# Infraestructura

## Networking.tf

Creamos la VPC

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

name = "${local.project\_name}-${terraform.workspace}"

cidr = "${local.vpc\_cidr}"

azs = ["${local.aws\_region}a", "${local.aws\_region}b", "${local.aws\_region}c"]

private\_subnets = ["${local.private\_subnet\_1}", "${local.private\_subnet\_2}", "${local.private\_subnet\_3}"]

public\_subnets = ["${local.public\_subnet\_1}", "${local.public\_subnet\_2}", "${local.public\_subnet\_3}"]

create\_vpc = true

create\_igw = true

enable\_nat\_gateway = true

single\_nat\_gateway = false

reuse\_nat\_ips = true

external\_nat\_ip\_ids = "${aws\_eip.nat.\*.id}"

}

Definimos el CIDR, región y nombre del proyecto para utilizar en los tags. Creamos 3 subredes públicas y 3 subredes privadas

locals {

cluster\_name = "${var.project\_name}-${terraform.workspace}"

#Private Subnets

private\_subnet\_1 = cidrsubnet("${local.vpc\_cidr}", 8, 1)

private\_subnet\_2 = cidrsubnet("${local.vpc\_cidr}", 8, 2)

private\_subnet\_3 = cidrsubnet("${local.vpc\_cidr}", 8, 3)

#Public Subnets

public\_subnet\_1 = cidrsubnet("${local.vpc\_cidr}", 8, 11)

public\_subnet\_2 = cidrsubnet("${local.vpc\_cidr}", 8, 12)

public\_subnet\_3 = cidrsubnet("${local.vpc\_cidr}", 8, 13)

project\_name = "obligatorio"

aws\_region = "us-east-1"

vpc\_cidr = "10.0.0.0/16"

}

Se asignan 3 elastic ips , las cuales se utilizaran en los nat-gateways. Esto proporciona redundancia en caso de que una zona se caiga.

count = 3

vpc = true

}

## Main.tf

Defininos los providers:

Hashicorp/aws - para todo la interacción con AWS

Hashicorp/Helm - para desplegar algún helmchart dentro de eks.

Gavinbunney/kubectl - para ejecutar comandos dentro de nuestro cluster.

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.18"

}

helm = {

source = "hashicorp/helm"

version = "~> 2.4"

}

kubectl = {

source = "gavinbunney/kubectl"

version = "~> 1.14"

}

}

Defininos el bucket de S3 que se creó a mano donde vamos a guardar el estado de la infra manejado por terraform.

backend "s3" {

bucket = "obligatorio-abdm-terraform"

key = "obligatorio.tfstate"

region = "us-east-1"

encrypt = true

}

aws s3api create-bucket --bucket obligatorio-abdm-terraform --region us-east-1

## Iam.tf

Se definan reglas de entropía de password para los usuarios de la cuenta.

module "iam\_account" {

source = "terraform-aws-modules/iam/aws//modules/iam-account"

version = "~> 4.3"

account\_alias = var.company\_name

minimum\_password\_length = 12

max\_password\_age = 30

password\_reuse\_prevention = 5

require\_lowercase\_characters = true

require\_uppercase\_characters = true

require\_symbols = true

require\_numbers = true

}

Se crean políticas y roles para los distintos componentes que corren desde eks.

(Por ser muy largo solo se pone un ejemplo)

resource "aws\_iam\_role" "route53-externaldns-controller" {

name = "route53-externaldns-controller"

assume\_role\_policy = data.aws\_iam\_policy\_document.external\_dns.json

}

## Route53.tf

Se define la zona pública de DNS.

resource "aws\_route53\_zone" "primary" {

name = var.route53\_domain\_name

}

Se crea un registro de DNS tipo A con el valor del balanceador que se crea más adelante.

resource "aws\_route53\_record" "obligatorio" {

zone\_id = aws\_route53\_zone.primary.zone\_id

name = "obligatorio"

type = "A"

alias {

name = aws\_lb.obligatorio.dns\_name

zone\_id = aws\_lb.obligatorio.zone\_id

evaluate\_target\_health = false

}

}

## Lb.tf

Se crea un balanceador externo y se vincula con el security group creado en sg.tf

resource "aws\_lb" "obligatorio" {

name = var.project\_name

internal = false

load\_balancer\_type = "application"

security\_groups = [module.sg\_external\_alb.security\_group\_id]

subnets = "${module.vpc.public\_subnets}"

enable\_deletion\_protection = true

}

Se crea un listener para el puerto 80 y se lo agrega al balanceador previamente creado.

resource "aws\_lb\_listener" "obligatorio" {

load\_balancer\_arn = aws\_lb.obligatorio.arn

port = "80"

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.obligatorio.arn

}

}

Se crea un target group que va a escuchar en el puerto 31234.

resource "aws\_lb\_target\_group" "obligatorio" {

name = "obligatorio-tg"

port = 31234

protocol = "HTTP"

vpc\_id = module.vpc.vpc\_id

}

Se listan todas las instancias de EC2 que tengan como tag Name = initial.

data "aws\_instances" "obligatorio" {

instance\_tags = {

Name = "initial"

}

instance\_state\_names = ["running", "stopped"]

}

Se agregan las instancias previamente listadas al target group.

resource "aws\_lb\_target\_group\_attachment" "obligatorio" {

target\_group\_arn = aws\_lb\_target\_group.obligatorio.id

count = length(data.aws\_instances.obligatorio.ids)

target\_id = data.aws\_instances.obligatorio.ids[count.index]

port = 31234

}

## Sg.tf

Se define el security group que se agrega al balanceador que recibe todo el tráfico externo.

module "sg\_external\_alb" {

source = "terraform-aws-modules/security-group/aws"

name = "external\_alb"

description = "Security group for external connections"

vpc\_id = module.vpc.vpc\_id

egress\_rules = ["all-all"]

ingress\_with\_cidr\_blocks = [

{

from\_port = 80

to\_port = 80

protocol = "tcp"

description = "external to LB"

cidr\_blocks = "0.0.0.0/0"

}

]

}

## Eks.tf

Se crea el cluster de k8s.

module "eks" {

source = "terraform-aws-modules/eks/aws"

version = "~> 18.0"

cluster\_name = "${var.project\_name}-${terraform.workspace}"

cluster\_version = "1.22"

cluster\_endpoint\_private\_access = true

cluster\_endpoint\_public\_access = true

enable\_irsa = true

cluster\_addons = {

coredns = {

resolve\_conflicts = "OVERWRITE"

}

kube-proxy = {}

vpc-cni = {

resolve\_conflicts = "OVERWRITE"

}

}

Se crean instancias auto manejadas tipo spot para los workers.

eks\_managed\_node\_groups = {

initial = {

min\_size = 1

max\_size = 1

desired\_size = 1

instance\_types = ["t3.medium"]

capacity\_type = "SPOT" # ON\_DEMAND or SPOT

}

}

Se asignan permisos al cluster para un grupo de usuario aws/iam llamado 2soAdmin.

# aws-auth configmap

manage\_aws\_auth\_configmap = true

aws\_auth\_roles = [

{

rolearn = "arn:aws:iam::813224394680:group/2soAdmin"

usergroup = "2soAdmin"

groups = ["system:masters"]

}

]

Graphical user interface, application

Description automatically generated

Diagrama de security groups.

Diagram

Description automatically generated

# Kubernet

## Microservicios

Diagram

Description automatically generated

## Nginx

La función de este microservicio es servir de reverse proxy y endpoint centralizado a todos los microservicios. Esta desplegado como NodePort y recibe todo el tráfico entrante que llega del balanceador de carga de AWS.

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: NodePort

selector:

app: nginx-app

ports:

- protocol: "TCP"

port: 80

nodePort: 31234

targetPort: 80

server {

access\_log /var/log/nginx/api\_access.log main;

listen 80;

server\_name \_;

location /orders {

proxy\_pass http://172.20.10.10/orders/;

}

location /payments {

proxy\_pass http://172.20.10.11/payments/;

}

location /products {

proxy\_pass http://172.20.10.13/products/;

}

location /shipping {

proxy\_pass http://172.20.10.12/shipping/;

}

}

## Orders-service-example

Microservicio que se encarga de generar las ordenes de compra. Este microservico está configurado en modo ClusterIp y tiene una ip interna fija, el contenedor escucha en el puerto 8080 y exponemos el puerto 80. Este microservicio necesita saber de antemano la url o ips de los servicios products, shipping y payments. Por esta razón se utilizan IPs fijas en la solución. (Lo malo es que no nos permite escalar en cantidad de pods, para solventar esto se debe pedir al área de desarrollo que reconsidere la arquitectura)

apiVersion: v1

kind: Service

metadata:

name: orders-service

spec:

clusterIP: 172.20.10.10

selector:

app: orders-app

ports:

- protocol: "TCP"

port: 80

targetPort: 8080

apiVersion: apps/v1

kind: Deployment

metadata:

name: orders-app

spec:

selector:

matchLabels:

app: orders-app

replicas: 1

template:

metadata:

labels:

app: orders-app

spec:

containers:

- name: orders-app

image: ortdevops2022/orders-service-example

env:

- name: APP\_ARGS

value: "http://172.20.10.11 http://172.20.10.12 http://172.20.10.13"

resources:

requests:

memory: "500Mi"

cpu: "250m"

limits:

memory: "1000Mi"

cpu: "500m"

imagePullPolicy: Always

ports:

- containerPort: 8080

## Payments-service-example

Microservicio que se encarga de generar los pagos. Este microservico está configurado en modo ClusterIp y tiene una ip interna fija, el contenedor escucha en el puerto 8080 y exponemos el puerto 80, el servicio orders se conecta directamente a este servicio.

apiVersion: v1

kind: Service

metadata:

name: payments-service

spec:

clusterIP: 172.20.10.11

selector:

app: payments-app

ports:

- protocol: "TCP"

port: 80

targetPort: 8080

apiVersion: apps/v1

kind: Deployment

metadata:

name: payments-app

spec:

selector:

matchLabels:

app: payments-app

replicas: 1

template:

metadata:

labels:

app: payments-app

spec:

containers:

- name: payments-app

image: ortdevops2022/payments-service-example

resources:

requests:

memory: "500Mi"

cpu: "250m"

limits:

memory: "1000Mi"

cpu: "500m"

imagePullPolicy: Always

ports:

- containerPort: 8080

## Products-service-example

Microservicio que se encarga de listar los productos. Este microservico está configurado en modo ClusterIp y tiene una ip interna fija, el contenedor escucha en el puerto 8080 y exponemos el puerto 80.

apiVersion: v1

kind: Service

metadata:

name: products-service

spec:

clusterIP: 172.20.10.13

selector:

app: products-app

ports:

- protocol: "TCP"

port: 80

targetPort: 8080

apiVersion: apps/v1

kind: Deployment

metadata:

name: products-app

spec:

selector:

matchLabels:

app: products-app

replicas: 1

template:

metadata:

labels:

app: products-app

spec:

containers:

- name: products-app

image: ortdevops2022/products-service-example

resources:

requests:

memory: "500Mi"

cpu: "250m"

limits:

memory: "1000Mi"

cpu: "500m"

imagePullPolicy: Always

ports:

- containerPort: 8080

## Shipping-service-example

Microservicio que se encarga de listar los envíos. Este microservico está configurado en modo ClusterIp y tiene una ip interna fija, el contenedor escucha en el puerto 8080 y exponemos el puerto 80.

apiVersion: v1

kind: Service

metadata:

name: shipping-service

spec:

clusterIP: 172.20.10.12

selector:

app: shipping-app

ports:

- protocol: "TCP"

port: 80

targetPort: 8080

apiVersion: apps/v1

kind: Deployment

metadata:

name: shipping-app

spec:

selector:

matchLabels:

app: shipping-app

replicas: 1

template:

metadata:

labels:

app: shipping-app

spec:

containers:

- name: shipping-app

image: ortdevops2022/shipping-service-example

resources:

requests:

memory: "500Mi"

cpu: "250m"

limits:

memory: "1000Mi"

cpu: "500m"

imagePullPolicy: Always

ports:

- containerPort: 8080

A picture containing text, electronics

Description automatically generated

## Reportes Sonarcloud

Graphical user interface, application

Description automatically generated

## Pruebas de los servicios

A picture containing table

Description automatically generated