Diego Maigualca 2024-02-22

Documentación del Proyecto de API de Gestión de Usuarios

Introducción

Este documento proporciona una guía completa para desarrolladores y operadores sobre cómo configurar, desplegar y mantener la API de gestión de usuarios. La API permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre usuarios, facilitando la gestión de datos de usuario en una variedad de aplicaciones.

El proyecto está construido utilizando tecnologías modernas como Node.js para el backend, contenerización con Docker para facilitar el despliegue y la portabilidad, y Terraform para la provisión de infraestructura como código en AWS. Además, se integra con Jenkins para automatizar el proceso de integración y despliegue continuos (CI/CD), asegurando que cada cambio en el código sea automáticamente construido, probado y desplegado en un entorno de producción.

La documentación está estructurada para guiar a los usuarios a través de los siguientes aspectos clave del proyecto:

- Desarrollo del Microservicio: Instrucciones sobre cómo configurar el entorno de desarrollo, la estructura del proyecto, y cómo ejecutar y probar la API localmente.
- Contenedorización con Docker: Pasos para contenerizar la aplicación, construir imágenes Docker, y usar Docker Compose para orquestar contenedores localmente.
- Despliegue con Terraform en AWS: Detalles sobre cómo usar Terraform para crear y gestionar la infraestructura necesaria en AWS, y cómo desplegar la aplicación en este entorno.
- Integración Continua con Jenkins: Configuración de un pipeline en Jenkins para automatizar la construcción, pruebas y despliegue de la aplicación.

Objetivo:

Este documento está destinado a desarrolladores de software, ingenieros de DevOps, y cualquier persona involucrada en el ciclo de vida del desarrollo y operación de aplicaciones web basadas en microservicios.

Desarrollo del Microservicio

1. **Configuración del Entorno de Desarrollo**

- Instalar Node.js y npm en tu sistema.
- Clonar el repositorio del proyecto a tu máquina local.
- Navegar al directorio del proyecto y ejecutar npm install para instalar las dependencias necesarias.

2. **Estructura del Proyecto**

```
├── src/                  # Código fuente de la aplicación
│   ├── api/              # Definiciones de rutas para la API
│   ├── config/           # Configuraciones, incluyendo la base de datos
│   ├── controllers/      # Controladores para manejar la lógica de negocio
│   ├── models/           # Modelos de datos
│   └── index.js          # Punto de entrada de la aplicación
│
├── terraform/            # Archivos de configuración de Terraform para infraestructura
│
├── .dockerignore         # Archivo para excluir archivos y carpetas del contexto de Docker
├── .env.example          # Plantilla para las variables de entorno necesarias
├── .gitignore            # Archivo para excluir archivos y carpetas de Git
├── Dockerfile            # Definiciones para construir la imagen Docker de la aplicación
├── docker-compose.yml    # Orquesta el microservicio y la base de datos para el despliegue local
├── Jenkinsfile           # Define el pipeline de Jenkins para CI/CD
├── package.json          # Define las dependencias del proyecto
```

Describir la estructura de directorios del proyecto, similar a como se hizo en el README, detallando la función de cada carpeta y archivo importante.

3. **Desarrollo Local**

Iniciar el servidor de desarrollo con *npm run dev* o el comando relevante definido en package.json.

Realizar cambios en el código y probarlos localmente accediendo a http://localhost:3000 o el puerto configurado.

4. **Pruebas**

- Ejecutar npm test para correr las pruebas unitarias y de integración.
- Asegurarse de que todas las pruebas pasen antes de realizar un commit de los cambios.

Uso de Docker

Construcción de la Imagen Docker

- Asegúrate de tener Docker instalado en tu sistema.
- Navega al directorio raíz de tu proyecto donde se encuentra el Dockerfile.
- Ejecuta el siguiente comando para construir la imagen Docker de tu aplicación:

```
Dockerfile > ...
    You, 2 hours ago | 1 author (You)
1   # Utiliza una imagen base oficial de Node.js como punto de partida      You, 2 hours
2   FROM node:20
3
4   # Establece el directorio de trabajo dentro del contenedor
5   WORKDIR /app
6
7   # Copia los archivos de definición de dependencias
8   COPY package*.json ./
9
10  # Instala las dependencias de tu proyecto
11  RUN npm install
12
13  # Copia el resto del código fuente de tu aplicación al contenedor
14  COPY . .
15
16  # Expone el puerto en el que tu aplicación se ejecutará dentro del contenedor
17  EXPOSE 3000
18
19  # Comando para ejecutar tu aplicación
20  CMD ["node", "src/index.js"]
21
```

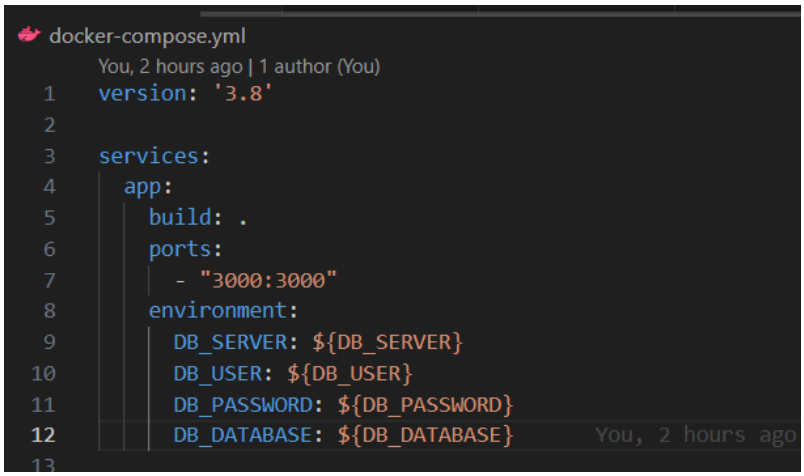
Uso de Docker Compose

Despliegue Local con Docker Compose

- Asegúrate de tener Docker Compose instalado.
- Navega al directorio donde se encuentra tu archivo docker-compose.yml.
- Ejecuta el siguiente comando para iniciar todos los servicios definidos en tu docker-compose.yml:

```
docker-compose up
```

```
PS C:\Users\PC\Documents\Proyectos\jelou> docker compose up
[+] Building 0.0s (0/0)                                                                                    docker:default
[+] Running 2/2
 ✓Network jelou_default  Created                                                                                    0.1s
 ✓Container jelou-app-1  Created                                                                                    0.1s
Attaching to jelou-app-1
Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:3000 -> 0.0.0.0:0: listen tcp 0.0.0.0:3000: bind: Only one usage of each sock
et address (protocol/network address/port) is normally permitted.
PS C:\Users\PC\Documents\Proyectos\jelou> docker compose up
[+] Building 0.0s (0/0)                                                                                    docker:default
Attaching to jelou-app-1
jelou-app-1  | {
jelou-app-1  |   user: 'R21MB',
jelou-app-1  |   password: 'R21MB',
jelou-app-1  |   server: '192.168.100.50',
jelou-app-1  |   database: 'jelou',
jelou-app-1  |   options: { encrypt: false, enableArithAbort: true }
jelou-app-1  | }
jelou-app-1  | Servidor corriendo en el puerto 3000
```



Integración Continua con Jenkins (Otra opción)

Configuración del Pipeline

- Asegúrate de tener Jenkins instalado y configurado correctamente.
- En la interfaz de Jenkins, crea un nuevo "Pipeline" y configura la fuente del código fuente para apuntar a tu repositorio donde se encuentra el Jenkinsfile.

Definición del Pipeline

- Tu Jenkinsfile debería definir los pasos necesarios para construir, probar y posiblemente desplegar tu aplicación. Asegúrate de que el Jenkinsfile esté en la raíz de tu proyecto.

Ejecución del Pipeline

- Una vez configurado el pipeline, Jenkins detectará automáticamente los cambios en tu repositorio (según la configuración) o puedes ejecutar el pipeline manualmente desde la interfaz de Jenkins.
- Monitorea la ejecución del pipeline en Jenkins para asegurarte de que todos los pasos se completen con éxito.

Despliegue

- Si tu pipeline incluye pasos de despliegue, asegúrate de que las credenciales y configuraciones necesarias estén correctamente establecidas en Jenkins para permitir el despliegue en el entorno deseado.

Notas Adicionales

- Docker: Asegúrate de que tus Dockerfile y docker-compose.yml estén correctamente configurados para tu aplicación, incluyendo puertos, volúmenes y otras dependencias necesarias.
- Jenkins: Dependiendo de la complejidad de tu pipeline, podrías necesitar plugins adicionales o configuraciones específicas para tu entorno de despliegue.
- Seguridad: Siempre considera las prácticas recomendadas de seguridad, especialmente cuando manejes credenciales y configuraciones sensibles en Jenkins y Docker.

Uso de Terraform

Requisitos Previos

5. Instala Terraform: Asegúrate de tener Terraform instalado en tu máquina. Puedes descargarlo desde terraform.io.
6. Configura AWS CLI: Instala y configura el AWS CLI con tus credenciales de AWS. Esto permitirá que Terraform acceda a tu cuenta de AWS. Puedes seguir la guía de AWS para instalar y configurar el CLI.
7. Prepara tu Aplicación: Asegúrate de que tu aplicación esté contenerizada y lista para ser desplegada. Si estás utilizando ECR, sube tu imagen de Docker al repositorio de ECR.

Pasos para el Despliegue con Terraform

1. **Inicialización de Terraform**
2. Navega al directorio donde se encuentra tu configuración de Terraform (donde están tus archivos .tf).

- Ejecuta el comando terraform init. Esto inicializará tu directorio de trabajo con Terraform, descargando los proveedores necesarios y preparando todo para el despliegue.

3. **Revisión del Plan de Despliegue**

- Ejecuta terraform plan para que Terraform evalúe tus archivos de configuración y muestre un resumen de los cambios que se aplicarán en tu infraestructura. Esto te permite revisar lo que Terraform hará antes de realizar cambios reales.

4. **Aplicar la Configuración**

- Una vez que hayas revisado y estés satisfecho con el plan de despliegue, ejecuta terraform apply.
- Terraform te pedirá que confirmes la acción antes de proceder. Escribe yes para continuar.
- Terraform creará los recursos definidos en tus archivos de configuración en AWS, incluyendo la infraestructura necesaria para tu microservicio.

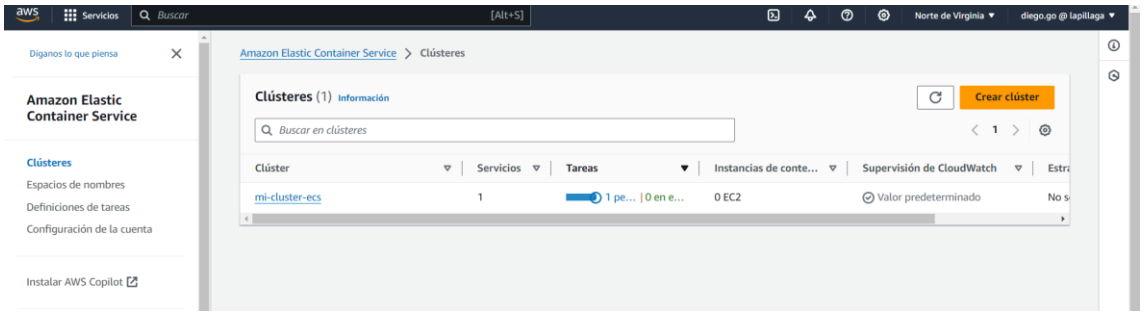
5. **Verificación**

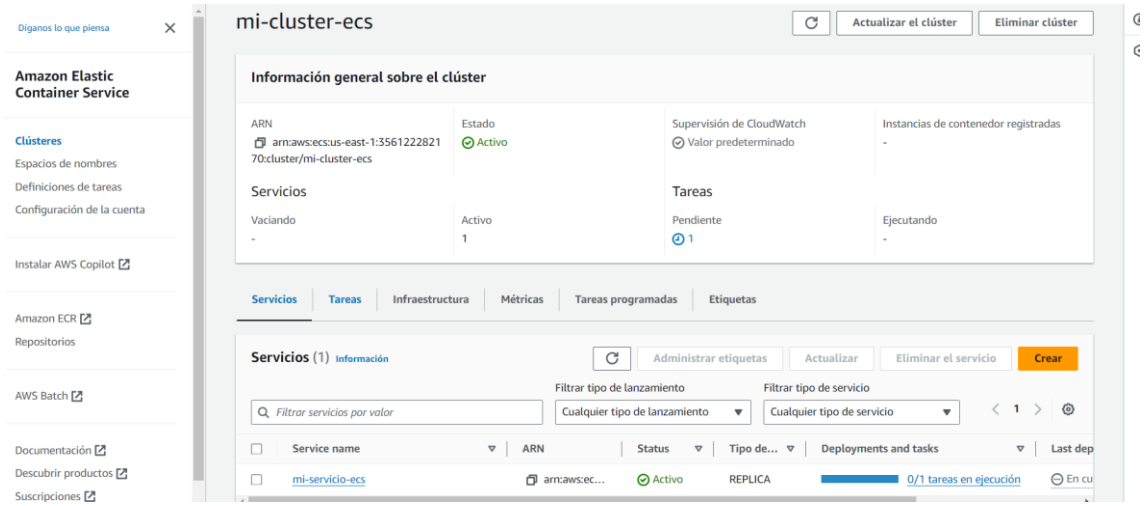
- Tras la finalización del proceso, Terraform mostrará los outputs definidos en tus archivos de configuración, los cuales pueden incluir direcciones IP, nombres DNS, y otros detalles importantes sobre los recursos creados.
- Verifica que todos los recursos se hayan creado correctamente en tu cuenta de AWS y que tu microservicio esté funcionando como se espera.

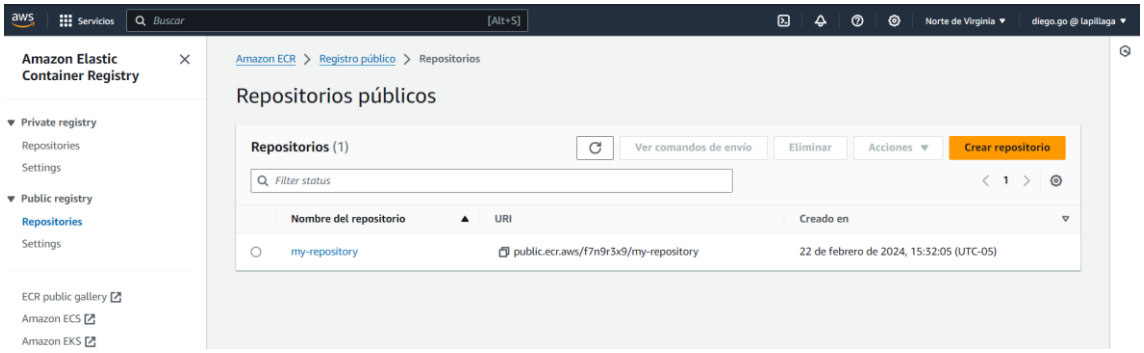
```
PS C:\Users\Usuario\OneDrive - Nagarro\NAGARRO\Documents\jelou\jelou_app_express> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.37.0...
- Installed hashicorp/aws v5.37.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

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

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

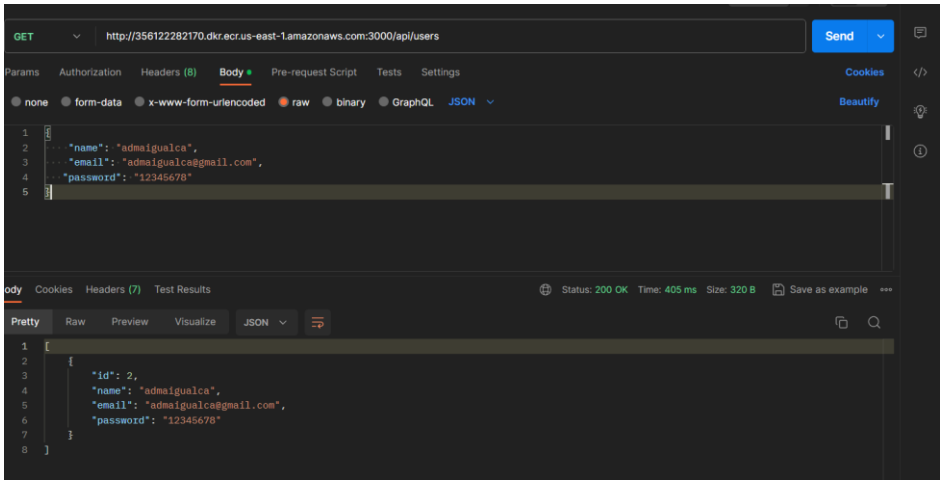
```
PS C:\Users\Usuario\OneDrive - Nagarro\NAGARRO\Documents\jelou\jelou_app_express> terraform apply
aws_vpc.example: Refreshing state... [id=vpc-0033035da05ab4ebe]
aws_subnet.example: Refreshing state... [id=subnet-0972c13547adef5e6]
aws_security_group.example: Refreshing state... [id=sg-01c3ece3f1c8726e2]

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:

  # aws_ecr_repository.app_repository will be created
  + resource "aws_ecr_repository" "app_repository" {
      + arn                  = (known after apply)
      + id                   = (known after apply)
      + image_tag_mutability = "MUTABLE"
      + name                 = "mi-aplicacion"
      + registry_id          = (known after apply)
      + repository_url       = (known after apply)
      + tags_all             = (known after apply)
    }

  # aws_ecs_cluster.app_cluster will be created
  + resource "aws_ecs_cluster" "app_cluster" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + name     = "mi-cluster-ecs"
      + tags_all = (known after apply)

      + setting {
          + name  = (known after apply)
          + value = (known after apply)
        }
    }

  # aws_ecs_service.app_service will be created
  + resource "aws_ecs_service" "app_service" {
      + cluster                            = (known after apply)
      + deployment_maximum_percent         = 200
      + deployment_minimum_healthy_percent = 100
      + desired_count                      = 1
      + enable_ecs_managed_tags            = false
```

```
PS C:\Users\Usuario\OneDrive - Nagarro\NAGARRO\Documents\jelou\jelou_app_express> terraform apply
aws_vpc.example: Refreshing state... [id=vpc-0033035da05ab4ebe]
aws_subnet.example: Refreshing state... [id=subnet-0972c13547adef5e6]
aws_security_group.example: Refreshing state... [id=sg-01c3ece3f1c8726e2]

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:

  # aws_ecr_repository.app_repository will be created
  + resource "aws_ecr_repository" "app_repository" {
      + arn                  = (known after apply)
      + id                   = (known after apply)
      + image_tag_mutability = "MUTABLE"
      + name                 = "mi-aplicacion"
      + registry_id          = (known after apply)
      + repository_url       = (known after apply)
      + tags_all             = (known after apply)
    }

  # aws_ecs_cluster.app_cluster will be created
  + resource "aws_ecs_cluster" "app_cluster" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + name     = "mi-cluster-ecs"
      + tags_all = (known after apply)

      + setting {
          + name  = (known after apply)
          + value = (known after apply)
        }
    }
```









Link Github:

https://github.com/damaigualca/jelou_app_express.git