Repaso de computación en la Nube y servicios AWS

Temas: Base de datos en AWS

Objetivos

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

Ejercicio 1: Bases de datos administradas vs. no Administradas

Pregunta:

- Define los conceptos de bases de datos administradas y no administradas.
- Discute las ventajas y desventajas de cada una.
- Proporciona ejemplos de escenarios en los que elegirías una base de datos administrada y otra en la que optarías por una base de datos no administrada. Justifica tu elección.

Ejercicio 2: Servicios de bases de datos para requisitos especializados

Pregunta:

- Investiga y explica al menos tres servicios de bases de datos de AWS diseñados para necesidades específicas (por ejemplo, Amazon Timestream, Amazon Quantum Ledger Database).
- Para cada servicio, describe un caso de uso ideal y cómo se diferenciaría de usar una base de datos más general.

Ejercicio 3: Conceptos y modelos de bases de datos

Pregunta:

- Describe los principales modelos de bases de datos (relacional, documental, de grafos, clave-valor, etc.).
- Para cada modelo, menciona un ejemplo de una base de datos de AWS que se ajuste a ese modelo y explica brevemente su uso principal.

Ejercicio 4: Introducción a Amazon RDS

Pregunta:

- Explica qué es Amazon RDS y los beneficios que ofrece sobre implementar una base de datos relacional en una instancia EC2.
- Describe el proceso de despliegue de una base de datos en Amazon RDS dentro de una Amazon VPC.

Ejercicio 5: Copias de seguridad y recuperación

Pregunta:

- ¿Qué mecanismos de respaldo y recuperación proporciona Amazon RDS?
- Diseña un plan de copias de seguridad y recuperación para una aplicación crítica utilizando Amazon RDS.

Ejercicio 6: Alta disponibilidad con Multi-AZ

Pregunta:

- Explica cómo funciona la arquitectura Multi-AZ en Amazon RDS.
- ¿Cuáles son los beneficios de usar Multi-AZ? Proporciona un escenario en el que sería crítico implementarlo.

Ejercicio 7: Escalabilidad horizontal con réplicas de lectura

Pregunta:

- Describe cómo se configuran y utilizan las réplicas de lectura en Amazon RDS.
- Proporciona un ejemplo de un caso de uso en el que las réplicas de lectura mejorarían significativamente el rendimiento.

Ejercicio 8: Introducción a Amazon Aurora

Pregunta:

- Explica qué es Amazon Aurora y cómo difiere de otros motores de base de datos compatibles con MySQL y PostgreSQL.

- Discute las características que hacen a Amazon Aurora una opción preferida para aplicaciones de alta disponibilidad y rendimiento.

Ejercicio 9: Amazon DynamoDB (solución de base de datos NoSQL)

Pregunta:

- Describe la estructura de tablas, elementos y atributos en Amazon DynamoDB.
- Explica cómo se puede provisionar la capacidad en DynamoDB y por qué es importante para gestionar el rendimiento y los costos.

Ejercicio 10: Uso de Amazon Redshift y almacenamiento de datos

Pregunta:

- Define qué es Amazon Redshift y para qué tipo de cargas de trabajo es más adecuado.
- Explica los conceptos de OLAP y cómo Redshift facilita estas operaciones.

Ejercicio 11: Importancia del caché en memoria con Amazon ElastiCache

Pregunta:

- ¿Qué es Amazon ElastiCache y cuáles son sus principales usos?
- Discute cómo el uso de caché en memoria puede mejorar el rendimiento de una aplicación web.

Ejercicio 12: Introducción a Amazon Neptune

Pregunta:

- Explica qué es Amazon Neptune y sus casos de uso más comunes.
- Proporciona un ejemplo de cómo una base de datos de grafos como Neptune podría ser utilizada en una aplicación de redes sociales.

Ejercicio 13: Amazon QLDB y la migración de bases de datos

Pregunta:

- Describe qué es Amazon QLDB y para qué tipo de aplicaciones es ideal.
- Explica el proceso general de migración de bases de datos utilizando el Servicio de Migración de Bases de Datos de AWS (DMS).

AWS Lab Learner

Ejercicio 1: Despliegue de Amazon RDS en una VPC

Objetivo: Configurar y desplegar una base de datos Amazon RDS dentro de una VPC.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Navega a Amazon RDS y crea una nueva instancia de base de datos (puede ser MySQL o PostgreSQL).
- Configura la instancia para que esté dentro de una VPC existente.
- Configura subnets, grupos de seguridad y ajustes de red para la instancia.
- Conéctate a la base de datos desde una instancia EC2 dentro de la misma VPC y realiza algunas operaciones básicas (creación de tablas, inserción de datos).

Resultados esperados:

- Una instancia de RDS funcional dentro de una VPC.
- Conexión exitosa a la base de datos desde una instancia EC2.
- Operaciones básicas de bases de datos realizadas correctamente.

Ejercicio 2: Configuración de Multi-AZ en Amazon RDS

Objetivo: Configurar una instancia de Amazon RDS con Multi-AZ para alta disponibilidad.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Crea una nueva instancia de Amazon RDS o modifica una existente para habilitar la opción Multi-AZ.
- Verifica la creación de la réplica en una zona de disponibilidad diferente.
- Simula una falla en la zona de disponibilidad primaria y observa la conmutación por error a la réplica secundaria.

Resultados esperados:

- Configuración exitosa de Multi-AZ en Amazon RDS.
- Conmutación por error automática a la réplica secundaria sin pérdida de datos.

Ejercicio 3: Uso de réplicas de lectura en Amazon RDS

Objetivo: Configurar y utilizar réplicas de lectura en Amazon RDS para mejorar la escalabilidad horizontal.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Crea una réplica de lectura para una instancia de Amazon RDS existente.
- Configura la aplicación para distribuir las consultas de lectura entre la instancia principal y la réplica de lectura.

1. Realiza pruebas de rendimiento para medir la mejora en la capacidad de lectura.

Resultados esperados:

- Réplica de lectura creada y funcionando.
- Aplicación configurada para usar la réplica de lectura.
- Mejora en el rendimiento de las consultas de lectura.

Ejercicio 4: Creación y configuración de una tabla en Amazon DynamoDB

Objetivo: Crear y configurar una tabla en Amazon DynamoDB y realizar operaciones CRUD.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Navega a Amazon DynamoDB y crea una nueva tabla con una clave primaria.
- Configura la capacidad provisionada para la tabla.
- Usa AWS CLI o SDK para insertar, leer, actualizar y eliminar elementos en la tabla.
- Configura índices secundarios globales (GSI) y realiza consultas utilizando los índices.

Resultados esperados:

- Tabla DynamoDB creada y configurada correctamente.
- Operaciones CRUD realizadas exitosamente.
- Índices secundarios configurados y utilizados en consultas.

Ejercicio 5: Despliegue y configuración de Amazon ElastiCache

Objetivo: Desplegar y configurar un clúster de ElastiCache para mejorar el rendimiento de una aplicación.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Crea un clúster de ElastiCache usando Redis.
- Conéctate al clúster desde una aplicación en una instancia EC2 y configura el caché para almacenar resultados de consultas frecuentes.
- Realiza pruebas de rendimiento para medir la mejora proporcionada por el caché.

Resultados esperados:

- Clúster de ElastiCache creado y configurado.
- Conexión exitosa desde la aplicación a ElastiCache.
- Mejora en el rendimiento de la aplicación debido al uso del caché.

Ejercicio 6: Creación y configuración de Amazon Redshift

Objetivo: Desplegar y configurar un clúster de Amazon Redshift y realizar consultas OLAP.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Crea un clúster de Amazon Redshift.
- Configura las tablas y carga datos en el clúster utilizando COPY desde Amazon S3.
- Realiza consultas OLAP complejas para analizar los datos cargados.

Resultados esperados:

- Clúster de Amazon Redshift creado y configurado.
- Datos cargados exitosamente desde Amazon S3.
- Consultas OLAP realizadas y resultados analizados.

Ejercicio 7: Configuración de Amazon Neptune

Objetivo: Configurar y usar Amazon Neptune para almacenar y consultar datos de grafos.

Instrucciones:

- Inicia sesión en AWS Lab Learner.
- Crea una instancia de Amazon Neptune.
- Carga un conjunto de datos de grafos en Neptune utilizando Gremlin o SPARQL.
- Realiza consultas para encontrar relaciones y patrones en los datos del grafo.

Resultados esperados:

- Instancia de Amazon Neptune creada y configurada.
- Datos de grafos cargados exitosamente.
- Consultas realizadas y resultados obtenidos.

Código

Ejercicio 1: Bases de Datos Administradas vs. No Administradas

Objetivo: Entender la diferencia entre bases de datos administradas y no administradas.

Instrucciones:

- Investiga y define bases de datos administradas y no administradas.
- Escribe un script en Python que simule la creación de una base de datos no administrada utilizando SQLite y una base de datos administrada (simulada) utilizando una clase personalizada que automatice tareas comunes (backup, recuperación, etc.).

```
import sqlite3
import time

# Simulación de base de datos no administrada
conn = sqlite3.connect('unmanaged.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
cursor.execute('''INSERT INTO users (name) VALUES ('Alice')''')
conn.commit()
conn.close()

# Simulación de base de datos administrada
class ManagedDatabase:
    def __init__(self, db_name):
        self.conn = sqlite3.connect(db_name)
        self.cursor = self.conn.cursor()
        self.create_backup()

    def execute_query(self, query):
        self.cursor.execute(query)
        self.conn.commit()
        self.create_backup()

    def create_backup(self):
        with open('backup.sql', 'w') as f:
            for line in self.conn.iterdump():
                f.write('%s\n' % line)
        print("Backup created at", time.ctime())

managed_db = ManagedDatabase('managed.db')
managed_db.execute_query('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name
TEXT)''')
managed_db.execute_query('''INSERT INTO users (name) VALUES ('ZZ')''')
```

Ejercicio 2: Servicios de bases de Datos para Requisitos Especializados

Objetivo: Explorar servicios de bases de datos especializados.

Instrucciones:

- Investiga tres tipos de bases de datos especializadas.
- Escribe un script en Python que simule operaciones básicas en tres bases de datos diferentes utilizando bibliotecas de Python como tinydb para bases de datos JSON, networkx para grafos y sqlalchemy para bases de datos SQL.

```
from tinydb import TinyDB, Query
import networkx as nx
from sqlalchemy import create_engine, Column, Integer, String, Base

# Base de datos JSON con TinyDB
db = TinyDB('db.json')
db.insert({'type': 'specialized', 'name': 'JSON Database'})

# Base de datos de grafos con NetworkX
G = nx.Graph()
G.add_node("Alice")
G.add_node("ZZ")
G.add_edge("Alice", "ZZ")
print("Graph nodes:", G.nodes)
print("Graph edges:", G.edges)

# Base de datos relacional con SQLAlchemy
engine = create_engine('sqlite:///specialized.db')
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)

Base.metadata.create_all(engine)
```

Ejercicio 3: Modelos de bases de datos

Objetivo: Comprender diferentes modelos de bases de datos.

Instrucciones:

- Escribe un script en Python que cree y manipule diferentes tipos de bases de datos: relacional (SQLite), documental (TinyDB), y de grafos (NetworkX).

```
# Relacional (SQLite)
import sqlite3

conn = sqlite3.connect('relational.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
conn.commit()
conn.close()

# Documental (TinyDB)
from tinydb import TinyDB, Query

db = TinyDB('documental.json')
db.insert({'name': 'Alice'})

# Grafos (NetworkX)
import networkx as nx

G = nx.Graph()
G.add_node("Alice")
G.add_node("ZZ")
G.add_edge("Alice", "ZZ")
```

Ejercicio 4: Simulación de Amazon RDS

Objetivo: Simular una base de datos administrada.

Instrucciones:

- Crea una clase en Python que simule un servicio de base de datos administrada, implementando características como copias de seguridad automáticas y recuperación ante fallos.

```
class SimulatedRDS:
    def __init__(self, db_name):
        self.conn = sqlite3.connect(db_name)
        self.cursor = self.conn.cursor()
```


```
        self.create_backup()

    def execute_query(self, query):
        self.cursor.execute(query)
        self.conn.commit()
        self.create_backup()

    def create_backup(self):
        with open('backup.sql', 'w') as f:
            for line in self.conn.iterdump():
                f.write('%s\n' % line)
        print("Backup created at", time.ctime())

    def restore_from_backup(self):
        with open('backup.sql', 'r') as f:
            sql_script = f.read()
        self.cursor.executescript(sql_script)
        self.conn.commit()
        print("Database restored from backup")

rds = SimulatedRDS('simulated_rds.db')
rds.execute_query('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
rds.execute_query('''INSERT INTO users (name) VALUES ('Alice')''')
```

Ejercicio 5: Simulación de Amazon DynamoDB

Objetivo: Simular operaciones de una base de datos NoSQL como DynamoDB.

Instrucciones:

- Utiliza tinydb para simular una base de datos NoSQL, realizando operaciones CRUD y gestionando la capacidad provisionada.

```
from tinydb import TinyDB, Query

# Creación de la tabla
db = TinyDB('dynamodb.json')

# Inserción de datos
db.insert({'id': 1, 'name': 'Alice'})
db.insert({'id': 2, 'name': 'ZZ'})
```


```
# Consulta de datos
User = Query()
result = db.search(User.name == 'Alice')
print(result)

# Actualización de datos
db.update({'name': 'Alice Updated'}, User.id == 1)

# Eliminación de datos
db.remove(User.id == 2)
```

Ejercicio 6: Simulación de Amazon Redshift

Objetivo: Simular un almacén de datos y operaciones OLAP.

Instrucciones:

- Utiliza pandas y sqlite3 para simular la carga de grandes volúmenes de datos y realizar consultas analíticas.

```
import pandas as pd
import sqlite3

# Creación de la base de datos
conn = sqlite3.connect('redshift.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS sales (id INTEGER PRIMARY KEY, amount REAL, date
TEXT)''')
conn.commit()

# Carga de datos
data = [(1, 100.0, '2023-01-01'), (2, 200.0, '2023-01-02')]
cursor.executemany('INSERT INTO sales VALUES (?, ?, ?)', data)
conn.commit()

# Consulta OLAP
df = pd.read_sql_query('SELECT date, SUM(amount) as total_sales FROM sales GROUP BY date',
conn)
print(df)
```

Ejercicio 7: Simulación de ElastiCache

Objetivo: Simular un caché en memoria.

Instrucciones:

Utiliza redis-py para simular operaciones de caché en memoria.

```
import redis

# Conexión a Redis
r = redis.Redis(host='localhost', port=6379, db=0)

# Inserción de datos en caché
r.set('user:1', 'Alice')
r.set('user:2', 'ZZ')

# Recuperación de datos del caché
print(r.get('user:1'))  # Output: Alice
print(r.get('user:2'))  # Output: ZZ
```

Ejercicio 8: Simulación de Amazon Neptune

Objetivo: Simular una base de datos de grafos.

Instrucciones:

- Utiliza networkx para crear y consultar una base de datos de grafos.

```
import networkx as nx

# Creación del grafo
G = nx.Graph()
G.add_node("Alice")
G.add_node("ZZ")
G.add_edge("Alice", "ZZ")

# Consultas en el grafo
print("Nodes:", G.nodes)
print("Edges:", G.edges)
```

Ejercicio 9: Simulación de Amazon QLDB

Objetivo: Simular una base de datos de libro mayor.

Instrucciones:

- Crea una clase en Python que registre transacciones y permita consultas de estado.

```
class Ledger:
    def __init__(self):
        self.transactions = []

    def record_transaction(self, transaction):
        self.transactions.append(transaction)

    def get_transactions(self):
        return self.transactions

ledger = Ledger()
ledger.record_transaction({'id': 1, 'action': 'create', 'data': 'Alice'})
ledger.record_transaction({'id': 2, 'action': 'update', 'data': 'Alice Updated'})

print(ledger.get_transactions())
```

Ejercicio 10: Simulación de Database Migration Service (DMS)

Objetivo: Simular la migración de datos entre bases de datos.

Instrucciones:

- Escribe un script en Python que copie datos de una base de datos SQLite a otra.

```
import sqlite3

# Base de datos de origen
src_conn = sqlite3.connect('source.db')
src_cursor = src_conn.cursor()
src_cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
src_cursor.execute('''INSERT INTO users (name) VALUES ('Alice')''')
src_conn.commit()

# Base de datos de destino
dst_conn = sqlite3.connect('destination.db')
dst_cursor = dst_conn.cursor()
```

```
dst_cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
dst_conn.commit()

# Migración de datos
for row in src_cursor.execute('SELECT * FROM users'):
    dst_cursor.execute('INSERT INTO users VALUES (?, ?)', row)
dst_conn.commit()

# Verificación de datos en la base de datos de destino
for row in dst_cursor.execute('SELECT * FROM users'):
    print(row)
```

Ejercicio 11: Simulación de unsistema de gestión de bases de datos administradas y No administradas

Objetivo: Desarrollar un sistema completo que gestione bases de datos administradas y no administradas, permitiendo operaciones CRUD, respaldo y recuperación automática, y monitoreo del rendimiento.

Instrucciones:

- Diseña una clase base Database que soporte operaciones CRUD.
- Implementa una clase UnmanagedDatabase que extienda Database y utilice SQLite.
- Implementa una clase ManagedDatabase que extienda Database, automatizando respaldos, recuperación, y monitoreo de rendimiento.
- Implementa un sistema de monitoreo que registre y muestre estadísticas de rendimiento (tiempo de respuesta, tasa de errores, etc.).
- Diseña un script que permita la creación, operación, y gestión de instancias de bases de datos administradas y no administradas.

Ejercicio 12: Implementación de un motor de bases de datos especializadas

Objetivo: Crear un sistema que soporte múltiples tipos de bases de datos especializadas y permita la interacción entre ellas.

Instrucciones:

- Implementa una base de datos documental utilizando tinydb.
- Implementa una base de datos de grafos utilizando networkx.
- Implementa una base de datos relacional utilizando sqlalchemy.
- Diseña un sistema que permita consultas y operaciones cruzadas entre las diferentes bases de datos.

- Implementa un sistema de sincronización de datos que mantenga la coherencia entre las bases de datos especializadas.

Ejercicio 13: Simulación de un Almacén de Datos y Procesamiento OLAP

Objetivo: Crear un sistema de almacén de datos que permita la carga masiva de datos, procesamiento OLAP, y generación de informes.

Instrucciones:

- Implementa un sistema de carga de datos que ingeste grandes volúmenes de datos desde diferentes fuentes (archivos CSV, bases de datos relacionales, etc.).
- Diseña un sistema de almacenamiento que organice los datos en esquemas de estrella y copo de nieve.
- Implementa un motor de consultas OLAP que soporte operaciones de agregación, filtrado y análisis de datos.
- Diseña un sistema de generación de informes que presente los resultados de las consultas OLAP en formatos visuales (tablas, gráficos).
- Desarrolla un script que simule diferentes escenarios de carga y consulta, evaluando el rendimiento del sistema.

Ejercicio 14: Desarrollo de un Sistema de Caché en Memoria

Objetivo: Crear un sistema de caché en memoria que emule las funcionalidades de Amazon ElastiCache utilizando Redis.

Instrucciones:

- Implementa una clase InMemoryCache que soporte operaciones de almacenamiento y recuperación de datos.
- Diseña un sistema que permita la configuración y gestión del caché (políticas de expiración, capacidad máxima, etc.).
- Implementa un mecanismo de respaldo que permita la persistencia de datos del caché en almacenamiento secundario.
- Diseña un sistema de invalidación y actualización de caché que mantenga la coherencia de datos con una base de datos principal.
- Desarrolla un script que simule diferentes escenarios de uso del caché y analice el impacto en el rendimiento de la aplicación.

Ejercicio 15 [Opcional]: Implementación de un servicio completo de migración de datos

Objetivo: Desarrollar un sistema de migración de datos que soporte múltiples fuentes y destinos de bases de datos, emulando las funcionalidades de AWS DMS.

Instrucciones:

- Implementa una clase DataSource que soporte la conexión a diferentes tipos de bases de datos (relacional, NoSQL).
- Diseña un sistema que permita la extracción, transformación y carga (ETL) de datos entre bases de datos de diferentes tipos.
- Implementa un mecanismo de sincronización en tiempo real que mantenga los datos consistentes entre las bases de datos origen y destino durante la migración.
- Diseña un sistema de monitoreo que registre el progreso, estado y rendimiento de la migración.
- Desarrolla un script que simule la migración de una base de datos grande, evaluando la integridad, consistencia y rendimiento del sistema.