Documento de Arquitectura de la Solución

1. Introducción..... 2
2. Descripción de la Arquitectura del Sistema..... 2
2.1. Componentes del sistema..... 2
2.2. Flujo de interacción..... 2
3. Arquitectura Cloud (En local en este caso)..... 3
3.1. Justificación del entorno local..... 3
4. Ciclo de vida del Software..... 4
4.1. Desarrollo local..... 4
4.2. Modelo de ramas..... 4
4.3. Integración continua..... 4
4.4. Despliegue..... 5
5. Infraestructura como Código..... 5
6. Despliegue con Ansible..... 5
7. Conclusión..... 6
8. Instrucciones para ejecución..... 6

1. Introducción

El presente documento describe detalladamente la arquitectura técnica y operativa de la solución diseñada para el reto final del Bootcamp DevOps. La solución gira en torno a una aplicación web escrita en Python (Flask) y aborda los principales bloques necesarios para poner en producción un software moderno: entorno local de desarrollo, integración continua, infraestructura como código y despliegue automatizado.

Este documento servirá como guía única de referencia durante el ciclo de vida de desarrollo, pruebas, despliegue y mantenimiento del producto.

2. Descripción de la Arquitectura del Sistema

2.1. Componentes del sistema

La solución consta de los siguientes componentes principales:

- Aplicación web: Desarrollada en Python con el framework Flask. Sirve como interfaz de usuario y lógica de negocio.
- Base de datos: PostgreSQL, utilizada para persistencia de datos estructurados.
- Docker: Utilizado para contenerizar tanto la aplicación web como la base de datos.
- Jenkins: Plataforma de automatización utilizada para definir y ejecutar pipelines de CI/CD.
- Terraform: Herramienta de IaC utilizada para definir la infraestructura mediante código.
- Ansible: Utilizado para automatizar el despliegue de versiones de la aplicación contenida en Docker.

2.2. Flujo de interacción

1. El desarrollador clona el repositorio desde GitHub.

2. Levanta el entorno local mediante docker-compose.
3. Desarrolla nuevas funcionalidades, ejecuta los tests (pytest) y valida el código con flake8.
4. Subida a Git activa un hook en Jenkins.
5. Jenkins ejecuta una pipeline compuesta por etapas de test, linting, construcción de imagen Docker y subida a un registry.
6. Terraform y Ansible permiten gestionar la infraestructura y desplegar la aplicación automáticamente.

3. Arquitectura Cloud (En local en este caso)

Debido a que se ha optado por una solución ejecutada localmente, no se ha utilizado una cuenta de AWS. Sin embargo, se ha simulado un entorno cloud mediante el uso de contenedores Docker orquestados por Terraform y Ansible, lo cual permite replicar escenarios productivos.

3.1. Justificación del entorno local

- Evita costes asociados a servicios cloud.
- Permite ejecución offline y control total del entorno.
- Facilita el aprendizaje práctico de herramientas como Docker, Terraform y Ansible.

De querer escalar a un entorno cloud en AWS, los servicios mapeables serían:

- EC2 → Para alojar contenedores Docker.
- ECR → Para alojar imágenes Docker.
- RDS → Para instancias PostgreSQL.
- S3 + CloudFront → Para servir contenido estático.

- CloudWatch → Para logs y métricas.

4. Ciclo de vida del Software

4.1. Desarrollo local

- Docker y Docker Compose permiten simular un entorno real de producción.
- PostgreSQL se ejecuta en contenedor, evitando dependencias en la máquina host.
- Tests automatizados y linting antes de subir cambios.

4.2. Modelo de ramas

Se sigue una estrategia basada en GitHub Flow:

- main: Rama estable que refleja el estado actual en producción.
- develop: Integración de nuevas funcionalidades.
- Ramas feature/, bugfix/ y hotfix/ se crean temporalmente y se integran mediante pull requests.

4.3. Integración continua

- Jenkins detecta cambios en el repositorio y ejecuta automáticamente las siguientes etapas:
    1. Clonado del repositorio
    2. Ejecución de tests con pytest
    3. Linting con flake8
    4. Construcción de imagen Docker
    5. Subida al registry (en este caso no se ha configurado uno, por lo que este paso se omite o se haría localmente)

4.4. Despliegue

El despliegue se realiza mediante Ansible, que:

1. Inicia el servicio Docker.
2. Elimina el contenedor anterior si existía.
3. Despliega la nueva versión de la imagen.

Esto permite:

- Despliegues sin interrupción.
- Rollback rápido en caso de fallos.

5. Infraestructura como Código

Se ha definido una infraestructura mínima con Terraform que:

- Crea una red Docker personalizada.
- Despliega un contenedor con una imagen nginx como simulación de entorno productivo.

Justificación de mantener IaC en el mismo repositorio:

- Evita desincronizaciones entre infraestructura y aplicación.
- Facilita el onboarding.
- Facilita el versionado conjunto.

6. Despliegue con Ansible

El playbook de Ansible permite desplegar automáticamente la aplicación:

- Utiliza el módulo docker_container para la gestión de contenedores.

- Variables como docker_image, docker_container_name, etc., permiten flexibilidad y reutilización.
- El inventario apunta a localhost, facilitando pruebas en entorno local.

7. Conclusión

La solución presentada replica de forma local una arquitectura de producción moderna, usando herramientas estándar de la industria y siguiendo buenas prácticas DevOps. Está pensada para ser mantenible, escalable y fácilmente ampliable a un entorno cloud si se dispone de los recursos necesarios.

8. Instrucciones para ejecución

```
# Requisitos previos:
- Docker
- docker-compose
- Terraform
- Ansible
- Python >= 3.8

# Clonar repositorio
$ git clone <URL>
$ cd reto_final_pythonV2

# Levantar entorno local
$ docker-compose up --build

# Ejecutar tests
$ docker-compose run --rm web pytest --cov=app --cov-report=term-missing

# Iniciar infraestructura con Terraform
$ cd infra
$ terraform init && terraform apply

# Desplegar con Ansible
$ cd ../ansible
$ ansible-playbook -i inventory.ini playbook.yml --ask-become-pass
```