Repaso de computación en la Nube y servicios AWS

Temas: IAM, S3

Objetivos

El conocimiento profundo de AWS IAM y S3 proporciona una base sólida para la gestión segura y eficiente de la infraestructura en la nube. Los profesionales que dominan estos servicios pueden implementar soluciones escalables, seguras y conformes a las normativas, lo que es altamente valorado en el mercado laboral actual. Además, estas habilidades facilitan la optimización de costos y la mejora continua de la infraestructura tecnológica, aportando un valor significativo a las organizaciones.

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:

IAM

Ejercicios teóricos de IAM para AWS

Introducción al Servicio AWS IAM

Pregunta: Explica la funcionalidad principal del servicio AWS IAM y su importancia en la administración de accesos en la nube de AWS.

La consola de AWS IAM

Pregunta: Describe las diferentes secciones de la consola de AWS IAM y cómo se utilizan para gestionar identidades y permisos.

Servicios de AWS IAM

Pregunta: Enumera y describe brevemente los servicios principales que ofrece AWS IAM y cómo se integran con otros servicios de AWS.

Diferencias clave entre usuarios IAM y Grupos IAM

Pregunta: Compara y contrasta las diferencias clave entre usuarios IAM y grupos IAM, incluyendo sus usos y mejores prácticas.

Usuarios IAM

Pregunta: Explica el proceso de creación y gestión de un usuario IAM, incluyendo la asignación de permisos y políticas.

Grupos IAM

Pregunta: Describe cómo crear y gestionar grupos IAM, y cómo se pueden utilizar para simplificar la administración de permisos.

Definición de permisos con políticas IAM

Pregunta: Explica qué son las políticas IAM y cómo se utilizan para definir permisos. Proporciona un ejemplo de una política simple.

La Cuenta Root y la implementación de MFA

Pregunta: Discute la importancia de la cuenta root en AWS y las mejores prácticas para asegurarla, incluyendo la implementación de MFA (Multi-Factor Authentication).

Configuración de MFA

Pregunta: Describe el proceso de configuración de MFA para una cuenta root y un usuario IAM. ¿Por qué es importante habilitar MFA?

Importancia de definir políticas de contraseña de IAM

Pregunta: Explica por qué es importante definir políticas de contraseña en IAM y qué elementos deben incluirse en una política de contraseña robusta.

Tipos de políticas basadas en Identidades

Pregunta: Enumera y explica los diferentes tipos de políticas basadas en identidades que se pueden definir en IAM.

Ejemplo de una política IAM

Pregunta: Proporciona y analiza un ejemplo de política IAM que otorgue permisos de solo lectura a un bucket de S3 específico.

Asignación de credenciales temporales con roles IAM

Pregunta: Explica qué son los roles IAM y cómo se utilizan para asignar credenciales temporales. Proporciona un caso de uso común.

Credenciales temporales

Pregunta: Describe el concepto de credenciales temporales en IAM y cómo se pueden generar y utilizar de forma segura.

Revisión de Informes de credenciales

Pregunta: Discute la importancia de revisar los informes de credenciales en IAM y cómo se puede utilizar esta información para mejorar la seguridad.

Usa el laboratorio de AWS Lab Learner

Ejercicio 1: Creación y gestión de usuarios IAM

- **Tarea:** Crea tres usuarios IAM con diferentes niveles de permisos. Asigna políticas específicas a cada uno y documenta los pasos realizados.

Ejercicio 2: Configuración de MFA para usuarios IAM

- **Tarea:** Configura MFA para un usuario IAM y documenta el proceso, incluyendo cómo verificar el estado de MFA y cómo manejar la autenticación de doble factor.

Ejercicio 3: Definición y asignación de políticas IAM

- **Tarea:** Crea una política personalizada que otorgue permisos específicos a un bucket de S3. Asigna esta política a un usuario o grupo y verifica que los permisos funcionen correctamente.

Ejercicio 4: Implementación de Roles IAM para credenciales temporales

- **Tarea:** Configura un rol IAM que permita a una instancia EC2 acceder a un bucket de S3. Documenta el proceso y prueba el acceso desde la instancia.

Código

Podemos crear una serie de scripts y funciones que representen la gestión de usuarios, grupos, políticas y roles, así como la implementación de MFA y la generación de credenciales temporales.

1. Introducción al servicio AWS IAM
Crear una clase `IAMService` que actúe como el punto de entrada para gestionar usuarios, grupos, políticas y roles.

```
class IAMService:
    def __init__(self):
        self.users = {}
        self.groups = {}
        self.policies = {}
        self.roles = {}

iam_service = IAMService()
```

2. La consola de AWS IAM

Crear funciones que representen la interfaz de usuario de la consola de AWS IAM, permitiendo la creación y gestión de usuarios y grupos.

```
class IAMConsole:
    def create_user(self, iam_service, user_name):
        iam_service.users[user_name] = {'policies': [], 'mfa_enabled': False}
        print(f"User '{user_name}' created.")

    def create_group(self, iam_service, group_name):
        iam_service.groups[group_name] = {'policies': [], 'users': []}
        print(f"Group '{group_name}' created.")

iam_console = IAMConsole()
iam_console.create_user(iam_service, "alice")
iam_console.create_group(iam_service, "admin-group")
```

3. Servicios de AWS IAM

Crear métodos dentro de IAMService para gestionar usuarios, grupos y políticas.

```
def list_users(iam_service):
    return list(iam_service.users.keys())

def list_groups(iam_service):
    return list(iam_service.groups.keys())

print("Users:", list_users(iam_service))
print("Groups:", list_groups(iam_service))
```

4. Diferencias clave entre usuarios IAM y Grupos IAM

Crear funciones para añadir usuarios a grupos y asignar políticas a usuarios y grupos.

```
def add_user_to_group(iam_service, user_name, group_name):
    if user_name in iam_service.users and group_name in iam_service.groups:
        iam_service.groups[group_name]['users'].append(user_name)
        print(f"User '{user_name}' added to group '{group_name}'.")

def assign_policy_to_user(iam_service, user_name, policy):
    if user_name in iam_service.users:
        iam_service.users[user_name]['policies'].append(policy)
        print(f"Policy assigned to user '{user_name}'.")

def assign_policy_to_group(iam_service, group_name, policy):
    if group_name in iam_service.groups:
        iam_service.groups[group_name]['policies'].append(policy)
        print(f"Policy assigned to group '{group_name}'.")

add_user_to_group(iam_service, "alice", "admin-group")
assign_policy_to_user(iam_service, "alice", "AdminPolicy")
assign_policy_to_group(iam_service, "admin-group", "GroupAdminPolicy")
```

5. Definición de permisos con políticas IAM
Crear una estructura básica para representar políticas y asignarlas a usuarios y grupos.

```
class Policy:
    def __init__(self, name, permissions):
        self.name = name
        self.permissions = permissions

admin_policy = Policy("AdminPolicy", ["s3:ListBucket", "ec2:StartInstances"])
```

```
group_admin_policy = Policy("GroupAdminPolicy", ["s3:*", "ec2:*"])

iam_service.policies["AdminPolicy"] = admin_policy
iam_service.policies["GroupAdminPolicy"] = group_admin_policy

assign_policy_to_user(iam_service, "alice", admin_policy)
assign_policy_to_group(iam_service, "admin-group", group_admin_policy)
```

6. La cuenta root y la implementación de MFA

Crear una función para habilitar MFA en una cuenta.

```
def enable_mfa_for_user(iam_service, user_name):
    if user_name in iam_service.users:
        iam_service.users[user_name]['mfa_enabled'] = True
        print(f"MFA enabled for user '{user_name}'.")

enable_mfa_for_user(iam_service, "alice")
```

7. Configuración de MFA

Simular el proceso de configuración de MFA con un código de verificación.

```
import random

def setup_mfa(user_name):
    code = random.randint(100000, 999999)
    print(f"MFA setup for user '{user_name}'. Verification code: {code}")

setup_mfa("alice")
```

8. Importancia de definir políticas de contraseña de IAM

Crear una función para establecer políticas de contraseñas.

```
def set_password_policy(min_length, require_symbols, require_numbers):
    policy = {
        "min_length": min_length,
        "require_symbols": require_symbols,
        "require_numbers": require_numbers
```

```
    }
    print("Password policy set:", policy)

set_password_policy(8, True, True)
```

9. Tipos de políticas basadas en identidades

Crear ejemplos de políticas basadas en identidades y asignarlas a usuarios y grupos.

```
read_only_policy = Policy("ReadOnlyPolicy", ["s3:GetObject"])
write_policy = Policy("WritePolicy", ["s3:PutObject"])

iam_service.policies["ReadOnlyPolicy"] = read_only_policy
iam_service.policies["WritePolicy"] = write_policy

assign_policy_to_user(iam_service, "alice", read_only_policy)
assign_policy_to_group(iam_service, "admin-group", write_policy)
```

10. Ejemplo de una política IAM

Proveer un ejemplo de política y asignarla.

```
example_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::example_bucket"
        }
    ]
}

def assign_json_policy_to_user(iam_service, user_name, policy_json):
    if user_name in iam_service.users:
        iam_service.users[user_name]['policies'].append(policy_json)
        print(f"JSON policy assigned to user '{user_name}'.")

assign_json_policy_to_user(iam_service, "alice", example_policy)
`
```

11. Asignación de credenciales temporales con roles IAM

Crear una función para generar credenciales temporales.

```
import uuid
```

```
def generate_temporary_credentials():
    access_key = str(uuid.uuid4())
    secret_key = str(uuid.uuid4())
    session_token = str(uuid.uuid4())
    print(f"Temporary credentials generated:\nAccess Key: {access_key}\nSecret Key:
{secret_key}\nSession Token: {session_token}")

generate_temporary_credentials()
```

12. Revisión de informes de credenciales

Crear una función para revisar informes de credenciales.

```
def generate_credential_report(iam_service):
    report = {}
    for user in iam_service.users:
        report[user] = {
            "policies": iam_service.users[user]['policies'],
            "mfa_enabled": iam_service.users[user]['mfa_enabled']
        }
    print("Credential Report:", report)

generate_credential_report(iam_service)
```

Ejercicios

Ejercicio 1: Gestión avanzada de políticas y permisos

Tarea: Implementa una función para validar si un usuario tiene permisos específicos para realizar una acción en un recurso determinado. La función debe considerar las políticas asignadas directamente al usuario y aquellas asignadas a los grupos a los que pertenece el usuario.

Ejercicio 2: Simulación completa de MFA

Tarea: Implementa una simulación completa de MFA que incluya la generación de un código de verificación y la verificación de ese código al momento de inicio de sesión del usuario.

Ejercicio 3: Implementación de políticas basadas en condiciones

Tarea: Extiende la funcionalidad de las políticas IAM para incluir condiciones. Implementa una política que permita acciones solo durante un horario específico del día y verifica que los permisos se apliquen correctamente.

Ejercicio 4: Gestión de roles y asignación de credenciales temporales

Tarea: Implementa un sistema de roles que permita a los usuarios asumir roles y recibir credenciales temporales con permisos específicos. Implementa la lógica para verificar los permisos basados en los roles asumidos.

Ejercicio 5: Informe detallado de credenciales y seguridad

Tarea: Implementa una función que genere un informe detallado de credenciales y seguridad para todos los usuarios, incluyendo información sobre políticas, estado de MFA, y roles asumidos. El informe debe ser exportado en formato JSON.

S3

Introducción a las opciones de almacenamiento en AWS

Pregunta: Explica las diferentes opciones de almacenamiento disponibles en AWS y sus casos de uso más comunes. ¿Cuáles son las diferencias clave entre el almacenamiento en bloque, almacenamiento de archivos y almacenamiento de objetos?

Almacenamiento en Bloque

Pregunta: Describe el almacenamiento en bloque en AWS y sus ventajas. ¿Cómo se utiliza Amazon EBS (Elastic Block Store) y cuáles son sus principales características?

Almacenamiento de archivos

Pregunta: Explica el almacenamiento de archivos en AWS. ¿Qué es Amazon EFS (Elastic File System) y cómo se diferencia de Amazon FSx?

Almacenamiento de objetos

Pregunta: Define el almacenamiento de objetos y explica cómo Amazon S3 implementa este tipo de almacenamiento. ¿Cuáles son las ventajas del almacenamiento de objetos sobre los otros tipos de almacenamiento?

Introducción a Amazon S3

Pregunta: Proporciona una visión general de Amazon S3, incluyendo su arquitectura y cómo se organizan los datos en buckets y objetos.

Buckets y Objetos

Pregunta: Describe cómo crear y gestionar buckets y objetos en Amazon S3. ¿Cuáles son las mejores prácticas para nombrar y organizar buckets y objetos?

Gestión de objetos en un Bucket

Pregunta: Explica las diferentes maneras de gestionar objetos en un bucket de Amazon S3, incluyendo la carga, descarga y eliminación de objetos. ¿Qué herramientas y métodos se pueden utilizar para estas operaciones?

Alojamiento regional – disponibilidad Global

Pregunta: Discute la importancia del alojamiento regional en Amazon S3 y cómo asegura la alta disponibilidad y durabilidad de los datos.

Permisos de acceso

Pregunta: Describe cómo se gestionan los permisos de acceso en Amazon S3. ¿Qué son las políticas de bucket y las listas de control de acceso (ACL)?

Versionado

Pregunta: Explica el concepto de versionado en Amazon S3 y cómo se configura. ¿Cuáles son las ventajas de habilitar el versionado en un bucket?

Replicación en la misma región y en diferentes regiones

Pregunta: Describe las opciones de replicación en Amazon S3, tanto en la misma región como en diferentes regiones. ¿Cuáles son los casos de uso para cada tipo de replicación?

Encriptación en S3

Pregunta: Explica las opciones de encriptación disponibles en Amazon S3. ¿Cómo se configuran y gestionan las claves de encriptación?

Alojamiento de sitios web estáticos

Pregunta: Describe cómo se puede utilizar Amazon S3 para alojar sitios web estáticos. ¿Cuáles son los pasos necesarios para configurar un bucket de S3 para alojamiento web?

Amazon S3 Transfer Acceleration (S3TA)

Pregunta: Explica qué es Amazon S3 Transfer Acceleration y cómo mejora la velocidad de carga y descarga de objetos en Amazon S3.

Soluciones de archivado con Amazon S3 Glacier

Pregunta: Describe Amazon S3 Glacier y sus casos de uso. ¿Cómo se integran las soluciones de archivado en una estrategia de almacenamiento en AWS?

Conectando el almacenamiento On-Premises a AWS con Amazon Storage Gateway

Pregunta: Explica qué es Amazon Storage Gateway y cómo se utiliza para conectar el almacenamiento on-premises con AWS. ¿Cuáles son las diferentes configuraciones disponibles?

Migración de grandes volúmenes de datos a AWS con la familia Snow de AWS

Pregunta: Describe las diferentes soluciones disponibles en la familia Snow de AWS para la migración de grandes volúmenes de datos. ¿Cuáles son las diferencias entre AWS Snowball, Amazon Snowcone y Amazon Snowmobile?

AWS Snowball

Pregunta: Explica el funcionamiento de AWS Snowball y sus casos de uso. ¿Cuáles son las consideraciones clave al planificar una migración de datos con Snowball?

Amazon Snowcone

Pregunta: Describe Amazon Snowcone y sus casos de uso específicos. ¿Cómo se diferencia de otros dispositivos de la familia Snow?

Amazon Snowmobile

Pregunta: Proporciona una visión general de Amazon Snowmobile y sus capacidades. ¿En qué situaciones es más adecuado utilizar Snowmobile para la migración de datos?

Usa el laboratorio de AWS Lab Learner

Ejercicio 1: Creación y gestión de un bucket de Amazon S3

Objetivo: Crear un bucket de Amazon S3 y gestionar objetos dentro de él.

Instrucciones:

1. Inicia sesión en AWS Management Console.
2. Navega a Amazon S3.
3. Crea un nuevo bucket con un nombre único.
4. Sube varios archivos al bucket.
5. Organiza los archivos en carpetas dentro del bucket.
6. Configura permisos para que algunos archivos sean públicos y otros privados.
7. Elimina uno de los archivos del bucket.

Preguntas:

- ¿Qué configuraciones adicionales puedes aplicar al bucket?
- ¿Cómo se gestionan los permisos para los archivos en el bucket?

Ejercicio 2: Configuración de versionado en un Bucket de Amazon S3

Objetivo: Configurar el versionado en un bucket de Amazon S3 y observar cómo se manejan las versiones de los objetos.

Instrucciones:

1. Utiliza el bucket creado en el ejercicio anterior.
2. Habilita el versionado para el bucket.
3. Sube un archivo con el mismo nombre varias veces y observa cómo se gestionan las versiones
4. Elimina una versión específica del archivo.
5. Restaura una versión anterior del archivo.

Preguntas:

- ¿Qué ventajas tiene habilitar el versionado en un bucket?
- ¿Cómo puedes recuperar una versión eliminada accidentalmente?

Ejercicio 3: Configuración de replicación en diferentes regiones

Objetivo: Configurar la replicación entre dos buckets de Amazon S3 en diferentes regiones.

Instrucciones:

1. Crea un segundo bucket en una región diferente.
2. Configura la replicación para que los objetos subidos al primer bucket se repliquen automáticamente en el segundo bucket.
3. Sube un archivo al primer bucket y verifica que se replica correctamente en el segundo bucket.

Preguntas:

- ¿Qué pasos adicionales son necesarios para configurar la replicación entre regiones?
- ¿Cómo afecta la replicación a los costos de almacenamiento?

Ejercicio 4: Uso de Amazon S3 para alojamiento de sitios web estáticos

Objetivo: Configurar un bucket de Amazon S3 para alojar un sitio web estático.

Instrucciones:

1. Crea un nuevo bucket de Amazon S3.
2. Sube los archivos de un sitio web estático al bucket.
3. Configura el bucket para el alojamiento de sitios web estáticos.
4. Accede al sitio web utilizando el endpoint de Amazon S3.

Preguntas:

- ¿Qué configuraciones adicionales puedes aplicar al sitio web alojado en Amazon S3?
- ¿Cómo puedes hacer que el sitio web sea accesible mediante un nombre de dominio personalizado?

Ejercicio 5: Implementación de Amazon S3 Glacier para archivado de datos

Objetivo: Utilizar Amazon S3 Glacier para archivar datos de un bucket de S3.

Instrucciones:

1. Crea una política de ciclo de vida para el bucket de Amazon S3.
2. Configura la política para mover objetos a S3 Glacier después de un período específico.
3. Sube varios archivos al bucket y verifica que se mueven a S3 Glacier según la política de ciclo de vida.

Preguntas:

- ¿Qué beneficios ofrece Amazon S3 Glacier para el archivado de datos?
- ¿Cómo puedes recuperar datos archivados en S3 Glacier?

Ejercicio 6: Configuración de Amazon Storage Gateway

Objetivo: Configurar Amazon Storage Gateway para conectar el almacenamiento on-premises con AWS.

Instrucciones:

1. Despliega una instancia de Storage Gateway en tu entorno on-premises (simulado).
2. Conecta el Storage Gateway con tu cuenta de AWS.
3. Configura un volumen de almacenamiento que utilice Amazon S3 como backend.
4. Sube datos al volumen de Storage Gateway y verifica que se almacenan en Amazon S3.

Preguntas:

- ¿Cuáles son las diferentes configuraciones disponibles para Amazon Storage Gateway?
- ¿Qué casos de uso son adecuados para Amazon Storage Gateway?

Ejercicio 7: Migración de datos con AWS Snowball

Objetivo: Simular una migración de datos utilizando AWS Snowball.

Instrucciones:

1. Solicita un dispositivo Snowball desde AWS Management Console (simulado).
2. Configura el dispositivo Snowball para recibir datos.
3. Sube un conjunto de datos al dispositivo Snowball.
4. Verifica que los datos se han transferido correctamente a Amazon S3.

Preguntas:

- ¿Qué consideraciones debes tener al planificar una migración de datos con AWS Snowball?
- ¿Cómo se asegura la seguridad de los datos durante la migración?

Código

1. Simulación de opciones de Almacenamiento en AWS

1.1. Almacenamiento en bloque (Block Storage)

Simula un almacenamiento en bloque con archivos binarios en Python.

```
class BlockStorage:

  def __init__(self, size):

    self.storage = bytearray(size)


  def write(self, data, offset):

    self.storage[offset:offset+len(data)] = data
```


```
    def read(self, offset, size):
        return self.storage[offset:offset+size]

# Ejemplo de uso
block_storage = BlockStorage(1024)
block_storage.write(b"Hello", 0)
print(block_storage.read(0, 5))  # Output: b'Hello'
```

1.2. Almacenamiento de Archivos (File Storage)

Simula un sistema de archivos básico con directorios y archivos.

```
import os
 class FileStorage:
    def __init__(self, root_dir):
        self.root_dir = root_dir
        os.makedirs(root_dir, exist_ok=True)

    def write(self, path, data):
        with open(os.path.join(self.root_dir, path), 'w') as f:
            f.write(data)
    def read(self, path):
        with open(os.path.join(self.root_dir, path), 'r') as f:
            return f.read()
# Ejemplo de uso
file_storage = FileStorage('/tmp/filestorage')
file_storage.write('example.txt', 'Hello, File Storage!')
```

```
print(file_storage.read('example.txt'))  # Output: 'Hello, File Storage!'
```

1.3. Almacenamiento de Objetos (Object Storage)

Simula el almacenamiento de objetos usando un diccionario.

```
class ObjectStorage:
    def __init__(self):
        self.storage = {}

    def put_object(self, key, data):
        self.storage[key] = data

    def get_object(self, key):
        return self.storage.get(key, None)

# Ejemplo de uso
object_storage = ObjectStorage()
object_storage.put_object('file1.txt', 'Hello, Object Storage!')
print(object_storage.get_object('file1.txt'))  # Output: 'Hello, Object Storage!'
```

2. Introducción a Amazon S3

2.1. Buckets y Objetos

Simula la creación y gestión de buckets y objetos.

```
class S3Bucket:
    def __init__(self):
        self.buckets = {}
```


```
    def create_bucket(self, name):
        self.buckets[name] = {}

    def put_object(self, bucket, key, data):
        if bucket in self.buckets:
            self.buckets[bucket][key] = data

    def get_object(self, bucket, key):
        return self.buckets.get(bucket, {}).get(key, None)

# Ejemplo de uso
s3 = S3Bucket()
s3.create_bucket('mybucket')
s3.put_object('mybucket', 'file1.txt', 'Hello, S3 Bucket!')
print(s3.get_object('mybucket', 'file1.txt'))  # Output: 'Hello, S3 Bucket!'
```

3. Gestión de objetos en un Bucket

3.1. Permisos de acceso

Simula el manejo de permisos mediante un diccionario de permisos.

```
class S3BucketWithPermissions(S3Bucket):
    def __init__(self):
        super().__init__()
        self.permissions = {}
```


```
    def set_permission(self, bucket, key, permission):
        if bucket not in self.permissions:
            self.permissions[bucket] = {}
        self.permissions[bucket][key] = permission

    def check_permission(self, bucket, key, action):
        return self.permissions.get(bucket, {}).get(key) == action

# Ejemplo de uso
s3p = S3BucketWithPermissions()
s3p.create_bucket('mybucket')
s3p.put_object('mybucket', 'file1.txt', 'Hello, S3 with Permissions!')
s3p.set_permission('mybucket', 'file1.txt', 'read')
print(s3p.check_permission('mybucket', 'file1.txt', 'read'))  # Output: True
```

4. Versionado

4.1. Implementación de Versionado

Simula el versionado de objetos con una lista.

```
class S3BucketWithVersioning(S3Bucket):
    def __init__(self):
        super().__init__()
        self.versions = {}

    def put_object(self, bucket, key, data):
```

```
        if bucket not in self.versions:
            self.versions[bucket] = {}
        if key not in self.versions[bucket]:
            self.versions[bucket][key] = []
        self.versions[bucket][key].append(data)
    def get_object(self, bucket, key, version=None):
        if version is None:
            return self.versions.get(bucket, {}).get(key, [])[-1]
        return self.versions.get(bucket, {}).get(key, [])[version]

# Ejemplo de uso
s3v = S3BucketWithVersioning()
s3v.create_bucket('mybucket')
s3v.put_object('mybucket', 'file1.txt', 'Version 1')
s3v.put_object('mybucket', 'file1.txt', 'Version 2')
print(s3v.get_object('mybucket', 'file1.txt'))  # Output: 'Version 2'
print(s3v.get_object('mybucket', 'file1.txt', 0))  # Output: 'Version 1'
```

5. Replicación

5.1. Replicación en Diferentes Regiones

Simula la replicación de objetos entre dos buckets.

```
class S3BucketWithReplication(S3Bucket):
    def replicate(self, source_bucket, target_bucket):
        if source_bucket in self.buckets and target_bucket in self.buckets:
            self.buckets[target_bucket] = self.buckets[source_bucket].copy()
```


```
# Ejemplo de uso
s3r = S3BucketWithReplication()
s3r.create_bucket('source_bucket')
s3r.create_bucket('target_bucket')
s3r.put_object('source_bucket', 'file1.txt', 'Data to Replicate')
s3r.replicate('source_bucket', 'target_bucket')
print(s3r.get_object('target_bucket', 'file1.txt'))  # Output: 'Data to Replicate'
```

6. Encriptación

6.1. Encriptación de Objetos

Simula la encriptación de objetos antes de almacenarlos.

```
from cryptography.fernet import Fernet
class S3BucketWithEncryption(S3Bucket):
    def __init__(self, key):
        super().__init__()
        self.cipher = Fernet(key)

    def put_object(self, bucket, key, data):
        encrypted_data = self.cipher.encrypt(data.encode())
        super().put_object(bucket, key, encrypted_data)

    def get_object(self, bucket, key):
        encrypted_data = super().get_object(bucket, key)
```


```
        return self.cipher.decrypt(encrypted_data).decode()


# Ejemplo de uso

key = Fernet.generate_key()

s3e = S3BucketWithEncryption(key)

s3e.create_bucket('mybucket')

s3e.put_object('mybucket', 'file1.txt', 'Encrypted Data')

print(s3e.get_object('mybucket', 'file1.txt'))  # Output: 'Encrypted Data'
```

Ejercicios

Ejercicio 1: Simulación completa de almacenamiento en Bloque con gestión de espacio

Objetivo: Implementar una clase AdvancedBlockStorage que gestione bloques de almacenamiento con funciones para asignar y liberar bloques de forma dinámica, incluyendo fragmentación y desfragmentación.

Requisitos:

Implementar una clase AdvancedBlockStorage que permita:

- Crear un espacio de almacenamiento dividido en bloques de tamaño fijo.
- Asignar y liberar bloques.
- Manejar la fragmentación y ofrecer una función para desfragmentar el almacenamiento.

Incluir métodos para:

- allocate(size) que asigne bloques contiguos suficientes para almacenar el tamaño especificado.
- free(block_id) que libere los bloques asignados.
- defragment() que reorganice los bloques para reducir la fragmentación.

Ejercicio 2: Sistema de Archivos con gestión de permisos y versionado

Objetivo: Crear una clase VersionedFileStorage que extienda FileStorage para incluir permisos de acceso y versionado de archivos.

Requisitos:

Implementar una clase VersionedFileStorage que:

- Gestione archivos con permisos de lectura/escritura.
- Mantenga versiones de cada archivo.

Incluir métodos para:

- set_permission(path, permission) para establecer permisos (lectura/escritura).
- write_versioned(path, data) para escribir datos y mantener versiones.
- read_version(path, version) para leer una versión específica del archivo.

Ejercicio 3: Sistema de almacenamiento de Objetos con encriptación y replicación

Objetivo: Crear una clase EncryptedReplicatedObjectStorage que combine encriptación y replicación de objetos.

Requisitos:

Implementar una clase EncryptedReplicatedObjectStorage que:

- Encripte objetos antes de almacenarlos.
- Replicación automática de objetos en múltiples ubicaciones.

Incluir métodos para:

- put_object(bucket, key, data) para almacenar y encriptar datos.
- get_object(bucket, key) para recuperar y desencriptar datos.
- replicate(source_bucket, target_bucket) para replicar objetos.

Ejercicio 4: Gestión de versiones y replicación de archivos

Objetivo: Implementar un sistema de almacenamiento que gestione versiones de archivos y permita la replicación entre diferentes regiones simuladas.

Requisitos:

Implementar una clase VersionedReplicatedFileStorage que:

- Gestione versiones de archivos.
- Permita la replicación entre regiones simuladas.

Incluir métodos para:

- add_version(path, data) para añadir una nueva versión de un archivo.
- get_version(path, version) para obtener una versión específica de un archivo.
- replicate_region(region_from, region_to) para replicar todos los archivos de una región a otra.

Ejercicio 5: Sistema de almacenamiento de Objetos con soporte para políticas de acceso

Objetivo: Crear una clase ObjectStorageWithPolicies que gestione objetos y aplique políticas de acceso basadas en roles.

Requisitos:

Implementar una clase ObjectStorageWithPolicies que:

- Gestione objetos con políticas de acceso basadas en roles.
- Permita la definición y aplicación de políticas de acceso.

Incluir métodos para:

- put_object(bucket, key, data, role) para almacenar datos y asignar roles.
- get_object(bucket, key, role) para recuperar datos si el rol tiene acceso.
- define_policy(role, action, resource) para definir políticas de acceso.