JBDL_82_L31

Quick recap

The instructor began a new lesson on caching, specifically focusing on Redis as a database for caching purposes and explaining the concept of caching as a technique to store data in memory for faster retrieval. The discussion covered various aspects of caching implementation, including API latency improvements, cache hit/miss scenarios, and the complexities of data synchronization between cache and database. The instructor concluded by demonstrating Redis installation and usage, explaining Redis data types and commands, and preparing students for the upcoming integration of Redis with a Spring Boot project.

Next steps

- Instructor: Cover Redis integration with Spring Boot project in next class
- Instructor: Start Spring Security topic next week
- Instructor: Cover Spring profiling in next class
- Students: Read and understand Redis commands for string, hash, list, set, and sorted set data types before next class
- Instructor: Upload meeting transcript and list of Redis commands for students to review

Summary

Redis Caching Lesson Overview

The instructor began a new lesson on caching, specifically using Redis as the database for caching purposes. They explained the concept of caching as a technique to store data in memory for faster retrieval, contrasting it with disk storage which is cheaper but slower. The instructor outlined the plan to cover Redis data types, commands, and integration into a Spring Boot project, with the goal of completing caching by the end of the week to begin Spring Security lessons the following week.

API Caching for Latency Reduction

The discussion focused on improving API latency through caching, where a cache layer is introduced between the server and database. The instructor explained the concepts of cache hit (10 ms response time) and cache miss (1.02 seconds response time) scenarios, and demonstrated how Step 3 (storing data back to cache) can be performed asynchronously with Step 2 (database retrieval) to reduce latency to 1.01 seconds. The class discussed when to use caching, with Prashant explaining that even a 50% cache hit ratio provides significant performance benefits, though the instructor noted that caching is most beneficial when hit ratios exceed 70% and should be evaluated based on performance testing results and cost considerations.

Cache Invalidation Strategies Explained

The discussion focused on the complexities of implementing caching in a system, particularly the challenges with data synchronization between cache and database. The speaker explained that while caching can improve performance by reducing database calls, it introduces complications during data updates, deletions, and insertions. They described two approaches to handle cache invalidation: manually updating the cache after database changes, and using database event listeners to automatically synchronize cache data. The speaker emphasized that while the first approach is simpler, the second method is more commonly used in companies due to its ability to maintain isolation between database, application, and cache layers, though it can be challenging to implement.

Database Event Triggers and Caching

The instructor explained how databases can trigger events that are consumed by a cache layer, demonstrating this with a code example showing how data is written to DynamoDB and then synchronized with Elasticsearch using event listeners. They also covered how Kinesis can be used as an alternative to Kafka for streaming data, and briefly mentioned PubNub as a tool for real-time messaging.

Server Communication and Event Architecture

The instructor explained server-side communication and event-driven architecture, focusing on how servers can communicate with clients using sockets and events. They discussed the use of Pub/Sub (or Message Bus) patterns for communication between systems, and mentioned that while Kafka will be covered later in the course, other queuing mechanisms like Kinesis or RabbitMQ can be used for similar purposes. The instructor also covered caching solutions, explaining the benefits and implementation of Redis as a distributed cache, and mentioned that while Redis is widely used by companies like Uber and Zomato, other caching solutions like Aerospike are also available with similar concepts but different commands and integrations.

Distributed Caching Approaches Comparison

The instructor discussed two approaches to caching in a distributed system: local caching on each application server and a global caching server. They explained that local caching can lead to data inconsistency and cache misses when using stateless load balancing algorithms, while a global cache avoids these issues by storing data in a single location. Prashant suggested that local caching could have lower latency, but the instructor clarified that the main advantage is the elimination of network calls between processes running on the same server, which significantly reduces latency.

Microservices Caching and Database Strategies

The discussion focused on how microservices handle shared caching and databases. Class explained that while Redis supports multiple databases like MySQL, creating multiple databases in Redis significantly increases costs, so most startups use a single Redis instance to store keys for various microservices. For larger companies like Microsoft or Zomato that require data segregation, they might use separate Redis instances for different microservices, but this is less common. Class also clarified that Redis is database-agnostic and can be used in conjunction with various underlying databases like MySQL, MongoDB, or DynamoDB, with multiple instances running in a cluster for replication and failover.

Load Balancers vs API Gateways

The discussion focused on explaining the differences between load balancers and API gateways, with Class explaining that while both perform routing functions, API gateways handle authentication, authorization, and service orchestration, while load balancers focus solely on distributing network traffic. Class also clarified that internal load balancers (ILB) are used for service-to-service communication within the same VPC, while external load balancers (GLB) handle public internet traffic. The conversation concluded with a brief mention of using Redis for caching, though this was not fully explored.

Redis Installation and Configuration Basics

The instructor demonstrated how to download and run Redis on a Mac, explaining that the default port is 6379 but can be changed by specifying a configuration file. They showed that when running Redis without a config file, it uses the default settings, but when using a specific config file, the port can be changed to 6380. The instructor also explained that the PID (Process ID) changes each time Redis is run, and mentioned that Redis logs include information about loading an RDB file created by version 7.2.7.

Understanding Redis Data Storage

The instructor explained that Redis is an in-memory data store that also provides the capability to replicate data on disk, allowing it to persist data even after the server is closed. They clarified that while Redis can be used as a caching solution, it should not be considered a replacement for traditional databases due to costs and performance issues when handling large amounts of data. The instructor also demonstrated how to connect to a Redis server using the Redis CLI and showed that Redis currently stores 13 keys in memory.

Redis Data Storage Basics Explained

The instructor explained the basics of Redis, focusing on how data is stored and managed. They discussed the 13 keys created earlier and how the data is saved to disk as an RDB file, which is not human-readable but can be somewhat interpreted. The instructor also covered the concept of data compression to speed up loading times and explained the process of saving and loading data from memory to disk. They concluded by mentioning the importance of understanding Redis commands for further learning.

Redis Data Types Overview

The instructor discussed Redis data types and commands, explaining that keys are always strings while values can be strings, hashes, lists, sets, or specialized types like geospatial and time series data. They emphasized that students should review the commands for these data types before the next class to save time, as they will begin integrating Redis with a Spring Boot project and cover string profiling in the 1.5-2 hour session tomorrow. The instructor also clarified that spaces in string values need to be encoded with a colon in Redis commands.