubectl describe node worker-node1.example.com

kubectl delete node worker-node1.example.com

kubectl get nodes

Create a new file with Node info,

vi nodereg.json

{

  "kind": "Node",

  "apiVersion": "v1",

  "metadata": {

   "name": "worker-node-1.example.com",

   "labels": {

   "name": "firstnode"

   }

  }

}

kubectl create -f nodereg.json

# **kubectl get nodes**

# **Deploying the first pod and accessing it**

kubectl run nginxpod --image=nginx --port 80

kubectl get pods

kubectl describe pod nginxpod

kubectl exec -it nginxpod /bin/sh

# **Kubernetes Dashboard**

# **Deploying the dashboard**

kubectl apply -f<https://raw.githubusercontent.com/kubernetes/dashboard/v2.5.0/aio/deploy/recommended.yaml>

# **Verifying the Dashboard resources**

kubectl get pods -n kubernetes-dashboard -o wide

kubectl get deployment -n kubernetes-dashboard -o wide

kubectl get svc -n kubernetes-dashboard -o wide

# **Editing the Service type of the dashboard**

kubectl edit svc -n kubernetes-dashboard kubernetes-dashboard

**Note:** Change the attribute after entering the deployment

         type: ClusterIP (image 1) to NodePort (image 2)

Graphical user interface, text, application

Description automatically generated

**Verifying the changes**

kubectl get svc -n kubernetes-dashboard -o wide

Note down the service(node-port) port number , here it is 31851

**Checking where the Pod is running**

kubectl get pods -n kubernetes-dashboard -o wide

kubectl get svc -n kubernetes-dashboard -o wide

kubectl get nodes -o wide

**Accessing Kubernetes Dashboard**

Click on the master tab on the lab, and then click on the desktop option.

Open Firefox browser

[**https://localhost**](https://localhost/)**:<<NodePort>>**

Example: https://localhost:31851

Click on Advanced -> Accept Risk and Continue

On the Kubernetes Dashboard,

Select **Token** from the given options and enter the token

**Upgrading the control plane(Master)**

sudo apt-get install -y kubeadm=1.23.17-00 --allow-change-held-packages

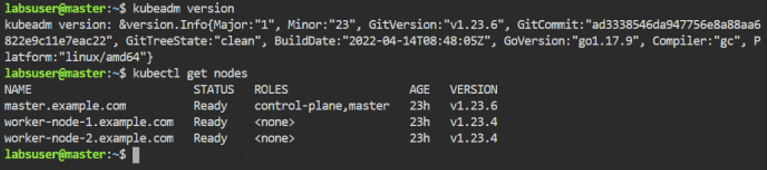
sudo apt-get install -y kubelet=1.23.17-00 kubectl=1.23.17-00 --allow-change-held-packages

**Verifying the updated version of Kubernetes**

kubeadm version

kubectl get nodes

sudo kubeadm upgrade plan

**

**Create a new user with permissions to create, list, get, update, and delete pods**

**Create the CertificateSigningRequest**

csr.yml

apiVersion: certificates.k8s.io/v1beta1  
kind: CertificateSigningRequest  
metadata:  
name: mike  
spec:  
request: $(cat mike.csr | base64 | tr -d '\n')  
usages:  
- digital signature  
- key encipherment  
- server auth

**Apply above yml file**

kubectl apply -f csr.yaml

**Create the Role with the permissions**

New yml file dev-role.yaml

apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
namespace: dev  
name: dev-role  
rules:  
- apiGroups: [""] # "" indicates the core API group  
resources: ["pods"]  
verbs: ["create","list", "get", "update","delete"]

kubectl apply -f dev-role.yaml

**Bind the user and role**

mike-roleBinding.yaml

apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
name: mike-rolebinding  
namespace: dev  
subjects:  
- kind: User  
name: mike  
apiGroup: rbac.authorization.k8s.io  
roleRef:  
kind: Role  
name: dev-role  
apiGroup: rbac.authorization.k8s.io

kubectl apply -f mike-roleBinding.yaml

kubectl get rolebindings -n dev  
​  
NAME AGE  
mike-rolebinding 6s

kubectl describe rolebinding mike-rolebinding -n dev  
​  
Name: mike-rolebinding  
Labels: <none>  
Annotations: kubectl.kubernetes.io/last-applied-configuration:  
{"apiVersion":"rbac.authorization.k8s.io/v1","kind":"RoleBinding","metadata":{"annotations":{},"name":"mike-rolebinding","namespace":"dev"...  
Role:  
Kind: Role  
Name: dev-role  
Subjects:  
Kind Name Namespace  
---- ---- ---------  
User mike

**Verify the access permissions**

k auth can-i list pods --as mike -n dev  
yes

**Configure application on the pod**

apiVersion: v1

kind: Pod

metadata:

  name: darwin

spec:

  containers:

- name: nginx

    image: nginx:1.21.4

    ports:

- containerPort: 80

**Take snapshot of ETCD database**

sudo apt install etcd-client

From the etcd static pod manifest file located at /etc/kubernetes/manifests/etcd.yaml the location.

kubectl get po -n kube-system kubectl describe pod etcd-master-node -n kube-system

Take an etcd snapshot backup using the following command

ETCDCTL\_API=3 etcdctl \ --endpoints=https://127.0.0.1:2379 \ --cacert=<ca-file> \ --cert=<cert-file> \ --key=<key-file> \ snapshot save <backup-file-location>

ETCDCTL\_API=3 etcdctl \ --endpoints=https://127.0.0.1:2379 \ --cacert=/etc/kubernetes/pki/etcd/ca.crt \ --cert=/etc/kubernetes/pki/etcd/server.crt \ --key=/etc/kubernetes/pki/etcd/server.key \ snapshot save /opt/backup/etcd.db

**Verify snapshot**

ETCDCTL\_API=3 etcdctl --write-out=table snapshot status /opt/backup/etcd.db

**Kubernetes etcd Restore Using Snapshot Backup**

ETCDCTL\_API=3 etcdctl snapshot restore /opt/backup/etcd.db

ETCDCTL\_API=3 etcdctl **--data-dir /opt/etcd** snapshot restore /opt/backup/etcd.db

**Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured**

oc autoscale deployment/image-registry --min=5 --max=7 --cpu-percent=75

horizontalpodautoscaler.autoscaling/image-registry autoscaled

Sample HPA for the image-registry Deployment object with minReplicas set to 3

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: image-registry

namespace: default

spec:

maxReplicas: 7

minReplicas: 3

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: image-registry

targetCPUUtilizationPercentage: 75

status:

currentReplicas: 5

desiredReplicas: 0

oc get deployment image-registry

**Sample HPA object with a scaling policy**

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: hpa-resource-metrics-memory

namespace: default

spec:

behavior:

scaleDown:

policies:

- type: Pods

value: 4

periodSeconds: 60

- type: Percent

value: 10

periodSeconds: 60

selectPolicy: Min

stabilizationWindowSeconds: 300

scaleUp:

policies:

- type: Pods

value: 5

periodSeconds: 70

- type: Percent

value: 12

periodSeconds: 80

selectPolicy: Max

stabilizationWindowSeconds: 0