Taller 1: Construcción de pipelines en Cloud

En este ejercicio debe realizar la construcción de un pipeline para el código disponible en:

https://github.com/bortizf/microservice-app-example

Para este proyecto, debe trabajar tanto del lado de desarrollo como de operaciones, considerando los diferentes aspectos para que el proyecto pueda ser utilizado por un equipo ágil. Inicialmente, deben escoger la metodología ágil a utilizar.

Aspectos a considerar para el desarrollo del taller:

1. 2.5% Estrategia de branching para desarrolladores
R//
La estrategia que proponemos para el equipo de desarrollo es Git Flow, un enfoque estructurado y ampliamente utilizado en equipos ágiles.

Estructura de ramas:

Rama :	Propósito
main :	Contiene código estable y listo para producción.
develop :	Contiene el código en estado estable para pruebas, con nuevas features integradas.
feature/* :	Ramas para el desarrollo de nuevas funcionalidades.
release/* :	Preparación de nuevas versiones a producción (pruebas finales, ajustes).
hotfix/* :	Corrección rápida de bugs en producción directamente sobre main.

Flujo de trabajo:

- Los desarrolladores crean ramas feature/nombre-funcionalidad desde develop.
- Una vez completada la feature, se hace un pull request (PR) a develop.
- Cuando varias funcionalidades están listas, se crea una rama release/x.x.x desde develop.
- Tras pruebas, release/x.x.x se fusiona a main y también a develop.
- Si hay bugs críticos en producción, se crea una rama hotfix/x.x.x desde main.

2. 2.5% Estrategia de branching para operaciones
R//
Para el equipo de operaciones, se define una estrategia de branching que permita mantener la infraestructura como código organizada, trazable y sincronizada con el desarrollo del sistema.

Enfoque: Infraestructura como Código (IaC)
Se utilizará una rama dedicada y una estructura de carpetas claras dentro del repositorio para gestionar scripts de infraestructura, pipelines y configuraciones relacionadas con despliegue.

Ramas utilizadas:

Rama	Propósito
infra	Rama dedicada para los archivos de infraestructura (IaC, Jenkins, Azure).
infra/dev	Rama para pruebas y cambios preliminares en la infraestructura.
infra/prod	Rama protegida para despliegue en producción.

3. 15.0% Patrones de diseño de nube (mínimo dos)
R//

Patrones implementados:

1. Retry Pattern

Descripción:

Este patrón permite que una operación fallida se intente nuevamente, con la esperanza de que el error haya sido transitorio (por ejemplo, pérdida temporal de conectividad con un microservicio).

Aplicación en el proyecto:

- Se implementa en el **Jenkinsfile**, usando el bloque retry(n) para reintentar pasos críticos como despliegues o builds fallidos.

2. Circuit Breaker Pattern

Descripción:

Este patrón evita que una aplicación siga intentando comunicarse con un servicio que está fallando, lo cual protege al sistema de sobrecarga y mejora su estabilidad.

Aplicación en el proyecto:

- Puede implementarse en alguno de los microservicios si se desea mostrar el patrón en ejecución.

3. Health Check

Justificación de elección:

Ambos patrones son muy utilizados en arquitecturas modernas basadas en microservicios y pipelines. Se complementan para mejorar tanto la **resiliencia operativa (Retry en Jenkins)** como la **estabilidad del sistema (Circuit Breaker en la lógica del microservicio)**.

4. 15.0% Diagrama de arquitectura
R//

5. 15.0% Pipelines de desarrollo (incluidos los scripts para las tareas que lo necesiten)
6. 5.0% Pipelines de infraestructura (incluidos los scripts para las tareas que lo necesiten)
7. 20.0% Implementación de la infraestructura
8. 15.0% Demostración en vivo de cambios en el pipeline
9. 10.0% Entrega de los resultados: debe incluir la documentación necesaria para todos los elementos desarrollados.

Presentación: 23 de abril. 8 minutos por grupo.

Documentación del Proyecto

1. Arquitectura y Patrones Implementados

- Patrón Circuit Breaker: Implementado en el Auth API para manejar fallos en la validación de JWT
- Patrón Retry: Implementado en el TODOs API para manejar reintentos en la escritura a Redis
- Arquitectura de Microservicios: Separación en módulos independientes
- Users API
- Auth API
- TODOs API
- Log Message Processor
- Frontend Vue

1. Configuración de Azure

- Recursos Creados:
- Resource Group: microservices-rg
- ACR: tuacr
- AKS: microservi-microservices-rg-7c197c
- Comandos Ejecutados:
- bash
- Apply to pipeline.yam...
- Run
- # Login y configuración
- az login
- az account set --subscription "tu-subscription-id"
-
- # Creación de recursos
- az group create --name microservices-rg --location eastus
- az acr create --resource-group microservices-rg --name tuacr --sku Basic
- az aks create --resource-group microservices-rg --name microservi-microservices-rg-7c197c --node-count 2 --enable-addons monitoring --generate-ssh-keys

-
- # Configuración de AKS
- az aks get-credentials --resource-group microservices-rg --name microservi-microservices-rg-7c197c

1. Configuración de Kubernetes

- Namespaces: dev y prod
- Servicios:
- Redis (ConfigMap, Deployment, Service)
- Ingress para enrutamiento
- Archivos de Configuración:
- infra/k8s/namespace.yaml
- infra/k8s/redis.yaml
- infra/k8s/ingress.yaml

1. Pipelines de Azure DevOps

- Conexiones de Servicio:
- acr-connection: Para Azure Container Registry
- k8s-dev-connection: Para Kubernetes
- Variables:
- ACR_NAME = tuacr
- RESOURCE_GROUP = microservices-rg
- AKS_NAME = microservi-microservices-rg-7c197c
- Pipelines por Servicio:
- Infraestructura
- Frontend
- Users API
- Auth API
- TODOs API
- Log Message Processor

1. Estructura de Directorios
2. text
3. Apply to pipeline.yam...
4. microservice-app-example/
5. ├── frontend/
6. ├── infra/
7. │ └── k8s/
8. ├── log-message-processor/
9. ├── todos-api/
10. ├── auth-api/
11. ├── users-api/
12. └── arch-img/