Repaso de computación en la Nube y servicios AWS

Temas: Automatización y despliegue en AWS

Objetivos

La automatización y el despliegue en AWS son fundamentales para optimizar los procesos de desarrollo y operaciones de software. Estas prácticas permiten a las organizaciones mejorar la confiabilidad, escalar aplicaciones de manera eficiente y reducir costos operativos.
La automatización y el despliegue en AWS son importantes porque:

- Mejora la eficiencia: Automatizar tareas y procesos permite una ejecución más rápida y confiable, reduciendo la intervención humana y el tiempo de respuesta.
- Aumenta la escalabilidad: Los servicios de AWS permiten escalar aplicaciones de manera eficiente, lo que es crucial para satisfacer las demandas de crecimiento y cambios en el mercado.
- Reduce costos operativos: La automatización y el despliegue en AWS reducen los costos operativos al minimizar la necesidad de personal y recursos para realizar tareas manuales.
- Mejora la confianza: La automatización y el despliegue en AWS mejoran la confiabilidad de las aplicaciones al minimizar la posibilidad de errores humanos y maximizar la consistencia en el despliegue.

Los objetivos de este repaso son profundizar estos temas en tres niveles: desde el punto de vista teórico, dsde el punto de vista páctico usando las herramientas de AWS y por codificación a través de conceptos rudimentarios de python.

Tareas
Prepara una presentación grupal para responder las siguientes preguntas y utiliza un ide de tu preferencia para las implementaciones solicitadas:

Preguntas

1 . Describe las ventajas de usar Amazon Elastic Beanstalk para la implementación de aplicaciones web en comparación con la configuración manual de una infraestructura en AWS.

Respuesta esperada:

- Abstracción de la infraestructura subyacente
- Gestión automática de la capacidad, balanceo de carga, escalado automático y monitoreo de la salud de la aplicación.
- Reducción de la complejidad operativa

- Soporte para múltiples lenguajes de programación y marcos de trabajo

2. Explica los componentes clave de Amazon Elastic Beanstalk y su función en el ciclo de vida de una aplicación.

Respuesta esperada:

- Environment: Contiene todos los recursos necesarios para ejecutar una aplicación.
- Application: Conjunto lógico de componentes que componen tu aplicación.
- Application version: Una versión específica del código de tu aplicación.
- Configuration template: Plantilla que define configuraciones predeterminadas para los recursos del entorno.

3. Define los beneficios del uso de Infraestructura como Código (IaC) y cómo Amazon CloudFormation facilita este enfoque.

Respuesta esperada:

- Consistencia y repetibilidad en la creación de infraestructura
- Reducción de errores humanos
- Simplificación del proceso de aprovisionamiento y despliegue
- Facilitación de la gestión del ciclo de vida de los recursos

4 . Explica qué son las plantillas de CloudFormation y qué elementos principales deben incluir.

Respuesta esperada:

- Resources: Los recursos AWS que se desean crear.
- Parameters: Valores de entrada para personalizar el comportamiento de la plantilla.
- Mappings: Asignaciones estáticas de valores.
- Conditions: Condiciones para crear recursos de manera condicional.
- Outputs: Salidas de los valores resultantes después de la creación de los recursos.

5. Describe qué son las pilas de CloudFormation y cómo se utilizan.

Respuesta esperada:

- Agrupación de todos los recursos AWS que se pueden gestionar como una sola unidad.
- Facilita el manejo y despliegue de múltiples recursos a la vez.

6. Explica qué son los conjuntos de cambios (Change Sets) y cómo se utilizan en CloudFormation.

Respuesta esperada:

- Permite previsualizar los cambios que se harán a los recursos antes de aplicarlos.
- Ayuda a identificar impactos y errores potenciales antes de la implementación.

7. Define qué es la detección de desvíos (Drift Detection) en CloudFormation y su importancia.

Respuesta esperada:

- Proceso de identificar recursos que han cambiado fuera del control de CloudFormation.
- Importancia de mantener la sincronización entre el estado real y el estado definido en la plantilla.

8 . Introduce AWS OpsWorks y cómo se integra con Chef y Puppet para la automatización de la configuración.

Respuesta esperada:

- Servicio que facilita la administración de configuraciones automatizadas usando Chef y Puppet.
- Simplificación de la configuración y despliegue de aplicaciones en servidores.

9. Explica qué son las pilas (Stacks) de AWS OpsWorks y cómo se estructuran.

Respuesta esperada:

- Grupo lógico de recursos gestionados por AWS OpsWorks.
- Incluye capas (layers), instancias (instances), y aplicaciones (apps).

10. Describe cómo AWS Lambda puede ser utilizado para la automatización de tareas IT y proporciona ejemplos específicos.

Respuesta esperada:

- Ejecución de código en respuesta a eventos, sin necesidad de gestionar servidores.
- Ejemplos: Automatización de backups, procesamiento de logs, gestión de infraestructura.

11. Explica el rol de AWS Lambda en la integración con otros servicios AWS para automatizar tareas de DevOps.

Respuesta esperada:

- Integración con Amazon CloudWatch para desencadenar funciones Lambda en base a eventos.
- Uso de AWS SDK dentro de funciones Lambda para interactuar con otros servicios AWS como S3, DynamoDB, EC2.

12 . Describe el proceso de despliegue de una aplicación utilizando Amazon Elastic Beanstalk y los diferentes métodos de despliegue disponibles (por ejemplo, Blue/Green, Rolling, Rolling with additional batch).

Respuesta esperada:
Proceso de despliegue:

- Preparación del código de la aplicación.
- Creación de una nueva versión de la aplicación en Elastic Beanstalk.
- Despliegue de la nueva versión en un entorno.

Métodos de despliegue:

- Blue/Green: Despliegue en un nuevo entorno, luego cambiar el tráfico.
- Rolling: Actualización de instancias en lotes.
- Rolling with additional batch: Adición de un nuevo lote de instancias antes de actualizar.

13, Explica cómo configurar y utilizar variables de entorno en Amazon Elastic Beanstalk.

Respuesta esperada:

- Configuración de variables de entorno en el archivo de configuración .ebextensions.
- Uso de la consola de Elastic Beanstalk para definir variables de entorno.
- Acceso a variables de entorno desde el código de la aplicación.

13. Proporciona un ejemplo de una plantilla básica de CloudFormation que cree una instancia EC2 y un bucket S3. Describe cada sección de la plantilla

Respuesta esperada:

Ejemplo de plantilla:

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyEC2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId: ami-0c55b159cbfafe1f0
  MyS3Bucket:
    Type: 'AWS::S3::Bucket'
```

Secciones de la plantilla:

- AWSTemplateFormatVersion: Versión del formato de la plantilla.
- Resources: Definición de los recursos a crear.

14. Describe cómo gestionar actualizaciones de infraestructura con CloudFormation sin causar interrupciones significativas en los servicios (por ejemplo, cambios que requieren recreación de recursos).

Respuesta esperada:

- Uso de conjuntos de cambios (Change Sets) para previsualizar y validar cambios.
- Aplicación de políticas de actualización que minimicen el impacto, como UpdatePolicy.
- Implementación de estrategias de despliegue como Blue/Green.

15. Explica cómo AWS OpsWorks for Chef Automate facilita la gestión de configuraciones y la orquestación de infraestructuras.

Respuesta esperada:

- Administración centralizada de recetas de Chef.
- Ejecución automática de configuraciones en respuesta a cambios.
- Monitoreo y reporte del estado de las configuraciones.

16. Describe un escenario donde la automatización de configuraciones con AWS OpsWorks y Chef puede mejorar la eficiencia operativa de una empresa.

Respuesta esperada:

- Ejemplo: Configuración automatizada de entornos de desarrollo, pruebas y producción.
- Beneficios: Reducción del tiempo de despliegue, consistencia en las configuraciones, escalabilidad.

4. Automatización de IT con AWS Lambda

17. Proporciona un ejemplo de cómo AWS Lambda puede integrarse con Amazon S3 para procesar archivos automáticamente cuando se suben a un bucket.

Respuesta esperada:

- Configuración de una función Lambda que se desencadena por eventos de S3.
- Ejemplo de código Lambda que procesa un archivo de texto subido:

```
import boto3
import json

def lambda_handler(event, context):
      s3 = boto3.client('s3')
    bucket = event['Records'][0]['s3']['bucket']['name']
     key = event['Records'][0]['s3']['object']['key']

     response = s3.get_object(Bucket=bucket, Key=key)
    content = response['Body'].read().decode('utf-8')

# Procesar el contenido del archivo
print("Contenido del archivo:", content)

return {
   'statusCode': 200,
            'body': json.dumps('Archivo procesado correctamente')
```

}

18. Discute las mejores prácticas para el manejo de errores y la monitorización de funciones Lambda en un entorno de producción.

Respuesta esperada:

- Implementación de manejo de errores y reintentos.
- Uso de AWS CloudWatch Logs para monitorear y registrar eventos.
- Configuración de métricas y alarmas en CloudWatch para detectar y responder a problemas.

AWS Lab learner

1. **Despliegue de Aplicaciones con Amazon Elastic Beanstalk**

Despliegue básico de una aplicación Web:

Objetivo: Desplegar una aplicación web sencilla utilizando Amazon Elastic Beanstalk.
Instrucciones:

- Crea una aplicación web sencilla en Python (puede ser una aplicación Flask).
- Empaqueta la aplicación en un archivo ZIP.
- Crear un nuevo entorno en Elastic Beanstalk y desplegar la aplicación.
- Accede a la URL proporcionada por Elastic Beanstalk y verifica que la aplicación esté funcionando.

Despliegue Blue/Green:

Objetivo: Implementar una estrategia de despliegue Blue/Green para minimizar el tiempo de inactividad.
Instrucciones:

- Crea dos entornos en Elastic Beanstalk (Blue y Green).
- Despliega la versión actual de la aplicación en el entorno Blue.
- Despliega una nueva versión de la aplicación en el entorno Green.
- Cambia el tráfico del entorno Blue al entorno Green.
- Verifica que la nueva versión de la aplicación esté funcionando correctamente.

2. Infraestructura como Código (IaC) con Amazon CloudFormation

Creación de una plantilla básica:

Objetivo: Crear y desplegar una plantilla de CloudFormation que configure una instancia EC2 y un bucket S3.
Instrucciones:

- Escribe una plantilla de CloudFormation en formato YAML.
- Define los recursos EC2 y S3 dentro de la plantilla.

- Sube la plantilla a Amazon S3.
- Crea una pila (stack) en CloudFormation utilizando la plantilla.
- Verifica que los recursos se hayan creado correctamente.

Uso de change sets:

Objetivo: Modificar una plantilla existente y utilizar un Change Set para aplicar los cambios.
Instrucciones:

- Crea una plantilla de CloudFormation que configure un grupo de seguridad para EC2.
- Despliegue la plantilla para crear la pila inicial.
- Modifica la plantilla para agregar reglas adicionales al grupo de seguridad.
- Crea un Change Set para la pila existente.
- Revisa y aplicar el Change Set.
- Verifica que los cambios se hayan aplicado correctamente.

3. Orquestación con Chef y Puppet utilizando AWS OpsWorks

Configuración de una pila de OpsWorks:
Objetivo: Configurar una pila de AWS OpsWorks para gestionar la configuración de un conjunto de instancias EC2 utilizando Chef.
Instrucciones:

- Crea una nueva pila en AWS OpsWorks.
- Configura una capa (layer) dentro de la pila para una aplicación web.
- Asocia una receta de Chef con la capa para instalar un servidor web (por ejemplo, Apache).
- Lanza una instancia EC2 dentro de la capa.
- Verifica que la instancia tenga el servidor web instalado y funcionando.

4. Automatización de IT con AWS Lambda

Procesamiento automático de archivos S3:

Objetivo: Crear una función Lambda que procese archivos automáticamente cuando se suben a un bucket S3.
Instrucciones:

- Crea un bucket S3 y configura una regla de evento para desencadenar una función Lambda en cargas de archivos.
- Escribe una función Lambda en Python que lea y procese el contenido de los archivos subidos.
- Despliega la función Lambda y asociarla con el evento del bucket S3.
- Sube un archivo al bucket S3 y verifica que la función Lambda se ejecute correctamente y procese el archivo.

Automatización de Backups:

Objetivo: Utilizar AWS Lambda para automatizar el backup de bases de datos RDS.
Instrucciones:

- Crea una base de datos RDS.
- Escribe una función Lambda que tome snapshots de la base de datos en intervalos regulares.
- Configura un evento de CloudWatch para desencadenar la función Lambda de acuerdo con un cronograma definido.
- Verifica que los snapshots de la base de datos se creen automáticamente según el cronograma.

Código

1. Despliegue de aplicaciones

Simulación de Elastic Beanstalk

Puedes crear un script de Python que simule el despliegue de una aplicación. Este script puede incluir:

- **Compresión del código de la aplicación** (simulando empaquetado)
- **Despliegue del paquete** en un entorno local (simulando el despliegue en Elastic Beanstalk)
- **Monitorización del estado del despliegue** (simulando la monitorización de la salud de la aplicación)

```
import os
import time
import zipfile

# Simulación de empaquetado de la aplicación
def package_app(app_dir, output_zip):
    with zipfile.ZipFile(output_zip, 'w') as zipf:
        for root, dirs, files in os.walk(app_dir):
            for file in files:
                zipf.write(os.path.join(root, file), os.path.relpath(os.path.join(root, file), app_dir))
    print(f"Aplicación empaquetada en {output_zip}")

# Simulación de despliegue de la aplicación
def deploy_app(zip_file):
    print(f"Desplegando {zip_file}...")
    time.sleep(2)
    print("Despliegue completado.")
    return "HEALTHY"

# Simulación de monitorización de la salud de la aplicación
def monitor_deployment(status):
```

```
    if status == "HEALTHY":
        print("La aplicación está funcionando correctamente.")
    else:
        print("Hay problemas con la aplicación.")

# Directorio de la aplicación y nombre del archivo ZIP
app_directory = 'mi_aplicacion'
output_zip_file = 'mi_aplicacion.zip'

package_app(app_directory, output_zip_file)
deployment_status = deploy_app(output_zip_file)
monitor_deployment(deployment_status)
```

2. Infraestructura como Código (IaC) con CloudFormation

Simulación de Plantillas CloudFormation

Puedes simular plantillas y despliegue de infraestructura utilizando JSON o YAML y Python para interpretarlas.

```
import json

import time

# Ejemplo de plantilla CloudFormation en JSON

cloudformation_template = {

    "AWSTemplateFormatVersion": "2010-09-09",

    "Resources": {

        "MyInstance": {

            "Type": "AWS::EC2::Instance",

            "Properties": {

                "InstanceType": "t2.micro",

                "ImageId": "ami-0c55b159cbfafe1f0"

            }

        },
```


```
        "MyBucket": {
            "Type": "AWS::S3::Bucket"
        }
    }
}
# Simulación de despliegue de la plantilla
def deploy_template(template):
    print("Desplegando plantilla CloudFormation...")
    time.sleep(2)
    print("Recursos creados:")
    for resource in template["Resources"]:
        print(f"- {resource}")
    return template["Resources"]


# Simulación de Change Set
def create_change_set(template, changes):
    print("Creando Change Set...")
    time.sleep(2)
    for change in changes:
        template["Resources"].update(change)
    print("Change Set aplicado.")
    return template
# Simulación de Drift Detection
def drift_detection(template, actual_state):
    print("Detectando desviaciones...")
```


```
    time.sleep(2)
    drifts = []
    for resource in template["Resources"]:
        if resource not in actual_state:
            drifts.append(resource)
    if drifts:
        print(f"Desviaciones detectadas en: {', '.join(drifts)}")
    else:
        print("No se detectaron desviaciones.")
# Estado actual simulado
actual_state = {
    "MyInstance": {
        "Type": "AWS::EC2::Instance",
        "Properties": {
            "InstanceType": "t2.micro",
            "ImageId": "ami-0c55b159cbfafe1f0"
        }
    }
}
# Despliegue inicial
deployed_resources = deploy_template(cloudformation_template)
# Aplicación de un Change Set
changes = {
    "MyBucket": {
        "Type": "AWS::S3::Bucket"
```


```
    }

}

updated_template = create_change_set(cloudformation_template, changes)

# Detección de desviaciones

drift_detection(updated_template, actual_state)
```

3. Orquestación con Chef y Puppet

Simulación con Scripts de Python

Puedes crear un script de Python que simule la ejecución de recetas de Chef o manifiestos de Puppet para configurar servidores.

```
import time

# Simulación de una receta de Chef
def chef_recipe():
    print("Ejecutando receta de Chef...")
    time.sleep(2)
    print("Servidor configurado con Apache.")

# Simulación de un manifiesto de Puppet
def puppet_manifest():
    print("Aplicando manifiesto de Puppet...")
    time.sleep(2)
    print("Servidor configurado con Nginx.")

# Ejecución de las simulaciones
chef_recipe()
puppet_manifest()
```

4. Automatización de IT con AWS Lambda

Simulación de AWS Lambda

Puedes crear funciones Lambda simuladas que se ejecuten en respuesta a eventos definidos en Python.


```
import time

# Simulación de una función Lambda
def lambda_function(event, context):
    print("Función Lambda desencadenada por evento.")
    print(f"Evento recibido: {event}")
    time.sleep(2)
    print("Función Lambda completada.")

# Simulación de un evento S3
s3_event = {
    "Records": [
        {
            "s3": {
                "bucket": {
                    "name": "mi_bucket"
                },
                "object": {
                    "key": "archivo.txt"
                }
            }
        }
    ]
}

# Ejecución de la función Lambda con el evento simulado
lambda_function(s3_event, None)
```

Código

Ejercicio 1: Simulación de un Ciclo Completo de Despliegue con Elastic Beanstalk

Objetivo

Simular un ciclo completo de despliegue de una aplicación web utilizando un script de Python. El ciclo incluye empaquetado, despliegue, monitorización y actualización de la aplicación.

Instrucciones

1. **Crear la aplicación web:** Desarrolla una pequeña aplicación web en Flask.
2. **Empaquetado:** Crea una función que empaquete la aplicación en un archivo ZIP.

3. **Despliegue inicial:** Crea una función que despliegue la aplicación (simulado).
4. **Monitorización:** Implementa la monitorización del estado de la aplicación.
5. **Actualización de la aplicación:** Simula una actualización de la aplicación y volver a desplegar.
6. **Registro y logs:** Implementarun sistema de registro de logs para cada etapa del ciclo de despliegue.

Ejercicio 2: Simulación de Infraestructura como Código (IaC) Completa con CloudFormation

Objetivo

Simular la gestión completa de la infraestructura utilizando CloudFormation con un script de Python. Incluye la creación de plantillas, despliegue de pilas, aplicación de conjuntos de cambios y detección de desviaciones.

Instrucciones

1. **Creación de una plantilla de CloudFormation:** Crea una plantilla que incluya una instancia EC2, un bucket S3 y una base de datos RDS.
2. **Despliegue de la pila inicial:** Despliega la plantilla inicial.
3. **Aplicación de un Change Set:** Modifica la plantilla y aplica los cambios usando un Change Set.
4. **Detección de desviaciones:** Implementa una función para detectar desviaciones entre la plantilla y el estado actual de la infraestructura.
5. **Registro y logs:** Implementa un sistema de registro de logs para cada etapa del ciclo de despliegue.

Ejercicio 3: Orquestación con Chef y Puppet Simulada con Python

Objetivo

Simular la orquestación de configuraciones de servidores utilizando Chef y Puppet con un script de Python. El script debe incluir la definición de recetas y manifiestos, la aplicación de configuraciones y la monitorización del estado.

Instrucciones

1. **Definición de recetas de Chef y manifiestos de Puppet:** Crea funciones que representen recetas de Chef y manifiestos de Puppet.
2. **Aplicación de configuraciones:** Implementa la aplicación de configuraciones a servidores simulados.
3. **Monitorización del estado de los servidores:** Implementa la monitorización del estado de los servidores después de aplicar las configuraciones.

4. **Registro y logs:** Implementa un sistema de registro de logs para cada etapa del proceso de orquestación.

```
import time
import logging

# Configuración del logger
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger()

# Simulación de una receta de Chef
def chef_recipe():
    logger.info("Ejecutando receta de Chef...")
    time.sleep(2)
    logger.info("Servidor configurado con Apache.")

# Simulación de un manifiesto de Puppet
def puppet_manifest():
    logger.info("Aplicando manifiesto de Puppet...")
    time.sleep(2)
    logger.info("Servidor configurado con Nginx.")

# Simulación de la aplicación de configuraciones a servidores
def apply_configuration(config_type):
    if config_type == "Chef":
        chef_recipe()
    elif config_type == "Puppet":
        puppet_manifest()
    else:
        logger.error("Tipo de configuración desconocido.")

# Simulación de monitorización del estado de los servidores
def monitor_servers():
    logger.info("Monitorizando el estado de los servidores...")
    time.sleep(2)
    logger.info("Todos los servidores están funcionando correctamente.")

# Aplicación de configuraciones y monitorización
apply_configuration("Chef")
apply_configuration("Puppet")
monitor_servers()
```

Ejercicio 4: Automatización de IT con Lambda simulada con Python

Objetivo: Simular la automatización de tareas de IT utilizando funciones Lambda con un script de Python. Incluye la creación de funciones Lambda, la configuración de eventos y la ejecución de funciones en respuesta a eventos.

Instrucciones

1. **Definición de funciones Lambda:** Crea funciones Lambda simuladas en Python.
2. **Simulación de eventos:** Crea eventos simulados que desencadenen las funciones Lambda.
3. **Ejecución de funciones Lambda:** Implementa la ejecución de las funciones Lambda en respuesta a eventos.
4. **Registro y logs:** Implementa un sistema de registro de logs para cada etapa del proceso de automatización.

Ejercicio 5: Simulación de despliegue y gestión de aplicaciones con Elastic Beanstalk

Objetivo: Simular el despliegue, gestión y escalado automático de una aplicación web utilizando Elastic Beanstalk. El script debe incluir empaquetado, despliegue, monitorización, escalado y actualización de la aplicación.

Instrucciones

1. Crea una aplicación web en Flask y empaquetarla en un archivo ZIP.
2. Implementa la simulación del despliegue de la aplicación en Elastic Beanstalk.
3. Monitorea el estado de la aplicación y simular el escalado automático basado en la carga
4. Actualiza la aplicación con una nueva versión y redeploy.
5. Implementa un sistema de registro de logs para cada etapa del ciclo de despliegue y escalado.

Ejercicio 6: Infraestructura Completa como Código (IaC) con CloudFormation

Objetivo: Simula la gestión completa de la infraestructura utilizando CloudFormation con un script de Python. El ejercicio incluye la creación de plantillas complejas, despliegue de pilas, aplicación de conjuntos de cambios, detección de desviaciones y manejo de dependencias entre recursos.

Instrucciones

1. Crea una plantilla de CloudFormation que incluya una instancia EC2, un bucket S3, una base de datos RDS y una función Lambda.
2. Despliega la plantilla inicial y manejar dependencias entre recursos.

3. Aplica un Change Set que modifique varios recursos a la vez.
4. Implementa una función para detectar desviaciones entre la plantilla y el estado actual de la infraestructura.
5. Registra los logs detallados de cada etapa del ciclo de despliegue y cambios aplicados.

Ejercicio 7: Orquestación avanzada con Chef y Puppet utilizando AWS OpsWorks

Objetivo: Simular la orquestación avanzada de configuraciones de servidores utilizando Chef y Puppet con un script de Python. Incluye la definición de recetas y manifiestos, la aplicación de configuraciones complejas, la monitorización del estado de los servidores y la gestión de roles y perfiles.

Instrucciones

1. Define recetas de Chef y manifiestos de Puppet que configuren servicios web complejos.
2. Aplica las configuraciones a servidores simulados y manejar dependencias entre servicios.
3. Implementa la monitorización del estado de los servidores y la reconfiguración automática en caso de fallos.
4. Gestiona roles y perfiles de servidores para diferentes entornos (desarrollo, pruebas, producción).
5. Registra los logs detallados de cada etapa del proceso de orquestación y configuración.

Ejercicio 8: Automatización completa de tareas de IT con AWS Lambda

Objetivo: Simular la automatización completa de tareas de IT utilizando funciones Lambda con un script de Python. Incluye la creación de funciones Lambda complejas, la configuración de eventos, la ejecución de funciones en respuesta a eventos y la integración con otros servicios simulados.

Instrucciones

1. Define varias funciones Lambda simuladas en Python que realicen tareas complejas (procesamiento de datos, generación de reportes, backups automáticos).
2. Crea eventos simulados que desencadenen las funciones Lambda (eventos de S3, CloudWatch, DynamoDB).
3. Implementa la ejecución de las funciones Lambda en respuesta a eventos y manejar la integración con otros servicios simulados (S3, DynamoDB, SNS).
4. Implementa un sistema de recuperación y reintento en caso de fallos en la ejecución de las funciones Lambda.
5. Registra los logs detallados de cada etapa del proceso de automatización y manejo de errores.

Ejercicio 9: Simulación de un ciclo de vida completo de una aplicación con Docker y Kubernetes

Objetivo_ Simular el ciclo de vida completo de una aplicación utilizando Docker y Kubernetes con un script de Python. Incluye la creación de imágenes Docker, despliegue en un clúster de Kubernetes, escalado automático, gestión de configuraciones y monitorización.

Instrucciones

- Crea un Dockerfile para una aplicación web en Flask y construir la imagen Docker.
- Implementa la simulación del despliegue de la imagen Docker en un clúster de Kubernetes.
- Simula el escalado automático del clúster basado en la carga y la gestión de configuraciones utilizando ConfigMaps y Secrets.
- Implementa la monitorización del estado del clúster y de los pods desplegados.
- Registra los logs detallados de cada etapa del ciclo de vida de la aplicación y del clúster de Kubernetes.