Spring Boot Microservices

Beginner to Guru

The Problem With Transactions

Transactions

- A database transaction allows you to have sequence of steps
  - All steps must complete to be committed
  - Else, a rollback occurs returning the database to the original state
- The Order Allocation Scenario
  - Allocate Inventory - Updating Inventory and Order with Allocation
  - Works well within a monolith
  - Order and Inventory are two different Microservices / Databases
  - Breaks traditional transactions

Important Terms

- **Transaction** - A unit of work. One or more operations
  - Can be just one; can be hundreds or thousands.
- **Commit** - Indicates the end of the transaction and tells database to make changes permanent.
  - More efficient to do multiple operations in a transaction. There is a 'cost' with commits.
- **Rollback** - Revert all changes of the transaction
- **Save Point** - Programatic point you can set, which allows you to rollback to (ie rollback part of a transaction)

A.C.I.D. Transactions

- A.C.I.D. - Typically one database
  - **Atomicity** - All operations are completed successfully or database is returned to previous state.
  - **Consistency** - Operations do not violate system integrity constraints.
  - **Isolated** - Results are independent of concurrent transactions.
  - **Durable** - Results are made persistent in case of system failure (ie written to disk)
- Database handles all locking and coordination to guarantee transaction
  - This is EXPENSIVE to-do - takes a lot of system resources

Distributed Transactions

- With Microservices, often multiple services are involved in what is considered a transaction
  - Order Allocation Example - Order Service, Inventory Service
- Java EE - Java Transaction API (JTA)
  - Enables distributed transactions for Java environments
  - Well supported by Spring
  - Transactions are managed across nodes by a Transaction Manager
  - Very Java Centric

Two Phase Commit - 2PC

- Happens in two phases - Voting and Commit
- Coordinator asks each node if proposed transaction is okay
  - If all respond okay
    - Commit message is sent
    - Each Node commits work and sends acknowledgement to coordinator
  - If any node responds no
    - Rollback message is sent
    - Each node rollsback and sends acknowledgement to coordinator

Problems with Two Phase Commit

- Problems with 2PC
  - Does not scale - expensive
  - Blocking Protocol - the various steps block and wait for other to complete
  - Performance is limited to the speed of the slowest node
  - Coordinator is a Single Point of Failure
  - Technology lock-in
    - Can be very difficult to mix technology stacks

Challenges with Microservices

- A transaction for a Microservice architecture will often span multiple microservices
- Each service should have its own database
  - Could be a mix of SQL and NoSQL databases
- Should be technology agnostic
  - Services can by in Java, .NET, Ruby, etc
- How to coordinate the 'Transaction' across multiple microservices???

SPRING FRAMEWORK

GURU