**CHAT APP**

Imaginate que eres un senior devops con 10 años de experiencia en devops, eks, ecs, kubernetes, aws, etc y se te pide para un cliente que:

**Desplegar una Aplicación de Chat en Tiempo Real con WebSockets en EKS**:

Con el objetivo de reunir conceptos los estudiantes crearán una aplicación de

chat en tiempo real utilizando WebSockets en EKS.

➔ La aplicación **se desplegará automáticamente con Terraform y GitHub Actions**

usando todos los modulos que hemos implementado anteriormente con el

objetivo de agilizar el desarrollo y el despliegue.

En el repositorio infra\_resources:

❖ Crea una *carpeta llamada chat\_app*:

❖ Crear *otra subcarpeta llamada terraform*.

❖ Crea una *carpeta para el pipeline* para desplegar la app llamado

*chat\_app\_deploy.yaml*.

❖ Crear los ficheros que aparecen el la imagen de forma que tu carpeta se vea igual y Dentro de subcarpeta llamada terraform crea la carpeta llamada modules y traete todos los módulos necesarios de forma que tu carpeta se vea asi y en la carpeta principal: chat\_app crea el resto de ficheros: main.tf, outputs.tf , variables.tf, provider.tf. Tambien Traer los módulos de redes y subredes creados antes a la carpeta de módulos actual.

github/workflows

chat\_app\_deploy.yaml

terrafrom

modules

alb

ecr

eks-cluster

eks-node-group

iam-roles

k8s

chat-app-deployment.yaml

chat-app-service.yaml

Dockerfile

index.js

package.json

main.tf

variables.tf

outputs.tf

provider.tf

Esta seria **la estructura de carpetas** Y TODOS LOS DATOS DE INTERES ahora debes codificarme todo correctamente y hacer un despliegue funcional de la aplicacion chat-app, a la aplicacion se accedera a través de la url del balanceador de carga

**chat\_app**/

│

├── *.github*/

│ └── *workflows*/

│ └── chat\_app\_deploy.yaml

│

├── terraform/

│ ├── main.tf

│ ├── provider.tf

│ ├── variables.tf

│ ├── outputs.tf

│ └── modules/

│ ├── alb/

│ ├── ecr/

│ ├── eks-cluster/

│ ├── eks-node-group/

│ └── iam-roles/

│

├── k8s/

│ ├── chat-app-deployment.yaml

│ └── chat-app-service.yaml

│

├── Dockerfile

├── index.js

├── package.json

**Archivos Terraform**

**terraform/provider.tf**

hcl

CopiarEditar

provider "aws" {

region = var.aws\_region

}

**terraform/variables.tf**

hcl

CopiarEditar

variable "aws\_region" {

description = "AWS region to deploy infrastructure"

type = string

default = "us-west-2"

}

variable "cluster\_name" {

description = "EKS Cluster name"

type = string

default = "chat-app-cluster"

}

variable "node\_group\_name" {

description = "EKS Node Group name"

type = string

default = "chat-app-node-group"

}

# Add other variables as needed for ECR, ALB, etc.

**terraform/main.tf**

hcl

CopiarEditar

module "eks\_cluster" {

source = "./modules/eks-cluster"

cluster\_name = var.cluster\_name

region = var.aws\_region

}

module "eks\_node\_group" {

source = "./modules/eks-node-group"

cluster\_name = module.eks\_cluster.cluster\_name

node\_group\_name = var.node\_group\_name

region = var.aws\_region

}

module "alb" {

source = "./modules/alb"

region = var.aws\_region

cluster\_name = module.eks\_cluster.cluster\_name

}

module "ecr" {

source = "./modules/ecr"

region = var.aws\_region

}

module "iam\_roles" {

source = "./modules/iam-roles"

region = var.aws\_region

}

**terraform/outputs.tf**

hcl

CopiarEditar

output "eks\_cluster\_endpoint" {

value = module.eks\_cluster.cluster\_endpoint

}

output "eks\_cluster\_name" {

value = module.eks\_cluster.cluster\_name

}

output "alb\_dns\_name" {

value = module.alb.dns\_name

}

**Módulos Terraform**

Los módulos en terraform/modules/ manejarán la creación de recursos como el ALB, ECR, EKS Cluster, Node Group, IAM Roles, etc. Por ejemplo, el módulo para EKS podría verse así:

**terraform/modules/eks-cluster/main.tf**

hcl

CopiarEditar

resource "aws\_eks\_cluster" "this" {

name = var.cluster\_name

role\_arn = var.iam\_role\_arn

vpc\_config {

subnet\_ids = var.subnet\_ids

}

}

**Archivos de Kubernetes**

**k8s/chat-app-deployment.yaml**

yaml

CopiarEditar

apiVersion: apps/v1

kind: Deployment

metadata:

name: chat-app

spec:

replicas: 2

selector:

matchLabels:

app: chat-app

template:

metadata:

labels:

app: chat-app

spec:

containers:

- name: chat-app

image: ${aws\_account\_id}.dkr.ecr.${aws\_region}.amazonaws.com/chat-app:latest

ports:

- containerPort: 8080

**k8s/chat-app-service.yaml**

yaml

CopiarEditar

apiVersion: v1

kind: Service

metadata:

name: chat-app-service

spec:

selector:

app: chat-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer

**Dockerfile y Node.js**

**Dockerfile**

dockerfile

CopiarEditar

FROM node:16

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

EXPOSE 8080

CMD ["node", "index.js"]

**index.js**

js

CopiarEditar

const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', function connection(ws) {

ws.on('message', function incoming(message) {

ws.send(`Received: ${message}`);

});

});

console.log('Chat server running on ws://localhost:8080');

**package.json**

json

CopiarEditar

{

"name": "chat-app",

"version": "1.0.0",

"description": "A simple chat app using WebSockets",

"main": "index.js",

"dependencies": {

"ws": "^8.7.0"

},

"scripts": {

"start": "node index.js"

}

}

**GitHub Actions Pipeline**

**.github/workflows/chat\_app\_deploy.yaml**

yaml

CopiarEditar

name: Deploy Chat App

on:

push:

branches:

- main

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Initialize Terraform

run: terraform init

- name: Plan Terraform

run: terraform plan

- name: Apply Terraform

run: terraform apply -auto-approve

deploy\_k8s:

runs-on: ubuntu-latest

needs: terraform

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up kubectl

uses: azure/setup-kubectl@v1

with:

version: 'v1.21.0'

- name: Configure kubectl

run: kubectl config set-cluster eks --server=${{ secrets.EKS\_CLUSTER\_ENDPOINT }} --certificate-authority=${{ secrets.EKS\_CA\_CERTIFICATE }}

env:

KUBECONFIG: ${{ secrets.KUBECONFIG }}

- name: Deploy to Kubernetes

run: |

kubectl apply -f k8s/chat-app-deployment.yaml

kubectl apply -f k8s/chat-app-service.yaml

Este flujo de trabajo creará y desplegará los recursos necesarios utilizando Terraform para EKS, ALB, ECR, etc. Luego, desplegará la aplicación en Kubernetes.

**Despliegue de la aplicación**

1. El flujo de trabajo de GitHub Actions se activará cuando haya un cambio en la rama main.
2. Terraform creará la infraestructura necesaria en AWS (EKS, ALB, ECR, etc.).
3. Luego, se desplegará la aplicación de chat en el clúster de EKS.
4. La aplicación será accesible a través del DNS del balanceador de carga creado por el módulo ALB.

Este es un esquema básico y funcional para desplegar la aplicación de chat en tiempo real en EKS usando WebSockets, Terraform y GitHub Actions.

4o mini

chat\_app/

│

├── terraform/

│ ├── main.tf

│ ├── provider.tf

│ ├── variables.tf

│ ├── outputs.tf

│ └── modules/

│ ├── alb/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── ecr/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── eks-cluster/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── eks-node-group/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── iam-roles/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ └── vpc/

│ ├── main.tf

│ ├── variables.tf

│ └── outputs.tf

│

├── k8s/

│ ├── chat-app-deployment.yaml

│ └── chat-app-service.yaml

│

├── Dockerfile

├── index.js

├── package.json

└── .github/

└── workflows/

└── chat\_app\_deploy.yaml

me falta que me proporciones el resto de los archivos como de la subcarpeta alb, los archivos main.tf, variables.tf y outputs.tf, al igual de la subcarpeta ecr, como de la eks-node-group, como de iam-roles y de eks-cluster que solo me has dado el main, tambien la subcarpeta vpc dentro de modules para las redes y subredes para hacerlo todo modular y funcional y este completamente integrado y que funcione el despliegue de la app de chat, por favor

**ChatGPT Plus**

ChatGPT

¡Claro! Te proporcionaré el código completo de los módulos para que todo funcione de manera modular y funcional. Incluyo los archivos que me mencionas para que puedas integrarlos y realizar el despliegue de la aplicación de chat en EKS correctamente.

**Estructura de carpetas (modificada con todos los módulos)**

css

CopiarEditar

chat\_app/

│

├── terraform/

│ ├── main.tf

│ ├── provider.tf

│ ├── variables.tf

│ ├── outputs.tf

│ └── modules/

│ ├── alb/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── ecr/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── eks-cluster/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── eks-node-group/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ ├── iam-roles/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── outputs.tf

│ └── vpc/

│ ├── main.tf

│ ├── variables.tf

│ └── outputs.tf

│

├── k8s/

│ ├── chat-app-deployment.yaml

│ └── chat-app-service.yaml

│

├── Dockerfile

├── index.js

├── package.json

└── .github/

└── workflows/

└── chat\_app\_deploy.yaml

**Módulo ALB (terraform/modules/alb/main.tf)**

hcl

CopiarEditar

resource "aws\_lb" "this" {

name = "${var.cluster\_name}-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = var.security\_groups

subnets = var.subnets

enable\_deletion\_protection = false

enable\_cross\_zone\_load\_balancing = true

idle\_timeout {

seconds = 60

}

}

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

name = "${var.cluster\_name}-tg"

port = 80

protocol = "HTTP"

vpc\_id = var.vpc\_id

health\_check {

path = "/"

interval = 30

timeout = 5

healthy\_threshold = 3

unhealthy\_threshold = 3

}

}

resource "aws\_lb\_listener" "this" {

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

port = "80"

default\_action {

type = "fixed-response"

fixed\_response {

status\_code = "200"

content\_type = "text/plain"

message\_body = "OK"

}

}

}

**terraform/modules/alb/variables.tf**

hcl

CopiarEditar

variable "cluster\_name" {

description = "Cluster name"

type = string

}

variable "security\_groups" {

description = "Security groups to attach to the ALB"

type = list(string)

}

variable "subnets" {

description = "Subnets for the ALB"

type = list(string)

}

variable "vpc\_id" {

description = "VPC ID"

type = string

}

**terraform/modules/alb/outputs.tf**

hcl

CopiarEditar

output "alb\_dns\_name" {

value = aws\_lb.this.dns\_name

}

**Módulo ECR (terraform/modules/ecr/main.tf)**

hcl

CopiarEditar

resource "aws\_ecr\_repository" "this" {

name = "chat-app"

}

resource "aws\_ecr\_lifecycle\_policy" "this" {

repository\_name = aws\_ecr\_repository.this.name

policy = jsonencode({

rule = [

{

rulePriority = 1

description = "Expire untagged images"

status = "enabled"

filter = {

tagStatus = "untagged"

}

action = {

type = "expire"

expiry = {

days = 30

}

}

}

]

})

}

**terraform/modules/ecr/variables.tf**

hcl

CopiarEditar

variable "region" {

description = "AWS region"

type = string

}

**terraform/modules/ecr/outputs.tf**

hcl

CopiarEditar

output "repository\_url" {

value = aws\_ecr\_repository.this.repository\_url

}

**Módulo EKS Cluster (terraform/modules/eks-cluster/main.tf)**

hcl

CopiarEditar

resource "aws\_eks\_cluster" "this" {

name = var.cluster\_name

role\_arn = var.iam\_role\_arn

vpc\_config {

subnet\_ids = var.subnet\_ids

}

}

**terraform/modules/eks-cluster/variables.tf**

hcl

CopiarEditar

variable "cluster\_name" {

description = "EKS Cluster name"

type = string

}

variable "iam\_role\_arn" {

description = "IAM role ARN for the EKS cluster"

type = string

}

variable "subnet\_ids" {

description = "List of subnet IDs"

type = list(string)

}

**terraform/modules/eks-cluster/outputs.tf**

hcl

CopiarEditar

output "cluster\_endpoint" {

value = aws\_eks\_cluster.this.endpoint

}

output "cluster\_name" {

value = aws\_eks\_cluster.this.name

}

**Módulo EKS Node Group (terraform/modules/eks-node-group/main.tf)**

hcl

CopiarEditar

resource "aws\_eks\_node\_group" "this" {

cluster\_name = var.cluster\_name

node\_group\_name = var.node\_group\_name

node\_role\_arn = var.node\_role\_arn

subnet\_ids = var.subnet\_ids

scaling\_config {

desired\_size = 2

max\_size = 3

min\_size = 1

}

instance\_types = ["t3.medium"]

}

**terraform/modules/eks-node-group/variables.tf**

hcl

CopiarEditar

variable "cluster\_name" {

description = "EKS Cluster name"

type = string

}

variable "node\_group\_name" {

description = "EKS Node Group name"

type = string

}

variable "node\_role\_arn" {

description = "IAM role ARN for the Node Group"

type = string

}

variable "subnet\_ids" {

description = "List of subnet IDs"

type = list(string)

}

**terraform/modules/eks-node-group/outputs.tf**

hcl

CopiarEditar

output "node\_group\_name" {

value = aws\_eks\_node\_group.this.node\_group\_name

}

**Módulo IAM Roles (terraform/modules/iam-roles/main.tf)**

hcl

CopiarEditar

resource "aws\_iam\_role" "this" {

name = var.role\_name

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Action = "sts:AssumeRole"

Effect = "Allow"

Principal = {

Service = "eks.amazonaws.com"

}

}

]

})

}

**terraform/modules/iam-roles/variables.tf**

hcl

CopiarEditar

variable "role\_name" {

description = "IAM Role name"

type = string

}

**terraform/modules/iam-roles/outputs.tf**

hcl

CopiarEditar

output "role\_arn" {

value = aws\_iam\_role.this.arn

}

**Módulo VPC (terraform/modules/vpc/main.tf)**

hcl

CopiarEditar

resource "aws\_vpc" "this" {

cidr\_block = var.cidr\_block

enable\_dns\_support = true

enable\_dns\_hostnames = true

}

resource "aws\_subnet" "this" {

count = length(var.subnet\_cidr\_blocks)

cidr\_block = element(var.subnet\_cidr\_blocks, count.index)

vpc\_id = aws\_vpc.this.id

availability\_zone = element(var.availability\_zones, count.index)

}

**terraform/modules/vpc/variables.tf**

hcl

CopiarEditar

variable "cidr\_block" {

description = "CIDR block for the VPC"

type = string

}

variable "subnet\_cidr\_blocks" {

description = "CIDR blocks for the subnets"

type = list(string)

}

variable "availability\_zones" {

description = "Availability zones for the subnets"

type = list(string)

}

**terraform/modules/vpc/outputs.tf**

hcl

CopiarEditar

output "vpc\_id" {

value = aws\_vpc.this.id

}

output "subnet\_ids" {

value = aws\_subnet.this[\*].id

}

**Integración final**

1. **Terraform**: Ejecutar **terraform init y terraform apply** en la **carpeta de terraform** para desplegar la infraestructura (VPC, ECR, EKS, ALB, etc.).
2. **Docker**: Construir y subir la imagen de Docker a ECR usando d**ocker build y docker push.**
3. **Kubernetes**: Desplegar la aplicación en EKS usando los archivos de Kubernetes **chat-app-deployment.yaml** y **chat-app-service.yaml.**

**GitHub Actions**

El archivo .**github/workflows/chat\_app\_deploy.yaml** seguirá siendo el mismo que te proporcioné anteriormente para automatizar el proceso de CI/CD.

Con estos archivos y módulos, deberías tener una infraestructura completa y funcional para el despliegue de la aplicación de chat en EKS. ¡Listo para ejecutarlo!

4o mini

Tien el modulo vpc que me has proporcionado varias zonas de disponibilidad para desplegar la app, asi como varias subnets publicas y varias privadas y grupos de seguridad, etc para un despliegue escalable, de alta disponibilidad, et.En caso de que no sea asi debes rehacerlo correctamente y compatible y sin fallos con el resto de modulos de esta aplicacion, por favor

**ChatGPT Plus**

ChatGPT

Entiendo, para asegurar un despliegue escalable, de alta disponibilidad y que sea compatible con el resto de los módulos, vamos a modificar el módulo de **VPC** para incluir **múltiples zonas de disponibilidad**, **subnets públicas y privadas**, **grupos de seguridad** y las configuraciones necesarias para una infraestructura robusta y escalable.

Aquí te proporciono un módulo de **VPC** actualizado y con estos requisitos:

**Módulo VPC con subnets públicas y privadas, múltiples zonas de disponibilidad y grupos de seguridad**

**terraform/modules/vpc/main.tf**

hcl

CopiarEditar

resource "aws\_vpc" "this" {

cidr\_block = var.cidr\_block

enable\_dns\_support = true

enable\_dns\_hostnames = true

tags = {

Name = "${var.cluster\_name}-vpc"

}

}

# Subnets públicas

resource "aws\_subnet" "public" {

count = length(var.public\_subnet\_cidr\_blocks)

cidr\_block = element(var.public\_subnet\_cidr\_blocks, count.index)

vpc\_id = aws\_vpc.this.id

availability\_zone = element(var.availability\_zones, count.index)

map\_public\_ip\_on\_launch = true

tags = {

Name = "${var.cluster\_name}-public-${count.index}"

}

}

# Subnets privadas

resource "aws\_subnet" "private" {

count = length(var.private\_subnet\_cidr\_blocks)

cidr\_block = element(var.private\_subnet\_cidr\_blocks, count.index)

vpc\_id = aws\_vpc.this.id

availability\_zone = element(var.availability\_zones, count.index)

tags = {

Name = "${var.cluster\_name}-private-${count.index}"

}

}

# Internet Gateway

resource "aws\_internet\_gateway" "this" {

vpc\_id = aws\_vpc.this.id

}

# NAT Gateway para las subnets privadas

resource "aws\_eip" "this" {

vpc = true

}

resource "aws\_nat\_gateway" "this" {

allocation\_id = aws\_eip.this.id

subnet\_id = aws\_subnet.public[0].id

depends\_on = [aws\_internet\_gateway.this]

}

# Route Tables

resource "aws\_route\_table" "public" {

vpc\_id = aws\_vpc.this.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.this.id

}

}

resource "aws\_route\_table\_association" "public" {

count = length(aws\_subnet.public)

subnet\_id = aws\_subnet.public[count.index].id

route\_table\_id = aws\_route\_table.public.id

}

resource "aws\_route\_table" "private" {

vpc\_id = aws\_vpc.this.id

route {

cidr\_block = "0.0.0.0/0"

nat\_gateway\_id = aws\_nat\_gateway.this.id

}

}

resource "aws\_route\_table\_association" "private" {

count = length(aws\_subnet.private)

subnet\_id = aws\_subnet.private[count.index].id

route\_table\_id = aws\_route\_table.private.id

}

# Seguridad: Grupos de Seguridad

resource "aws\_security\_group" "this" {

name = "${var.cluster\_name}-sg"

description = "Security group for EKS and ALB"

vpc\_id = aws\_vpc.this.id

egress {

cidr\_blocks = ["0.0.0.0/0"]

from\_port = 0

to\_port = 0

protocol = "-1"

}

ingress {

cidr\_blocks = ["0.0.0.0/0"]

from\_port = 80

to\_port = 80

protocol = "tcp"

}

tags = {

Name = "${var.cluster\_name}-sg"

}

}

**terraform/modules/vpc/variables.tf**

hcl

CopiarEditar

variable "cidr\_block" {

description = "CIDR block for the VPC"

type = string

}

variable "public\_subnet\_cidr\_blocks" {

description = "CIDR blocks for public subnets"

type = list(string)

}

variable "private\_subnet\_cidr\_blocks" {

description = "CIDR blocks for private subnets"

type = list(string)

}

variable "availability\_zones" {

description = "Availability zones for the subnets"

type = list(string)

}

variable "cluster\_name" {

description = "Cluster name"

type = string

}

**terraform/modules/vpc/outputs.tf**

hcl

CopiarEditar

output "vpc\_id" {

value = aws\_vpc.this.id

}

output "public\_subnet\_ids" {

value = aws\_subnet.public[\*].id

}

output "private\_subnet\_ids" {

value = aws\_subnet.private[\*].id

}

output "security\_group\_id" {

value = aws\_security\_group.this.id

}

**Explicación de las modificaciones:**

1. **VPC**: He añadido la creación de un VPC con DNS y soporte de nombres de host habilitado.
2. **Subnets públicas y privadas**:
   * Se crean subnets públicas y privadas utilizando CIDR blocks definidos en las variables.
   * Las subnets públicas tienen habilitada la opción de asignar IPs públicas a las instancias que se lancen en ellas.
3. **Internet Gateway**: Se crea un Internet Gateway para las subnets públicas.
4. **NAT Gateway**: Se añade un NAT Gateway para permitir que las instancias en subnets privadas tengan acceso a internet.
5. **Route Tables**:
   * Las subnets públicas se enrutan directamente al Internet Gateway.
   * Las subnets privadas se enrutan a través del NAT Gateway.
6. **Grupos de Seguridad**:
   * Se crea un grupo de seguridad que permite el acceso HTTP (puerto 80) a través de ALB y acceso general a través de egress.

**Cómo usar este módulo en el main.tf de la raíz:**

En tu archivo main.tf de la carpeta terraform, debes utilizar este módulo y proporcionar los valores de las variables.

**terraform/main.tf**

hcl

CopiarEditar

module "vpc" {

source = "./modules/vpc"

cidr\_block = "10.0.0.0/16"

public\_subnet\_cidr\_blocks = ["10.0.1.0/24", "10.0.2.0/24"]

private\_subnet\_cidr\_blocks = ["10.0.3.0/24", "10.0.4.0/24"]

availability\_zones = ["us-west-2a", "us-west-2b", "us-west-2c"]

cluster\_name = "chat-app-cluster"

}

output "vpc\_id" {

value = module.vpc.vpc\_id

}

output "public\_subnet\_ids" {

value = module.vpc.public\_subnet\_ids

}

output "private\_subnet\_ids" {

value = module.vpc.private\_subnet\_ids

}

output "security\_group\_id" {

value = module.vpc.security\_group\_id

}

**Resumen:**

* Este módulo ahora crea una **infraestructura escalable** con **subnets públicas y privadas** distribuidas en varias **zonas de disponibilidad**.
* El **NAT Gateway** garantiza que las subnets privadas tengan acceso a internet, lo cual es útil para desplegar aplicaciones en un entorno seguro.
* El **grupo de seguridad** habilita el tráfico HTTP, pero se puede ajustar según los requerimientos de tu aplicación.
* El resto de los módulos, como los de **EKS**, **ALB**, **ECR**, etc., pueden interactuar con este módulo sin problemas.

Con esta configuración, deberías poder desplegar una infraestructura robusta y escalable para la aplicación de chat en EKS.

4o mini

Puedes especificar detalladamente, cada uno de estos pasos : Integración final Terraform: Ejecutar terraform init y terraform apply en la carpeta de terraform para desplegar la infraestructura (VPC, ECR, EKS, ALB, etc.). Docker: Construir y subir la imagen de Docker a ECR usando docker build y docker push. Kubernetes: Desplegar la aplicación en EKS usando los archivos de Kubernetes chat-app-deployment.yaml y chat-app-service.yaml. GitHub Actions El archivo .github/workflows/chat\_app\_deploy.yaml seguirá siendo el mismo que te proporcioné anteriormente para automatizar el proceso de CI/CD.

**ChatGPT Plus**

ChatGPT

Aquí tienes una guía detallada para realizar la **integración final** del despliegue de la aplicación de chat en tiempo real con WebSockets en **EKS**, utilizando **Terraform**, **Docker**, **Kubernetes**, y **GitHub Actions**.

**Paso 1: Terraform: Ejecutar terraform init y terraform apply**

Primero, debes desplegar la infraestructura utilizando **Terraform**. Esto incluye la creación de la **VPC**, **ECR**, **EKS**, **ALB**, y otros recursos necesarios.

**1.1 Inicializar Terraform**

Para comenzar, asegúrate de estar en la carpeta donde se encuentra tu archivo main.tf de Terraform (en el directorio de terraform), y luego ejecuta:

**cd terraform**

**terraform init**

Este comando descargará los proveedores necesarios y los módulos de Terraform, asegurando que todo esté preparado para ejecutar el despliegue.

**1.2 Ejecutar terraform apply**

A continuación, ejecuta el siguiente comando para aplicar la configuración de Terraform:

**terraform apply**

Este comando mostrará un resumen de las acciones que Terraform tomará, como la **creación de subnets**, **VPC**, **clusters de EKS**, **ALB**, y **otros recursos**. Cuando estés listo para aplicar los cambios, escribe yes para confirmar.

Esto creará toda la **infraestructura en AWS** y te proporcionará **salida**s como **las ID de los recursos**, incluyendo **las subnets, VPC**, el **EKS cluster**, etc.

**Paso 2: Docker: Construir y subir la imagen de Docker a ECR**

Ahora que la infraestructura de AWS está configurada, el siguiente paso es construir la imagen de Docker para la aplicación de chat en tiempo real y cargarla en **Amazon ECR**.

**2.1 Configuración de Amazon ECR**

Antes de construir y subir la imagen, debes configurar el repositorio de **ECR** y obtener el URI para subir la imagen. **Si no lo has hecho en Terraform automaticamente**, puedes hacerlo **manualmente** **desde la consola de AWS** o **utilizar el siguiente comando**:

**aws ecr create-repository --repository-name chat-app --region <your-region>**

**aws ecr describe-repositories --repository-names chat-app-repo --region eu-west-1 --profile 248189943700\_EKS-alumnos**

**javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ aws ecr describe-repositories --repository-names chat-app-repo --region eu-west-1 --profile 248189943700\_EKS-alumnos**

**{**

**"repositories": [**

**{**

**"repositoryArn": "arn:aws:ecr:eu-west-1:248189943700:repository/chat-app-repo",**

**"registryId": "248189943700",**

**"repositoryName": "chat-app-repo",**

**"repositoryUri": "248189943700.dkr.ecr.eu-west-1.amazonaws.com/chat-app-repo",**

**"createdAt": "2025-02-02T14:35:46.010000+01:00",**

**"imageTagMutability": "MUTABLE",**

**"imageScanningConfiguration": {**

**"scanOnPush": false**

**},**

**"encryptionConfiguration": {**

**"encryptionType": "AES256"**

**}**

**}**

**]**

**}**

**2.2 Construir la imagen de Docker**

Asegúrate de que el archivo **Dockerfil**e esté configurado correctamente para tu aplicación. Luego, construye la imagen con el siguiente comando:

**docker build -t <ecr-repository-uri>:latest .**

**docker build -t 248189943700.dkr.ecr.eu-west-1.amazonaws.com/chat-app-repo:latest .**

Aquí <ecr-repository-uri> es el URI del repositorio ECR que obtuviste en el paso anterior. Un ejemplo de URI sería**: 123456789012.dkr.ecr.us-west-2.amazonaws.com/chat-app.**

**2.3 Autenticación en ECR**

Autentícate con ECR para poder subir la imagen usando el siguiente comando:

**aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <ecr-repository-uri>**

**aws ecr get-login-password --region eu-west-1 --profile 248189943700\_EKS-alumnos | docker login --username AWS --password-stdin 248189943700.dkr.ecr.eu-west-1.amazonaws.com/chat-app-repo**

**2.4 Subir la imagen de Docker a ECR**

Una vez que la imagen esté construida, sube la imagen a ECR:

**docker push <ecr-repository-uri>:latest**

**docker push 248189943700.dkr.ecr.eu-west-1.amazonaws.com/chat-app-repo:latest**

**2.5.Configurar kubectl para tu cluster EKS**:

bash

Copy

aws eks update-kubeconfig --name chat-app-cluster --region eu-west-1

aws eks update-kubeconfig --name chat-app-cluster --region eu-west-1 --profile 248189943700\_EKS-alumnos

**2.6.Verificar la conexión con el cluster**:

bash

Copy

kubectl get nodes

javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-10-0-3-91.eu-west-1.compute.internal Ready <none> 34m v1.31.4-eks-aeac579

ip-10-0-4-81.eu-west-1.compute.internal Ready <none> 34m v1.31.4-eks-aeac579

Para verificar que todo está configurado correctamente:

bash

Copy

*# Verificar que puedes ver tu cluster*

kubectl config current-context

*# Verificar los nodos*

kubectl get nodes

*# Verificar los namespaces*

kubectl get namespaces

app$ kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 38m

javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ kubectl get pods

No resources found in default namespace.

El endpoint de tu cluster EKS es:

https://4D9A9A4C5867125C133EE285248B1611.gr7.eu-west-1.eks.amazonaws.com

Tu ALB DNS es:

Copy

chat-app-cluster-alb-117147314.eu-west-1.elb.amazonaws.com

Tus subnets están distribuidas así:

* Públicas: subnet-08838f765322bda56, subnet-0fcf7804f2ea064ce
* Privadas: subnet-0dca16b6c61733cec, subnet-0eeaa3d2bc52b470f

Para verificar que los servicios están funcionando correctamente después del despliegue:

bash

Copy

*# Verificar pods*

kubectl get pods

*# Verificar servicios*

kubectl get services

*# Verificar despliegues*

kubectl get deployments

Este comando subirá tu imagen Docker a ECR, donde luego podrás utilizarla para crear los pods en **EKS**.

**Paso 3: Kubernetes: Desplegar la aplicación en EKS**

Ahora, vamos a crear los archivos de despliegue de Kubernetes necesarios para lanzar la aplicación en **EKS**.

**3.1 Actualizar el archivo chat-app-deployment.yaml**

Este archivo se encargará de crear el **despliegue de la aplicación** en **EKS**. Asegúrate de que la imagen de Docker en ECR esté correctamente configurada en el archivo **chat-app-deployment.yaml.**

Aquí tienes un ejemplo básico:

apiVersion: apps/v1

kind: Deployment

metadata:

name: chat-app-deployment

labels:

app: chat-app

spec:

replicas: 3 # Número de réplicas para alta disponibilidad

selector:

matchLabels:

app: chat-app

template:

metadata:

labels:

app: chat-app

spec:

containers:

- name: chat-app

image: <ecr-repository-uri>:latest

ports:

- containerPort: 8080

env:

- name: NODE\_ENV

value: "production"

---

apiVersion: v1

kind: Service

metadata:

name: chat-app-service

spec:

selector:

app: chat-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer # Esto crea un ALB o NLB para el acceso público

**3.2 Desplegar en EKS**

Una vez que el archivo esté listo, aplica el despliegue con **kubectl**. Primero, asegúrate de tener configurado el **kubeconfig para tu cluster de EKS**.

**Cd k8s**

**aws eks --region <your-region> update-kubeconfig --name <your-cluster-name>**

**aws eks --region eu-west-1 --profile 248189943700\_EKS-alumnos update-kubeconfig --name chat-app-cluster**

Luego, aplica el despliegue:

**kubectl apply -f chat-app-deployment.yaml**

kubectl apply -f chat-app-service.yaml

Este comando desplegará la aplicación en tu cluster **EKS**, creando un **deployment** con múltiples réplicas y exponiéndolo a través de un **LoadBalancer**.

1. **Verificar el despliegue**:

bash

Copy

*# Verificar el deployment*

kubectl get deployments

kubectl get pods

*# Verificar el service*

kubectl get services

*# Verificar los eventos*

kubectl get events

*# Ver los logs de los pods (reemplaza POD\_NAME con el nombre real del pod)*

kubectl logs POD\_NAME

app/k8s$ kubectl get deployments

**NAME READY UP-TO-DATE AVAILABLE AGE**

**chat-app-deployment 0/3 3 0 2m19s**

**app/k8s$ kubectl get services**

**NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE**

**chat-app-service LoadBalancer 172.20.94.94 a1283580e771f447a9423d15f060c4cf-1754866871.eu-west-1.elb.amazonaws.com 80:30804/TCP 77m**

**kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 124m**

1. **Monitorear el estado del despliegue**:

bash

Copy

*# Ver detalles del deployment*

kubectl describe deployment chat-app

*# Ver detalles del service*

kubectl describe service chat-app-service

1. **Verificar la accesibilidad**:

* Espera a que el LoadBalancer esté listo (puede tomar unos minutos)
* Obtén la URL del LoadBalancer:

bash

Copy

kubectl get service chat-app-service -o wide

1. **Escalar el deployment (opcional)**:

bash

Copy

*# Para aumentar o disminuir el número de réplicas*

kubectl scale deployment chat-app --replicas=3

1. **Troubleshooting (si es necesario)**:

bash

bash

Copy

*# Ver logs detallados*

kubectl logs -f deployment/chat-app

*# Ver descripción detallada de los pods*

kubectl describe pods

*# Ver eventos del cluster*

kubectl get events --sort-by='.lastTimestamp'

Comandos adicionales útiles:

bash

Copy

*# Reiniciar el deployment*

kubectl rollout restart deployment chat-app

*# Ver historial de rollouts*

kubectl rollout history deployment chat-app

*# Eliminar recursos si necesitas empezar de nuevo*

kubectl delete -f chat-app-deployment.yaml

kubectl delete -f chat-app-service.yaml

Recuerda que:

1. La imagen debe existir en ECR antes del despliegue
2. Los nodos deben tener acceso a ECR
3. Los security groups deben permitir el tráfico necesario
4. El ALB debe estar configurado correctamente

¿Necesitas ayuda con algún paso específico o quieres que profundicemos en algún aspecto en particular?

**Paso 4: GitHub Actions: Automatización del CI/CD con chat\_app\_deploy.yaml**

El archivo **chat\_app\_deploy.yaml** en la carpeta **.github/workflows**/ se encargará de automatizar todo el **proceso de CI/CD**.

Aquí tienes el contenido básico de **chat\_app\_deploy.yaml**, que deberías adaptar a tu configuración:

name: **Deploy Chat App to EKS**

on:

push:

branches:

- dev # Cambia esto según la rama que usas

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply -auto-approve

docker:

runs-on: ubuntu-latest

needs: terraform

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Docker

uses: docker/setup-buildx-action@v1

- name: Log in to Amazon ECR

run: |

aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <ecr-repository-uri>

- name: Build Docker image

run: docker build -t <ecr-repository-uri>:latest .

- name: Push Docker image to ECR

run: docker push <ecr-repository-uri>:latest

kubernetes:

runs-on: ubuntu-latest

needs: docker

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Configure AWS CLI

run: |

aws configure set aws\_access\_key\_id ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws configure set aws\_secret\_access\_key ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws configure set default.region <your-region>

- name: Update kubeconfig

run: aws eks --region <your-region> update-kubeconfig --name <your-cluster-name>

- name: Deploy to EKS

run: kubectl apply -f kubernetes/chat-app-deployment.yaml

5. Resumen Pasos de CI/CD:

1. Terraform:
   * Se inicializa Terraform, se aplica la infraestructura.
2. Docker:
   * Construye la imagen de Docker y la sube a Amazon ECR.
3. Kubernetes:
   * Actualiza el kubeconfig para EKS y aplica el archivo chat-app-deployment.yaml para desplegar la aplicación.

Resumen del flujo:

1. Terraform: Prepara la infraestructura (VPC, ECR, EKS, ALB, etc.).
2. Docker: Construye y sube la imagen Docker a ECR.
3. Kubernetes: Despliega la aplicación en EKS.
4. GitHub Actions: Automatiza el flujo CI/CD para la infraestructura y despliegue.

Este es el proceso completo para realizar el despliegue de tu aplicación de chat en tiempo real usando WebSockets en EKS, asegurando que todo esté automatizado y modularizado adecuadamente.

Este es el terraform plan primero:  
  
javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ 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.alb.aws\_lb.this will be created

+ resource "aws\_lb" "this" {

+ arn = (known after apply)

+ arn\_suffix = (known after apply)

+ client\_keep\_alive = 3600

+ desync\_mitigation\_mode = "defensive"

+ dns\_name = (known after apply)

+ drop\_invalid\_header\_fields = false

+ enable\_deletion\_protection = false

+ enable\_http2 = true

+ enable\_tls\_version\_and\_cipher\_suite\_headers = false

+ enable\_waf\_fail\_open = false

+ enable\_xff\_client\_port = false

+ enforce\_security\_group\_inbound\_rules\_on\_private\_link\_traffic = (known after apply)

+ id = (known after apply)

+ idle\_timeout = 60

+ internal = false

+ ip\_address\_type = (known after apply)

+ load\_balancer\_type = "application"

+ name = "chat-app-cluster-alb"

+ name\_prefix = (known after apply)

+ preserve\_host\_header = false

+ security\_groups = (known after apply)

+ subnets = (known after apply)

+ tags\_all = (known after apply)

+ vpc\_id = (known after apply)

+ xff\_header\_processing\_mode = "append"

+ zone\_id = (known after apply)

+ subnet\_mapping (known after apply)

}

# module.alb.aws\_lb\_listener.this will be created

+ resource "aws\_lb\_listener" "this" {

+ arn = (known after apply)

+ id = (known after apply)

+ load\_balancer\_arn = (known after apply)

+ port = 80

+ protocol = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_issuer\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_leaf\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_serial\_number\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_subject\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_mtls\_clientcert\_validity\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_tls\_cipher\_suite\_header\_name = (known after apply)

+ routing\_http\_request\_x\_amzn\_tls\_version\_header\_name = (known after apply)

+ routing\_http\_response\_access\_control\_allow\_credentials\_header\_value = (known after apply)

+ routing\_http\_response\_access\_control\_allow\_headers\_header\_value = (known after apply)

+ routing\_http\_response\_access\_control\_allow\_methods\_header\_value = (known after apply)

+ routing\_http\_response\_access\_control\_allow\_origin\_header\_value = (known after apply)

+ routing\_http\_response\_access\_control\_expose\_headers\_header\_value = (known after apply)

+ routing\_http\_response\_access\_control\_max\_age\_header\_value = (known after apply)

+ routing\_http\_response\_content\_security\_policy\_header\_value = (known after apply)

+ routing\_http\_response\_server\_enabled = (known after apply)

+ routing\_http\_response\_strict\_transport\_security\_header\_value = (known after apply)

+ routing\_http\_response\_x\_content\_type\_options\_header\_value = (known after apply)

+ routing\_http\_response\_x\_frame\_options\_header\_value = (known after apply)

+ ssl\_policy = (known after apply)

+ tags\_all = (known after apply)

+ tcp\_idle\_timeout\_seconds = (known after apply)

+ default\_action {

+ order = (known after apply)

+ type = "fixed-response"

+ fixed\_response {

+ content\_type = "text/plain"

+ message\_body = "OK"

+ status\_code = "200"

}

}

+ mutual\_authentication (known after apply)

}

# module.alb.aws\_lb\_target\_group.this will be created

+ resource "aws\_lb\_target\_group" "this" {

+ arn = (known after apply)

+ arn\_suffix = (known after apply)

+ connection\_termination = (known after apply)

+ deregistration\_delay = "300"

+ id = (known after apply)

+ ip\_address\_type = (known after apply)

+ lambda\_multi\_value\_headers\_enabled = false

+ load\_balancer\_arns = (known after apply)

+ load\_balancing\_algorithm\_type = (known after apply)

+ load\_balancing\_anomaly\_mitigation = (known after apply)

+ load\_balancing\_cross\_zone\_enabled = (known after apply)

+ name = "chat-app-cluster-tg"

+ name\_prefix = (known after apply)

+ port = 80

+ preserve\_client\_ip = (known after apply)

+ protocol = "HTTP"

+ protocol\_version = (known after apply)

+ proxy\_protocol\_v2 = false

+ slow\_start = 0

+ tags\_all = (known after apply)

+ target\_type = "instance"

+ vpc\_id = (known after apply)

+ health\_check {

+ enabled = true

+ healthy\_threshold = 3

+ interval = 30

+ matcher = (known after apply)

+ path = "/"

+ port = "traffic-port"

+ protocol = "HTTP"

+ timeout = 5

+ unhealthy\_threshold = 3

}

+ stickiness (known after apply)

+ target\_failover (known after apply)

+ target\_group\_health (known after apply)

+ target\_health\_state (known after apply)

}

# module.ecr.aws\_ecr\_lifecycle\_policy.this will be created

+ resource "aws\_ecr\_lifecycle\_policy" "this" {

+ id = (known after apply)

+ policy = jsonencode(

{

+ rules = [

+ {

+ action = {

+ type = "expire"

}

+ description = "Expire untagged images after 30 days"

+ rulePriority = 1

+ selection = {

+ countNumber = 30

+ countType = "sinceImagePushed"

+ countUnit = "days"

+ tagStatus = "untagged"

}

},

]

}

)

+ registry\_id = (known after apply)

+ repository = "chat-app-repo"

}

# module.ecr.aws\_ecr\_repository.this will be created

+ resource "aws\_ecr\_repository" "this" {

+ arn = (known after apply)

+ id = (known after apply)

+ image\_tag\_mutability = "MUTABLE"

+ name = "chat-app-repo"

+ registry\_id = (known after apply)

+ repository\_url = (known after apply)

+ tags\_all = (known after apply)

}

# module.eks\_cluster.aws\_eks\_cluster.this will be created

+ resource "aws\_eks\_cluster" "this" {

+ arn = (known after apply)

+ bootstrap\_self\_managed\_addons = true

+ certificate\_authority = (known after apply)

+ cluster\_id = (known after apply)

+ created\_at = (known after apply)

+ endpoint = (known after apply)

+ id = (known after apply)

+ identity = (known after apply)

+ name = "chat-app-cluster"

+ platform\_version = (known after apply)

+ role\_arn = (known after apply)

+ status = (known after apply)

+ tags\_all = (known after apply)

+ version = (known after apply)

+ access\_config (known after apply)

+ kubernetes\_network\_config (known after apply)

+ upgrade\_policy (known after apply)

+ vpc\_config {

+ cluster\_security\_group\_id = (known after apply)

+ endpoint\_private\_access = false

+ endpoint\_public\_access = true

+ public\_access\_cidrs = (known after apply)

+ subnet\_ids = (known after apply)

+ vpc\_id = (known after apply)

}

}

# module.eks\_node\_group.aws\_eks\_node\_group.this will be created

+ resource "aws\_eks\_node\_group" "this" {

+ ami\_type = (known after apply)

+ arn = (known after apply)

+ capacity\_type = (known after apply)

+ cluster\_name = "chat-app-cluster"

+ disk\_size = (known after apply)

+ id = (known after apply)

+ instance\_types = [

+ "t3.small",

]

+ node\_group\_name = "chat-app-nodes"

+ node\_group\_name\_prefix = (known after apply)

+ node\_role\_arn = (known after apply)

+ release\_version = (known after apply)

+ resources = (known after apply)

+ status = (known after apply)

+ subnet\_ids = (known after apply)

+ tags\_all = (known after apply)

+ version = (known after apply)

+ node\_repair\_config (known after apply)

+ scaling\_config {

+ desired\_size = 2

+ max\_size = 3

+ min\_size = 1

}

+ update\_config (known after apply)

}

# module.iam\_roles.aws\_iam\_role.this will be created

+ resource "aws\_iam\_role" "this" {

+ arn = (known after apply)

+ assume\_role\_policy = jsonencode(

{

+ Statement = [

+ {

+ Action = "sts:AssumeRole"

+ Effect = "Allow"

+ Principal = {

+ Service = "eks.amazonaws.com"

}

},

]

+ Version = "2012-10-17"

}

)

+ create\_date = (known after apply)

+ force\_detach\_policies = false

+ id = (known after apply)

+ managed\_policy\_arns = (known after apply)

+ max\_session\_duration = 3600

+ name = "chat-app-eks-role"

+ name\_prefix = (known after apply)

+ path = "/"

+ tags\_all = (known after apply)

+ unique\_id = (known after apply)

+ inline\_policy (known after apply)

}

# module.vpc.aws\_eip.this will be created

+ resource "aws\_eip" "this" {

+ 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)

+ ipam\_pool\_id = (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)

+ public\_ipv4\_pool = (known after apply)

+ tags\_all = (known after apply)

+ vpc = (known after apply)

}

# module.vpc.aws\_internet\_gateway.this will be created

+ resource "aws\_internet\_gateway" "this" {

+ arn = (known after apply)

+ id = (known after apply)

+ owner\_id = (known after apply)

+ tags\_all = (known after apply)

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_nat\_gateway.this will be created

+ resource "aws\_nat\_gateway" "this" {

+ allocation\_id = (known after apply)

+ association\_id = (known after apply)

+ connectivity\_type = "public"

+ id = (known after apply)

+ network\_interface\_id = (known after apply)

+ private\_ip = (known after apply)

+ public\_ip = (known after apply)

+ secondary\_private\_ip\_address\_count = (known after apply)

+ secondary\_private\_ip\_addresses = (known after apply)

+ subnet\_id = (known after apply)

+ tags\_all = (known after apply)

}

# module.vpc.aws\_route\_table.private will be created

+ resource "aws\_route\_table" "private" {

+ arn = (known after apply)

+ id = (known after apply)

+ owner\_id = (known after apply)

+ propagating\_vgws = (known after apply)

+ route = [

+ {

+ cidr\_block = "0.0.0.0/0"

+ nat\_gateway\_id = (known after apply)

# (11 unchanged attributes hidden)

},

]

+ tags\_all = (known after apply)

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_route\_table.public will be created

+ resource "aws\_route\_table" "public" {

+ arn = (known after apply)

+ id = (known after apply)

+ owner\_id = (known after apply)

+ propagating\_vgws = (known after apply)

+ route = [

+ {

+ cidr\_block = "0.0.0.0/0"

+ gateway\_id = (known after apply)

# (11 unchanged attributes hidden)

},

]

+ tags\_all = (known after apply)

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_route\_table\_association.private[0] will be created

+ resource "aws\_route\_table\_association" "private" {

+ id = (known after apply)

+ route\_table\_id = (known after apply)

+ subnet\_id = (known after apply)

}

# module.vpc.aws\_route\_table\_association.private[1] will be created

+ resource "aws\_route\_table\_association" "private" {

+ id = (known after apply)

+ route\_table\_id = (known after apply)

+ subnet\_id = (known after apply)

}

# module.vpc.aws\_route\_table\_association.public[0] will be created

+ resource "aws\_route\_table\_association" "public" {

+ id = (known after apply)

+ route\_table\_id = (known after apply)

+ subnet\_id = (known after apply)

}

# module.vpc.aws\_route\_table\_association.public[1] will be created

+ resource "aws\_route\_table\_association" "public" {

+ id = (known after apply)

+ route\_table\_id = (known after apply)

+ subnet\_id = (known after apply)

}

# module.vpc.aws\_security\_group.this will be created

+ resource "aws\_security\_group" "this" {

+ arn = (known after apply)

+ description = "Security group for EKS and ALB"

+ egress = [

+ {

+ cidr\_blocks = [

+ "0.0.0.0/0",

]

+ from\_port = 0

+ ipv6\_cidr\_blocks = []

+ prefix\_list\_ids = []

+ protocol = "-1"

+ security\_groups = []

+ self = false

+ to\_port = 0

# (1 unchanged attribute hidden)

},

]

+ id = (known after apply)

+ ingress = [

+ {

+ cidr\_blocks = [

+ "0.0.0.0/0",

]

+ from\_port = 80

+ ipv6\_cidr\_blocks = []

+ prefix\_list\_ids = []

+ protocol = "tcp"

+ security\_groups = []

+ self = false

+ to\_port = 80

# (1 unchanged attribute hidden)

},

]

+ name = "chat-app-cluster-sg"

+ name\_prefix = (known after apply)

+ owner\_id = (known after apply)

+ revoke\_rules\_on\_delete = false

+ tags = {

+ "Name" = "chat-app-cluster-sg"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-sg"

}

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_subnet.private[0] will be created

+ resource "aws\_subnet" "private" {

+ arn = (known after apply)

+ assign\_ipv6\_address\_on\_creation = false

+ availability\_zone = "eu-west-1a"

+ availability\_zone\_id = (known after apply)

+ cidr\_block = "10.0.3.0/24"

+ enable\_dns64 = false

+ enable\_resource\_name\_dns\_a\_record\_on\_launch = false

+ enable\_resource\_name\_dns\_aaaa\_record\_on\_launch = false

+ id = (known after apply)

+ ipv6\_cidr\_block\_association\_id = (known after apply)

+ ipv6\_native = false

+ map\_public\_ip\_on\_launch = false

+ owner\_id = (known after apply)

+ private\_dns\_hostname\_type\_on\_launch = (known after apply)

+ tags = {

+ "Name" = "chat-app-cluster-private-0"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-private-0"

}

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_subnet.private[1] will be created

+ resource "aws\_subnet" "private" {

+ arn = (known after apply)

+ assign\_ipv6\_address\_on\_creation = false

+ availability\_zone = "eu-west-1b"

+ availability\_zone\_id = (known after apply)

+ cidr\_block = "10.0.4.0/24"

+ enable\_dns64 = false

+ enable\_resource\_name\_dns\_a\_record\_on\_launch = false

+ enable\_resource\_name\_dns\_aaaa\_record\_on\_launch = false

+ id = (known after apply)

+ ipv6\_cidr\_block\_association\_id = (known after apply)

+ ipv6\_native = false

+ map\_public\_ip\_on\_launch = false

+ owner\_id = (known after apply)

+ private\_dns\_hostname\_type\_on\_launch = (known after apply)

+ tags = {

+ "Name" = "chat-app-cluster-private-1"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-private-1"

}

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_subnet.public[0] will be created

+ resource "aws\_subnet" "public" {

+ arn = (known after apply)

+ assign\_ipv6\_address\_on\_creation = false

+ availability\_zone = "eu-west-1a"

+ availability\_zone\_id = (known after apply)

+ cidr\_block = "10.0.1.0/24"

+ enable\_dns64 = false

+ enable\_resource\_name\_dns\_a\_record\_on\_launch = false

+ enable\_resource\_name\_dns\_aaaa\_record\_on\_launch = false

+ id = (known after apply)

+ ipv6\_cidr\_block\_association\_id = (known after apply)

+ ipv6\_native = false

+ map\_public\_ip\_on\_launch = true

+ owner\_id = (known after apply)

+ private\_dns\_hostname\_type\_on\_launch = (known after apply)

+ tags = {

+ "Name" = "chat-app-cluster-public-0"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-public-0"

}

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_subnet.public[1] will be created

+ resource "aws\_subnet" "public" {

+ arn = (known after apply)

+ assign\_ipv6\_address\_on\_creation = false

+ availability\_zone = "eu-west-1b"

+ availability\_zone\_id = (known after apply)

+ cidr\_block = "10.0.2.0/24"

+ enable\_dns64 = false

+ enable\_resource\_name\_dns\_a\_record\_on\_launch = false

+ enable\_resource\_name\_dns\_aaaa\_record\_on\_launch = false

+ id = (known after apply)

+ ipv6\_cidr\_block\_association\_id = (known after apply)

+ ipv6\_native = false

+ map\_public\_ip\_on\_launch = true

+ owner\_id = (known after apply)

+ private\_dns\_hostname\_type\_on\_launch = (known after apply)

+ tags = {

+ "Name" = "chat-app-cluster-public-1"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-public-1"

}

+ vpc\_id = (known after apply)

}

# module.vpc.aws\_vpc.this will be created

+ resource "aws\_vpc" "this" {

+ arn = (known after apply)

+ cidr\_block = "10.0.0.0/16"

+ default\_network\_acl\_id = (known after apply)

+ default\_route\_table\_id = (known after apply)

+ default\_security\_group\_id = (known after apply)

+ dhcp\_options\_id = (known after apply)

+ enable\_dns\_hostnames = true

+ enable\_dns\_support = true

+ enable\_network\_address\_usage\_metrics = (known after apply)

+ id = (known after apply)

+ instance\_tenancy = "default"

+ ipv6\_association\_id = (known after apply)

+ ipv6\_cidr\_block = (known after apply)

+ ipv6\_cidr\_block\_network\_border\_group = (known after apply)

+ main\_route\_table\_id = (known after apply)

+ owner\_id = (known after apply)

+ tags = {

+ "Name" = "chat-app-cluster-vpc"

}

+ tags\_all = {

+ "Name" = "chat-app-cluster-vpc"

}

}

Plan: 23 to add, 0 to change, 0 to destroy.

Changes to Outputs:

+ alb\_dns\_name = (known after apply)

+ eks\_cluster\_endpoint = (known after apply)

+ eks\_cluster\_name = "chat-app-cluster"

+ private\_subnet\_ids = [

+ (known after apply),

+ (known after apply),

]

+ public\_subnet\_ids = [

+ (known after apply),

+ (known after apply),

]

+ security\_group\_id = (known after apply)

+ vpc\_id = (known after apply)

─────────────────────────────────────────────────────────────────────

¡Claro! Estoy revisando la primera parte del terraform plan que has compartido. Aquí hay un resumen de lo que veo y algunos puntos a considerar:

**Resumen de la Primera Parte del Plan**

1. **Creación de Recursos**: Todos los recursos que se enumeran están marcados para ser creados (+ create), lo que significa que no hay conflictos o problemas evidentes en esta parte del plan.
2. **AWS Load Balancer (ALB)**:
   * El recurso aws\_lb se creará correctamente y tiene configuraciones adecuadas, como idle\_timeout, security\_groups, y subnets.
   * El atributo dns\_name se generará después de la aplicación, lo cual es normal.
3. **Listener y Target Group**:
   * El listener y el target group también están configurados correctamente y se crearán sin problemas.
4. **ECR (Elastic Container Registry)**:
   * Se crearán tanto la política de ciclo de vida como el repositorio de ECR, lo cual es necesario para almacenar las imágenes de tu aplicación.
5. **EKS (Elastic Kubernetes Service)**:
   * El clúster EKS y el grupo de nodos están configurados para ser creados. Asegúrate de que los roles de IAM estén correctamente configurados para permitir que EKS funcione.
6. **IAM Roles**:
   * El rol de IAM para EKS se creará correctamente, lo cual es esencial para que el clúster tenga los permisos necesarios.

**Consideraciones**

* **Revisar la Configuración de Seguridad**: Asegúrate de que los grupos de seguridad y las subredes estén configurados correctamente para permitir el tráfico necesario entre los componentes (ALB, EKS, etc.).
* **Verificar las Políticas de IAM**: Asegúrate de que las políticas de IAM asociadas al rol de EKS tengan los permisos necesarios para interactuar con otros servicios de AWS.
* **Configuración de Salud**: En el target group, verifica que la configuración de salud esté bien definida para que EKS pueda gestionar correctamente las instancias.

**Esperando la Segunda Parte**

Gracias por compartir la segunda parte del terraform plan. Aquí hay un resumen de lo que veo y algunas consideraciones para asegurarte de que la aplicación se despliegue correctamente.

**Resumen de la Segunda Parte del Plan**

1. **Creación de Recursos VPC**:
   * Se crearán varios recursos relacionados con la VPC, incluidos Elastic IPs, gateways de Internet, NAT gateways, tablas de rutas, subredes (tanto públicas como privadas) y el VPC en sí.
   * Todo parece estar configurado correctamente para permitir la conectividad entre los recursos.
2. **Grupos de Seguridad**:
   * Se creará un grupo de seguridad que permite el tráfico HTTP (puerto 80) desde cualquier dirección IP (0.0.0.0/0). Esto es adecuado para un ALB, pero asegúrate de que sea lo que deseas en términos de seguridad.
3. **Subredes**:
   * Las subredes públicas están configuradas para asignar direcciones IP públicas, lo cual es correcto para los recursos que necesitan ser accesibles desde Internet.
   * Las subredes privadas no asignan direcciones IP públicas, lo cual es adecuado para los nodos de EKS.
4. **Plan de Ejecución**:
   * El plan indica que se agregarán 23 recursos, lo que significa que no hay conflictos ni problemas evidentes en la configuración actual.

**Consideraciones**

* **Seguridad**:
  + Revisa las reglas del grupo de seguridad. Permitir tráfico desde 0.0.0.0/0 puede ser un riesgo de seguridad. Considera restringir el acceso a direcciones IP específicas si es posible.
* **Configuración de Rutas**:
  + Asegúrate de que las tablas de rutas estén correctamente asociadas con las subredes correspondientes. Las subredes públicas deben tener una ruta hacia el gateway de Internet, y las subredes privadas deben tener una ruta hacia el NAT gateway.
* **Verificación de Recursos**:
  + Asegúrate de que todos los recursos necesarios para tu aplicación estén incluidos en el plan. Por ejemplo, verifica que los nodos de EKS y cualquier otro recurso que necesites estén correctamente configurados.
* **Outputs**:
  + Los outputs que se generarán después de la aplicación (como alb\_dns\_name, eks\_cluster\_endpoint, etc.) son útiles para acceder a los recursos una vez que se hayan creado.

**Conclusión**

El plan de Terraform parece estar bien configurado y no muestra errores evidentes. Si has revisado las consideraciones de seguridad y conectividad, deberías estar listo para proceder con terraform apply.

Si tienes alguna pregunta específica o si hay algo más que te gustaría revisar, no dudes en decírmelo. ¡Estoy aquí para ayudarte!

**Listar VPC disponibles:**

javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ aws ec2 describe-vpcs --query 'Vpcs[\*].[VpcId,Tags[?Key==`Name`].Value|[0],CidrBlock,State]' --output table --profile 248189943700\_EKS-alumnos

-------------------------------------------------------------------------------

| DescribeVpcs |

+------------------------+----------------------+----------------+------------+

| vpc-04908424cd9342959 | None | 10.0.0.0/16 | available |

| vpc-0f89d9c4fccb3dace | None | 10.0.0.0/16 | available |

| vpc-0aad847febf809cf4 | None | 172.31.0.0/16 | available |

| vpc-06ea1d353c5b299d2 | react-app-2048-vpc | 10.0.0.0/16 | available |

| vpc-085866b18ccdf1ae8 | None | 10.0.0.0/16 | available |

+------------------------+----------------------+----------------+------------+

javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app$ aws ec2 delete-vpc --vpc-id vpc-06ea1d353c5b299d2 --profile 248189943700\_EKS-alumnos

An error occurred (DependencyViolation) when calling the DeleteVpc operation: The vpc 'vpc-06ea1d353c5b299d2' has dependencies and cannot be deleted.

TODOS LOS OUTPUTS DE MI INFRAESTRUCTURA CHAT-APP:  
  
Outputs:

**alb\_dns\_name** = "chat-app-cluster-alb-117147314.eu-west-1.elb.amazonaws.com"

**eks\_cluster\_endpoint** = "https://4D9A9A4C5867125C133EE285248B1611.gr7.eu-west-1.eks.amazonaws.com"

**eks\_cluster\_name** = "chat-app-cluster"

**private\_subnet\_ids** = [

"subnet-0dca16b6c61733cec",

"subnet-0eeaa3d2bc52b470f",

]

**public\_subnet\_ids** = [

"subnet-08838f765322bda56",

"subnet-0fcf7804f2ea064ce",

]

**security\_group\_id** = "sg-02764c8716b871773"

**vpc\_id** = "vpc-0b690e66858f12ecc"

**repositoryUri": "248189943700.dkr.ecr.eu-west-1.amazonaws.com/chat-app-repo**

app/k8s$ kubectl create secret docker-registry ecr-secret --docker-server=248189943700.dkr.ecr.eu-west-1.amazonaws.com --docker-username=AWS --docker-password=$(aws ecr get-login-password --region eu-west-1 --profile 248189943700\_EKS-alumnos) --docker-email=francisco.guerra@campusdual.com

secret/ecr-secret created

javier@DESKTOP-JV8P0GR:/mnt/e/CURSO DEVOP&CLOUD\_CAMPUSDUAL\_CLUSTERTIC/2025-Kubernetes-Terraform/10 DECIMA CLASE\_ECS\_EKS\_PFM3\_03022025/cloud-resources1/chat\_app/k8s$ kubectl get secrets

NAME TYPE DATA AGE

ecr-secret kubernetes.io/dockerconfigjson 1 45s

VARIABLES DE ENTORNO PARA GITHUB ACTIONS:  
  
#aws eks update-kubeconfig --name ${{ secrets.EKS\_CLUSTER\_NAME }} --region eu-west-1

kubectl config set-cluster eks --server=${{ secrets.EKS\_CLUSTER\_ENDPOINT }} --certificate-authority=${{ secrets.EKS\_CA\_CERTIFICATE }}

env:

        KUBECONFIG: ${{ secrets.KUBECONFIG }}

kubectl apply -f k8s/chat-app-deployment.yaml

        kubectl apply -f k8s/chat-app-service.yaml

**ENLACE PARA VER LA APLICACIÓN DESPLEGADA:**

[a87de81cbb66f451b88520bcc17fc533-2068815550.eu-west-1.elb.amazonaws.com](http://a87de81cbb66f451b88520bcc17fc533-2068815550.eu-west-1.elb.amazonaws.com/)

**Pasos para Configurar Terraform con S3 como Backend**

**1️ Crear el bucket de S3 para almacenar el terraform.tfstate**

Ejecuta en la terminal de AWS CLI:

aws s3api create-bucket --bucket mi-terraform-tfstate --region eu-west-1 --create-bucket-configuration LocationConstraint=eu-west-1

Si necesitas habilitar versionado (para mantener cambios en el estado):

aws s3api put-bucket-versioning --bucket mi-terraform-tfstate --versioning-configuration

**aws s3api create-bucket --bucket mi-terraform-tfstate-248189943700 --region eu-west-1 --create-bucket-configuration LocationConstraint=eu-west-1 --profile 248189943700\_EKS-alumnos**

aws s3 ls --profile 248189943700\_EKS-alumnos

**http://mi-terraform-tfstate-248189943700.s3.amazonaws.com/**

**2️Crear un terraform.tf que use S3 como backend**

En chat\_app/terraform/ (o el directorio correcto), crea o modifica el archivo backend.tf:

hcl

CopiarEditar

terraform {

backend "s3" {

bucket = "mi-terraform-tfstate"

key = "terraform/state/terraform.tfstate"

region = "eu-west-1"

encrypt = true

#dynamodb\_table = "terraform-lock"

}

}

Si trabajo solo comento linea : dynamodb\_table = "terraform-lock"

Esto almacena el estado en s3://mi-terraform-tfstate/terraform/state/terraform.tfstate.

**🔒 Bloqueo con DynamoDB (opcional, recomendado en entornos colaborativos)**  
Para evitar conflictos en ejecuciones simultáneas, **crea una tabla DynamoDB**:

sh

CopiarEditar

aws dynamodb create-table \

--table-name terraform-lock \

--attribute-definitions AttributeName=LockID,AttributeType=S \

--key-schema AttributeName=LockID,KeyType=HASH \

--billing-mode PAY\_PER\_REQUEST

**3.Actualizar GitHub Actions para usar S3**

Modifica chat\_app\_deploy.yaml para usar el backend de S3:

yaml

CopiarEditar

- name: Initialize Terraform

working-directory: chat\_app/terraform

run: terraform init -backend-config="bucket=mi-terraform-tfstate" -backend-config="key=terraform/state/terraform.tfstate" -backend-config="region=eu-west-1"

- name: Validate Terraform configuration

working-directory: chat\_app/terraform

run: terraform validate

- name: Plan Terraform

working-directory: chat\_app/terraform

run: terraform plan

- name: Apply Terraform

working-directory: chat\_app/terraform

run: terraform apply -auto-approve

**🔹 Terraform ahora guardará el estado en S3 en lugar de localmente.**  
Esto evitará el error **"No configuration files"**, ya que el estado ya existirá en S3.

**📌 Beneficios de Usar S3 como Backend**

✅ **Evita pérdida del terraform.tfstate** en cada ejecución.  
✅ **Permite ejecutar Terraform desde múltiples máquinas/GitHub Actions sin conflictos.**  
✅ **Facilita la colaboración en equipo.**  
✅ **Usando DynamoDB, evita que dos personas modifiquen la infraestructura al mismo tiempo.**

**📌 Próximos pasos**

1️⃣ Ejecuta terraform init nuevamente para migrar el estado a S3.  
2️⃣ Confirma que terraform apply funciona correctamente.  
3️⃣ Ejecuta el workflow en GitHub Actions.

🔹 ¡Con esto, Terraform funcionará correctamente en CI/CD! 🚀

4o

**WEBEX 04/02/2025**

**CI CD & GITHUB ACTIONS**

**Ejercicio 1 - Crear tu Primer Workflow**

Archivo: .github/workflows/ci.yml

name: CI Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v3

- name: Print Message

run: echo "Hello, CI/CD!"

0s

[1](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:1)Current runner version: '2.322.0'

[2](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:2)Operating System

[3](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:3) Ubuntu

[4](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:4) 24.04.1

[5](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:5) LTS

[6](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:7)Runner Image

[7](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:8) Image: ubuntu-24.04

[8](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:9) Version: 20250126.1.0

[9](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:10) Included Software: <https://github.com/actions/runner-images/blob/ubuntu24/20250126.1/images/ubuntu/Ubuntu2404-Readme.md>

[10](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:11) Image Release: <https://github.com/actions/runner-images/releases/tag/ubuntu24%2F20250126.1>

[11](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:13)Runner Image Provisioner

[12](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:14) 2.0.422.1

[13](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:16)GITHUB\_TOKEN Permissions

[14](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:17) Contents: read

[15](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:18) Metadata: read

[16](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:19) Packages: read

[17](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:21)Secret source: Actions

[18](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:22)Prepare workflow directory

[19](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:23)Prepare all required actions

[20](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:24)Getting action download info

[21](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:25)Download action repository 'actions/checkout@v3' (SHA:f43a0e5ff2bd294095638e18286ca9a3d1956744)

[22](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:1:26)Complete job name: build

Checkout Code

1s

[1](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:1)Run actions/checkout@v3

[14](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:15)Syncing repository: gruposdosdual/cloud-resources1

[15](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:16)Getting Git version info

[19](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:21)Temporarily overriding HOME='/home/runner/work/\_temp/b458b10d-d834-4bee-a010-95785d0f58f0' before making global git config changes

[20](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:22)Adding repository directory to the temporary git global config as a safe directory

[21](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:23)/usr/bin/git config --global --add safe.directory /home/runner/work/cloud-resources1/cloud-resources1

[22](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:24)Deleting the contents of '/home/runner/work/cloud-resources1/cloud-resources1'

[23](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:25)Initializing the repository

[24](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:26) /usr/bin/git init /home/runner/work/cloud-resources1/cloud-resources1

[25](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:27) hint: Using 'master' as the name for the initial branch. This default branch name

[26](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:28) hint: is subject to change. To configure the initial branch name to use in all

[27](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:29) hint: of your new repositories, which will suppress this warning, call:

[28](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:30) hint:

[29](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:31) hint: git config --global init.defaultBranch <name>

[30](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:32) hint:

[31](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:33) hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and

[32](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:34) hint: 'development'. The just-created branch can be renamed via this command:

[33](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:35) hint:

[34](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:36) hint: git branch -m <name>

[35](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:37) Initialized empty Git repository in /home/runner/work/cloud-resources1/cloud-resources1/.git/

[36](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:38) /usr/bin/git remote add origin <https://github.com/gruposdosdual/cloud-resources1>

[37](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:40)Disabling automatic garbage collection

[38](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:41) /usr/bin/git config --local gc.auto 0

[39](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:43)Setting up auth

[40](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:44) /usr/bin/git config --local --name-only --get-regexp core\.sshCommand

[41](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:45) /usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all 'core.sshCommand' || :"

[42](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:46) /usr/bin/git config --local --name-only --get-regexp http\.https\:\/\/github\.com\/\.extraheader

[43](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:47) /usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https\:\/\/github\.com\/\.extraheader' && git config --local --unset-all 'http.<https://github.com/.extraheader>' || :"

[44](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:48) /usr/bin/git config --local http.<https://github.com/.extraheader> AUTHORIZATION: basic \*\*\*

[45](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:50)Fetching the repository

[46](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:51) /usr/bin/git -c protocol.version=2 fetch --no-tags --prune --progress --no-recurse-submodules --depth=1 origin +9422030458be709987e0b6c8795b61872854109b:refs/remotes/origin/dev

[47](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:52) remote: Enumerating objects: 229, done.

Print Message

0s

Post Checkout Code

0s

Complete job

0s

Resolving deltas: 96% (26/27)

[382](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:387) Resolving deltas: 100% (27/27)

[383](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:388) Resolving deltas: 100% (27/27), done.

[384](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:389) From <https://github.com/gruposdosdual/cloud-resources1>

[385](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:390) \* [new ref] 9422030458be709987e0b6c8795b61872854109b -> origin/dev

[386](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:392)Determining the checkout info

[387](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:394)Checking out the ref

[388](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:395) /usr/bin/git checkout --progress --force -B dev refs/remotes/origin/dev

[389](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:396) Switched to a new branch 'dev'

[390](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:397) branch 'dev' set up to track 'origin/dev'.

[391](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:399)/usr/bin/git log -1 --format='%H'

[392](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:2:400)'9422030458be709987e0b6c8795b61872854109b'

Print Message

0s

[1](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:3:1)Run echo "Hello, CI/CD!"

[2](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:3:2) echo "Hello, CI/CD!"

[3](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:3:3) shell: /usr/bin/bash -e {0}

[4](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:3:5)Hello, CI/CD!

Post Checkout Code

0s

[1](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:1)Post job cleanup.

[2](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:2)/usr/bin/git version

[3](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:3)git version 2.48.1

[4](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:4)Temporarily overriding HOME='/home/runner/work/\_temp/9460aef4-05b3-4d2e-ad90-3f672eeb7121' before making global git config changes

[5](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:5)Adding repository directory to the temporary git global config as a safe directory

[6](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:6)/usr/bin/git config --global --add safe.directory /home/runner/work/cloud-resources1/cloud-resources1

[7](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:7)/usr/bin/git config --local --name-only --get-regexp core\.sshCommand

[8](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:8)/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all 'core.sshCommand' || :"

[9](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:9)/usr/bin/git config --local --name-only --get-regexp http\.https\:\/\/github\.com\/\.extraheader

[10](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:10)http.<https://github.com/.extraheader>

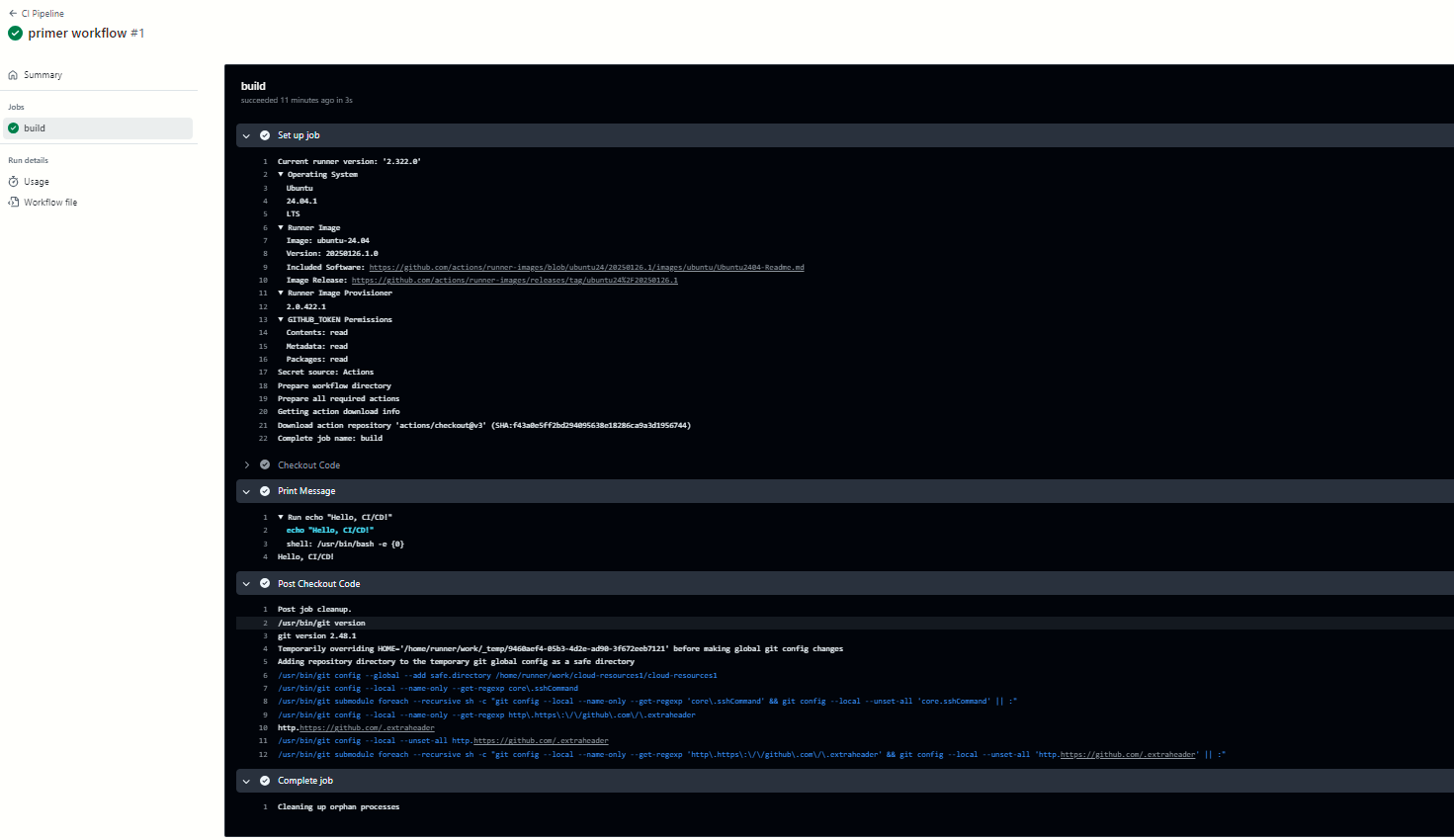
[11](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:11)/usr/bin/git config --local --unset-all http.<https://github.com/.extraheader>

[12](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:6:12)/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https\:\/\/github\.com\/\.extraheader' && git config --local --unset-all 'http.<https://github.com/.extraheader>' || :"

Complete job

0s

[1](https://github.com/gruposdosdual/cloud-resources1/actions/runs/13138665722/job/36660146973#step:7:1)Cleaning up orphan processes



**Ejercicio 2 - Validar Archivos YAML**

Modificar el workflow para incluir validación con yamllint.

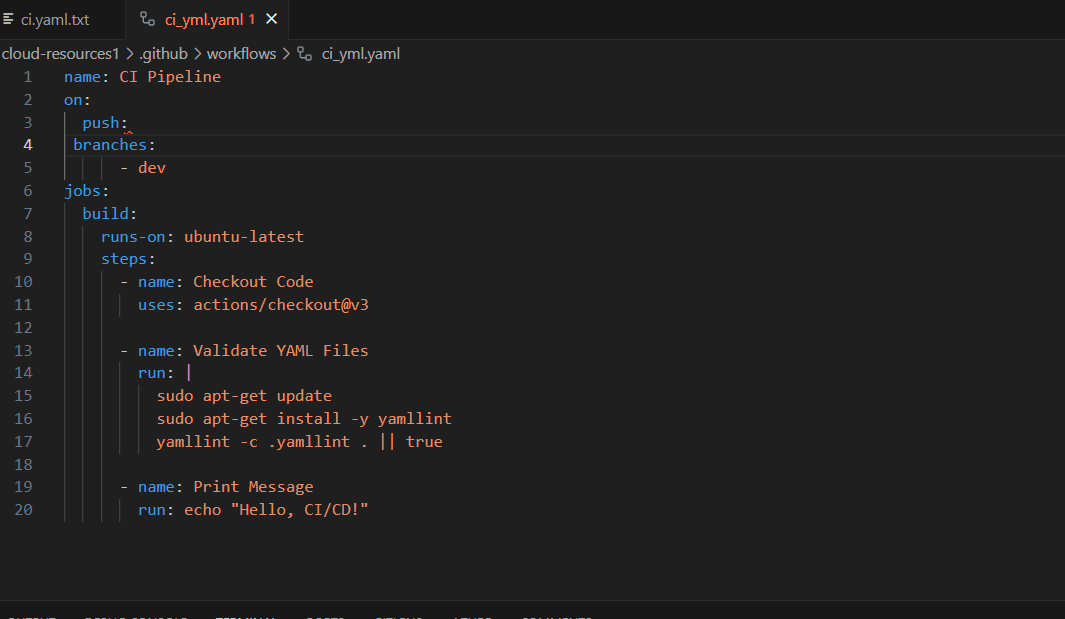
- name: Validate YAML Files

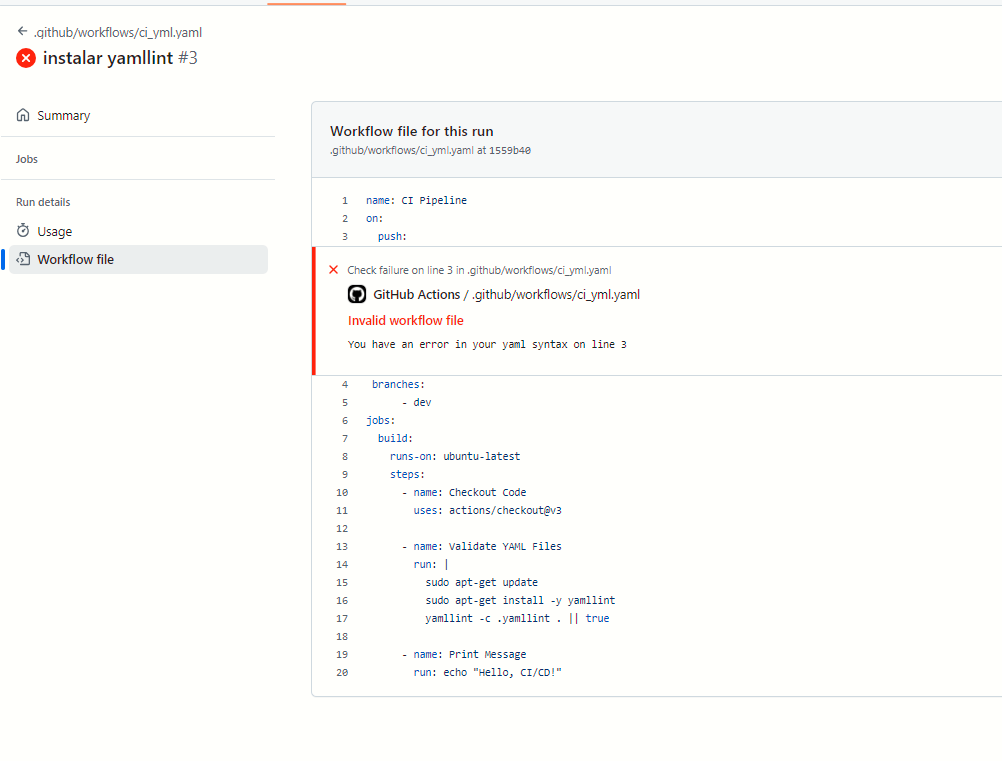
run: |

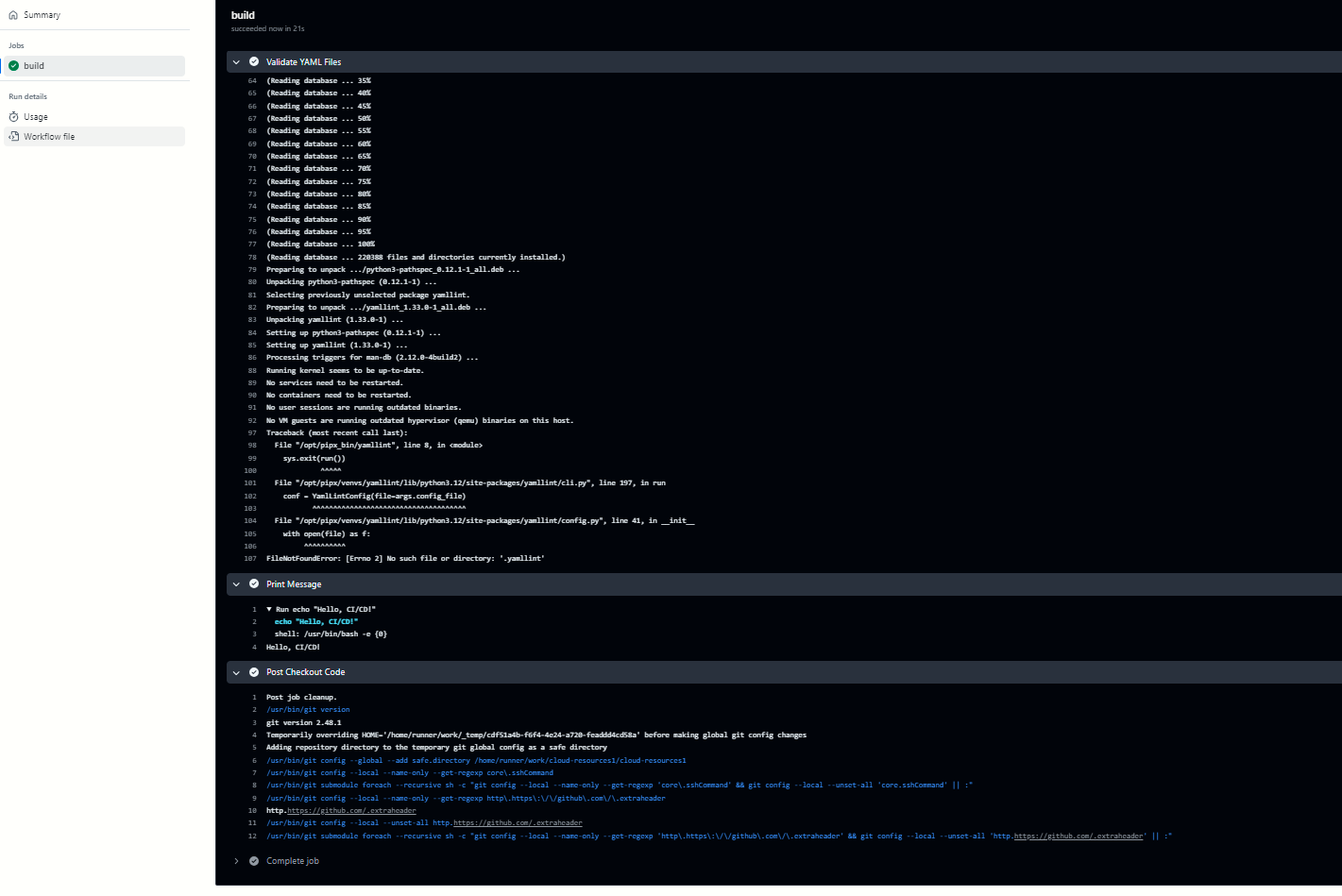
sudo apt-get install yamllint

yamllint .

- Introducir un error en YAML, probar y corregirlo.







**Ejercicio 3 - Badge de Estado**

Generar y agregar un badge de estado al README.md.

- Confirmar la visualización en el repositorio.

**CODIGO NO VALIDO Y COMPLEJO**

name: CI Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Build and Test

run: |

echo "Running tests..."

# Simula un test

- name: Create Status Badge

if: always()

uses: actions/github-script@v6

with:

github-token: ${{secrets.GITHUB\_TOKEN}}

script: |

const fs = require('fs');

const workflowName = 'CI Pipeline';

const badgeUrl = `https://github.com/${context.repo.owner}/${context.repo.repo}/actions/workflows/ci.yml/badge.svg`;

const badgeMarkdown = `![${workflowName}](${badgeUrl})`;

let readme = fs.readFileSync('README.md', 'utf8');

if (!readme.includes(badgeMarkdown)) {

readme = badgeMarkdown + '\n\n' + readme;

fs.writeFileSync('README.md', readme);

}

**CODIGO VALIDO Y SENCILLO:**

name: CI Pipeline

on:

  push:

    branches:

      - dev

jobs:

  build:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v3

      - name: Add README Badge

        run: |

          if [ ! -f README.md ]; then

            echo "# Mi Proyecto" > README.md

          fi

          BADGE\_URL="https://github.com/${{ github.repository }}/actions/workflows/ci.yml/badge.svg"

          BADGE\_LINE="![CI Pipeline Status]($BADGE\_URL)"

          # Añadir badge al inicio del README si no existe

          if ! grep -q "$BADGE\_LINE" README.md; then

            sed -i "1i $BADGE\_LINE\n" README.md

          fi

      - name: Commit Changes

        run: |

          git config --local user.email "action@github.com"

          git config --local user.name "GitHub Action"

          git add README.md

          git commit -m "Add workflow status badge" || echo "No changes to commit"

          git push

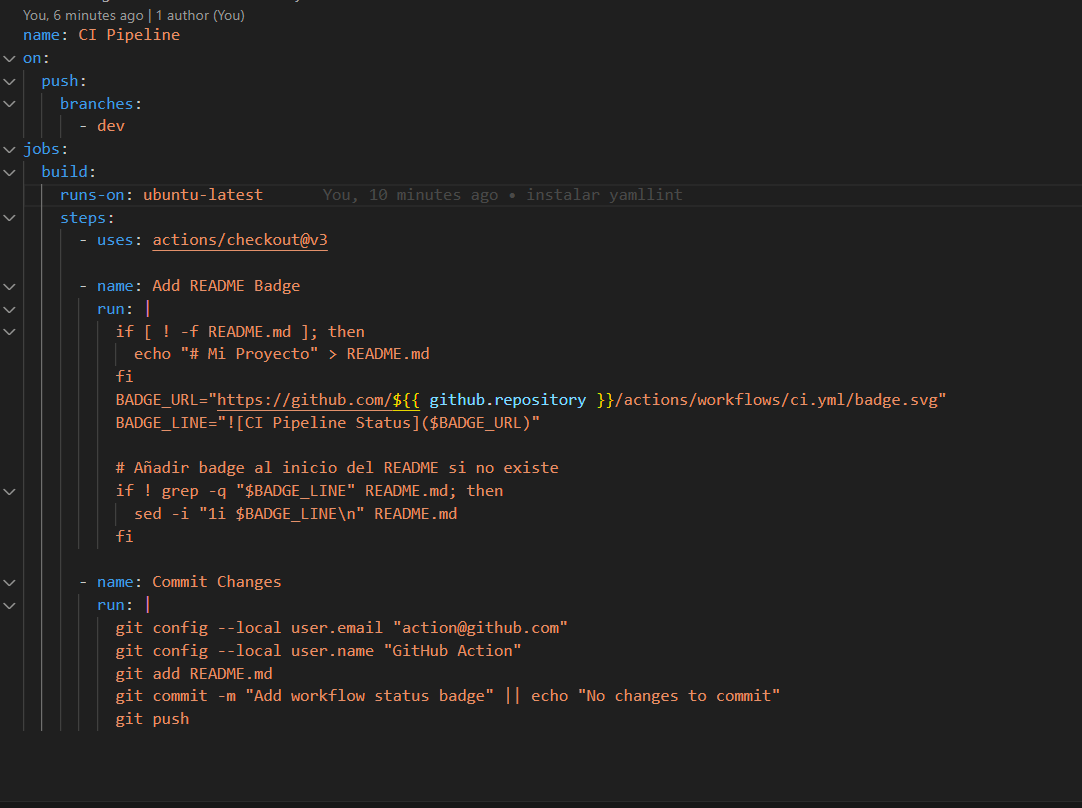
ANTES DEBES AUTORIZAR READ AND WRITES EN GITHUB:

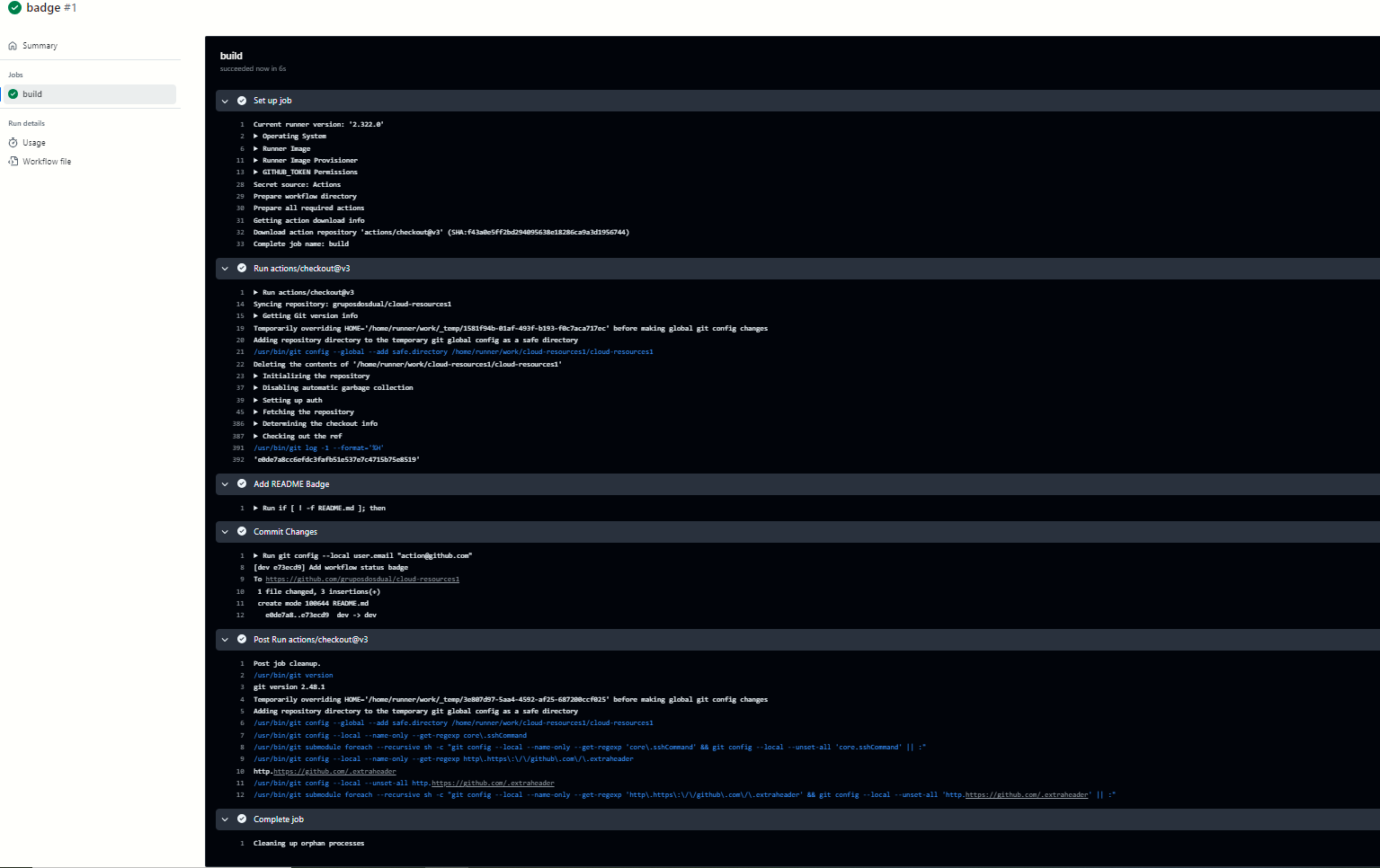
El error indica que el GitHub Actions bot no tiene permisos para hacer push al repositorio. Necesitas configurar los permisos correctos:

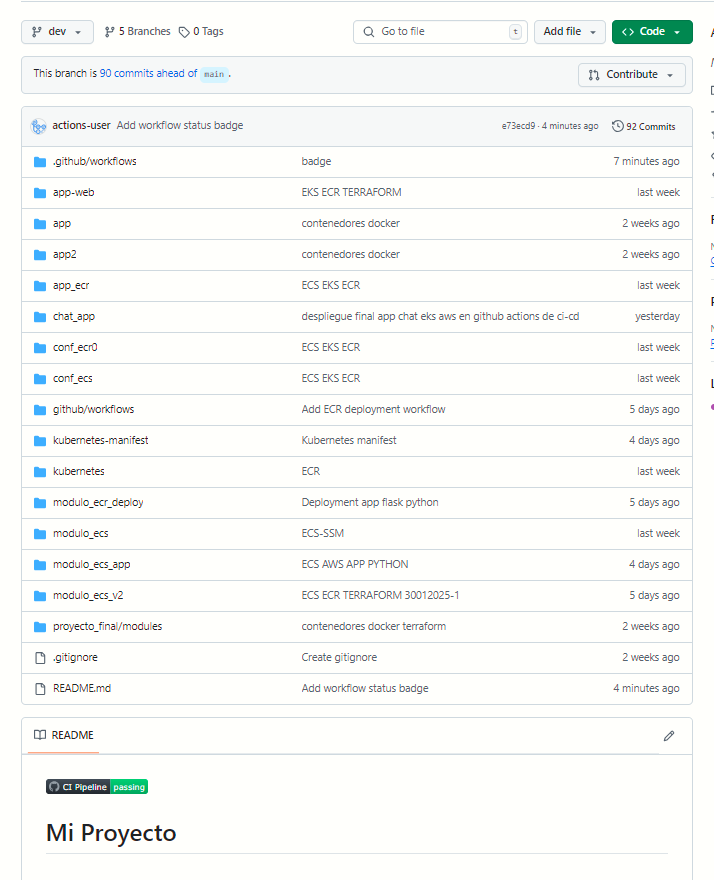
Solución:

1. Ve a tu repositorio en GitHub
2. Configuraciones (Settings)
3. Actions > General
4. Workflow permissions
5. Selecciona "Read and write permissions"
6. Guarda los cambios

Actualiza también el workflow:







**EJERCICIO 4 EN EQUIPOS:**Configuración de Pipeline Completo

1. Crear archivo .github/workflows/full-pipeline.yml:

yaml

Copy

name: Full CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build-and-test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v3

with:

python-version: '3.9'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install pytest

pip install -r requirements.txt

- name: Run Tests

run: |

pytest tests/ --junitxml=test-results/junit.xml

- name: Upload Test Results

uses: actions/upload-artifact@v3

with:

name: test-results

path: test-results/junit.xml

- name: Generate Coverage Report

run: |

pip install coverage

coverage run -m pytest

coverage xml -o coverage-report/coverage.xml

- name: Upload Coverage Report

uses: actions/upload-artifact@v3

with:

name: coverage-report

path: coverage-report/coverage.xml

1. Crear estructura de proyecto:

Copy

.

├── .github/

│ └── workflows/

│ └── full-pipeline.yml

├── requirements.txt

├── src/

│ └── ejemplo.py

└── tests/

└── test\_ejemplo.py

1. Contenido de requirements.txt:

Copy

pytest

1. Código de ejemplo en src/ejemplo.py:

python

Copy

def suma(a, b):

return a + b

def resta(a, b):

return a - b

1. Pruebas en tests/test\_ejemplo.py:

python

Copy

from src.ejemplo import suma, resta

def test\_suma():

assert suma(2, 3) == 5

def test\_resta():

assert resta(5, 3) == 2

Pasos en GitHub:

1. Crear repositorio
2. Subir archivos
3. Ir a Actions
4. Verificar ejecución del workflow

Puntos clave:

* Instala dependencias
* Ejecuta pruebas
* Genera informe de tests
* Genera informe de cobertura
* Almacena artefactos

 CopyRetry

Claude does not have the ability to run the code it generates yet.

**EJERCICIO GRUPAL:**

[toni.publico@gmail.com](mailto:toni.publico@gmail.com) PROFESOR ANTONIO VEGA

**Crear un proyecto en el front que mas os guste, node, react,...  
  
Crear el ecr para este proyecto  
  
Crear el workflow de app  
  
Crear el workflow de iac  
  
Crear llamada al modulo por TF**

**1.- Crear definicion TF  
  
2.- Crea Worflow TF  
  
3.- Crea proyecto con scaffolding, pruebas unitarias  
  
4.- Crea dockerfile  
  
5.- Crear workflow donde se ejecuten las pruebas, se construya el front y se publique la imagen de docker**

Proyecto de Aplicación Web con Backend Python y Frontend React

Estructura de Carpetas:

Copy

task-management-app/

│

├── backend/

│ ├── src/

│ │ ├── \_\_init\_\_.py

│ │ ├── main.py

│ │ ├── models.py

│ │ ├── database.py

│ │ └── routes.py

│ ├── tests/

│ │ ├── \_\_init\_\_.py

│ │ ├── test\_routes.py

│ │ └── test\_models.py

│ ├── requirements.txt

│ └── Dockerfile

│

├── frontend/

│ ├── public/

│ ├── src/

│ │ ├── components/

│ │ ├── pages/

│ │ ├── App.js

│ │ └── index.js

│ ├── package.json

│ └── Dockerfile

│

├── .github/

│ └── workflows/

│ └── ci-cd.yml

│

└── docker-compose.yml

Backend (Python con FastAPI):

1. backend/requirements.txt:

Copy

fastapi

uvicorn

sqlalchemy

pydantic

pytest

requests

1. backend/src/main.py:

python

Copy

from fastapi import FastAPI

from routes import task\_router

app = FastAPI()

app.include\_router(task\_router)

@app.get("/")

def read\_root():

return {"message": "Task Management API"}

1. backend/src/models.py:

python

Copy

from sqlalchemy import Column, Integer, String, Boolean

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

class Task(Base):

\_\_tablename\_\_ = "tasks"

id = Column(Integer, primary\_key=True, index=True)

title = Column(String, index=True)

description = Column(String)

completed = Column(Boolean, default=False)

1. backend/src/routes.py:

python

Copy

from fastapi import APIRouter, HTTPException

from pydantic import BaseModel

task\_router = APIRouter()

class TaskCreate(BaseModel):

title: str

description: str = None

completed: bool = False

tasks = []

@task\_router.post("/tasks/")

def create\_task(task: TaskCreate):

new\_task = task.dict()

new\_task['id'] = len(tasks) + 1

tasks.append(new\_task)

return new\_task

@task\_router.get("/tasks/")

def list\_tasks():

return tasks

Frontend (React):

1. frontend/package.json:

json

Copy

{

"dependencies": {

"react": "^17.0.2",

"react-dom": "^17.0.2",

"axios": "^0.21.1"

}

}

1. frontend/src/App.js:

javascript

Copy

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function App() {

const [tasks, setTasks] = useState([]);

const [newTask, setNewTask] = useState('');

useEffect(() => {

fetchTasks();

}, []);

const fetchTasks = async () => {

const response = await axios.get('/api/tasks/');

setTasks(response.data);

};

const handleSubmit = async (e) => {

e.preventDefault();

await axios.post('/api/tasks/', {

title: newTask,

description: ''

});

fetchTasks();

setNewTask('');

};

return (

<div>

<h1>Task Management App</h1>

<form onSubmit={handleSubmit}>

<input

value={newTask}

onChange={(e) => setNewTask(e.target.value)}

placeholder="Nueva tarea"

/>

<button type="submit">Agregar Tarea</button>

</form>

<ul>

{tasks.map(task => (

<li key={task.id}>{task.title}</li>

))}

</ul>

</div>

);

}

export default App;

GitHub Actions Workflow: .github/workflows/ci-cd.yml:

yaml

Copy

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

test-backend:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v3

with:

python-version: '3.9'

- name: Install dependencies

run: |

cd backend

pip install -r requirements.txt

- name: Run Backend Tests

run: |

cd backend

python -m pytest tests/

build-and-deploy:

needs: test-backend

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Build Docker Compose

run: docker-compose up --build -d

Pasos para implementar:

1. Crear repositorio en GitHub
2. Clonar repositorio localmente
3. Copiar estructura de carpetas
4. Commit y push
5. Configurar GitHub Actions
6. Verificar pipeline en repositorio

Nota: Este es un ejemplo básico. Requiere configuraciones adicionales como variables de entorno, manejo de bases de datos, etc.

Aquí tienes una versión mejorada de tu texto, con algunas sugerencias para hacerlo más claro y detallado:

**Tareas para el Proyecto**

1. **Diseñar la aplicación Node.js**:  
   Desarrollar la estructura de la aplicación Node.js, implementando la funcionalidad principal según los requisitos del proyecto.
2. **Diseñar el Dockerfile**:  
   Crear un archivo Dockerfile eficiente que permita construir una imagen Docker de la aplicación Node.js, incluyendo las dependencias necesarias, variables de entorno y configuraciones pertinentes.
3. **Crear un repositorio ECR en AWS utilizando Terraform**:  
   Utilizar **Terraform** para crear un repositorio **Elastic Container Registry (ECR)** en AWS, que almacenará las imágenes Docker generadas para la aplicación.
4. **Crear pruebas unitarias para la aplicación Node.js**:  
   Desarrollar pruebas unitarias utilizando herramientas como **Jest** o **Mocha** para garantizar que la funcionalidad de la aplicación esté bien cubierta y que el código sea confiable.
5. **Configurar el workflow para las pruebas unitarias**:  
   Crear un **workflow** de integración continua (CI) que ejecute las pruebas unitarias automáticamente con cada *push* al repositorio, asegurando que cualquier cambio en el código pase las pruebas antes de ser integrado.
6. **Configurar el workflow de construcción y *push* de la imagen Docker de la aplicación Node.js**:  
   Crear un **workflow** de CI/CD que, si las pruebas unitarias son exitosas, construya la imagen Docker de la aplicación Node.js y la suba automáticamente al repositorio ECR creado previamente. Este paso asegura que siempre se tenga la última versión de la imagen lista para su despliegue.
7. **Diseñar la infraestructura de despliegue de la aplicación utilizando Terraform**:  
   Utilizar **Terraform** para diseñar y provisionar la infraestructura necesaria para el despliegue de la aplicación en AWS (por ejemplo, instancias EC2, Elastic Beanstalk, o ECS, dependiendo de los requisitos). Esto debe incluir la configuración de la red, seguridad y almacenamiento adecuados.
8. **Configurar el workflow de despliegue automático de la aplicación**:  
   Crear un **workflow** de **CD (despliegue continuo)** que despliegue automáticamente la aplicación en la infraestructura AWS configurada, siempre que una nueva imagen Docker sea subida a ECR. Esto puede implicar la configuración de una herramienta como **AWS ECS**, **Elastic Beanstalk**, o **Amazon EKS**, según las necesidades del proyecto.

**Sugerencias para mejorar el flujo de trabajo:**

1. **Automatizar pruebas de integración y end-to-end**:  
   Además de las pruebas unitarias, sería ideal incluir pruebas de integración y pruebas end-to-end (E2E) para verificar que la aplicación funciona correctamente en un entorno completo y no solo en unidades aisladas.
2. **Incluir un sistema de monitoreo y alertas**:  
   Considera agregar un paso en el flujo de trabajo para implementar herramientas de monitoreo, como **AWS CloudWatch** o **Prometheus**, para seguir el estado de la aplicación después del despliegue y recibir alertas en caso de fallos o anomalías.
3. **Asegurar el acceso al repositorio ECR**:  
   Implementa un proceso seguro para la autenticación con el repositorio ECR (por ejemplo, utilizando **AWS CLI** y roles de IAM para la autenticación de los *CI/CD pipelines*).
4. **Versionado de imágenes Docker**:  
   En el paso de construcción y *push* de la imagen Docker, asegúrate de aplicar un esquema de versionado claro (como el uso de tags basados en el número de versión de la aplicación o el hash del commit) para poder hacer un seguimiento de las versiones de la imagen desplegada.
5. **Revisión de seguridad y buenas prácticas**:  
   Asegúrate de que las configuraciones de seguridad (como roles de IAM, configuraciones de seguridad en las instancias de EC2 o en contenedores Docker, etc.) estén alineadas con las mejores prácticas de AWS. Esto podría incluir la revisión de permisos mínimos necesarios y el uso de redes privadas.

**Tareas y Objetivos Recomendados:**

1. **Refinar la estructura de la aplicación** para asegurar que sea escalable, modular y fácil de mantener.
2. **Automatizar la creación de entornos de prueba** para ejecutar pruebas unitarias en diferentes entornos antes de integrar cambios en producción.
3. **Configurar un pipeline de CD** que permita la implementación de cambios en producción solo después de pasar pruebas rigurosas, asegurando la calidad del software en todo momento.

Estas mejoras y sugerencias ayudarán a que el flujo de trabajo sea más robusto, escalable y seguro, al mismo tiempo que se asegura la calidad del código y la eficiencia del proceso de despliegue.

 a6fbeb104a2694b4b908b02ff58ea699-1684171358.eu-west-2.elb.amazonaws.com/treasure/

**TRABAJO SUPLENTARIO DE ANTONIO VEGA:**  
<https://www.linkedin.com/posts/claudenederlia_exciting-news-from-the-semiconductor-industry-activity-7292470589661802497-PT9Z?utm_source=share&utm_medium=member_desktop>

https://docs.google.com/spreadsheets/d/1EQKz27FhMaimjYqHykqquxWySlOtlj-yxQpzXGrVDR4/edit?usp=sharing

**WEBEX 05/02/2005**

**WORKFLOWS AVANZADOS EN GITHUB ACTIONS**

**EJERCICIOS 1:**

**Múltiples Jobs con Dependencias:**

**Ejercicio 1: Crear Múltiples Jobs con Dependencias:**

Crear `advanced-workflow.yml` con `build`, `test` y `deploy`.

- Definir dependencias entre jobs con `needs`.

- Confirmar ejecución en la pestaña Actions.

name: Advanced CI/CD Workflow

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Set up Node.js

uses: actions/setup-node@v4

with:

node-version: '20'

- name: Install dependencies

run: npm install

- name: Build application

run: npm run build

- name: Upload build artifacts

uses: actions/upload-artifact@v4

with:

name: build-output

path: dist/

test:

needs: build

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Download build artifacts

uses: actions/download-artifact@v4

with:

name: build-output

path: dist/

- name: Run unit tests

run: npm test

- name: Run integration tests

run: npm run test:integration

deploy:

needs: [build, test]

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Download build artifacts

uses: actions/download-artifact@v4

with:

name: build-output

path: dist/

- name: Deploy to production

run: |

echo "Deploying to production server..."

# Aquí podrías añadir comandos de despliegue específicos

# Por ejemplo: scp, ssh, docker push, etc.

Para usar este workflow, debes seguir estos pasos:

1. Crear el archivo de workflow:
   * En tu repositorio de GitHub, navega a .github/workflows/
   * Crea un nuevo archivo llamado advanced-workflow.yml
   * Copia y pega el contenido del artifact que he generado
2. Estructura del workflow:
   * build: Construye la aplicación y sube los artefactos
   * test: Depende de build, ejecuta pruebas unitarias e integración
   * deploy: Depende de build y test, simula un despliegue
3. Requisitos previos:
   * Tener un proyecto Node.js configurado
   * Scripts de build, test y test:integration en tu package.json
4. Configuración adicional:
   * Ajusta las versiones de Node.js según tu proyecto
   * Personaliza los comandos de construcción, pruebas y despliegue

🔑 Puntos importantes:

* El job test solo se ejecutará si build es exitoso
* El job deploy requiere que build y test sean exitosos
* Se utilizan artefactos para compartir la salida entre jobs

Recomendaciones:

* Asegúrate de tener configurados los scripts en tu package.json
* Verifica que los comandos de pruebas y construcción sean correctos
* En un escenario real, reemplaza el comentario de despliegue con tus comandos específicos

Aquí te explico el flujo completo de cómo hacer un git pull de manera segura y luego subir tus cambios, con los comandos y explicaciones detalladas:

1. Primero, verifica en qué rama estás:

bash

Copy

git branch

Explicación: Este comando muestra en qué rama estás actualmente. Es importante saber esto antes de hacer cualquier operación.

1. Cambia a la rama principal (generalmente main o master):

bash

Copy

git checkout main

Explicación: Te aseguras de estar en la rama principal antes de actualizar.

1. Actualiza tu repositorio local de manera segura:

bash

Copy

git fetch origin

Explicación: Descarga todos los cambios del repositorio remoto sin modificar tu trabajo local. Es como una "revisión" previa de los cambios.

1. Revisa las diferencias antes de hacer merge:

bash

Copy

git log origin/main ^main

Explicación: Este comando te muestra los commits que están en el repositorio remoto pero no en tu repositorio local.

1. Realiza el pull con rebase (la opción más limpia):

bash

Copy

git pull --rebase origin main

Explicación:

* --rebase aplica tus cambios locales encima de los cambios remotos
* Evita crear commits de merge innecesarios
* Mantiene el historial de commits más limpio y lineal

1. Si hay conflictos, resuélvelos:

bash

Copy

*# Abre los archivos con conflictos*

nano archivo\_con\_conflicto.txt

*# Marca los archivos como resueltos*

git add archivo\_con\_conflicto.txt

*# Continúa el rebase*

git rebase --continue

Explicación: Si hay conflictos, los editas manualmente, los marcas como resueltos y continúas el rebase.

1. Añade tus cambios locales:

bash

Copy

*# Añade todos los cambios*

git add .

*# O añade cambios específicos*

git add archivo1.txt archivo2.txt

Explicación: Preparas los archivos que quieres subir al repositorio.

1. Haz commit de tus cambios:

bash

Copy

git commit -m "Descripción clara de los cambios"

Explicación: Crea un nuevo commit con un mensaje descriptivo de lo que has modificado.

1. Sube tus cambios al repositorio remoto:

bash

Copy

git push origin main

Explicación: Envía tus commits locales al repositorio remoto en GitHub.

🔑 Consejos adicionales:

* Siempre haz pull antes de empezar a trabajar
* Usa --rebase para mantener un historial limpio
* Escribe mensajes de commit claros y concisos
* Resuelve conflictos con cuidado

❗ Precauciones:

* No hagas git push -f (forzar) a menos que estés completamente seguro
* Comunícate con tu equipo antes de hacer cambios grandes
* Si no estás seguro, pregunta a un compañero

¿Quieres que profundice en alguno de estos pasos o tienes alguna duda específica sobre el flujo de trabajo?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

[.](https://support.anthropic.com/en/articles/8525154-claude-is-providing-incorrect-or-misleading-responses-what-s-going-on)

**Ejercicio 2: Configurar una Matriz de Estrategia**

jobs:

test:

runs-on: ubuntu-latest

needs: build

strategy:

matrix:

node-version: [12, 14, 16]

steps:

- name: Set Node.js Version

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

- name: Run Tests

run: npm test

- Confirmar ejecución en diferentes versiones

name: Node.js Matrix Strategy CI

*# Desencadenantes del workflow*

on:

push:

branches: [ main, develop ]

pull\_request:

branches: [ main, develop ]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Set up Node.js

uses: actions/setup-node@v4

with:

node-version: 16

- name: Install dependencies

run: npm ci

- name: Build project

run: npm run build

- name: Upload build artifacts

uses: actions/upload-artifact@v4

with:

name: build-output

path: dist/

test:

needs: build

runs-on: ubuntu-latest

*# Estrategia de matriz para probar en múltiples versiones*

strategy:

*# Configura una matriz para probar en Node.js 12, 14 y 16*

matrix:

node-version: [12, 14, 16]

*# Puedes añadir más configuraciones si es necesario*

*# Por ejemplo, probando en diferentes sistemas operativos*

*# os: [ubuntu-latest, windows-latest, macos-latest]*

*# Nombre del job con la versión de Node.js*

name: Test on Node.js ${{ matrix.node-version }}

steps:

- uses: actions/checkout@v4

- name: Use Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v4

with:

node-version: ${{ matrix.node-version }}

- name: Install dependencies

run: npm ci

- name: Download build artifacts

uses: actions/download-artifact@v4

with:

name: build-output

path: dist/

- name: Run Tests

run: npm test

*# Ejemplo de pruebas adicionales*

- name: Run Coverage

run: npm run test:coverage

*# Opcional: Subir resultados de cobertura*

- name: Upload coverage

uses: codecov/codecov-action@v3

with:

fail\_ci\_if\_error: true

**1. Desencadenantes del Workflow**

* push en ramas main y develop
* pull\_request en ramas main y develop
* Esto garantiza que se ejecuten pruebas en cada push o solicitud de PR

**2. Job de Build (build)**

* Configura el entorno Node.js
* Instala dependencias con npm ci (más seguro para CI)
* Construye el proyecto
* Sube artefactos de construcción

**3. Job de Test con Matriz de Estrategia (test)**

* Depende del job de build
* Usa strategy.matrix para probar en Node.js 12, 14 y 16
* Cada combinación de la matriz se ejecuta como un job separado
* Descarga artefactos del job de build
* Ejecuta pruebas

**4. Características Avanzadas**

* Usa última versión de acciones de GitHub
* Configuración flexible para múltiples versiones
* Manejo de artefactos entre jobs
* Opcional: Reporte de cobertura con Codecov

**Pasos para Implementar:**

1. Crear archivo de workflow:

bash

Copy

mkdir -p .github/workflows

nano .github/workflows/node-matrix-ci.yml

1. Copiar contenido del artifact
2. Asegurar package.json tenga scripts:

json

Copy

{

"scripts": {

"build": "...",

"test": "jest",

"test:coverage": "jest --coverage"

}

}

1. Commit y push:

bash

Copy

git add .github/workflows/node-matrix-ci.yml

git commit -m "Add matrix strategy CI workflow"

git push origin main

🚀 Beneficios de esta Estrategia:

* Pruebas en múltiples versiones Node.js
* Detección temprana de incompatibilidades
* Configuración escalable
* Integración continua robusta

❗ Consideraciones:

* Ajusta versiones según tu proyecto
* Añade más configuraciones si es necesario
* Mantén actualizadas las dependencias

¿Quieres que profundice en algún aspecto del workflow o tienes preguntas sobre su implementación?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

[Claude can make mistakes.   
Please double-check responses.](https://support.anthropic.com/en/articles/8525154-claude-is-providing-incorrect-or-misleading-responses-what-s-going-on)

**Ejercicio 3: Usar Secretos en un Workflow**

Configurar `API\_KEY` en `Settings > Secrets > Actions`.

- Añadir el uso del secreto en el job `deploy` de forma segura.

name: CI/CD with Secrets

on:

push:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [16, 18]

steps:

- uses: actions/checkout@v4

- name: Use Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v4

with:

node-version: ${{ matrix.node-version }}

- name: Install dependencies

run: npm install

- name: Run Tests

run: npm test

deploy:

needs: test

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Deploy to Production

*# Uso seguro de un secreto*

env:

API\_KEY: ${{ secrets.API\_KEY }}

DEPLOY\_HOST: ${{ secrets.DEPLOY\_HOST }}

run: |

echo "Desplegando con clave de API segura"

# Ejemplo de uso de secretos en un script de despliegue

curl -X POST https://$DEPLOY\_HOST/deploy \

-H "Authorization: Bearer $API\_KEY" \

-d "version=$(git rev-parse HEAD)"

**Guía Completa para Configurar Secretos en GitHub**

**Paso 1: Crear Secretos en GitHub**

1. Ve a tu repositorio en GitHub
2. Haz clic en Settings
3. En el menú lateral, selecciona Secrets and variables
4. Selecciona Actions
5. Haz clic en New repository secret
6. Añade dos secretos:
   * API\_KEY: Tu clave de API
   * DEPLOY\_HOST: El host de despliegue

**Paso 2: Estructura del Workflow**

* Añadí un nuevo job deploy que depende de test
* Uso de env: para declarar variables secretas
* ${{ secrets.NOMBRE\_SECRETO }} para acceder de forma segura

**Paso 3: Implementación**

bash

Copy

*# Crear directorio de workflows*

mkdir -p .github/workflows

*# Crear archivo de workflow*

nano .github/workflows/secrets-workflow.yml

*# Copiar contenido del artifact*

**🔑 Características Clave:**

* Secretos nunca se exponen en logs
* Protección contra exposición accidental
* Uso seguro en scripts de despliegue

**Ejemplo de Uso Real**

Puedes usar este patrón para:

* Desplegar a servidores
* Hacer llamadas a APIs
* Configurar credenciales de servicios

❗ Consejos de Seguridad:

* Nunca imprimas secretos
* Rota tus claves periódicamente
* Limita el alcance de las claves

Ejemplo de script de despliegue real:

bash

Copy

#!/bin/bash

*# Script de despliegue de ejemplo*

if [ -z "$API\_KEY" ]; then

echo "Error: API\_KEY no configurada"

exit 1

fi

*# Lógica de despliegue segura*

**Práctica Colaborativa: Configurar un Workflow Completo**

Objetivo: Implementar un workflow con matrices, secretos y artefactos.

- Roles: Configurar jobs, añadir secretos, subir artefactos y validar errores

Crear una estructura de proyecto completa para un proyecto de Node.js con un workflow de GitHub Actions robusto.

mi-proyecto-devops/

│

├── .github/

│ └── workflows/

│ └── ci-cd-workflow.yml

│

├── src/

│ ├── controllers/

│ │ └── userController.js

│ ├── models/

│ │ └── userModel.js

│ ├── routes/

│ │ └── userRoutes.js

│ └── app.js

│

├── tests/

│ ├── unit/

│ │ └── userController.test.js

│ └── integration/

│ └── userRoutes.test.js

│

├── config/

│ ├── database.js

│ └── environment.js

│

├── scripts/

│ ├── deploy.sh

│ └── setup.sh

│

├── .env.example

├── package.json

├── README.md

└── Dockerfile

name: Complete CI/CD Pipeline

on:

push:

branches: [main, develop]

pull\_request:

branches: [main, develop]

jobs:

lint-and-test:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [16.x, 18.x]

steps:

- uses: actions/checkout@v4

- name: Use Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v4

with:

node-version: ${{ matrix.node-version }}

- name: Cache npm dependencies

uses: actions/cache@v3

with:

path: ~/.npm

key: ${{ runner.os }}-node-${{ hashFiles('\*\*/package-lock.json') }}

- name: Install Dependencies

run: npm ci

- name: Run Linter

run: npm run lint

- name: Run Unit Tests

run: npm run test:unit

- name: Run Integration Tests

run: npm run test:integration

build:

needs: lint-and-test

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Build Project

run: npm run build

- name: Upload Build Artifacts

uses: actions/upload-artifact@v4

with:

name: build-output

path: dist/

deploy:

needs: build

runs-on: ubuntu-latest

*# Solo despliega en main*

if: github.ref == 'refs/heads/main'

steps:

- uses: actions/checkout@v4

- name: Download Build Artifacts

uses: actions/download-artifact@v4

with:

name: build-output

path: dist/

- name: Deploy to Production

env:

DEPLOY\_TOKEN: ${{ secrets.DEPLOY\_TOKEN }}

DEPLOY\_HOST: ${{ secrets.DEPLOY\_HOST }}

run: |

echo "Desplegando a producción..."

ssh user@$DEPLOY\_HOST "cd /path/to/app && ./deploy.sh"

**Explicación Detallada:**

**🔍 Estructura del Proyecto**

* Workflow de CI/CD completo
* Múltiples jobs con dependencias
* Pruebas en diferentes versiones
* Manejo de secretos
* Construcción y despliegue automatizado

**🛡️ Características Principales**

* Validación de código
* Pruebas automatizadas
* Construcción de artefactos
* Despliegue condicional
* Seguridad con secretos

**🚦 Flujo de Trabajo**

1. Desarrollador hace push/PR
2. GitHub Actions inicia workflow
3. Ejecuta pruebas en múltiples versiones
4. Construye artefactos
5. Despliega si todo es correcto

¿Quieres que profundice en algún aspecto del proyecto o necesitas más detalles sobre alguna parte específica?

**royecto DevOps: Gestión de Usuarios con CI/CD**

**🚀 Descripción del Proyecto**

Este proyecto es una aplicación completa de gestión de usuarios con un flujo de integración y despliegue continuo (CI/CD) utilizando GitHub Actions, Docker y buenas prácticas de DevOps.

**📋 Características Principales**

* API RESTful para gestión de usuarios
* Autenticación y autorización
* Integración continua con GitHub Actions
* Despliegue automatizado
* Contenerización con Docker
* Gestión de base de datos con Sequelize

**🛠️ Tecnologías Utilizadas**

* Node.js
* Express.js
* Sequelize (ORM)
* PostgreSQL
* Docker
* GitHub Actions
* bcryptjs (Encriptación)

**🔧 Requisitos Previos**

* Node.js (16.x o superior)
* Docker
* PostgreSQL
* Cuenta de GitHub

**📦 Instalación y Configuración**

**1. Clonar Repositorio**

bash

Copy

git clone https://github.com/tu-usuario/proyecto-devops.git

cd proyecto-devops

**2. Configurar Variables de Entorno**

Crear archivo .env basado en .env.example:

Copy

# Configuración de Base de Datos

DB\_HOST=localhost

DB\_NAME=usuarios

DB\_USER=admin

DB\_PASSWORD=secreto

# Configuración del Servidor

PORT=3000

JWT\_SECRET=tu\_secreto\_jwt

**3. Instalación de Dependencias**

bash

Copy

npm install

**4. Iniciar Base de Datos**

bash

Copy

npm run db:create

npm run db:migrate

**🧪 Ejecución de Pruebas**

bash

Copy

*# Pruebas unitarias*

npm run test:unit

*# Pruebas de integración*

npm run test:integration

*# Cobertura de código*

npm run test:coverage

**🚢 Despliegue**

**Desarrollo Local**

bash

Copy

npm run dev

**Producción con Docker**

bash

Copy

*# Construir imagen*

docker build -t proyecto-devops .

*# Ejecutar contenedor*

docker run -p 3000:3000 proyecto-devops

**🔄 Workflow de CI/CD**

**Flujo de GitHub Actions**

1. **Lint y Validación**
   * Revisar código con ESLint
   * Verificar formato
2. **Pruebas**
   * Ejecutar pruebas unitarias
   * Ejecutar pruebas de integración
   * Verificar cobertura de código
3. **Construcción**
   * Generar artefactos
   * Construir imagen Docker
4. **Despliegue**
   * Desplegar en servidor de producción
   * Ejecutar migraciones
   * Reiniciar servicios

**🔒 Seguridad**

* Encriptación de contraseñas con bcrypt
* Validación de datos de entrada
* Secretos de GitHub para credenciales
* Configuración de CORS
* Validación de tokens JWT

**📝 Endpoints de API**

**Usuarios**

* POST /api/users: Crear usuario
* GET /api/users: Listar usuarios
* GET /api/users/:id: Obtener usuario por ID

**🤝 Contribución**

1. Fork del repositorio
2. Crear rama de feature
3. Commit de cambios
4. Push a la rama
5. Crear Pull Request

**📄 Licencia**

MIT License

**📞 Contacto**

* Correo: [tu-email@ejemplo.com](mailto:tu-email@ejemplo.com)
* GitHub: @tu-usuario

Copy

## Explicación Adicional:

### 🔍 Estructura Detallada

- Controlador: Maneja la lógica de negocio de usuarios

- Modelo: Define estructura de datos de usuarios

- Rutas: Define endpoints de la API

- Configuración BD: Conexión y sincronización

- Script Deploy: Automatización de despliegue

- Dockerfile: Contenerización de la aplicación

### 🚦 Flujo Completo

1. Desarrollador hace cambios

2. Push a repositorio

3. GitHub Actions inicia workflow

4. Ejecuta pruebas y validaciones

5. Construye artefactos

6. Despliega en producción

### 🛡️ Consideraciones de Seguridad

- Contraseñas encriptadas

- Validación de datos

- Gestión segura de secretos

- Configuración de entornos

¿Quieres que profundice en algún aspecto específico del proyecto o necesitas más detalles sob

**WEBEX 06/02/2025**

**CICD**

**EJERCICIO 1:**

**Ejercicio 1: Crear y Configurar un Bucket S3 con Terraform y politica de retencion de 30dias antes borrado**

Usar módulo de Terraform para S3.

- Código base:

```hcl

module "s3\_bucket" {

source = "./modules/s3"

bucket\_name = "ci-cd-static-app-grupo1"

region = "us-east-1"

public\_access = true

}

```

- Ejecutar `terraform init` y `terraform apply -auto-approve`.

Para crear y configurar un bucket S3 con Terraform, utilizando un módulo y estableciendo una política de retención de 30 días antes de su eliminación, primero debemos definir el módulo de S3 y luego utilizarlo en nuestro código base. A continuación, te proporcionaré un ejemplo detallado de cómo hacerlo.

**Paso 1: Crear el módulo de S3**

Primero, crea un directorio para tu módulo de S3. En este caso, lo llamaremos modules/s3. Dentro de este directorio, crea un archivo llamado main.tf donde definiremos la configuración del bucket S3.

Estructura de directorios

Code

Copiar código

.

├── main.tf

└── modules

└── s3

└── main.tf

Contenido de modules/s3/main.tf

hcl

Copiar código

provider "aws" {

region = var.region

}

resource "aws\_s3\_bucket" "this" {

bucket = var.bucket\_name

acl = var.public\_access ? "public-read" : "private"

versioning {

enabled = true

}

lifecycle {

prevent\_destroy = true

}

}

resource "aws\_s3\_bucket\_lifecycle\_configuration" "this" {

bucket = aws\_s3\_bucket.this.id

rule {

id = "retention-rule"

status = "Enabled"

expiration {

days = 30

}

}

}

variable "bucket\_name" {

description = "The name of the S3 bucket"

type = string

}

variable "region" {

description = "The AWS region to create the bucket in"

type = string

}

variable "public\_access" {

description = "Should the bucket be publicly accessible?"

type = bool

}

**Explicación del módulo**

1. **Provider**: Se especifica el proveedor de AWS y la región donde se creará el bucket.
2. **aws\_s3\_bucket**: Se define el recurso del bucket S3. Se utiliza la variable bucket\_name para nombrar el bucket y public\_access para establecer el ACL (Access Control List).
3. **Versioning**: Se habilita el versionado del bucket, lo que permite mantener múltiples versiones de los objetos.
4. **Lifecycle**: Se establece prevent\_destroy en true para evitar que el bucket se elimine accidentalmente.
5. **aws\_s3\_bucket\_lifecycle\_configuration**: Se configura una regla de ciclo de vida que elimina los objetos en el bucket después de 30 días.
6. **Variables**: Se definen las variables necesarias para el módulo.

**Paso 2: Crear el archivo principal**

Ahora, en el archivo main.tf en la raíz del proyecto, utilizaremos el módulo que acabamos de crear.

Contenido de main.tf

hcl

Copiar código

module "s3\_bucket" {

source = "./modules/s3"

bucket\_name = "ci-cd-static-app-grupo1"

region = "eu-east-2"

public\_access = true

}

**Paso 3: Inicializar y aplicar la configuración**

Ahora que hemos configurado nuestro módulo y el archivo principal, podemos inicializar y aplicar la configuración de Terraform.

1. Abre una terminal y navega al directorio donde se encuentra tu archivo main.tf.
2. Ejecuta el siguiente comando para inicializar el proyecto de Terraform:

bash

Copiar código

terraform init

Este comando descargará los proveedores necesarios y preparará el entorno de trabajo.

1. Luego, ejecuta el siguiente comando para aplicar la configuración:

bash

Copiar código

terraform apply -auto-approve

El flag -auto-approve se utiliza para omitir la confirmación interactiva y aplicar los cambios automáticamente.

**Resumen**

Con este enfoque, hemos creado un módulo de Terraform para un bucket S3 que incluye una política de retención de 30 días. La configuración permite que el bucket sea público y habilita el versionado. Además, hemos implementado una regla de ciclo de vida que elimina los objetos después de 30 días, lo que ayuda a gestionar el almacenamiento y los costos en AWS.

podrias definir el archivo variables, por favor?

.

GPT-4o mini

¡Por supuesto! Para mejorar la organización y la reutilización de tu módulo de Terraform, es una buena práctica definir un archivo separado para las variables. Esto facilita la gestión de las variables y su documentación.

**Paso 1: Crear el archivo de variables**

Dentro del directorio del módulo modules/s3, crea un archivo llamado variables.tf. Este archivo contendrá la definición de las variables que utilizamos en el módulo.

Contenido de modules/s3/variables.tf

hcl

Copiar código

variable "bucket\_name" {

description = "The name of the S3 bucket"

type = string

}

variable "region" {

description = "The AWS region to create the bucket in"

type = string

}

variable "public\_access" {

description = "Should the bucket be publicly accessible?"

type = bool

default = false *# Puedes establecer un valor por defecto si lo deseas*

}

**Explicación del archivo de variables**

1. **bucket\_name**: Esta variable es de tipo string y se utiliza para definir el nombre del bucket S3. La descripción proporciona contexto sobre su uso.
2. **region**: Esta variable también es de tipo string y se utiliza para especificar la región de AWS donde se creará el bucket. La descripción ayuda a entender su propósito.
3. **public\_access**: Esta variable es de tipo bool y se utiliza para determinar si el bucket debe ser accesible públicamente. Se ha añadido un valor por defecto de false, lo que significa que, si no se proporciona un valor al usar el módulo, el bucket no será público.

**Paso 2: Actualizar el módulo**

Con el archivo variables.tf creado, asegúrate de que el archivo main.tf del módulo modules/s3 esté utilizando estas variables correctamente. No es necesario realizar cambios en main.tf, ya que ya está utilizando las variables definidas.

**Resumen**

Ahora tienes un archivo variables.tf que define las variables utilizadas en tu módulo de S3. Esto mejora la claridad y la organización de tu código Terraform, facilitando la comprensión y el mantenimiento en el futuro.

Principio del formulario

**Ejercicio 2: Configurar el Workflow de Despliegue**

Configurar credenciales en GitHub Actions (`AWS\_ACCESS\_KEY\_ID`, `AWS\_SECRET\_ACCESS\_KEY`).

- Crear archivo `.github/workflows/deploy.yml` con este código:

```yaml

name: Deploy to S3

on:

push:

branches:

- main

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v3

- name: Setup AWS Credentials

uses: aws-actions/configure-aws-credentials@v3

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-east-1

- name: Deploy to S3

run: aws s3 sync . s3://ci-cd-static-app-grupo1 --delete

```

**Paso : Configurar GitHub Actions**

Para automatizar la aplicación de Terraform, configuraremos un flujo de trabajo de GitHub Actions.

1. Crea un directorio para las acciones:

bash

Copiar código

mkdir -p .github/workflows

1. Crea un archivo llamado terraform.yml dentro de .github/workflows:

Contenido de .github/workflows/terraform.yml

yaml

Copiar código

name: Terraform

on:

push:

branches:

- main

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply

no me has incluido el setup aes credenciales en el yaml, puedes reconfigurarlo, por favor?

.

GPT-4o mini

¡Claro! Para que GitHub Actions pueda interactuar con AWS y aplicar la configuración de Terraform, necesitas configurar las credenciales de AWS en el flujo de trabajo. Esto se hace utilizando secretos de GitHub para almacenar de forma segura las credenciales de acceso a AWS.

**Paso 1: Configurar las Credenciales de AWS en GitHub**

1. **Obtener las credenciales de AWS**:
   * Ve a la consola de AWS y crea un usuario IAM con permisos para gestionar S3 (por ejemplo, AmazonS3FullAccess).
   * Anota el Access Key ID y el Secret Access Key de este usuario.
2. **Agregar secretos en GitHub**:
   * Ve a tu repositorio en GitHub.
   * Haz clic en "Settings" (Configuración).
   * En el menú de la izquierda, selecciona "Secrets and variables" y luego "Actions".
   * Haz clic en "New repository secret".
   * Agrega los siguientes secretos:
     + **Nombre**: AWS\_ACCESS\_KEY\_ID - **Valor**: tu Access Key ID.
     + **Nombre**: AWS\_SECRET\_ACCESS\_KEY - **Valor**: tu Secret Access Key.

**Paso 2: Actualizar el Archivo de Flujo de Trabajo**

Ahora que has configurado los secretos, actualizaremos el archivo de flujo de trabajo para incluir la configuración de las credenciales de AWS.

Contenido actualizado de .github/workflows/terraform.yml

yaml

Copiar código

name: Terraform

on:

push:

branches:

- main

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-east-1 *# Cambia esto si usas otra región*

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply -auto-approve

**Explicación de los Cambios**

1. **Configure AWS credentials**: Este paso utiliza la acción aws-actions/configure-aws-credentials para configurar las credenciales de AWS en el entorno de ejecución. Utiliza los secretos que configuraste anteriormente para autenticarte con AWS.
2. **aws-region**: Asegúrate de que la región especificada aquí coincida con la región que estás utilizando en tu configuración de Terraform.

**Paso 3: Probar el Flujo de Trabajo**

Ahora que has configurado el flujo de trabajo, cada vez que hagas un push a la rama main, GitHub Actions ejecutará el flujo de trabajo y aplicará la configuración de Terraform automáticamente.

1. Realiza un cambio en tu código (puedes agregar un comentario en main.tf o en cualquier archivo).
2. Haz un commit y un push a la rama main:

bash

Copiar código

git add .

git commit -m "Trigger GitHub Actions workflow"

git push origin main

1. Ve a la pestaña "Actions" en tu repositorio de GitHub para ver el progreso del flujo de trabajo.

**Resumen**

Hemos configurado un flujo de trabajo de GitHub Actions que utiliza las credenciales de AWS para aplicar la configuración de Terraform automáticamente. Esto permite que cada cambio en la rama main se implemente en AWS, facilitando la gestión de la infraestructura como código.

**Ejercicio 3: Validar el Despliegue**

Crear `index.html` en el repositorio.

- Hacer commit y push a `main`.

- Verificar ejecución del workflow en `Actions`.

- Confirmar que el archivo aparece en S3 y es accesible desde la URL generada.

Ejercicio 3: Validar el Despliegue Crear `index.html` en el repositorio. - Hacer commit y push a `main`. - Verificar ejecución del workflow en `Actions`. - Confirmar que el archivo aparece en S3 y es accesible desde la URL generada.

.

GPT-4o mini

Para validar el despliegue de nuestro bucket S3 y asegurarnos de que el archivo index.html se haya subido correctamente, seguiremos los pasos que mencionaste. Vamos a crear el archivo index.html, configurarlo para que se suba al bucket S3, y luego verificaremos que el archivo esté accesible.

**Paso 1: Crear el Archivo index.html**

Primero, crearemos un archivo index.html en la raíz de nuestro repositorio. Este archivo contendrá contenido básico que se mostrará cuando accedamos a la URL del bucket S3.

Contenido de index.html

Crea un archivo llamado index.html en la raíz de tu repositorio y agrega el siguiente contenido:

html

Copiar código

*<!DOCTYPE html>*

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bienvenido a mi Bucket S3</title>

</head>

<body>

<h1>¡Hola, mundo!</h1>

<p>Este es un archivo estático servido desde un bucket S3.</p>

</body>

</html>

**Paso 2: Actualizar el Flujo de Trabajo para Subir index.html**

Ahora, actualizaremos el flujo de trabajo de GitHub Actions para que suba el archivo index.html al bucket S3 después de aplicar la configuración de Terraform.

Contenido actualizado de .github/workflows/terraform.yml

Agregaremos un paso adicional para subir el archivo index.html al bucket S3. Asegúrate de que el flujo de trabajo se vea así:

yaml

Copiar código

name: Terraform

on:

push:

branches:

- main

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-east-1 *# Cambia esto si usas otra región*

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply -auto-approve

- name: Upload index.html to S3

run: aws s3 cp index.html s3://ci-cd-static-app-grupo1/

**Explicación de los Cambios**

1. **Upload index.html to S3**: Este paso utiliza el comando de AWS CLI aws s3 cp para copiar el archivo index.html al bucket S3 que hemos creado. Asegúrate de que el nombre del bucket coincida con el que has definido en tu módulo de Terraform.

**Paso 3: Hacer Commit y Push**

Ahora que hemos creado el archivo index.html y actualizado el flujo de trabajo, haremos un commit y un push a la rama main.

bash

Copiar código

git add index.html .github/workflows/terraform.yml

git commit -m "Add index.html and update workflow to upload to S3"

git push origin main

**Paso 4: Verificar la Ejecución del Workflow en GitHub**

1. Ve a la pestaña "Actions" en tu repositorio de GitHub.
2. Deberías ver que se ha iniciado un nuevo flujo de trabajo. Haz clic en el último flujo de trabajo para ver los detalles.
3. Verifica que todos los pasos se hayan completado correctamente, especialmente el paso de "Upload index.html to S3".

**Paso 5: Confirmar que el Archivo Aparece en S3**

1. Ve a la consola de AWS S3.
2. Navega hasta el bucket que has creado (ci-cd-static-app-grupo1).
3. Deberías ver el archivo index.html en la lista de objetos del bucket.

**Paso 6: Acceder al Archivo desde la URL Generada**

Para acceder al archivo index.html, utiliza la URL del bucket S3. La URL generalmente tiene el siguiente formato:

Code

Copiar código

http://<bucket-name>.s3.amazonaws.com/index.html

En tu caso, sería:

Code

Copiar código

http://ci-cd-static-app-grupo1.s3.amazonaws.com/index.html

Abre esta URL en tu navegador. Deberías ver el contenido del archivo index.html que creaste.

**Resumen**

Hemos creado un archivo index.html, actualizado el flujo de trabajo de GitHub Actions para subirlo al bucket S

Principio del formulario

**Ejercicio 4: Actualizar el Despliegue**

Modificar `index.html` agregando una nueva línea.

- Hacer commit y push nuevamente.

- Confirmar que el cambio se refleja en la URL del bucket.

Para actualizar el despliegue, simplemente modificaremos el archivo index.html para agregar una nueva línea de contenido. Luego, haremos un commit y un push a la rama main, lo que activará el flujo de trabajo de GitHub Actions para aplicar los cambios y subir el archivo actualizado al bucket S3. A continuación, te detallo los pasos a seguir.

**Paso 1: Modificar index.html**

Abre el archivo index.html en tu editor de texto y agrega una nueva línea. Por ejemplo, puedes agregar una línea que diga "Este contenido ha sido actualizado.".

Contenido actualizado de index.html

html

Copiar código

*<!DOCTYPE html>*

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bienvenido a mi Bucket S3</title>

</head>

<body>

<h1>¡Hola, mundo!</h1>

<p>Este es un archivo estático servido desde un bucket S3.</p>

<p>Este contenido ha sido actualizado.</p> *<!-- Nueva línea agregada -->*

</body>

</html>

**Paso 2: Hacer Commit y Push**

Una vez que hayas guardado los cambios en index.html, realiza un commit y un push a la rama main para que GitHub Actions ejecute el flujo de trabajo y suba el archivo actualizado al bucket S3.

bash

Copiar código

git add index.html

git commit -m "Update index.html to add a new line"

git push origin main

**Paso 3: Verificar la Ejecución del Workflow en GitHub**

1. Ve a la pestaña "Actions" en tu repositorio de GitHub.
2. Deberías ver que se ha iniciado un nuevo flujo de trabajo. Haz clic en el último flujo de trabajo para ver los detalles.
3. Verifica que todos los pasos se hayan completado correctamente, especialmente el paso de "Upload index.html to S3".

**Paso 4: Confirmar que el Cambio se Refleja en la URL del Bucket**

1. Abre tu navegador y accede a la URL del archivo index.html en tu bucket S3:

Code

Copiar código

http://ci-cd-static-app-grupo1.s3.amazonaws.com/index.html

1. Deberías ver el contenido actualizado, incluyendo la nueva línea que agregaste: "Este contenido ha sido actualizado."

**Resumen**

Hemos actualizado el archivo index.html, realizado un commit y un push a la rama main, lo que activó el flujo de trabajo de GitHub Actions para aplicar los cambios y subir el archivo actualizado al bucket S3. Finalmente, confirmamos que el cambio se refleja correctamente en la URL del bucket. Este proceso demuestra cómo se puede gestionar y actualizar la infraestructura y el contenido estático utilizando Terraform y GitHub Actions.

**EJERCICIO 5: AÑADIR OICD PERSISTENCIA**

**rol\_github**

 Información

Eliminar

**Resumen**

Editar

Fecha de creación

February 06, 2025, 16:16 (UTC+01:00)

ARN

**arn:aws:iam::248189943700:role/rol\_github**

Última actividad

-

Duración máxima de la sesión

1 hora

Para implementar el ejercicio completo utilizando OpenID Connect (OIDC) en lugar de tokens de acceso, vamos a configurar GitHub Actions para que se autentique con AWS utilizando OIDC. Esto permite que GitHub Actions asuma un rol de IAM en AWS sin necesidad de almacenar credenciales de acceso en los secretos del repositorio.

**Paso 1: Configurar un Rol de IAM en AWS**

1. **Iniciar sesión en la consola de AWS**:
   * Ve a la consola de AWS y accede a tu cuenta.
2. **Crear un nuevo rol**:
   * Navega a **IAM** (Identity and Access Management).
   * En el menú de la izquierda, selecciona **Roles** y luego haz clic en **Create role**.
   * Selecciona **Web identity** como tipo de entidad de confianza.
   * En el proveedor de identidad, selecciona **GitHub**.
   * En el campo **Audience**, ingresa sts.amazonaws.com.
3. **Configurar la política de permisos**:
   * En la siguiente pantalla, selecciona las políticas que deseas adjuntar al rol. Para este ejercicio, puedes usar AmazonS3FullAccess para permitir el acceso completo a S3.
   * Haz clic en **Next: Tags** y luego en **Next: Review**.
   * Asigna un nombre al rol, por ejemplo, GitHubActionsS3AccessRole, y haz clic en **Create role**.
4. **Anotar el ARN del rol**:
   * Después de crear el rol, anota el ARN del rol, ya que lo necesitaremos más adelante.

**Paso 2: Configurar el Repositorio de GitHub**

1. **Configurar el flujo de trabajo de GitHub Actions**:
   * Abre tu repositorio en GitHub y ve a la pestaña **Settings**.
   * En el menú de la izquierda, selecciona **Actions** y luego **General**.
   * Desplázate hacia abajo hasta la sección **Workflow permissions** y asegúrate de que la opción **Read and write permissions** esté seleccionada.
2. **Configurar el flujo de trabajo**:
   * Crea o edita el archivo .github/workflows/terraform.yml en tu repositorio para que se vea así:

Contenido de .github/workflows/terraform.yml

yaml

Copiar código

name: Terraform

on:

push:

branches:

- main

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Configure AWS credentials using OIDC

id: aws-creds

uses: aws-actions/configure-aws-credentials@v1

with:

role-to-assume: arn:aws:iam::<YOUR\_AWS\_ACCOUNT\_ID>:role/GitHubActionsS3AccessRole

aws-region: us-east-1 *# Cambia esto si usas otra región*

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply -auto-approve

- name: Upload index.html to S3

run: aws s3 cp index.html s3://ci-cd-static-app-grupo1/

**Explicación del Flujo de Trabajo**

1. **Checkout code**: Este paso utiliza la acción actions/checkout para clonar el repositorio en el entorno de ejecución.
2. **Set up Terraform**: Este paso configura Terraform en el entorno de ejecución.
3. **Configure AWS credentials using OIDC**: Este paso utiliza la acción aws-actions/configure-aws-credentials para asumir el rol de IAM que creamos anteriormente. Debes reemplazar <YOUR\_AWS\_ACCOUNT\_ID> con tu ID de cuenta de AWS.
4. **Terraform Init**: Inicializa el directorio de trabajo de Terraform.
5. **Terraform Apply**: Aplica la configuración de Terraform para crear el bucket S3.
6. **Upload index.html to S3**: Este paso sube el archivo index.html al bucket S3.

**Paso 3: Crear el Archivo index.html**

Crea un archivo index.html en la raíz de tu repositorio con el siguiente contenido:

html

Copiar código

*<!DOCTYPE html>*

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bienvenido a mi Bucket S3</title>

</head>

<body>

<h1>¡Hola, mundo!</h1>

<p>Este es un archivo estático servido desde un bucket S3.</p>

</

**PRACTICA COLABORATIVA:**

**Configurar un Pipeline Completo**

Implementar un workflow completo en equipos.

- Usar Terraform para crear infraestructura en AWS.

- Sincronizar archivos en S3 y limpiar archivos obsoletos.

- Permitir ejecución manual con `workflow\_dispatch`.

Para implementar un pipeline completo utilizando Terraform, GitHub Actions y AWS, seguiremos una serie de pasos que incluyen la creación de la infraestructura en AWS, la sincronización de archivos en S3, la limpieza de archivos obsoletos y la configuración de un flujo de trabajo que permita la ejecución manual. A continuación, te proporcionaré la estructura de carpetas, el código completo para cada archivo y una explicación detallada de cada parte.

**Estructura de Carpetas**

La estructura de carpetas para este proyecto será la siguiente:

Code

Copiar código

.

├── .github

│ └── workflows

│ └── terraform.yml

├── modules

│ └── s3

│ ├── main.tf

│ └── variables.tf

├── index.html

└── main.tf

**Paso 1: Crear los Archivos de Terraform**

Contenido de modules/s3/main.tf

Este archivo define el bucket S3 y la configuración de ciclo de vida.

hcl

Copiar código

provider "aws" {

region = var.region

}

resource "aws\_s3\_bucket" "this" {

bucket = var.bucket\_name

acl = var.public\_access ? "public-read" : "private"

versioning {

enabled = true

}

lifecycle {

prevent\_destroy = true

}

}

resource "aws\_s3\_bucket\_lifecycle\_configuration" "this" {

bucket = aws\_s3\_bucket.this.id

rule {

id = "retention-rule"

status = "Enabled"

expiration {

days = 30

}

}

}

Contenido de modules/s3/variables.tf

Este archivo define las variables necesarias para el módulo S3.

hcl

Copiar código

variable "bucket\_name" {

description = "The name of the S3 bucket"

type = string

}

variable "region" {

description = "The AWS region to create the bucket in"

type = string

}

variable "public\_access" {

description = "Should the bucket be publicly accessible?"

type = bool

default = false

}

Contenido de main.tf

Este archivo principal utiliza el módulo S3 y define la configuración de Terraform.

hcl

Copiar código

module "s3\_bucket" {

source = "./modules/s3"

bucket\_name = "ci-cd-static-app-grupo1"

region = "us-east-1"

public\_access = true

}

**Paso 2: Crear el Archivo index.html**

Este archivo contendrá el contenido que se subirá al bucket S3.

Contenido de index.html

html

Copiar código

*<!DOCTYPE html>*

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Bienvenido a mi Bucket S3</title>

</head>

<body>

<h1>¡Hola, mundo!</h1>

<p>Este es un archivo estático servido desde un bucket S3.</p>

</body>

</html>

**Paso 3: Configurar el Flujo de Trabajo de GitHub Actions**

Contenido de .github/workflows/terraform.yml

Este archivo define el flujo de trabajo de GitHub Actions para crear la infraestructura, sincronizar archivos y limpiar archivos obsoletos.

yaml

Copiar código

name: Terraform

on:

push:

branches:

- main

workflow\_dispatch: *# Permite la ejecución manual del flujo de trabajo*

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Terraform

uses: hashicorp/setup-terraform@v1

with:

terraform\_version: 1.0.0

- name: Configure AWS credentials using OIDC

id: aws-creds

uses: aws-actions/configure-aws-credentials@v1

with:

role-to-assume: arn:aws:iam::<YOUR\_AWS\_ACCOUNT\_ID>:role/GitHubActionsS3AccessRole

aws-region: us-east-1 *# Cambia esto si usas otra región*

- name: Terraform Init

run: terraform init

- name: Terraform Apply

run: terraform apply -auto-approve

- name: Upload index.html to S3

run: aws s3 cp index.html s3://ci-cd-static-app-grupo1/

- name: Sync files to S3

run: aws s3 sync . s3://ci-cd-static-app-grupo1/ --exclude "**\*.tfstate"** --delete

- name: Clean up old files

run: aws s3 rm s

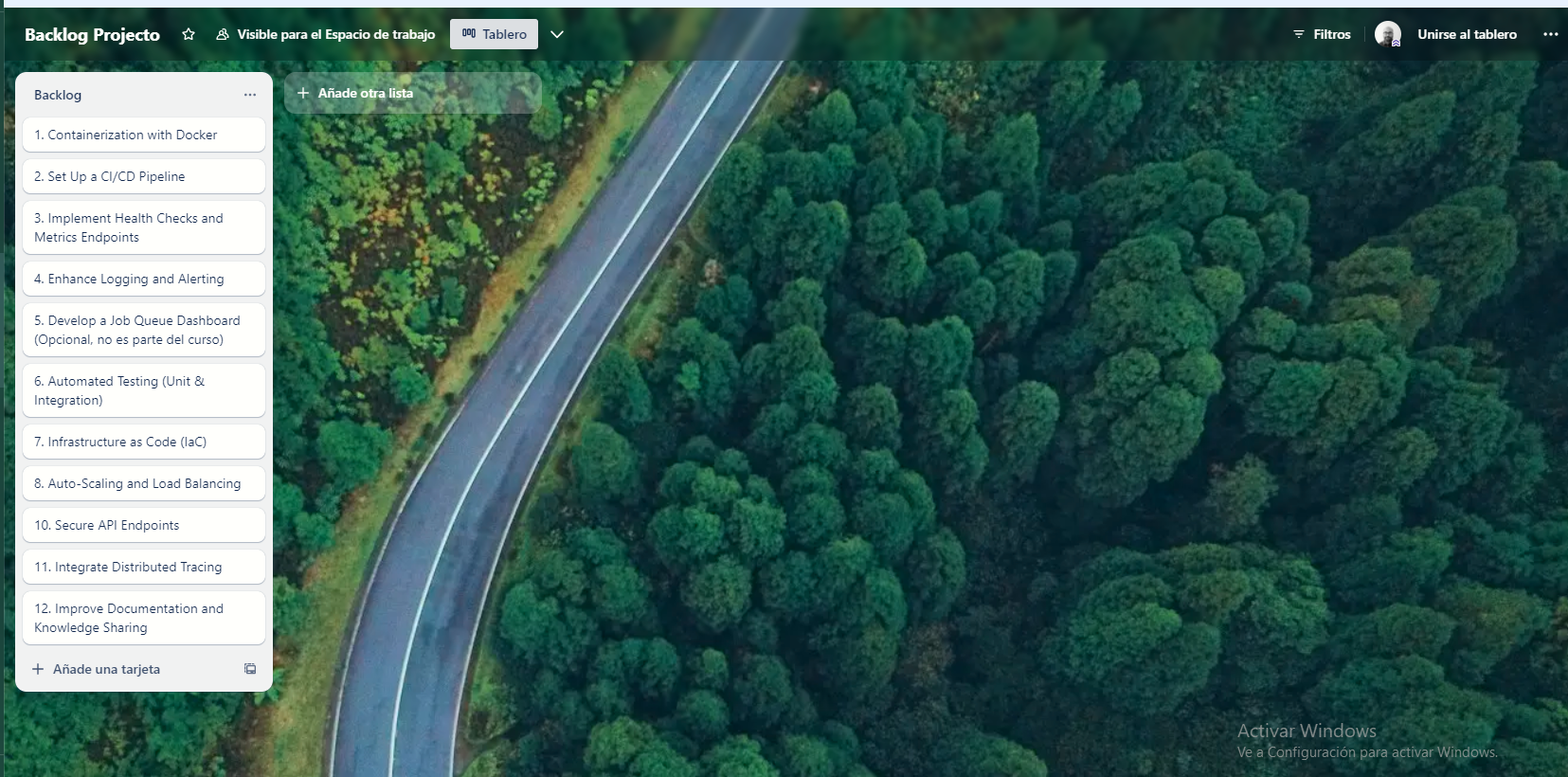
**TRABAJO DEL PROFESOR ANTONIO VEGA**

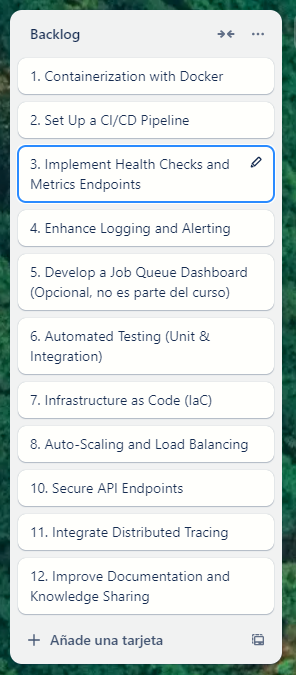
[**https://github.com/tonivega/simple-worker**](https://github.com/tonivega/simple-worker)

[antonio.vega@campusdual.com](mailto:antonio.vega@campusdual.com)

[rosa.ruanova.lagares@xunta.gal](mailto:rosa.ruanova.lagares@xunta.gal)

<https://trello.com/invite/67a50494bec03be9aa24fa2e/ATTI4afd000f2c3273c797a1cb5e11ff560111C116D7>





his branch is up to date with tonivega/simple-worker:develop.

ContributeSync fork

**Folders and files**

| **Name** |  |  |
| --- | --- | --- |
| **Latest commit**  author  **Toni Vega**  [[develop] Initial commit](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15)  [f1ada9e](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15) · 1 hour ago  **History**  [5 Commits](https://github.com/campusdualdevopsGrupo2/simple-worker/commits/develop/) | | |
| [README.md](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/README.md) | [Update README.md](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/223a076d638db63fb45fb0b72d52a898a0da37e9) | 4 days ago |
| [bluengo-worker](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/bluengo-worker) | [Add files via upload](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/29c89ebdbaa98cf4b8cd88af238a4d55f066f518) | 4 days ago |
| [build.sh](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/build.sh) | [[develop] Initial commit](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15) | 1 hour ago |
| [go.mod](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/go.mod) | [[develop] Initial commit](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15) | 1 hour ago |
| [go.sum](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/go.sum) | [[develop] Initial commit](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15) | 1 hour ago |
| [main.go](https://github.com/campusdualdevopsGrupo2/simple-worker/blob/develop/main.go) | [[develop] Initial commit](https://github.com/campusdualdevopsGrupo2/simple-worker/commit/f1ada9e295d2b57caf525219e3c36f38f4d8bd15) | 1 hour ago |

**Repository files navigation**

* [README](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop)

**Bluengo Simple Worker**

Poor's man, yet effective, job queue/worker. Should run fine as is in modern Linux releases.

**See our office webcam at**[**https://www.bluengo.com**](https://www.bluengo.com/)

Bluengo Simple Worker is a lightweight distributed job processing system written in Go. It consists of three subcommands:

* **server**: Runs the HTTP job server.
* **worker**: Runs a worker that polls the server for jobs and executes them.
* **add**: Adds a new job to the server's job queue.

**Table of Contents**

* [Usage](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#usage)
  + [Server](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#server)
  + [Worker](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#worker)
  + [Add Job](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#add-job)
* [Examples](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#examples)
* [Debug Mode](https://github.com/campusdualdevopsGrupo2/simple-worker/tree/develop#debug-mode)

**Usage**

The program uses subcommands to start the server, worker, or add jobs. The general usage is:

./bluengo <subcommand> [flags...]

**Server**

Start the job server which listens for incoming job requests and job polls from workers.

**Usage:**

./bluengo server [flags]

**Flags:**

* -port  
  Port for the server (default: 8080).
* -password  
  Password for authenticating requests. If set, workers and job submissions must include this password in the X-Job-Password header.
* -debug  
  Enable debug logging (default: false).

**Example:**

./bluengo server -port 8080 -password mysecret -debug

**Worker**

Run a worker that polls the job server and executes jobs concurrently.

**Usage:**

./bluengo worker [flags]

**Flags:**

* -server  
  URL of the job server (default: http://localhost:8080).
* -slots  
  Number of concurrent job slots (default: number of CPU cores).
* -poll  
  Poll interval in seconds (default: 1).
* -password  
  Password for authenticating with the server.
* -debug  
  Enable debug logging (default: false).

**Example:**

./bluengo worker -server http://localhost:8080 -slots 4 -poll 2 -password mysecret -debug

**Add Job**

Add a new job to the job server's queue.

**Usage:**

./bluengo add [flags]

**Flags:**

* -server  
  URL of the job server (default: http://localhost:8080).
* -cmd  
  The shell command to execute (required).
* -timeout  
  Timeout in seconds for the job execution (default: 10).
* -password  
  Password for authenticating with the server.
* -debug  
  Enable debug logging (default: false).

**Example:**

./bluengo add -server http://localhost:8080 -cmd "echo 'Hello, World!'" -timeout 5 -password mysecret -debug

**Examples**

1. **Start the server:**

./bluengo server -port 8080 -password mysecret

1. **Start a worker:**

./bluengo worker -server http://localhost:8080 -slots 4 -poll 2 -password mysecret

1. **Submit a job:**

./bluengo add -server http://localhost:8080 -cmd "sleep 10" -timeout 15 -password mysecret

**Debug Mode**

To enable verbose logging for troubleshooting, add the -debug flag to any subcommand:

./bluengo worker -server http://localhost:8080 -slots 4 -debug

**License**

This project is a proprietary project.

Imaginate que eres un senior devops ayudame con este ejerciciode manera codificadopara dockerizar esta aplicacion:

his branch is up to date with tonivega/simple-worker:develop. ContributeSync fork Folders and files Name Latest commit **Toni Vega** [develop] Initial commit f1ada9e · 1 hour ago History 5 Commits README.md Update README.md 4 days ago bluengo-worker Add files via upload 4 days ago build.sh [develop] Initial commit 1 hour ago go.mod [develop] Initial commit 1 hour ago go.sum [develop] Initial commit 1 hour ago main.go [develop] Initial commit 1 hour ago Repository files navigation \* README Bluengo Simple Worker Poor's man, yet effective, job queue/worker. Should run fine as is in modern Linux releases. **See our office webcam at** [**https://www.bluengo.com**](https://www.bluengo.com/) Bluengo Simple Worker is a lightweight distributed job processing system written in Go. It consists of three subcommands: \* **server**: Runs the HTTP job server. \* **worker**: Runs a worker that polls the server for jobs and executes them. \* **add**: Adds a new job to the server's job queue. Table of Contents \* Usage \* Server \* Worker \* Add Job \* Examples \* Debug Mode Usage The program uses subcommands to start the server, worker, or add jobs. The general usage is:

Copy

./bluengo <subcommand> [flags...]

Server Start the job server which listens for incoming job requests and job polls from workers. **Usage:**

Copy

./bluengo server [flags]

**Flags:** \* -port Port for the server (default: 8080). \* -password Password for authenticating requests. If set, workers and job submissions must include this password in the X-Job-Password header. \* -debug Enable debug logging (default: false). **Example:**

Copy

./bluengo server -port 8080 -password mysecret -debug

Worker Run a worker that polls the job server and executes jobs concurrently. **Usage:**

Copy

./bluengo worker [flags]

**Flags:** \* -server URL of the job server (default: http://localhost:8080). \* -slots Number of concurrent job slots (default: number of CPU cores). \* -poll Poll interval in seconds (default: 1). \* -password Password for authenticating with the server. \* -debug Enable debug logging (default: false). **Example:**

Copy

./bluengo worker -server http://localhost:8080 -slots 4 -poll 2 -password mysecret -debug

Add Job Add a new job to the job server's queue. **Usage:**

Copy

./bluengo add [flags]

**Flags:** \* -server URL of the job server (default: http://localhost:8080). \* -cmd The shell command to execute (required). \* -timeout Timeout in seconds for the job execution (default: 10). \* -password Password for authenticating with the server. \* -debug Enable debug logging (default: false). **Example:**

Copy

./bluengo add -server http://localhost:8080 -cmd "echo 'Hello, World!'" -timeout 5 -password mysecret -debug

Examples 1. **Start the server:**

Copy

./bluengo server -port 8080 -password mysecret

1. **Start a worker:**

Copy

./bluengo worker -server http://localhost:8080 -slots 4 -poll 2 -password mysecret

1. **Submit a job:**

Copy

./bluengo add -server http://localhost:8080 -cmd "sleep 10" -timeout 15 -password mysecret

Debug Mode To enable verbose logging for troubleshooting, add the -debug flag to any subcommand:

Copy

./bluengo worker -server http://localhost:8080 -slots 4 -debug

License This project is a proprietary project.

Te ayudo a dockerizar esta aplicación Go. Necesitaremos crear un Dockerfile multistage para optimizar el tamaño de la imagen.

Dockerfile for Bluengo Worker

Click to open code

También necesitaremos un script para construir la imagen:

Docker Build Script

Click to open code

Para usar la aplicación dockerizada:

1. Da permisos de ejecución al script:

bash

Copy

chmod +x build.sh

1. Construye la imagen:

bash

Copy

./build.sh

1. Ejecuta el servidor:

bash

Copy

docker run -p 8080:8080 bluengo-worker:latest server -port 8080

1. Ejecuta workers en contenedores separados:

bash

Copy

docker run bluengo-worker:latest worker -server http://host.docker.internal:8080

¿Necesitas alguna aclaración o ajuste específico?

*# Build stage*

FROM golang:1.21-alpine AS builder

WORKDIR /app

COPY . .

RUN CGO\_ENABLED=0 GOOS=linux go build -o bluengo

*# Final stage*

FROM alpine:3.19

WORKDIR /app

COPY --from=builder /app/bluengo .

EXPOSE 8080

ENTRYPOINT ["./bluengo"]

CMD ["server", "-port", "8080"]

#!/bin/bash

IMAGE\_NAME="bluengo-worker"

TAG="latest"

*# Build the Docker image*

docker build -t ${IMAGE\_NAME}:${TAG} .

*# Instrucciones de uso*

echo "Imagen construida como ${IMAGE\_NAME}:${TAG}"

echo ""

echo "Para ejecutar el servidor:"

echo "docker run -p 8080:8080 ${IMAGE\_NAME}:${TAG} server -port 8080"

echo ""

echo "Para ejecutar un worker:"

echo "docker run ${IMAGE\_NAME}:${TAG} worker -server http://host.docker.internal:8080"

se puede hacer lel build de la imagen con docker buil en lugar que con el buil script, si es asi detallamelo?

Sí, aquí tienes el comando directo de Docker build para construir la imagen:

Dockerfile Multistage

Click to open code

Comando para construir:

bash

Copy

docker build -t bluengo-worker:latest .

Para ejecutar:

bash

Copy

*# Servidor*

docker run -p 8080:8080 bluengo-worker:latest server -port 8080

*# Worker*

docker run bluengo-worker:latest worker -server http://host.docker.internal:8080

 CopyRetry

Claude does not have the ability to run the code it generates yet.

**WEBEX 07/02/2025**

**PROYECTO FINAL**

**GOLANG**

Ahora como se haria para hacer infraestructura como codigo con este ejercicio de la aplicacion en go, dame la estructura de carpetas, con la codificacion de archivos y su explicacion, por favor?

Te proporcionaré una estructura de Infrastructure as Code (IaC) para la aplicación Bluengo Worker usando Terraform:

Estructura de carpetas:

Copy

bluengo-iac/

│

├── terraform/

│ ├── main.tf

│ ├── variables.tf

│ ├── outputs.tf

│ └── versions.tf

│

├── docker/

│ ├── Dockerfile

│ └── docker-compose.yml

│

├── k8s/

│ ├── deployment.yaml

│ ├── service.yaml

│ └── configmap.yaml

│

└── scripts/

├── build.sh

└── deploy.sh

1. Configuración de Terraform (AWS):

Terraform Main Configuration

Click to open code

1. Variables de Terraform:

Terraform Variables

Click to open code

1. Docker Compose:

Docker Compose Configuration

Click to open code

1. Kubernetes Deployment:

Kubernetes Deployment

Click to open code

1. Script de Despliegue:

Deployment Script

Click to open code

Pasos para implementar:

1. Construir imagen:

bash

Copy

docker build -t bluengo-worker:latest .

1. Desplegar con Docker Compose:

bash

Copy

docker-compose up -d

1. Desplegar en Kubernetes:

bash

Copy

kubectl apply -f k8s/

1. Desplegar en AWS (Terraform):

bash

Copy

terraform init

terraform plan

terraform apply

¿Necesitas más detalles sobre alguna parte de la infraestructura?

**AKIATTSKF76KMMWVBNY6  
ACCESS ID**

**SECRET  
qKkph84dfgkqB9X+2EcfPZWaxxXXXL6GFTzwkEDB**

**WEBEX 10/02/2025**

**CICD GITHUB ACTIONS**

**CONTINUACION PARTE 2 TRELLO:**

**PRACTICA COLABORATIVA SETUP CI/CD PIPELINE CON TERRAFORM EN ECS DE AWS**

Ahora una vez dockerizado deberiamos hacer un setup ci/cd pipeline para con terraform levantar infraestructura en un ec2.Podrias en base a los ejercicios y datos anteriores, proporcionarme la estructura de carpetas correcta, asi como la codificacion de cada uno de los archivos y explicacion detallada, por favor?

Te ayudo a crear una estructura completa de CI/CD con Terraform para desplegar en EC2:

Estructura de carpetas:

Copy

proyecto/

├── .github/

│ └── workflows/

│ ├── deploy.yml

│ └── terraform.yml

├── terraform/

│ ├── main.tf

│ ├── variables.tf

│ ├── outputs.tf

│ ├── provider.tf

│ └── data.tf

├── docker/

│ ├── Dockerfile.server

│ └── Dockerfile.worker

└── scripts/

└── user\_data.sh

1. GitHub Actions Workflow para Terraform:

Terraform GitHub Actions Workflow

Click to open code

1. Configuración principal de Terraform:

Main Terraform Configuration

Click to open code

1. Variables de Terraform:

Terraform Variables

Click to open code

1. Script de inicialización del servidor:

EC2 User Data Script

Click to open code

Explicación del flujo:

1. Cuando se hace push a main o develop:
   * GitHub Actions ejecuta el workflow de Terraform
   * Se configuran las credenciales de AWS
   * Se inicializa Terraform
   * Se planifica y aplica la infraestructura
2. La infraestructura creada incluye:
   * VPC con subnet pública
   * Security Group para el tráfico necesario
   * Instancia EC2 con user\_data script
3. El script user\_data:
   * Instala Docker y AWS CLI
   * Configura acceso a ECR
   * Descarga y ejecuta los contenedores
4. Variables configurables:
   * Región AWS
   * CIDR blocks
   * Tipo de instancia
   * AMI ID

¿Necesitas más detalles sobre alguna parte específica?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

ecr\_repository\_url = "248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy"

instance\_id = "i-02a994e6d509a74c1"

instance\_public\_ip = "18.201.113.185"

public\_subnet\_id = "subnet-05e7139f63178cc58"

security\_group\_id = "sg-0843ffb383442264d"

vpc\_id = "vpc-09dd839bace5349a5"

**Fases del Flujo CI/CD**

Este proceso se compone de tres grandes fases:

1️⃣ **Infraestructura como Código (Terraform)** → Se asegura que AWS tenga todo listo.  
2️⃣ **Construcción y despliegue de contenedores (Docker + ECR)** → Se crean y suben las imágenes.  
3️⃣ **Ejecución en EC2 (o ECS si decides escalar)** → Se actualizan y ejecutan los contenedores en la infraestructura.

**Paso a Paso: Despliegue Completo**

**1️⃣ Fase de Infraestructura (Terraform)**

✅ **Acción:** Aplicamos Terraform para crear/actualizar la infraestructura.  
✅ **Resultado:** AWS tendrá:

* Una **VPC con subred pública**.
* Una **instancia EC2** con permisos para extraer imágenes de **ECR**.
* Un **grupo de seguridad** que permite el tráfico a la aplicación.
* Un **repositorio en ECR** para almacenar imágenes Docker.

🔹 **Comandos en la máquina local o en GitHub Actions**:

**2️⃣ Fase de CI: Construcción y Publicación de la Imagen en AWS ECR**

✅ **Acción:** Se construyen y publican las imágenes Docker en Amazon ECR.  
✅ **Resultado:** La imagen estará disponible en **ECR** para ser utilizada por EC2.

🔹 **Flujo del pipeline (deploy\_ecr.yml)**:

**3️⃣ Fase de CD: Despliegue en la instancia EC2**

✅ **Acción:** EC2 detecta la nueva imagen y la ejecuta automáticamente.  
✅ **Resultado:** La aplicación actualizada se ejecuta en EC2.

**📌 Métodos posibles para desplegar la imagen en EC2**

Hay tres formas de ejecutar el contenedor en EC2 sin usar scripts (user\_data.sh):

🔹 **Método 1: Terraform con aws\_instance + remote-exec**  
Terraform ejecuta los comandos en la instancia EC2:

**4️⃣ Validación**

✅ **Verificar que el contenedor está corriendo en EC2**

sh

CopiarEditar

ssh -i my-key.pem ec2-user@<EC2\_PUBLIC\_IP>

docker ps

✅ **Probar que la aplicación responde**

sh

CopiarEditar

curl http://<EC2\_PUBLIC\_IP>:8080

**Resumen Final**

| **Paso** | **Acción** | **Resultado** |
| --- | --- | --- |
| **1️.Terraform** | terraform apply | Infraestructura creada en AWS |
| **2️.CI** (GitHub Actions) | Build & Push a ECR | Imagen Docker disponible en AWS ECR |
| **3️.CD** (Terraform o EC2) | EC2 descarga imagen y ejecuta contenedor | Aplicación corriendo en EC2 |
| **4️.Validación** | docker ps y curl | Se verifica que todo funciona |
| **📢 Conclusión**  ✔️ Terraform **gestiona toda la infraestructura** sin scripts manuales. ✔️ GitHub Actions **automiza la compilación y despliegue** de imágenes. ✔️ EC2 **ejecuta los contenedores** sin necesidad de user\_data.sh.  🔹 **¿Te gustaría integrar autoscaling con ECS en lugar de EC2 para hacerlo más escalable?** |  |  |

El archivo terraform\_deploy.yaml **no suple completamente** la necesidad de arrancar el contenedor en EC2 automáticamente. Lo que hace este pipeline es **ejecutar Terraform**, pero **Terraform solo provisiona la infraestructura** (EC2, ECR, redes, etc.), no gestiona directamente el arranque de contenedores dentro de EC2 después de crearse.

**🔍 ¿Por qué deploy.yaml no es suficiente?**

El pipeline de GitHub Actions (deploy.yaml) **ejecuta Terraform** en una máquina de GitHub, pero **no tiene acceso a la EC2 para ejecutar comandos dentro de ella**.

Cuando Terraform crea una instancia EC2, **esta es un servidor vacío** que aún no sabe que tiene que ejecutar el contenedor. Por eso, se necesitan mecanismos adicionales dentro de la EC2 para que esta descargue y ejecute el contenedor de forma automática.

**🛠 Opciones para que EC2 arranque automáticamente el contenedor**

Si quieres **evitar scripts externos**, aquí hay **tres formas recomendadas** para que EC2 arranque automáticamente el contenedor tras la creación de la infraestructura.

**1️⃣ Opción: Provisioners en Terraform (remote-exec)**

Terraform puede conectarse a la EC2 vía SSH y ejecutar los comandos necesarios para instalar Docker y ejecutar el contenedor:

hcl

CopiarEditar

resource "aws\_instance" "bluengo\_server" {

ami = var.ami\_id

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.public.id

vpc\_security\_group\_ids = [aws\_security\_group.bluengo\_sg.id]

key\_name = var.key\_name # Clave SSH para conectarse a EC2

provisioner "remote-exec" {

inline = [

"sudo yum install -y docker",

"sudo service docker start",

"aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin 248189943700.dkr.ecr.eu-west-1.amazonaws.com",

"docker pull 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest",

"docker run -d -p 8080:8080 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest"

]

}

connection {

type = "ssh"

user = "ec2-user"

private\_key = file("~/.ssh/my-key.pem") # Ruta de la clave privada

host = self.public\_ip

}

tags = {

Name = "bluengo-server"

}

}

📌 **Ventaja**: Se ejecuta al instante tras la creación de EC2.  
📌 **Desventaja**: Si EC2 se reinicia, el contenedor se pierde y no se ejecutará automáticamente.

**2️⃣ Opción: user\_data en Terraform (Sin scripts externos)**

En lugar de usar remote-exec, puedes usar user\_data para que EC2 arranque el contenedor automáticamente en cada inicio:

hcl

CopiarEditar

resource "aws\_instance" "bluengo\_server" {

ami = var.ami\_id

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.public.id

vpc\_security\_group\_ids = [aws\_security\_group.bluengo\_sg.id]

user\_data = <<-EOF

#!/bin/bash

sudo yum update -y

sudo yum install -y docker

sudo service docker start

sudo systemctl enable docker

aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin 248189943700.dkr.ecr.eu-west-1.amazonaws.com

docker pull 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest

docker run -d -p 8080:8080 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest

EOF

tags = {

Name = "bluengo-server"

}

}

📌 **Ventaja**: Funciona **sin necesidad de SSH ni scripts adicionales**.  
📌 **Desventaja**: Si la instancia se reinicia, el contenedor **no se ejecuta automáticamente**.

**3️⃣ Opción Recomendada: systemd para arranque persistente en EC2**

Para asegurarte de que el contenedor **se inicie automáticamente después de un reinicio**, usa systemd para crear un servicio persistente:

hcl

CopiarEditar

resource "aws\_instance" "bluengo\_server" {

ami = var.ami\_id

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.public.id

vpc\_security\_group\_ids = [aws\_security\_group.bluengo\_sg.id]

user\_data = <<-EOF

#!/bin/bash

sudo yum update -y

sudo yum install -y docker

sudo service docker start

sudo systemctl enable docker

aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin 248189943700.dkr.ecr.eu-west-1.amazonaws.com

docker pull 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest

echo "[Unit]

Description=MyApp Container

After=network.target

[Service]

ExecStart=/usr/bin/docker run -p 8080:8080 --restart unless-stopped 248189943700.dkr.ecr.eu-west-1.amazonaws.com/ecr-jgl-deploy:latest

Restart=always

[Install]

WantedBy=multi-user.target" | sudo tee /etc/systemd/system/myapp.service

sudo systemctl enable myapp

sudo systemctl start myapp

EOF

tags = {

Name = "bluengo-server"

}

}

📌 **Ventaja**:  
✅ **Arranca automáticamente** cuando EC2 se reinicia.  
✅ **No depende de Terraform después del primer arranque**.  
✅ **Evita manualmente correr docker run cada vez**.

📌 **Desventaja**: No hay muchas. **Es la mejor opción** si usas EC2.

**🔍 Resumen y Elección Final**

| **Método** | **Auto-arranque tras terraform apply** | **Auto-arranque tras reinicio de EC2** | **Requiere conexión SSH** |
| --- | --- | --- | --- |
| **remote-exec** (SSH desde Terraform) | ✅ Sí | ❌ No | ✅ Sí |
| **user\_data** (Bootstrap en Terraform) | ✅ Sí | ❌ No | ❌ No |
| **systemd en EC2** (**Recomendada**) | ✅ Sí | ✅ Sí | ❌ No |

Si **quieres que el contenedor siempre se ejecute**, la opción con systemd es **la más recomendada**.

🔹 **¿Quieres eliminar EC2 y usar ECS con Fargate para mayor escalabilidad?** 🚀

**🚀 Paso a Paso: Puesta en Marcha del Proyecto**

Aquí tienes el proceso detallado de cómo todo se pone en marcha:

**1️⃣ Primera Configuración en GitHub y AWS**

* Crear un **repositorio GitHub** y agregar los secretos (AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY).
* Configurar AWS con los permisos necesarios para EC2, ECR y S3.

**2️⃣ Construcción y Subida de la Imagen a ECR**

📌 Este paso lo hace automáticamente el pipeline deploy\_ecr.yaml cuando subes código a develop:

bash

CopiarEditar

git push origin develop

🔹 Se ejecutan los siguientes pasos:

1. Se construye la imagen con docker build
2. Se sube la imagen a Amazon ECR con docker push

**3️⃣ Creación de la Infraestructura con Terraform**

📌 El pipeline deploy.yaml en GitHub Actions ejecuta terraform apply, lo cual:

1. Crea la VPC, subredes y security groups.
2. Lanza una instancia EC2.
3. EC2 ejecuta automáticamente el user\_data para instalar Docker y ejecutar el contenedor.

**4️⃣ Verificación**

Cuando la infraestructura esté lista:

bash

CopiarEditar

terraform output

Esto te dará la **IP pública** de EC2.  
Luego accede a:

plaintext

CopiarEditar

http://<EC2\_PUBLIC\_IP>:8080

para verificar que el servicio está corriendo.

**5️⃣ Deploys Automatizados**

* Cada vez que **hagas un push** a develop, el pipeline:
  + Construirá la nueva imagen Docker y la subirá a ECR.
  + Terraform NO será ejecutado, ya que la infraestructura no cambia.
  + EC2 **no descargará la nueva imagen automáticamente**.

**6️⃣ 🚀 (Opcional) Implementar un systemd o un cron job en EC2**

Para que EC2 actualice automáticamente la imagen cada cierto tiempo, puedes usar un cron job o un systemd service.

**📌 Conclusión**

1. **deploy\_ecr.yaml** maneja la construcción y subida de imágenes Docker a ECR.
2. **deploy.yaml** usa Terraform para crear y administrar la infraestructura en AWS.
3. **EC2 ejecuta automáticamente el contenedor** gracias al user\_data.
4. **Cada push a develop sube una nueva imagen a ECR**, pero **EC2 no se actualiza automáticamente** sin un mecanismo extra.

Si necesitas que EC2 **descargue y actualice automáticamente** la imagen de ECR sin reiniciarse, puedo ayudarte a configurar systemd o un cron job en EC2. 🚀

**WEBEX 10/02/2025**

**CICD GITHUB ACTIONS**

1. **Organizar Github**

<https://nvie.com/post/a-successful-git-branching-model>.

[www.toptal.com/software/trunk-based-development-git-flow](http://www.toptal.com/software/trunk-based-development-git-flow)

**Tranbase es ideal para trabajar en este caso**.

construir ejecutables para windows, linux (glib 2.27), arm64 y amd64  
aumentar revision de semver  
publicar version en github

<https://stackoverflow.com/a/47837371/2335377>

<https://stackoverflow.com/a/74955554/2335377>

<https://github.com/softprops/action-gh-release>

Te ayudaré a crear una estructura de CI/CD para compilar y publicar los ejecutables para diferentes plataformas usando GitHub Actions.

GitHub Actions Workflow for Multi-Platform Build

Click to open code

Para implementar esta solución, sigue estos pasos:

1. Crea la estructura de carpetas:

Copy

.github/

workflows/

release.yml # El archivo de workflow que creamos arriba

1. Asegúrate de que tu main.go tenga una variable para la versión:

go

Copy

package main

var Version = "dev"

func main() {

*// Tu código actual*

}

1. Actualiza el sistema de versiones usando semver en los tags de git:

bash

Copy

*# Para una nueva versión*

git tag v1.0.0

git push origin v1.0.0

*# Para una revisión*

git tag v1.0.1

git push origin v1.0.1

1. Configuración adicional recomendada:

Añade un archivo .goreleaser.yml en la raíz del proyecto para una configuración más avanzada de releases:

GoReleaser Configuration

Click to open code

1. Flujo de trabajo para publicar una nueva versión:

bash

Copy

*# 1. Actualiza tu código y haz commit*

git add .

git commit -m "feat: nueva funcionalidad"

*# 2. Crea y pushea un nuevo tag siguiendo semver*

git tag v1.0.0

git push origin v1.0.0

*# 3. El workflow se ejecutará automáticamente*

El workflow hará lo siguiente:

* Se activará cuando se pushee un nuevo tag que comience con 'v'
* Compilará el binario para Windows, Linux AMD64 y ARM64
* Creará checksums de los archivos
* Creará una nueva release en GitHub con los binarios y checksums
* Generará notas de release automáticamente

Consideraciones adicionales:

* Los binarios de Linux se compilarán con CGO\_ENABLED=0 para evitar dependencias de glibc
* Los nombres de los archivos incluirán la plataforma y arquitectura
* Se generan checksums SHA256 para verificar la integridad
* La versión se inyecta en el binario durante la compilación

¿Necesitas que ajuste alguna parte de la configuración o que explique algo con más detalle?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

HE

para que desde el codigo lo lance a la rema develop como lo hago y sin usar goreleaser aconfiguration como cosas avanzadas, por favor

Edit

Te ayudo a simplificar el proceso para lanzar los builds desde la rama develop.

GitHub Actions Workflow for Develop Branch

Click to open code

Para implementar esto, sigue estos pasos:

1. Crea o modifica el archivo en .github/workflows/develop-build.yml con el contenido del artifact anterior.
2. Asegúrate de que tu main.go tenga la variable de versión:

go

Copy

package main

var Version = "dev"

func main() {

*// Tu código actual*

}

1. Workflow de trabajo:

bash

Copy

*# 1. Trabaja en tu rama local*

git checkout -b feature/nueva-funcionalidad

*# 2. Haz tus cambios y commits*

git add .

git commit -m "feat: nueva funcionalidad"

*# 3. Merge a develop*

git checkout develop

git merge feature/nueva-funcionalidad

*# 4. Push a develop*

git push origin develop

Este workflow:

* Se ejecuta automáticamente cada vez que hay un push a la rama develop
* Genera una versión basada en la fecha (ejemplo: 2025.02.10-develop)
* Compila para las tres plataformas solicitadas
* Crea una release marcada como "pre-release" en GitHub
* No requiere configuración adicional ni herramientas externas

Los archivos generados serán:

Copy

build/

├── bluengo-worker-windows-amd64.exe

├── bluengo-worker-linux-amd64

├── bluengo-worker-linux-arm64

└── checksums.txt

¿Necesitas que ajuste algo o tienes alguna pregunta sobre el proceso?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

HE

este codigo no mefunciona se queda el github actions en espera con este error:

Requested labels: ubuntu-18.04 Job defined at: javier-guerra-campusdual/simple-worker/.github/workflows/release.yaml@refs/heads/develop Waiting for a runner to pick up this job...

el codigo que yo he usado:

name: Build and Release

on: push: branches: - develop tags: - 'v\*'

env: GO\_VERSION: '1.23' BINARY\_NAME: 'bluengo-worker'

jobs: build: name: Build Multi-Platform runs-on: ubuntu-18.04

steps: - name: Checkout code uses: actions/checkout@v4

* name: Set up Go uses: actions/setup-go@v4 with: go-version: ${{ env.GO\_VERSION }}
* name: Get version from tag id: get\_version run: echo "VERSION=${GITHUB\_REF#refs/tags/}" >> $GITHUB\_ENV
* name: Build for multiple platforms run: | # Create build directory mkdir -p build # Build for Windows AMD64 GOOS=windows GOARCH=amd64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_windows\_amd64.exe main.go # Build for Linux AMD64 GOOS=linux GOARCH=amd64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_linux\_amd64 main.go # Build for Linux ARM64 GOOS=linux GOARCH=arm64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_linux\_arm64 main.go # Create checksums cd build sha256sum \* > checksums.txt
* name: Create Release id: create\_release uses: softprops/action-gh-release@v1 with: files: | build/\* draft: false prerelease: false generate\_release\_notes: true env: GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}

Edit

Veo varios problemas en el código que están causando el error. Te ayudo a corregirlos:

1. ubuntu-18.04 está deprecado, debes usar una versión más reciente
2. La versión de Go 1.23 no existe (la última estable es 1.21)
3. Hay una mezcla de triggers que puede causar confusión
4. Falta el manejo de versión cuando se ejecuta desde develop

Aquí está el código corregido:

Fixed GitHub Actions Workflow

Click to open code

Cambios principales realizados:

1. Cambiado ubuntu-18.04 a ubuntu-latest
2. Corregida la versión de Go a 1.21
3. Mejorado el manejo de versiones:
   * Para tags: usa el número de versión del tag
   * Para develop: usa la fecha como versión
4. Añadido tag\_name en la creación del release
5. Configurado prerelease para que sea true en develop y false en tags

Para usar este workflow:

1. Para desarrollo normal:

bash

Copy

git checkout develop

git add .

git commit -m "feat: nueva funcionalidad"

git push origin develop

1. Para releases:

bash

Copy

git tag v1.0.0

git push origin v1.0.0

Esto debería resolver el problema del runner y permitir que el workflow se ejecute correctamente. ¿Necesitas que ajuste algo más?

podrias explicarme el codigo linea por linea y decirme por que no usas matrix strategy, por favor?

Te explico el código línea por línea y luego te mostraré una versión con matrix strategy para que veas la diferencia.

yaml

Copy

*# Define el nombre del workflow que aparecerá en GitHub Actions*

name: Build and Release

*# Define cuándo se ejecutará el workflow*

on:

push:

branches:

- develop *# Se ejecuta cuando hay push a develop*

tags:

- 'v\*' *# Se ejecuta cuando se crea un tag que empiece por 'v'*

*# Variables de entorno globales para todo el workflow*

env:

GO\_VERSION: '1.21' *# Versión de Go a usar*

BINARY\_NAME: 'bluengo-worker' *# Nombre base del ejecutable*

*# Lista de trabajos a ejecutar*

jobs:

build:

name: Build Multi-Platform

runs-on: ubuntu-latest *# Usa la última versión de Ubuntu*

steps:

*# Paso 1: Obtiene el código del repositorio*

- name: Checkout code

uses: actions/checkout@v4

*# Paso 2: Configura Go en el runner*

- name: Set up Go

uses: actions/setup-go@v4

with:

go-version: ${{ env.GO\_VERSION }}

*# Paso 3: Establece la versión según el contexto*

- name: Set version

id: set\_version

run: |

if [[ $GITHUB\_REF == refs/tags/\* ]]; then

echo "VERSION=${GITHUB\_REF#refs/tags/}" >> $GITHUB\_ENV # Si es un tag, usa su valor

else

echo "VERSION=$(date +'%Y.%m.%d')-develop" >> $GITHUB\_ENV # Si no, usa la fecha

fi

*# Paso 4: Compila para múltiples plataformas*

- name: Build for multiple platforms

run: |

# Crea el directorio de build

mkdir -p build

*# Compila para Windows AMD64*

GOOS=windows GOARCH=amd64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_windows\_amd64.exe main.go

*# Compila para Linux AMD64*

GOOS=linux GOARCH=amd64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_linux\_amd64 main.go

*# Compila para Linux ARM64*

GOOS=linux GOARCH=arm64 go build -ldflags="-X main.Version=${{ env.VERSION }}" -o build/${BINARY\_NAME}\_linux\_arm64 main.go

*# Genera checksums*

cd build

sha256sum \* > checksums.txt

*# Paso 5: Crea la release en GitHub*

- name: Create Release

id: create\_release

uses: softprops/action-gh-release@v1

with:

files: |

build/\* # Todos los archivos en build/

tag\_name: ${{ env.VERSION }} *# Usa la versión como nombre del tag*

draft: false *# No es un borrador*

prerelease: ${{ github.ref != 'refs/heads/develop' }} *# Es prerelease si no viene de develop*

generate\_release\_notes: true *# Genera notas automáticamente*

env:

GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }} *# Token para autenticación*

No usé matrix strategy por dos razones principales:

1. Simplicidad: Para casos simples con pocos targets, un script directo es más fácil de entender y mantener
2. Control de Release: Al hacer todo en un solo job, es más fácil crear una única release con todos los artefactos

Sin embargo, aquí te muestro cómo sería usando matrix strategy:

GitHub Actions Workflow with Matrix Strategy

Click to open code

Ventajas de usar matrix:

1. Paralelización: Las builds se ejecutan en paralelo
2. Mejor organización: Cada combinación OS/arch es un job separado
3. Más flexible: Fácil de añadir nuevas plataformas
4. Mejor manejo de errores: Si falla una build, las otras continúan

Desventajas:

1. Más complejo de configurar
2. Requiere un job adicional para la release
3. Mayor uso de recursos de GitHub Actions
4. Puede ser más lento en total debido al overhead de múltiples jobs

¿Te gustaría que profundice en algún aspecto específico de cualquiera de las dos implementaciones?

name: Build and Release

on:

push:

branches:

- develop

tags:

- 'v\*'

env:

GO\_VERSION: '1.21'

BINARY\_NAME: 'bluengo-worker'

jobs:

build:

name: Build ${{ matrix.os }}-${{ matrix.arch }}

runs-on: ubuntu-latest

strategy:

matrix:

include:

- os: windows

arch: amd64

extension: .exe

- os: linux

arch: amd64

extension: ''

- os: linux

arch: arm64

extension: ''

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Set up Go

uses: actions/setup-go@v4

with:

go-version: ${{ env.GO\_VERSION }}

- name: Set version

id: set\_version

run: |

if [[ $GITHUB\_REF == refs/tags/\* ]]; then

echo "VERSION=${GITHUB\_REF#refs/tags/}" >> $GITHUB\_ENV

else

echo "VERSION=$(date +'%Y.%m.%d')-develop" >> $GITHUB\_ENV

fi

- name: Build

run: |

mkdir -p build

GOOS=${{ matrix.os }} GOARCH=${{ matrix.arch }} go build \

-ldflags="-X main.Version=${{ env.VERSION }}" \

-o build/${BINARY\_NAME}\_${{ matrix.os }}\_${{ matrix.arch }}${{ matrix.extension }} \

main.go

- name: Upload artifact

uses: actions/upload-artifact@v3

with:

name: ${{ matrix.os }}-${{ matrix.arch }}

path: build/\*

release:

needs: build

runs-on: ubuntu-latest

steps:

- name: Download all artifacts

uses: actions/download-artifact@v3

with:

path: build

- name: Generate checksums

run: |

cd build

find . -type f -exec sha256sum {} \; > checksums.txt

- name: Create Release

uses: softprops/action-gh-release@v1

with:

files: build/\*\*/\*

tag\_name: ${{ env.VERSION }}

draft: false

prerelease: ${{ github.ref != 'refs/heads/develop' }}

generate\_release\_notes: true

env:

GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}

<https://github.com/dalessi93>

<https://github.com/Jorge382/simple-worker>

<https://github.com/Grupo-1-Campus-Dual-Tic-Org/simple-worker/>

<https://github.com/Jorge382/simple-worker>

<https://github.com/Grupo-1-Campus-Dual-Tic-Org/simple-worker/>

<https://github.com/Grupo-1-Campus-Dual-Tic-Org/simple-worker/>

**WEBEX 11/02/2025**

**CICD & GITHUB ACTIONS**

* 1. MONTAR UNA PIPELINE PARA COMPILAR KERNEL DE LINUX O CHROME

<https://github.com/tonivega/simple-worker/pull/3/commits/5e02de88e6e51be4e39b63d749e123cbbdc2c36e>

**Crear un pipeline de CI/CD para compilar un navegador Chrome** personalizado implica varios pasos, desde la **configuración del entorno** hasta **la construcción y despliegue del navegador**. A continuación, te proporcionaré un ejemplo de cómo podrías estructurar un **workflow de GitHub Actions** para **compilar un navegador Chrome personalizado**. Este ejemplo **asume que tienes un repositorio que contiene el código fuente** y las **configuraciones necesarias para compilar Chrome**.

**Estructura del Repositorio**

Supongamos que tu repositorio tiene la siguiente estructura:

Code

Copiar código

my-chrome-custom/

├── .github/

│ └── workflows/

│ └── ci-cd.yml

├── chrome/

│ ├── custom\_config/

│ └── build\_script.sh

└── README.md

**Paso 1: Crear el Script de Construcción**

Primero, necesitas un **script que compile Chrome**. Aquí hay un ejemplo básico de un script de construcción (**build\_script.sh**):

bash

Copiar código

**#!/bin/bash**

*# Cambiar a la carpeta del código fuente de Chrome*

cd chrome

*# Clonar el repositorio de Chromium (puedes ajustar la URL según sea necesario)*

git clone https://chromium.googlesource.com/chromium/src.git

*# Cambiar al directorio del repositorio clonado*

cd src

*# Configurar el entorno (esto puede variar según tus necesidades)*

gn gen out/Default --args='is\_debug=false'

*# Compilar Chrome*

ninja -C out/Default chrome

*# Copiar el binario compilado a una ubicación deseada*

cp out/Default/chrome /path/to/your/output/directory/

Asegúrate de que el script tenga permisos de ejecución:

bash

Copiar código

**chmod +x chrome/build\_script.sh**

**Paso 2: Crear el Workflow de GitHub Actions**

Ahora, crea el archivo de workflow en **.github/workflows/ci-cd.yml**:

yaml

Copiar código

name: CI/CD for Custom Chrome

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Set up dependencies

run: |

sudo apt-get update

sudo apt-get install -y git python3 python3-pip ninja-build

- name: Run build script

run: |

chmod +x chrome/build\_script.sh

./chrome/build\_script.sh

- name: Upload compiled Chrome

uses: actions/upload-artifact@v2

with:

name: custom-chrome

path: /path/to/your/output/directory/chrome

**Detalle del Workflow**

1. **Nombre del Workflow**: Se llama **CI/CD for Custom Chrome**.
2. **Eventos que Desencadenan el Workflow**: Se ejecuta en cada push a la rama main.
3. **Jobs**: Define un trabajo llamado build que se ejecuta en un entorno ubuntu-latest.
4. **Pasos**:
   * **Checkout del Repositorio**: Usa la acción actions/checkout@v2 para clonar el repositorio.
   * **Configuración de Dependencias**: Actualiza el sistema y instala las dependencias necesarias (como git, python3, ninja-build).
   * **Ejecutar el Script de Construcción**: Cambia los permisos del script de construcción y lo ejecuta.
   * **Subir el Chrome Compilado**: Usa actions/upload-artifact@v2 para subir el binario compilado como un artefacto del workflow.

**Paso 3: Probar el Workflow**

1. **Sube tu código al repositorio**: Asegúrate de que todos los archivos estén en su lugar y sube los cambios a la rama main.
2. **Verifica la Ejecución**: Ve a la pestaña de "Actions" en tu repositorio de GitHub y verifica que el workflow se ejecute correctamente.
3. **Descargar el Artefacto**: Una vez que el workflow se complete, podrás **descargar el artefacto compilado desde la interfaz de GitHub Actions.**

**Consideraciones Finales**

* **Dependencias**: Asegúrate de que todas las dependencias necesarias para compilar Chrome estén instaladas en el entorno de CI/CD.
* **Configuración de Compilación**: El script de construcción puede necesitar ajustes según tus requisitos específicos y la configuración de Chrome que desees personalizar.
* **Recursos**: La compilación de Chrome puede ser intensiva en recursos, así que asegúrate de que el entorno de CI/CD tenga suficiente capacidad.

**Este es un ejemplo básico y puede requerir ajustes según tus necesidades específicas y la configuración de tu proyecto. Si tienes más preguntas o necesitas más detalles,**

si no tiene esto que asume, proporcioamelo tambien codificado con la estructura de carpetas completas y explicado, por favor? un repositorio que contiene el código fuente y las configuraciones necesarias para compilar Chrome.

.

GPT-4o mini

Para crear un repositorio que **contenga el código fuente** y las **configuraciones necesarias para compilar un navegador Chrome personalizado**, necesitarás seguir varios pasos. A continuación, te proporcionaré una estructura de carpetas completa y el código necesario para que puedas configurar tu propio repositorio.

**Estructura del Repositorio**

Aquí tienes una estructura de carpetas sugerida para tu repositorio:

Code

Copiar código

my-chrome-custom/

├── .github/

│ └── workflows/

│ └── ci-cd.yml

├── chrome/

│ ├── custom\_config/

│ │ └── custom\_flags.gn

│ ├── build\_script.sh

│ └── Dockerfile

├── README.md

└── requirements.txt

**Descripción de la Estructura**

* **.github/workflows/**: Contiene los archivos de configuración para GitHub Actions.
* **chrome/**: Contiene el código y los scripts necesarios para compilar Chrome.
  + **custom\_config/**: Carpeta para almacenar configuraciones personalizadas.
  + **build\_script.sh**: Script que se encargará de compilar Chrome.
  + **Dockerfile**: Archivo para crear una imagen de Docker que contenga todas las dependencias necesarias para compilar Chrome.
* **README.md**: Archivo de documentación para tu proyecto.
* **requirements.txt**: Archivo que puede contener **dependencias de Python** si es necesario.

**Paso 1: Crear el Script de Construcción**

Crea el archivo **build\_script.sh** en la carpeta chrome/:

bash

Copiar código

**#!/bin/bash**

*# Cambiar a la carpeta del código fuente de Chrome*

cd chrome

*# Clonar el repositorio de Chromium (puedes ajustar la URL según sea necesario)*

git clone https://chromium.googlesource.com/chromium/src.git

*# Cambiar al directorio del repositorio clonado*

cd src

*# Configurar el entorno (esto puede variar según tus necesidades)*

gn gen out/Default --args='is\_debug=false'

*# Compilar Chrome*

ninja -C out/Default chrome

*# Copiar el binario compilado a una ubicación deseada*

cp out/Default/chrome /path/to/your/output/directory/

Asegúrate de que el script tenga permisos de ejecución:

bash

Copiar código

**chmod +x chrome/build\_script.sh**

**Paso 2: Crear el Dockerfile**

Crea un archivo **Dockerfile** en la carpeta chrome/:

**dockerfile**

Copiar código

*# chrome/Dockerfile*

FROM ubuntu:20.04

*# Instalar dependencias necesarias*

RUN apt-get update && \

apt-get install -y \

git \

python3 \

python3-pip \

ninja-build \

gn \

build-essential \

curl \

&& rm -rf /var/lib/apt/lists/\*

*# Crear un directorio de trabajo*

WORKDIR /app

*# Copiar el script de construcción*

COPY build\_script.sh .

*# Hacer el script ejecutable*

RUN chmod +x build\_script.sh

*# Comando por defecto para ejecutar el script de construcción*

CMD ["./build\_script.sh"]

**Paso 3: Crear el Workflow de GitHub Actions**

Crea el archivo de workflow en **.github/workflows/ci-cd.yml**:

yaml

Copiar código

name: CI/CD for Custom Chrome

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Build Docker image

run: |

docker build -t custom-chrome ./chrome

- name: Run Docker container

run: |

docker run --rm custom-chrome

**Paso 4: Crear el Archivo README.md**

Crea un archivo **README.md** en la raíz del repositorio:

markdown

Copiar código

**# My Custom Chrome**

Este repositorio contiene el código y las configuraciones necesarias para compilar un navegador Chrome personalizado.

**## Estructura del Repositorio**

my-chrome-custom/ ├── .github/ │ └── workflows/ │ └── ci-cd.yml ├── chrome/ │ ├── custom\_config/ │ │ └── custom\_flags.gn │ ├── build\_script.sh │ └── Dockerfile ├── README.md └── requirements.txt

Code

Copiar código

## Cómo Compilar Chrome

1. **Clona este repositorio**.

2. **Asegúrate de tener Docker instalado**.

3. **Ejecuta el siguiente comando para construir la imagen de Docker y compilar Chrome**:

```bash

**git add .**

**git commit -m "Initial commit"**

**git push origin main**

1. El proceso de compilación se ejecutará automáticamente a través de

continua, por favor

.

GPT-4o mini

Claro, continuemos con la documentación en el archivo README.md y algunos detalles adicionales sobre cómo usar el repositorio y el proceso de compilación.

**Continuación del Archivo README.md**

markdown

Copiar código

**## Cómo Compilar Chrome (continuación)**

4. **El proceso de compilación se ejecutará automáticamente a través de GitHub Actions**. Puedes verificar el progreso y los resultados en la pestaña "Actions" de tu repositorio en GitHub.

5. Una vez que la compilación se complete, **podrás encontrar el binario de Chrome** en **el directorio especificado en el script de construcción** (`build\_script.sh`). Asegúrate de ajustar la ruta de salida en el script según tus necesidades.

**## Personalización**

**Si deseas personalizar la configuración de Chrome**, puedes hacerlo **editando el archivo** `**custom\_flags.gn**` en la carpeta `**chrome/custom\_config**/`. Este archivo puede contener argumentos específicos que se pasarán al proceso de compilación de GN.

**## Requisitos**

- **\*\*Docker\*\***: Asegúrate de tener **Docker instalado** en tu máquina para construir la imagen y ejecutar el contenedor.

- **\*\*GitHub\*\***: Necesitarás una **cuenta de GitHub** para crear el repositorio y utilizar GitHub Actions.

**## Notas**

- La compilación de Chrome puede ser intensiva en recursos y puede tardar un tiempo considerable, dependiendo de la configuración de tu entorno y la capacidad de los servidores de GitHub Actions.

- Asegúrate de que tu cuenta de GitHub tenga suficiente cuota de uso de GitHub Actions, especialmente si planeas realizar compilaciones frecuentes.

**## Contribuciones**

Si deseas contribuir a este proyecto, siéntete libre de abrir un "issue" o enviar un "pull request". Todas las contribuciones son bienvenidas.

**## Licencia**

Este proyecto está bajo la Licencia MIT. Consulta el archivo LICENSE para más detalles.

**Paso 5: Crear el Archivo requirements.txt (Opcional)**

Si tu script de construcción o cualquier otra parte de tu proyecto **requiere dependencias de Python**, puedes crear un archivo **requirements.txt** en la raíz del repositorio. Por ejemplo:

plaintext

Copiar código

# requirements.txt

# Lista de dependencias de Python (si es necesario)

**Resumen de la Configuración**

1. **Estructura del Repositorio**: Hemos creado una **estructura de carpetas** que incluye todos los archivos necesarios para compilar un **navegador Chrome personalizado.**
2. **Script de Construcción**: El script build\_script.sh se encarga de **clonar el repositorio de Chromium**, **configurar el entorno** y **compilar Chrome**.
3. **Dockerfile**: El **Dockerfile** define un entorno de construcción que **incluye todas las dependencias** necesarias para **compilar Chrome**.
4. **Workflow de GitHub Actions**: El archivo **ci-cd.yml** configura un **pipeline de CI/CD** que se ejecuta en cada push a la rama main, **construyendo la imagen de Docker** y **ejecutando el script de construcción**.
5. **Documentación**: El archivo **README.md** proporciona instrucciones sobre cómo usar el repositorio, compilar Chrome y personalizar la configuración.

**Cómo Usar el Repositorio**

1. **Clona el Repositorio**: Clona el repositorio en tu máquina local.

bash

Copiar código

**git clone https://github.com/tu\_usuario/my-chrome-custom.git**

**cd my-chrome-custom**

1. **Configura el Repositorio**: Asegúrate de que todos los archivos estén en su lugar y realiza cualquier personalización necesaria.
2. **Sube los Cambios**: Realiza **un commit** y sube los cambios a la **rama main**.

bash

Copiar código

**git add .**

**git commit -m "Initial commit"**

**git push origin main**

1. **Verifica la Ejecución**: Ve a la pestaña "Actions" en tu repositorio de GitHub para verificar que el **workflow se ejecute correctamente**.
2. **Descarga el Binario Compilado**: Una vez que la compilación se complete, podrás encontrar el **binario de Chrome en la ubicación especificada en el script de construcción.**

**Consideraciones Finales**

* **Ajustes en el Script**: Dependiendo de tus necesidades, es posible que debas ajustar el script de construcción y los argumentos de GN para personalizar la compilación de Chrome.
* **Recursos de Compilación**: La compilación de Chrome puede **requerir una cantidad significativa de recursos**, así que asegúrate de que tu entorno de CI/CD tenga suficiente capacidad.
* **Mantenimiento**: Mantén tu repositorio actualizado con las últimas versiones de Chromium y ajusta las configuraciones según sea necesario.

Con esta guía, deberías tener un **repositorio completamente** funcional para compilar un navegador Chrome personalizado utilizando GitHub Actions y Docker. Si tienes más preguntas o necesitas más detalles, no dudes en preguntar. ¡Buena suerte con tu proyecto

Principio del formulario

**WEBEX 12/02/2025**

**CICD GITHUB**

**EJERCICIO 1:**

**Compilar de forma distribuida el Kernel de Linux, montando la pipeline correspondiente:**

Clonar repo kernel

Usar rama 6.13.2

Configurar el kernel con default config

Instalar distcc (ver documentacion en <https://hub.docker.com/r/bensuperpc/distcc>)

Instalar distcc en EKS usando un puerto del nodo tcp 55000 con sus ingress y un balenceador de carga tipo tcp (o ssh, según use el distcc), asi debemos publicar un nodo

Compilar kernel usando distcc

Arrancar kernel con un QEMU (Maquina virtual)

clonar repo del kernel  
usar rama 6.13.2  
configurar el kernel con default config  
instalar distcc (ver documentación de distcc) (<https://hub.docker.com/r/bensuperpc/distcc>)  
instalar distcc en EKS usando un puerto del nodo (puerto TCP 55000)  
compilar kernel usando distcc  
testear kernel con QEMU

¡Por supuesto! Te ayudaré a crear una pipeline para compilar el kernel de Linux de forma distribuida. Vamos a dividirlo en pasos claros y crear una estructura organizada.

Estructura del Proyecto y Archivos de Pipeline

Click to open code

Vamos a explicar paso a paso cada componente y cómo funciona todo el proceso:

1. **Estructura del Proyecto**:
   * Dockerfile.builder: Contiene la imagen base para compilar el kernel
   * Dockerfile.distcc: Configura el worker de distcc
   * k8s/: Directorio con las configuraciones de Kubernetes
   * scripts/: Scripts de construcción y prueba
   * Jenkinsfile: Define la pipeline de CI/CD
2. **Configuración de DistCC en EKS**:
   * Desplegamos **workers de distcc en EKS usando el deployment**
   * Exponemos el **servicio en el puerto 55000 TCP**
   * Configuramos un **LoadBalancer** para distribuir la carga
   * El ingress permite el acceso externo al servicio
3. **Proceso de Compilación**:
   * **Clonamos el kernel 6.13.2**
   * Usamos la **configuración por defecto** (defconfig)
   * **Configuramos distcc para usar los workers remotos**
   * **Compilamos usando distcc como compilador distribuido**
4. **Pruebas**:
   * Usamos **QEMU para probar el kernel compilado**
   * Configuramos **una consola serial para ver el output**
5. **Pipeline de Jenkins**: La pipeline tiene cuatro etapas principales:
   * Preparación del **entorno de compilación**
   * **Despliegue de DistCC en EKS**
   * **Compilación del kernel**
   * **Pruebas con QEMU**

Para ejecutar este proyecto, necesitarás:

1. Un cluster EKS configurado
2. Jenkins instalado y configurado
3. Docker instalado en el nodo de Jenkins
4. kubectl configurado para acceder a tu cluster EKS

¿Hay alguna parte específica que quieras que te explique con más detalle?

├── README.md

├── Dockerfile.builder

├── Dockerfile.distcc

├── k8s/

│ ├── distcc-deployment.yaml

│ ├── distcc-service.yaml

│ ├── distcc-ingress.yaml

│ └── distcc-configmap.yaml

├── scripts/

│ ├── setup-distcc.sh

│ ├── build-kernel.sh

│ └── test-kernel.sh

└── Jenkinsfile

*# Dockerfile.builder*

FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \

git \

build-essential \

libncurses-dev \

flex \

bison \

libssl-dev \

libelf-dev \

distcc \

qemu-system-x86 \

&& rm -rf /var/lib/apt/lists/\*

WORKDIR /kernel

COPY scripts/build-kernel.sh /usr/local/bin/

RUN chmod +x /usr/local/bin/build-kernel.sh

*# Dockerfile.distcc*

FROM bensuperpc/distcc:latest

EXPOSE 3632/tcp

EXPOSE 3632/udp

*# k8s/distcc-deployment.yaml*

apiVersion: apps/v1

kind: Deployment

metadata:

name: distcc-worker

spec:

replicas: 3

selector:

matchLabels:

app: distcc-worker

template:

metadata:

labels:

app: distcc-worker

spec:

containers:

- name: distcc

image: bensuperpc/distcc:latest

ports:

- containerPort: 3632

protocol: TCP

env:

- name: ALLOW

value: "0.0.0.0/0"

- name: JOBS

value: "8"

*# k8s/distcc-service.yaml*

apiVersion: v1

kind: Service

metadata:

name: distcc-service

spec:

type: LoadBalancer

ports:

- port: 55000

targetPort: 3632

protocol: TCP

selector:

app: distcc-worker

*# k8s/distcc-ingress.yaml*

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: distcc-ingress

annotations:

kubernetes.io/ingress.class: "nginx"

spec:

rules:

- http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: distcc-service

port:

number: 55000

*# scripts/build-kernel.sh*

*#!/bin/bash*

set -e

*# Clone kernel*

git clone --depth 1 --branch v6.13.2 https://github.com/torvalds/linux.git

cd linux

*# Configure distcc*

export DISTCC\_HOSTS="localhost distcc-service:55000"

export PATH="/usr/lib/distcc:$PATH"

*# Configure kernel*

make defconfig

*# Build kernel with distcc*

make -j$(nproc) CC="distcc gcc"

*# scripts/test-kernel.sh*

*#!/bin/bash*

set -e

qemu-system-x86\_64 \

-kernel arch/x86/boot/bzImage \

-append "console=ttyS0" \

-nographic

*# Jenkinsfile*

pipeline {

agent any

stages {

stage('Setup Builder') {

steps {

sh 'docker build -t kernel-builder -f Dockerfile.builder .'

}

}

stage('Deploy DistCC to EKS') {

steps {

sh '''

kubectl apply -f k8s/distcc-deployment.yaml

kubectl apply -f k8s/distcc-service.yaml

kubectl apply -f k8s/distcc-ingress.yaml

'''

}

}

stage('Build Kernel') {

steps {

sh '''

docker run --network=host \

-v ${WORKSPACE}:/kernel \

kernel-builder \

/usr/local/bin/build-kernel.sh

'''

}

}

stage('Test Kernel') {

steps {

sh '''

docker run --network=host \

-v ${WORKSPACE}:/kernel \

kernel-builder \

/usr/local/bin/test-kernel.sh

'''

}

}

}

}

Last edited just now

Primero podrias darme la pipeline cicd para desplegar con Github Actions en lugar de Jenkins asi como seria la estructura de carpetas y archivos y su codificacion con esta modificacion, por favor

Por supuesto. Vamos a adaptar la estructura para usar GitHub Actions en lugar de Jenkins.

Estructura del Proyecto con GitHub Actions

Click to open code

Los principales cambios y diferencias con respecto a la versión de Jenkins son:

1. **Estructura de Directorios**:
   * Añadimos el directorio .github/workflows/ que contiene la definición de la pipeline
   * Eliminamos el Jenkinsfile ya que no lo necesitamos
   * Mantenemos la misma estructura para los demás directorios
2. **GitHub Actions Workflow**:
   * Se ejecuta en pushes y pull requests a main
   * Usa secrets de GitHub para las credenciales de AWS
   * Integración con AWS ECR para almacenar las imágenes Docker
   * Incluye pasos para:
     + Configurar credenciales AWS
     + Construir y subir la imagen Docker
     + Desplegar DistCC en EKS
     + Compilar y probar el kernel
     + Subir los artefactos generados
3. **Mejoras Adicionales**:
   * Uso de GitHub Actions para subir artefactos (kernel compilado)
   * Espera automática a que los pods de DistCC estén listos
   * Uso de versiones específicas de las actions para mayor estabilidad

Para usar esta pipeline, necesitarás configurar en tu repositorio de GitHub:

1. **Secrets**:
   * AWS\_ACCESS\_KEY\_ID
   * AWS\_SECRET\_ACCESS\_KEY
2. **Variables de Entorno** (puedes modificarlas en el workflow):
   * AWS\_REGION
   * EKS\_CLUSTER\_NAME
   * ECR\_REPOSITORY
3. **Permisos**:
   * El IAM user/role debe tener permisos para:
     + ECR (push/pull de imágenes)
     + EKS (administración del cluster)

¿Te gustaría que profundice en algún aspecto específico de la implementación?

.

├── README.md

├── Dockerfile.builder

├── Dockerfile.distcc

├── .github/

│ └── workflows/

│ └── kernel-build.yml

├── k8s/

│ ├── distcc-deployment.yaml

│ ├── distcc-service.yaml

│ ├── distcc-ingress.yaml

│ └── distcc-configmap.yaml

└── scripts/

├── setup-distcc.sh

├── build-kernel.sh

└── test-kernel.sh

*# .github/workflows/kernel-build.yml*

name: Build Linux Kernel

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

env:

AWS\_REGION: us-east-1

EKS\_CLUSTER\_NAME: my-cluster

ECR\_REPOSITORY: kernel-builder

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v4

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ env.AWS\_REGION }}

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

- name: Build and push builder image

uses: docker/build-push-action@v5

with:

context: .

file: Dockerfile.builder

push: true

tags: |

${{ steps.login-ecr.outputs.registry }}/${{ env.ECR\_REPOSITORY }}:${{ github.sha }}

${{ steps.login-ecr.outputs.registry }}/${{ env.ECR\_REPOSITORY }}:latest

- name: Update kube config

run: |

aws eks update-kubeconfig --name ${{ env.EKS\_CLUSTER\_NAME }} --region ${{ env.AWS\_REGION }}

- name: Deploy DistCC to EKS

run: |

kubectl apply -f k8s/distcc-deployment.yaml

kubectl apply -f k8s/distcc-service.yaml

kubectl apply -f k8s/distcc-ingress.yaml

# Esperar a que los pods estén listos

kubectl wait --for=condition=ready pod -l app=distcc-worker --timeout=300s

- name: Build Kernel

run: |

docker run --network=host \

-v ${{ github.workspace }}:/kernel \

${{ steps.login-ecr.outputs.registry }}/${{ env.ECR\_REPOSITORY }}:${{ github.sha }} \

/usr/local/bin/build-kernel.sh

- name: Test Kernel

run: |

docker run --network=host \

-v ${{ github.workspace }}:/kernel \

${{ steps.login-ecr.outputs.registry }}/${{ env.ECR\_REPOSITORY }}:${{ github.sha }} \

/usr/local/bin/test-kernel.sh

- name: Upload Kernel Artifacts

uses: actions/upload-artifact@v4

with:

name: kernel-build

path: |

linux/arch/x86/boot/bzImage

linux/.config

*# Dockerfile.builder (mismo contenido que antes)*

FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \

git \

build-essential \

libncurses-dev \

flex \

bison \

libssl-dev \

libelf-dev \

distcc \

qemu-system-x86 \

&& rm -rf /var/lib/apt/lists/\*

WORKDIR /kernel

COPY scripts/build-kernel.sh /usr/local/bin/

COPY scripts/test-kernel.sh /usr/local/bin/

RUN chmod +x /usr/local/bin/build-kernel.sh /usr/local/bin/test-kernel.sh

*# scripts/build-kernel.sh (mismo contenido que antes)*

*#!/bin/bash*

set -e

git clone --depth 1 --branch v6.13.2 https://github.com/torvalds/linux.git

cd linux

export DISTCC\_HOSTS="localhost distcc-service:55000"

export PATH="/usr/lib/distcc:$PATH"

make defconfig

make -j$(nproc) CC="distcc gcc"

*# Los archivos k8s/\* mantienen el mismo contenido que en la versión anterior*

<https://stackoverflow.com/questions/28136815/linux-kernel-how-to-obtain-a-particular-version-right-upto-sublevel>

de los archivos que me has dado cuals seria un daemontset para desplegar por nodos, o sea el yaml?

Voy a mostrarte cómo convertir el deployment de DistCC a un DaemonSet para asegurar que cada nodo del cluster tenga un worker de DistCC.

DaemonSet Configuration for DistCC

Click to open code

Las principales diferencias y ventajas de usar un DaemonSet en lugar de un Deployment son:

1. **Garantía de Cobertura**:
   * Se ejecuta un pod en cada nodo del cluster
   * Ideal para servicios que necesitan estar presentes en todos los nodos
2. **Configuración de Red**:
   * hostNetwork: true: Usa la red del host para mejor rendimiento
   * hostPort: 55000: Expone el servicio directamente en el puerto del nodo
   * No necesitas un Service adicional ya que cada pod es accesible directamente
3. **Toleraciones**:
   * Incluye toleraciones para ejecutarse incluso en nodos master/control-plane
   * Asegura una cobertura completa del cluster
4. **Recursos y Límites**:
   * Define recursos específicos para cada pod
   * Configurable según las capacidades de tus nodos

Para **aplicar este DaemonSet**:

bash

Copy

kubectl apply -f k8s/distcc-daemonset.yaml

Para verificar que está funcionando:

bash

Copy

*# Ver pods en cada nodo*

kubectl get pods -o wide

*# Ver logs de un pod específico*

kubectl logs -f <nombre-del-pod>

*# Ver estado del DaemonSet*

kubectl get daemonset distcc-worker

¿Necesitas que te explique algún aspecto específico de la configuración?

 CopyRetry

Claude does not have the ability to run the code it generates yet.

JORGE : YAML BUENO DE KERNEL

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: distcc

namespace: default

labels:

app: distcc-devtool

spec:

selector:

matchLabels:

name: distcc-devtool

template:

metadata:

labels:

name: distcc-devtool

spec:

containers:

name: distcc

image: bensuperpc/distcc:ubuntu-22.04

args: ["sleep", "0" ]

ports:

containerPort: 3632

hostPort: 55000

protocol: TCP

18.134.143.117  
13.40.125.123  
18.170.112.170

[3.67.95.35](https://3.67.95.35)

[18.193.149.207](https://18.193.149.207)

[3.10.228.71](https://3.10.228.71)

[18.175.120.93](https://18.175.120.93)

[15.188.84.28](https://15.188.84.28)

**WEBEX 13/02/2025**

**CICD**

**LANZAR KERNEL LINUX:**

export DISTCC\_POTENTIAL\_HOSTS="[18.134.143.117](https://18.134.143.117) [13.40.125.123](https://13.40.125.123) [18.170.112.170](https://18.170.112.170) [3.67.95.35](https://3.67.95.35) [18.193.149.207](https://18.193.149.207) [3.10.228.71](https://3.10.228.71) [18.175.120.93](https://18.175.120.93) [15.188.84.28](https://15.188.84.28)"

sudo chwon -R javier: .

make defconfig

**Lanzar compilaciion:**

make -j$(nproc) CC-“distcc gcc” CXX=”distcc g++”

export DISTCC\_POTENTIAL\_HOSTS="[18.134.143.117:55000](https://18.134.143.117:55000) [13.40.125.123:55000](https://13.40.125.123:55000) [18.170.112.170:55000](https://18.170.112.170:55000) [3.67.95.35:55000](https://3.67.95.35:55000) [18.193.149.207:55000](https://18.193.149.207:55000) [3.10.228.71:55000](https://3.10.228.71:55000) [18.175.120.93:55000](https://18.175.120.93:55000) [15.188.84.28:55000](https://15.188.84.28:55000)"

make -j14 CC=“distcc gcc” bzImage

WEBEX 14/02/2025

CICD

REGISTRY.OPENDATA.AWS

<https://registry.opendata.aws/commoncrawl/>

<https://data.commoncrawl.org/crawl-data/CC-MAIN-2025-05/index.html>

RFC URI IDENTIFICADOR TECNICO DE ALGO : DATATRACKER.IETF.ORG/DOC/HTML/

Imaginate que eres un senior devops y debes hacer un proyecto de una pipeline con infraestructura como codigo donde uses common crawl que almacena toda internet en aws, buscando rss, de empleos de internet, o sea ofertas de empleo.

Te ayudaré a diseñar una pipeline DevOps para procesar datos de Common Crawl y extraer ofertas de empleo de feeds RSS. Vamos a crear una infraestructura como código (IaC) usando AWS.

Infrastructure as Code for Job Crawler Pipeline

Click to open code

Ahora vamos a crear el código Python que procesará los datos de Common Crawl:

Common Crawl Job RSS Processor

Click to open code

Esta solución incluye:

1. **Infraestructura como Código (Terraform)**:
   * VPC dedicada para el proyecto
   * Buckets S3 para datos crudos y procesados
   * Cluster EMR para procesamiento distribuido
   * Lambda para orquestación
   * Step Functions para el workflow
   * CloudWatch Events para programación
2. **Procesador de Datos (Python/PySpark)**:
   * Integración con Common Crawl
   * Filtrado de URLs de RSS relacionadas con empleos
   * Procesamiento de feeds RSS
   * Extracción y almacenamiento de ofertas de trabajo
3. **Pipeline de Datos**:
   * Ejecución diaria automática
   * Procesamiento distribuido con Spark
   * Almacenamiento eficiente en formato Parquet
   * Manejo de errores y recuperación

Para implementar esto necesitarías:

1. Configurar las credenciales de AWS
2. Ejecutar el código Terraform para crear la infraestructura
3. Empaquetar el código Python en un JAR para EMR
4. Configurar los roles IAM necesarios

¿Te gustaría que profundice en algún aspecto específico de la solución?

 CopyRetry

**HE**

no me dejaste terminar debe montarse un s3 commoncrawl arn:aws:s3:commoncrawl sobre un elasticsearch para hacer consultas de empleo y concretar sobre empleos de devops.Hazme toda la infraestructura, asi como estructura de carpetas, codificacion y explicacion detallada, por favor?

Entiendo, vamos a expandir la solución para incluir Elasticsearch y enfocarnos específicamente en empleos DevOps.

Primero, la estructura de carpetas del proyecto:

DevOps Job Crawler Project Structure

Click to open code

Ahora, la infraestructura actualizada con Terraform:

Updated Infrastructure with Elasticsearch

Click to open code

El procesador de Common Crawl específico para DevOps:

DevOps Job Crawler and Processor

Click to open code

Utilidades para el procesamiento de texto:

Text Processing Utilities

Click to open code

Para implementar esta solución:

1. **Preparación del Entorno**:

bash

Copy

*# Crear entorno virtual*

python -m venv venv

source venv/bin/activate

*# Instalar dependencias*

 CopyRetry

Claude’s response was limited as it hit the maximum length allowed at this time. Claude does not have the ability to run the code it generates yet.

devops-job-crawler/

├── infrastructure/

│ ├── terraform/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ ├── outputs.tf

│ │ └── elasticsearch.tf

│ └── docker/

│ └── Dockerfile

├── src/

│ ├── crawler/

│ │ ├── common\_crawl\_processor.py

│ │ ├── elastic\_indexer.py

│ │ └── devops\_classifier.py

│ ├── utils/

│ │ ├── text\_processor.py

│ │ └── elasticsearch\_utils.py

│ └── config/

│ └── config.yaml

├── tests/

│ ├── test\_crawler.py

│ └── test\_classifier.py

├── requirements.txt

└── README.md

*# main.tf*

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.0"

}

}

}

provider "aws" {

region = var.aws\_region

}

*# VPC Configuration*

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags = {

Name = "devops-job-crawler-vpc"

}

}

*# Subnets*

resource "aws\_subnet" "private" {

count = 3

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.${count.index + 1}.0/24"

availability\_zone = data.aws\_availability\_zones.available.names[count.index]

tags = {

Name = "private-subnet-${count.index + 1}"

}

}

*# Elasticsearch Domain*

resource "aws\_elasticsearch\_domain" "jobs\_search" {

domain\_name = "devops-jobs-search"

elasticsearch\_version = "7.10"

cluster\_config {

instance\_type = "m5.large.elasticsearch"

instance\_count = 3

zone\_awareness\_enabled = true

zone\_awareness\_config {

availability\_zone\_count = 3

}

}

ebs\_options {

ebs\_enabled = true

volume\_size = 100

}

vpc\_options {

subnet\_ids = aws\_subnet.private[\*].id

security\_group\_ids = [aws\_security\_group.es\_sg.id]

}

snapshot\_options {

automated\_snapshot\_start\_hour = 23

}

advanced\_options = {

"rest.action.multi.allow\_explicit\_index" = "true"

}

tags = {

Domain = "DevOpsJobsSearch"

}

}

*# S3 Buckets*

resource "aws\_s3\_bucket" "commoncrawl\_mirror" {

bucket = "devops-jobs-commoncrawl-mirror"

}

resource "aws\_s3\_bucket" "processed\_data" {

bucket = "devops-jobs-processed-data"

}

*# EMR Cluster*

resource "aws\_emr\_cluster" "processing\_cluster" {

name = "devops-jobs-processor"

release\_label = "emr-6.10.0"

applications = ["Spark", "Hadoop"]

service\_role = aws\_iam\_role.emr\_service\_role.arn

master\_instance\_group {

instance\_type = "m5.xlarge"

}

core\_instance\_group {

instance\_type = "m5.xlarge"

instance\_count = 3

}

ec2\_attributes {

subnet\_id = aws\_subnet.private[0].id

emr\_managed\_master\_security\_group = aws\_security\_group.emr\_master.id

emr\_managed\_slave\_security\_group = aws\_security\_group.emr\_slave.id

instance\_profile = aws\_iam\_instance\_profile.emr\_profile.arn

}

}

*# Lambda for Elasticsearch indexing*

resource "aws\_lambda\_function" "es\_indexer" {

filename = "../src/lambda/es\_indexer.zip"

function\_name = "devops-jobs-es-indexer"

role = aws\_iam\_role.lambda\_role.arn

handler = "es\_indexer.handler"

runtime = "python3.9"

timeout = 300

environment {

variables = {

ES\_ENDPOINT = aws\_elasticsearch\_domain.jobs\_search.endpoint

REGION = var.aws\_region

}

}

vpc\_config {

subnet\_ids = aws\_subnet.private[\*].id

security\_group\_ids = [aws\_security\_group.lambda\_sg.id]

}

}

*# src/crawler/common\_crawl\_processor.py*

from typing import List, Dict

import boto3

import json

import warcio

import elasticsearch

from elasticsearch import Elasticsearch, helpers

from utils.text\_processor import TextProcessor

from utils.elasticsearch\_utils import ElasticsearchClient

class DevOpsJobCrawler:

def \_\_init\_\_(self, es\_host: str, es\_port: int):

self.s3 = boto3.client('s3')

self.text\_processor = TextProcessor()

self.es\_client = ElasticsearchClient(es\_host, es\_port)

self.devops\_keywords = self.\_load\_devops\_keywords()

def \_load\_devops\_keywords(self) -> Dict[str, List[str]]:

return {

'tools': [

'kubernetes', 'docker', 'jenkins', 'terraform', 'ansible',

'aws', 'gcp', 'azure', 'gitlab', 'github', 'prometheus',

'grafana', 'elk', 'elasticsearch', 'kafka', 'redis'

],

'skills': [

'ci/cd', 'infrastructure as code', 'containerization',

'orchestration', 'automation', 'monitoring', 'logging',

'security', 'cloud architecture', 'microservices'

],

'titles': [

'devops', 'sre', 'platform engineer', 'cloud engineer',

'infrastructure engineer', 'reliability engineer'

]

}

def process\_warc\_record(self, record) -> Dict:

"""Procesa un registro WARC y extrae información relevante de DevOps"""

if record.type != 'response':

return None

content = record.content\_stream().read().decode('utf-8', errors='ignore')

*# Verifica si es una oferta de trabajo DevOps*

if not any(keyword in content.lower() for keyword in self.devops\_keywords['titles']):

return None

*# Extrae información relevante*

job\_info = {

'url': record.url,

'timestamp': record.date,

'title': self.text\_processor.extract\_title(content),

'company': self.text\_processor.extract\_company(content),

'description': self.text\_processor.extract\_description(content),

'tools': self.text\_processor.extract\_tools(content, self.devops\_keywords['tools']),

'skills': self.text\_processor.extract\_skills(content, self.devops\_keywords['skills']),

'location': self.text\_processor.extract\_location(content),

'salary\_range': self.text\_processor.extract\_salary(content)

}

return job\_info

def index\_job(self, job\_info: Dict):

"""Indexa la oferta de trabajo en Elasticsearch"""

doc = {

'index': 'devops-jobs',

'body': job\_info

}

self.es\_client.index\_document(doc)

def process\_commoncrawl\_segment(self, segment\_path: str):

"""Procesa un segmento de Common Crawl"""

warc\_data = self.s3.get\_object(

Bucket='commoncrawl',

Key=segment\_path

)['Body']

for record in warcio.ArchiveIterator(warc\_data):

job\_info = self.process\_warc\_record(record)

if job\_info:

self.index\_job(job\_info)

def create\_elasticsearch\_mappings(self):

"""Crea el mapping de Elasticsearch para trabajos DevOps"""

mapping = {

"mappings": {

"properties": {

"url": {"type": "keyword"},

"timestamp": {"type": "date"},

"title": {

"type": "text",

"fields": {

"keyword": {"type": "keyword"}

}

},

"company": {

"type": "text",

"fields": {

"keyword": {"type": "keyword"}

}

},

"description": {"type": "text"},

"tools": {"type": "keyword"},

"skills": {"type": "keyword"},

"location": {

"type": "text",

"fields": {

"keyword": {"type": "keyword"}

}

},

"salary\_range": {

"type": "object",

"properties": {

"min": {"type": "float"},

"max": {"type": "float"},

"currency": {"type": "keyword"}

}

}

}

}

}

self.es\_client.create\_index('devops-jobs', mapping)

*# src/utils/text\_processor.py*

import re

from typing import List, Dict

import spacy

from bs4 import BeautifulSoup

class TextProcessor:

def \_\_init\_\_(self):

self.nlp = spacy.load("en\_core\_web\_sm")

def extract\_title(self, content: str) -> str:

"""Extrae el título del trabajo del contenido HTML"""

soup = BeautifulSoup(content, 'html.parser')

*# Intenta diferentes estrategias para encontrar el título*

title\_candidates = [

soup.find('h1'),

soup.find('title'),

soup.find(class\_=re.compile(r'job.\*title|position.\*title', re.I))

]

for candidate in title\_candidates:

if candidate and candidate.text.strip():

return self.clean\_text(candidate.text)

return None

def extract\_company(self, content: str) -> str:

"""Extrae el nombre de la empresa"""

soup = BeautifulSoup(content, 'html.parser')

company\_patterns = [

r'company.\*name',

r'organization.\*name',

r'employer.\*name'

]

for pattern in company\_patterns:

company\_element = soup.find(class\_=re.compile(pattern, re.I))

if company\_element:

return self.clean\_text(company\_element.text)

return None

def extract\_description(self, content: str) -> str:

"""Extrae la descripción del trabajo"""

soup = BeautifulSoup(content, 'html.parser')

description\_patterns = [

r'job.\*description',

r'position.\*description',

r'role.\*description'

]

for pattern in description\_patterns:

desc\_element = soup.find(class\_=re.compile(pattern, re.I))

if desc\_element:

return self.clean\_text(desc\_element.text)

return None

def extract\_tools(self, content: str, tool\_list: List[str]) -> List[str]:

"""Extrae las herramientas mencionadas en el contenido"""

content\_lower = content.lower()

found\_tools = []

for tool in tool\_list:

if tool.lower() in content\_lower:

found\_tools.append(tool)

return found\_tools

def extract\_skills(self, content: str, skill\_list: List[str]) -> List[str]:

"""Extrae las habilidades mencionadas en el contenido"""

content\_lower = content.lower()

found\_skills = []

for skill in skill\_list:

if skill.lower() in content\_lower:

found\_skills.append(skill)

return found\_skills

def extract\_location(self, content: str) -> str:

"""Extrae la ubicación del trabajo"""

doc = self.nlp(content)

locations = []

for ent in doc.ents:

if ent.label\_ in ['GPE', 'LOC']:

locations.append(ent.text)

return locations[0] if locations else None

def extract\_salary(self, content: str) -> Dict:

"""Extrae el rango salarial"""

salary\_pattern = r'(?:salary|compensation).\*?(\$[\d,]+(?:,\d{3})\*(?:\.\d{2})?(?:\s\*-\s\*\$[\d,]+(?:,\d{3})\*(?:\.\d{2})?)?)'

match = re.search(salary\_pattern, content, re.I)

if match:

salary\_text = match.group(1)

salary\_numbers = re.findall(r'\$[\d,]+(?:,\d{3})\*(?:\.\d{2})?', salary\_text)

if len(salary\_numbers) == 2:

return {

'min': float(salary\_numbers[0].replace('$', '').replace(',', '')),

'max': float(salary\_numbers[1].replace('$', '').replace(',', '')),

'currency': 'USD'

}

elif len(salary\_numbers) == 1:

salary\_value = float(salary\_numbers[0].replace('$', '').replace(',', ''))

return {

'min': salary\_value,

'max': salary\_value,

'currency': 'USD'

}

return None

def clean\_text(self, text: str) -> str:

"""Limpia el texto de caracteres no deseados"""

if not text:

return None

text = re.sub(r'\s+', ' ', text)

text = text.strip()

return text