Building an App with NoSQL Database Cassandra

Corrales Solis Moises Alessandro

Abstract:
Apache Cassandra is a powerful NoSQL database known for its high availability, scalability, and ability to manage large volumes of data. Originally developed by Facebook, Cassandra is widely used today in applications where low latency and high fault tolerance are critical. This article will guide you through the basics of building an app with Cassandra, from understanding its architecture to setting up a simple CRUD application.

Resumen:
Apache Cassandra es una potente base de datos NoSQL conocida por su alta disponibilidad, escalabilidad y capacidad para gestionar grandes volúmenes de datos. Desarrollada originalmente por Facebook, Cassandra es ampliamente utilizada hoy en día en aplicaciones donde la baja latencia y la alta tolerancia a fallos son críticas. Este artículo le guiará a través de los conceptos básicos de la construcción de una aplicación con Cassandra, desde la comprensión de su arquitectura hasta la creación de una sencilla aplicación CRUD.

Keywords: Apache Cassandra, NoSQL Database, High Availability, Scalability, Distributed Architecture, CRUD Operations, Data Model, Keyspace, Partition Key, Node.js Integration.

Why Choose Cassandra?

Cassandra is optimized for distributed data storage, making it ideal for applications that need:

- High Write Throughput: Cassandra's write-oriented architecture enables it to handle large amounts of data, especially suited for data logging and real-time applications.
- Fault Tolerance and High Availability: Cassandra's distributed and decentralized nature means it has no single point of failure. Data is replicated across multiple nodes, ensuring that the app remains operational even if some nodes go down.
- Linear Scalability: Cassandra can scale horizontally by adding more nodes to the cluster, making it well-suited for applications that need to grow over time.

Cassandra Architecture Overview

Cassandra's architecture differs significantly from traditional relational databases. Here's a quick look at some of its key components:

- Nodes: Each server in a Cassandra cluster is called a node, and they store data and respond to read/write requests.
- Clusters and Rings: Nodes are organized into clusters and communicate via a "ring" structure. Data is distributed across nodes in a ring-like topology using consistent hashing.
- Partition Keys and Replication: Cassandra uses partition keys to determine data distribution across nodes, and replication ensures copies of data are stored on multiple nodes.
- Data Model: Unlike relational databases, Cassandra's schema design uses keyspaces, tables, and partitions. Tables are organized by rows and columns, but it's optimized for high-speed reads and writes rather than complex queries.

Setting Up Cassandra

To start building an app with Cassandra, you need to set up a local or cloud-based Cassandra database environment. Here's how you can get started:

Step 1: Install Cassandra

1. Download the latest version of Cassandra from the Apache Cassandra website.
2. Follow the installation guide for your operating system.
3. Start the Cassandra server using the following command:

"cassandra -f"

Step 2: Set Up the CQL Shell

CQL (Cassandra Query Language) is the language used to interact with Cassandra. After starting Cassandra, open the CQL shell using:

"cqlsh"

Designing Your Data Model

Cassandra's data modeling is different from traditional relational databases. Instead of normalizing tables, Cassandra encourages denormalized tables to improve read performance.

Key Concepts of Cassandra Data Model

- Keyspace: Similar to a database in an RDBMS, a keyspace defines the scope of data replication across nodes. Each keyspace has one or more tables.
- Tables and Columns: Tables in Cassandra are more flexible than relational tables and allow for wide rows.
- Partition Key: Partition keys determine data distribution across nodes and should be chosen carefully to avoid uneven data distribution.
- Primary Key: Consists of the partition key and clustering columns, used to uniquely identify each row.

Example Data Model

For an example app, let's assume we're building a simple social media app with a "users" table and a "posts" table.

```
CREATE KEYSPACE social_app
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': '3'
};

USE social_app;

CREATE TABLE users (
  user_id UUID PRIMARY KEY,
  name TEXT,
  email TEXT,
  created_at TIMESTAMP
);

CREATE TABLE posts (
  post_id UUID PRIMARY KEY,
  user_id UUID,
  content TEXT,
  created_at TIMESTAMP
);
```

The users table will store information about each user, while the `posts` table will store the posts created by each user.

Building the App with Cassandra

We'll build a simple CRUD (Create, Read, Update, Delete) API to interact with the Cassandra database. This example uses Node.js and the cassandra-driver package, but the concepts apply to other programming languages as well.

Step 1: Set Up Node.js Project

1. Create a new directory for the project and initialize it:

   mkdir cassandra-app

   cd cassandra-app

   npm init -y

2. Install the Cassandra driver:

   npm install cassandra-driver

Step 2: Connect to Cassandra

Create a file called database.js to set up the connection.

```
// database.js

const cassandra = require('cassandra-driver');


const client = new cassandra.Client({
  contactPoints: ['127.0.0.1'],
  localDataCenter: 'datacenter1',
  keyspace: 'social_app'
});
client.connect()
  .then(() => console.log('Connected to Cassandra'))
  .catch(err => console.error('Failed to connect to Cassandra', err));
```

```
module.exports = client;
```

Step 3: Create CRUD Operations

In app.js, we'll define the CRUD operations for our users and posts tables.

- Create a User:

```
// app.js

const client = require('./database');

const createUser = async (name, email) => {

  const query = 'INSERT INTO users (user_id, name, email, created_at) VALUES (uuid(), ?, ?, toTimestamp(now()))';

  await client.execute(query, [name, email], { prepare: true });

  console.log('User created');

};
```

- Read User Information:

```
const getUser = async (userId) => {

  const query = 'SELECT * FROM users WHERE user_id = ?';

  const result = await client.execute(query, [userId], { prepare: true });

  return result.rows[0];

};
```

- Update User Information:

```
const updateUser = async (userId, name, email) => {

  const query = 'UPDATE users SET name = ?, email = ? WHERE user_id = ?';

  await client.execute(query, [name, email, userId], { prepare: true });
```

```
  console.log('User updated');
};
```

- Delete User:

```
const deleteUser = async (userId) => {
  const query = 'DELETE FROM users WHERE user_id = ?';
  await client.execute(query, [userId], { prepare: true });
  console.log('User deleted');
};
```

Step 4: Integrate with a REST API

Using Express for a simple REST API, we can expose these CRUD operations to interact with the Cassandra database.

1. Install Express:

   ```
   npm install express
   ```

2. Create server.js to set up the API:

   ```
   const express = require('express');
   const { createUser, getUser, updateUser, deleteUser } = require('./app');
   const app = express();
   app.use(express.json());

   app.post('/users', (req, res) => {
     createUser(req.body.name, req.body.email)
       .then(() => res.send('User created'))
   ```

```
    .catch(err => res.status(500).send(err));
});

app.get('/users/:id', (req, res) => {
  getUser(req.params.id)
    .then(user => res.json(user))
    .catch(err => res.status(500).send(err));
});

app.put('/users/:id', (req, res) => {
  updateUser(req.params.id, req.body.name, req.body.email)
    .then(() => res.send('User updated'))
    .catch(err => res.status(500).send(err));
});
app.delete('/users/:id', (req, res) => {
  deleteUser(req.params.id)
    .then(() => res.send('User deleted'))
    .catch(err => res.status(500).send(err));
});
app.listen(3000, () => console.log('Server running on port 3000'));
```

Run your server with node server.js, and you can now interact with your Cassandra-backed app via the REST API.

Conclusion

Building an app with Apache Cassandra enables you to take advantage of a high-performance, scalable NoSQL database. Its distributed architecture is designed for low-latency and high availability, making it ideal for large-scale applications. By using the Cassandra data model and applying careful schema design, you can build efficient and responsive applications that can grow with your data.