Technical test - Imaginamos

Status	Done
Finalizacion	@January 17, 2025

Repositorio

https://github.com/AgnerVillaFabrega/technical-test-we-imagine

▼ 1. Creación de un repositorio con CI/CD

▼ a. Descripción General

Este repositorio contiene la implementación de la **primera parte** de la prueba técnica, que incluye:

1. Creación de un repositorio en GitHub con un servicio web minimalista (CRUD de usuarios en NestJS).
2. Configuración de un pipeline de CI/CD utilizando **GitHub Actions** que realiza las siguientes tareas:
   - **Build:** Compila el proyecto.
   - **Test:** Ejecuta pruebas unitarias y de integración, enviando los resultados a **SonarQube**.
   - **Deploy:** Despliega la aplicación en un clúster **EKS** en **AWS**.

▼ b. Estructura del Repositorio

```
agnervillafabrega-technical-test-we-imagine/
├── docker-compose.yml     # SonarQube local para prue
bas (no requerido en la prueba)
├── docs/                  # Documentación del proyect
o
├── k8/                    # Manifests de Kubernetes
```

```
(Deployment y Service)
│   ├── deployment.yaml
│   └── service.yaml
├── step-1/                # Proyecto principal (CRUD
de usuarios en NestJS)
│   ├── Dockerfile
│   ├── sonar-project.properties
│   ├── src/               # Código fuente de la aplic
ación
│   ├── test/              # Pruebas de integración y
unitarias
│   └── ...
└── .github/
    └── workflows/         # Configuración del pipelin
e CI/CD
        ├── cicd.yaml      # Pipeline de CI/CD para de
spliegue en EKS
        └── pr.yaml        # Pipeline para validación
en pull requests
```

▼ c. Detalles del Pipeline CI/CD con GitHub Action

El pipeline se implementó en dos archivos YAML bajo `.github/workflows/`:

▼ Archivo: `cicd.yaml`

1. **Objetivo:** Automatizar el flujo completo de CI/CD para cada push en la rama `main`.
2. **Pasos del Pipeline:**
   - **Build:**
     - Instala dependencias con `yarn`.
     - Compila la aplicación en NestJS.
   - **Test:**
     - Ejecuta pruebas unitarias y genera reportes de cobertura.
     - Envia los resultados de análisis estático a **SonarQube**.
   - **Deploy:**

- Construye y sube la imagen Docker al repositorio configurado.
- Despliega la aplicación al clúster **EKS**.
- Verifica el estado del despliegue.

▼ Archivo: `pr.yaml`

1. **Objetivo:** Validar los cambios antes de hacer merge en la rama principal.
2. **Pasos del Pipeline:**
   - Realiza el build de la aplicación.
   - Ejecuta pruebas unitarias y de integración.
   - Corre análisis estático de código con SonarQube.
   - Construye la imagen Docker (sin deploy).

▼ d. Configuración de Entornos

▼ Variables y Secretos en GitHub

Se utilizaron las siguientes configuraciones:

1. **Secretos:**
   - `AWS_ACCESS_KEY`: Credencial de acceso AWS.
   - `AWS_SECRET_KEY`: Llave secreta AWS.
   - `DOCKER_USERNAME`: Usuario de Docker Hub.
   - `DOCKER_PASSWORD`: Contraseña de Docker Hub.
   - `SONAR_TOKEN`: Token de autenticación SonarQube.
2. **Variables:**
   - `AWS_REGION`: Región del clúster AWS (por ejemplo, `us-east-1`).
   - `EKS_CLUSTER_NAME`: Nombre del clúster EKS.
   - `EKS_DEPLOYMENT_NAME`: Nombre del deployment en Kubernetes.
   - `EKS_SERVICE_NAME`: Nombre del servicio en Kubernetes.
   - `EKS_CONTAINER_NAME`: Nombre del contenedor dentro del clúster.
   - `IMAGE_NAME`: Nombre de la imagen Docker.

- `IMAGE_TAG`: Etiqueta de la imagen Docker (generalmente el commit SHA).
- `SONAR_HOST_URL`: URL del servidor SonarQube.

e. Despliegue en EKS

Se incluyó un directorio `k8/` con los manifests de Kubernetes necesarios:

Archivo: `deployment.yaml`

Despliega la aplicación como un Deployment con una réplica. El contenedor usa la imagen Docker generada en el pipeline.

Archivo: `service.yaml`

Expone la aplicación a través de un LoadBalancer para que sea accesible públicamente.

f. Uso de SonarQube

El análisis estático de código utiliza un archivo `sonar-project.properties` ubicado en el directorio `step-1/`. Este archivo contiene:

```
sonar.projectKey=agnervillafabrega-technical-test-we-imagine
sonar.sources=src
sonar.tests=test
sonar.test.inclusions=src/**/*.spec.ts
sonar.typescript.lcov.reportPaths=coverage/lcov.info
```

g. Errores y Soluciones Propuestas

Errores al usar EKS de forma local

1. Listando los clusters:

```
$ aws eks list-clusters --region us-east-2
An error occurred (AccessDeniedException) when calling the ListClusters operation: User: arn:aws:iam::267419910235:user/DevopsUserExam is not authorized to perform: eks:ListClusters on resource: arn:aws:eks:us-east-2:267419910235:cluster/*
```

Posible Solución:

- Verificar que el usuario IAM tenga los permisos necesarios, como `eks:ListClusters`.

2. Ejecutando el deployment y servicios:

```
$ kubectl apply -f deployment.yaml
error: error validating "deployment.yaml": error
validating data: failed to download openapi: the
server has asked for the client to provide crede
ntials; if you choose to ignore these errors, tu
rn validation off with --validate=false
```

Posible Solución:

- Verificar las credenciales configuradas en AWS CLI y asegurarse de que sean válidas.

3. Error al ejecutar comandos de despliegue:

```
$ aws eks update-kubeconfig --name rappi-web-eks
-cluster --region us-east-1
Updated context arn:aws:eks:us-east-1:2674199102
35:cluster/rappi-web-eks-cluster in C:\Users\Agn
er\.kube\config

$ kubectl get nodes
E0115 16:19:35.335811   28188 memcache.go:265] c
ouldn't get current server API group list: the s
erver has asked for the client to provide creden
tials
error: You must be logged in to the server (the
server has asked for the client to provide crede
ntials)
```

Posible Solución:

- Asegurarse de que el usuario IAM tenga permisos para acceder a los nodos del cluster (por ejemplo, `eks:DescribeCluster`, `ec2:DescribeInstances`).

- Revisar si el cluster está correctamente configurado con el rol IAM asociado.

▼ Error en el pipeline de CI/CD al momento de hacer el deploy

```
Run aws eks update-kubeconfig --region us-east-1 --
name rappi-web-eks-cluster
Added new context arn:aws:eks:us-east-1:***:cluste
r/rappi-web-eks-cluster to /home/runner/.kube/confi
g
E0115 20:55:08.332146    2309 memcache.go:265] "Unh
andled Error" err="couldn't get current server API
group list: the server has asked for the client to
provide credentials"
error: You must be logged in to the server (the ser
ver has asked for the client to provide credential
s)
```

Posible Solución:

- Asegurarse de que las credenciales de AWS configuradas en el pipeline son correctas y tienen los permisos necesarios.
- Confirmar que el rol IAM tenga permisos adecuados para interactuar con el cluster de EKS.

▼ Permisos de SonarQube

```
ERROR You're not authorized to analyze this project
or the project doesn't exist on SonarQube and you'r
e not authorized to create it. Please contact an ad
ministrator.
```

Posible Solución:

- Verificar los permisos del token de autenticación utilizado.
- Confirmar con el administrador de SonarQube que el usuario tenga permisos para analizar o crear el proyecto.

▼ 2. Generación de infraestructura con Terraform

Este proyecto contiene la configuración de Terraform para desplegar una infraestructura completa en AWS, incluyendo VPC, EKS, RDS/DocumentDB y recursos relacionados.

▼ Requisitos Previos

- Terraform >= 1.10.4
- AWS CLI configurado con las credenciales apropiadas
- kubectl (para interactuar con el cluster EKS)

▼ Componentes Principales

1. **VPC y Networking**
   - VPC con subnets públicas y privadas
   - Internet Gateway y NAT Gateway
   - Tablas de rutas configuradas
2. **EKS (Elastic Kubernetes Service)**
   - Cluster EKS con nodos auto-escalables
   - IAM roles y políticas necesarias
   - Security Groups configurados
3. **Base de Datos**
   - RDS (PostgreSQL/MySQL) o DocumentDB según configuración
   - Subnet groups y security groups dedicados
4. **Storage (Opcional)**
   - Bucket S3 con versionamiento y encriptación
   - IAM roles para acceso

▼ Estructura del Proyecto

```
.
├── main.tf           # Archivo principal y recursos c
```

```
ore
├── variables.tf      # Declaración de variables
├── outputs.tf        # Outputs/salidas del proyecto
├── versions.tf       # Versiones de providers
├── vpc.tf           # Configuración de red
├── security.tf      # Grupos de seguridad
├── databases.tf     # Recursos de bases de datos
├── eks.tf           # Configuración del cluster EKS
├── iam.tf           # Roles y políticas IAM
└── terraform.tfvars.example  # Ejemplo de variables
```

▼ Explicación Detallada por Archivo

1. main.tf (Archivo Principal)

El archivo `main.tf` es el punto de entrada principal de la configuración y contiene:

1. **Backend Configuration**:
   - Configuración para almacenar el estado de Terraform en S3 (comentado por defecto)
   - Crucial para trabajar en equipo y mantener el estado seguro
2. **Data Sources**:
   - `aws_caller_identity`: Obtiene información de la cuenta AWS actual
   - `aws_region`: Obtiene la región AWS actual
   - `aws_availability_zones`: Obtiene zonas disponibles
3. **Recursos Core**:
   - KMS Key para encriptación
   - VPC Endpoints para servicios AWS (S3, ECR)
   - Parámetros SSM para almacenar información
4. **Locals**:
   - Define variables locales reutilizables
   - Establece lógica condicional para tipos de proyecto
   - Define tags comunes

2. versions.tf

Define las versiones de los providers necesarios:

- AWS Provider
- Kubernetes Provider
- Versión mínima de Terraform

Propósito: Asegurar consistencia y compatibilidad entre diferentes entornos.

3. variables.tf

Declara todas las variables utilizadas en el proyecto:

1. **Variables de Proyecto**:
   - `project_name`
   - `environment`
   - `project_type`
2. **Variables de Red**:
   - CIDRs
   - Zonas de disponibilidad
   - Configuración de subnets
3. **Variables de EKS**:
   - Versión del cluster
   - Configuración de nodos
   - Tipos de instancia
4. **Variables de Base de Datos**:
   - Tipo de instancia
   - Motor de base de datos
   - Versión del motor

4. vpc.tf

Configura toda la infraestructura de red:

1. **VPC Principal**:
    - CIDR block principal
    - DNS hostnames y support
2. **Subnets**:
    - Subnets públicas (para balanceadores y NAT Gateway)
    - Subnets privadas (para EKS y RDS)
    - Tags específicos para Kubernetes
3. **Gateways**:
    - Internet Gateway para acceso a internet
    - NAT Gateway para subnets privadas
4. **Tablas de Ruteo**:
    - Rutas públicas hacia Internet Gateway
    - Rutas privadas hacia NAT Gateway
    - Asociaciones de subnets

5. security.tf

Define los grupos de seguridad:

1. **EKS Security Group**:
    - Reglas de ingreso para el API server
    - Reglas de egreso para los nodos
2. **RDS Security Group**:
    - Acceso desde el cluster EKS
    - Puertos específicos según el motor (5432/3306)

6. databases.tf

Configura recursos de base de datos:

1. **RDS**:
    - Instancia de base de datos
    - Subnet group

- Configuración de backup

2. **DocumentDB (Opcional)**:
    - Cluster DocumentDB
    - Instancias del cluster
3. **S3 (Opcional)**:
    - Bucket con versionamiento
    - Encriptación por defecto

7. eks.tf

Implementa el cluster Kubernetes:

1. **Cluster EKS**:
    - Configuración del control plane
    - Networking
    - Endpoint access
2. **Node Group**:
    - Configuración de auto-scaling
    - Tipos de instancia
    - IAM roles asociados

8. iam.tf

Gestiona todos los permisos:

1. **Roles EKS**:
    - Role para el cluster
    - Role para los nodos
2. **Políticas**:
    - Políticas para EKS
    - Políticas para nodos
    - Políticas para S3 (opcional)

Flujo de Despliegue

1. **Preparación**:

```
cp terraform.tfvars.example terraform.tfvars
# Editar terraform.tfvars con valores específicos
```

2. **Inicialización**:

```
terraform init
```

3. **Planificación**:

```
terraform plan
```

4. **Aplicación**:

```
terraform apply
```

5. **Eliminar todos los recursos creados**

```
terraform destroy
```

Outputs de Terraform

Descripción General

El archivo `outputs.tf` define los valores que serán mostrados después de aplicar la configuración de Terraform y que pueden ser utilizados por otros módulos o para referencia.

Uso de los Outputs

Consulta de Outputs

```
# Ver todos los outputs
terraform output
```

```
# Ver un output específico
terraform output vpc_id
```

Exportación de Outputs

```
# Exportar todos los outputs en formato JSON
terraform output -json > outputs.json

# Usar en scripts
VPC_ID=$(terraform output -raw vpc_id)
```

▼ 3. Monitoreo Centralizado de Clusters EKS - Documentación y Plan de Implementación

Estado Actual del Proyecto

Durante mi investigación sobre la solución de monitoreo centralizado para clusters EKS, encontré una documentación detallada que se alinea perfectamente con nuestros requerimientos. Si bien no he implementado a detalle esta solución completa, he logrado configurar algunos componentes básicos y los he documentado en el actual documento.

Implementación Actual

1. Instalación básica de Grafana y Prometheus en un servidor EC2:

   ```
   docker-compose up -d grafana prometheus
   ```

2. Visualización inicial de métricas del sistema host.
3. Configuración básica de alertas.

Conocimientos a Adquirir

- Optimización de consultas en Prometheus.
- Gestión avanzada de logs con Loki.

Documentación de Referencia para Implementación Futura

1. Arquitectura

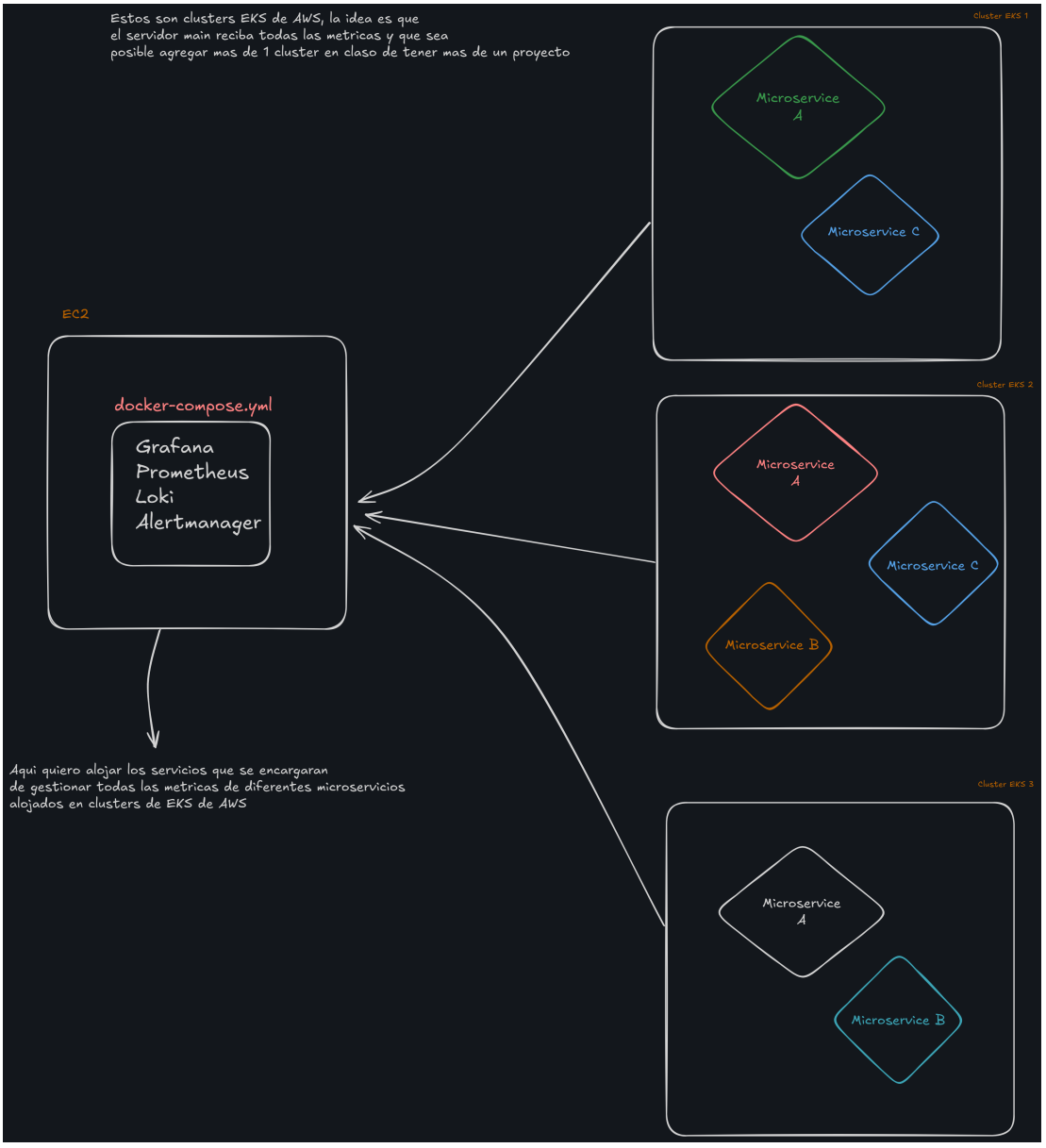
La arquitectura propuesta se basa en un servidor central de monitoreo (EC2) que aloja:

- **Grafana**: Visualización.
- **Prometheus**: Métricas.
- **Loki**: Logs.
- **Alertmanager**: Gestión de alertas.

1.1 Diagrama de Arquitectura

La solución implementa un modelo hub-and-spoke donde:

- El servidor central (EC2) actúa como hub.
- Los clusters EKS actúan como spokes.
- Cada cluster EKS envía datos a través de agentes de recolección.



2. Componentes

2.1 Stack de Monitoreo

- **Grafana**: Frontend unificado para visualización.
- **Prometheus**: Almacenamiento y consulta de métricas.
- **Loki**: Agregación y consulta de logs.
- **Alertmanager**: Gestión y enrutamiento de alertas.

2.2 Agentes

- **Prometheus Node Exporter**: Métricas de nodos.

- **Promtail**: Recolección de logs para Loki.
- **kube-state-metrics**: Métricas de Kubernetes.

---

3. Implementación

3.1 Prerequisitos

```
# Instalar herramientas necesarias
sudo yum update -y
sudo yum install -y docker git
sudo service docker start
sudo usermod -a -G docker ec2-user
```

3.2 Configuración del Servidor Central

```
mkdir -p /opt/monitoring
cd /opt/monitoring

# Crear docker-compose.yml
cat << 'EOF' > docker-compose.yml
[Contenido del archivo docker-compose.yml aquí]
EOF
```

3.3 Configuración de Prometheus

```
mkdir -p prometheus
cat << 'EOF' > prometheus/prometheus.yml
[Contenido del archivo prometheus.yml aquí]
EOF
```

---

4. Configuración

4.1 Configuración de EKS

```
# Aplicar en cada cluster EKS
kubectl create namespace monitoring
```

```
# Crear ServiceAccount para Prometheus
kubectl apply -f - <<EOF
[Contenido YAML aquí]
EOF
```

4.2 Configuración de Logs

```
# Configurar Promtail en cada cluster
kubectl apply -f - <<EOF
[Contenido YAML aquí]
EOF
```

---

5. Escalabilidad

5.1 Agregar Nuevo Cluster

1. Agregar configuración en `prometheus.yml`.
2. Configurar recolección de logs.
3. Actualizar Prometheus:

```
curl -X POST http://localhost:9090/-/reload
```

---

6. Pruebas

6.1 Verificación de Métricas

```
# Verificar targets en Prometheus
curl http://localhost:9090/api/v1/targets

# Verificar métricas básicas
curl http://localhost:9090/api/v1/query?query=up
```

6.2 Verificación de Logs

```
# Verificar ingesta de logs en Loki
curl -G -s "http://localhost:3100/loki/api/v1/query" \
  --data-urlencode 'query={job="kubernetes-pods"}' \
  --data-urlencode 'limit=10'
```

6.3 Dashboard Básico

1. Acceder a Grafana (`http://servidor:3000`).
2. Crear nuevo dashboard.
3. Añadir paneles para:
    - CPU por nodo.
    - Memoria por pod.
    - Tasa de errores.
    - Logs por severidad.

---

7. Consulta de Fallos en Tiempo Real

7.1 Acceso para Desarrolladores

Los desarrolladores pueden acceder a la información de fallos a través de múltiples interfaces:

1. **Grafana (Dashboard Principal)**:
    - URL: `http://servidor:3000`.
    - Credenciales: Proporcionadas por el equipo de DevOps.
    - Dashboards predefinidos para:
        - Errores HTTP por servicio.
        - Latencia de endpoints.
        - Logs de error agregados.
        - Estado de pods y servicios.

Referencias Bibliográficas

1. Guía de Monitoreo y Logging en EKS - Blog de Ankit Jodhani

2. Tutorial práctico sobre Kubernetes - Anvesh Muppeda
3. Serie de configuración local - Ahmad Bilal
4. Integración de Loki en EKS - Pavithra Sandamini

Recursos Adicionales Recomendados

- Documentación oficial de Grafana.
- Documentación oficial de Prometheus.
- Guías de AWS EKS.
- Documentación de Loki.