Investigación

Bases de Datos SQL:

- Definición: Son bases de datos relacionales que almacenan datos en tablas con filas y columnas, y utilizan el lenguaje SQL para realizar operaciones de gestión de datos.
- Uso: Se utilizan en aplicaciones que requieren un esquema de datos estructurado y consistente, con un alto nivel de integridad de datos y transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- Casos de uso: Son adecuadas para aplicaciones bancarias, sistemas de reservas, sistemas de gestión de recursos humanos, etc., donde los datos tienen una estructura bien definida y se requiere una alta consistencia.
- Ejemplos: MySQL, PostgreSQL, Oracle, SQL Server, SQLite.

Bases de Datos NoSQL:

- Definición: Son bases de datos no relacionales que almacenan datos en estructuras diferentes a las tablas, como documentos, pares clave-valor, grafos o columnas de familia.
- Uso: Se utilizan en aplicaciones que requieren escalabilidad horizontal, alto rendimiento y flexibilidad en el esquema de datos.
- Casos de uso: Son adecuadas para aplicaciones de redes sociales, análisis de datos a gran escala, almacenamiento de datos no estructurados, aplicaciones de Internet de las Cosas (IoT), etc.
- Ejemplos: MongoDB (bases de datos de documentos), Redis (bases de datos clave-valor), Neo4j (bases de datos de grafos), Apache Cassandra (bases de datos de columnas de familia).

Implementación en NodeJS

Para conectar Node.js con bases de datos relacionales SQL, existen varios paquetes y bibliotecas que podemos utilizar. Aquí están algunos ejemplos:

1. **Sequelize**: Es un ORM (Object-Relational Mapping) para Node.js que admite varios motores de bases de datos SQL, como MySQL,

PostgreSQL, SQLite y Microsoft SQL Server. Sequelize nos permite interactuar con la base de datos utilizando código JavaScript en lugar de consultas SQL directas. Ejemplo de uso:

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('database', 'username',
'password', {
  host: 'localhost',
  dialect: 'mysql'
});

const User = sequelize.define('User', {
  username: DataTypes.STRING,
  email: DataTypes.STRING
});

// Realizar operaciones CRUD
User.create({ username: 'john', email: 'john@example.com'
});
User.findAll();
User.update({ email: 'newemail@example.com' }, { where: {
id: 1 } });
User.destroy({ where: { id: 1 } });
```

Node-MySQL: Es un controlador nativo de MySQL para Node.js que nos permite ejecutar consultas SQL directamente en la base de datos. Ejemplo de uso:

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'username',
  password: 'password',
  database: 'database'
});
```

```
connection.query('SELECT * FROM users', (err, results) =>
{
  if (err) throw err;
  console.log(results);
});
```

Bases de Datos NoSQL en Node.js:

1. **MongoDB**: Es una base de datos NoSQL de documentos ampliamente utilizada. Para trabajar con MongoDB en Node.js, podemos utilizar el controlador oficial `mongodb` o el ODM (Object Document Mapping) `mongoose`. Ejemplo con `mongoose`:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

const User = mongoose.model('User', { name: String,
email: String });

const user = new User({ name: 'John', email:
'john@example.com' });
user.save((err) => {
  if (err) return handleError(err);
  // Usuario guardado
});
```

Redis: Es una base de datos NoSQL de estructura clave-valor en memoria. Podemos utilizar el paquete `redis` en Node.js para interactuar con Redis.
Ejemplo:

```
const redis = require('redis');
const client = redis.createClient();

client.on('error', (err) => console.log('Redis Client
Error', err));

client.set('key1', 'value1', (err, reply) => {
  if (err) throw err;
  console.log(reply); // OK
});

client.get('key1', (err, reply) => {
  if (err) throw err;
  console.log(reply); // 'value1'
});
```

Caso practico

Razones para utilizar Amazon ElastiCache para Redis en este servicio:

1. **Velocidad y rendimiento**: Redis es una base de datos en memoria, lo que significa que las operaciones de lectura y escritura son extremadamente rápidas. Esto es crucial para un servicio de notificaciones, donde se requiere un alto rendimiento para procesar y enviar notificaciones en tiempo real.
2. **Estructura de datos clave-valor simple**: Las notificaciones se pueden almacenar de manera eficiente como pares clave-valor en Redis. La clave puede ser un identificador único de la notificación, y el valor puede contener los detalles de la notificación, como el destinatario, el contenido, el canal de entrega, etc.

3. **Escalabilidad y alta disponibilidad**: Amazon ElastiCache para Redis ofrece escalabilidad horizontal y alta disponibilidad integradas. Esto permite que el servicio de notificaciones se escale fácilmente a medida que crece la demanda, y garantiza una disponibilidad confiable.
4. **Persistencia opcional**: Redis ofrece la opción de almacenar datos en memoria o persistirlos en disco. En Amazon ElastiCache para Redis, puedes configurar la persistencia en disco para garantizar la durabilidad de los datos en caso de un reinicio o falla.
5. **Integración con otros servicios AWS**: Amazon ElastiCache para Redis se integra de manera fluida con otros servicios de AWS, como AWS Lambda, Amazon SNS (Simple Notification Service), Amazon SQS (Simple Queue Service), y más. Esto facilita la construcción de un flujo de trabajo de notificaciones completo y escalable.
6. **Funciones adicionales**: Redis ofrece funciones adicionales útiles para un servicio de notificaciones, como colas de mensajes (listas), conjuntos, caché, pub/sub, entre otros.