System Design Day 7: Distributed Caching Basics

.

🔷 What Is Caching?

Caching is the process of storing frequently accessed data in temporary memory to reduce the time and resources required to access it from the original data source (like a database or an API).

🗃️ Centralized Caching (Traditional Approach)

📌 Overview:

- Uses a **single cache server** to store frequently accessed data.
- Application checks the central cache before querying the main database.

✅ Pros:

- Simple setup and easy to manage.
- Reduces latency for repeat data fetches.

❌ Cons:

- **Single point of failure.**
- **Scalability bottleneck** – can't handle high concurrent load.
- Not resilient to node or network failures.

🌐 Distributed Caching (Modern Approach)

📌 What It Is:

A **distributed cache** stores data across multiple nodes in a network using a **Distributed Hash Table (DHT)**. Nodes are horizontally scalable and automatically rebalance data and traffic.

✅ Advantages:

- **High availability**
- **Fault tolerance**
- **Horizontal scalability**
- **Low latency and fast response**

⚙️ Key Features

✅ Consistency

- Ensures data accuracy across distributed nodes.
- Some tools (e.g., **Redis Cluster**) provide **eventual consistency**, while others like **Apache Ignite** offer **strong consistency** via transactional and ACID-compliant caches.

🔄 Eviction Policies

- Automatically remove stale or least-used items to free memory.
- Common strategies:
  - **LRU (Least Recently Used)**
  - **LFU (Least Frequently Used)**
  - **TTL (Time-To-Live)**

Most distributed caches support configuring eviction based on time, size, or usage.

🧱 Multi-Node Setup

- Nodes collaborate and replicate data.
- Load is distributed evenly.
- In case of node failure, others take over (high availability).
- New nodes auto-join and data is rebalanced dynamically.

🛠️ Popular Distributed Caching Tools

🔷 Redis

- In-memory key-value store.
- Supports data structures (Lists, Sets, Hashes).
- Offers pub/sub and basic persistence.
- Great for Spring Boot apps using `spring-boot-starter-data-redis`.

🔷 Memcached

- Lightweight key-value store.
- High performance, stateless, distributed.
- No advanced structures or persistence.

🔷 Hazelcast

- In-memory data grid.
- Java-native.
- Supports distributed maps, queues, and executors.
- Easily integrated with Spring.

🔷 Apache Ignite ✅

> Yes, Apache Ignite is a powerful distributed caching solution — and much more.

🔧 Features:

- **In-memory distributed cache** (with optional persistence).
- **Strong consistency** with support for **ACID transactions**.
- **SQL support** (run SQL queries over distributed cache).
- **Compute grid** & **distributed database** capabilities.
- **Automatic rebalancing** and **partitioning** of data across nodes.
- **Supports near-cache** for local fast access in co-located applications.

🔌 Spring Integration:

- Fully compatible with **Spring Cache abstraction** (`@Cacheable`, `@CacheEvict`).
- Can be embedded or used in a client/server mode.

- Example:

```
@Cacheable(value = "products", key = "#productId")
public Product getProductById(String productId) {
    return productRepository.find(productId);
}
```

```
spring:
  cache:
    type: ignite
ignite:
  config: classpath:ignite-config.xml
```

💡 Use Cases for Distributed Caching

- **Session management** (user tokens, preferences)
- **Reduce database load** with frequently queried data
- **Accelerate microservice communication** (avoid service-to-service calls)
- **Real-time analytics / leaderboards / notifications**
- **Storing reference data** (product info, currency rates, etc.)

🧠 Design Considerations

Concern	Design Consideration
Consistency	Choose between **eventual**, **strong**, or **transactional** consistency.
Eviction	Configure TTLs, LRU/LFU, or size limits to avoid memory overflow.
Fault Tolerance	Ensure cluster replication and automatic failover.
Persistence	Use in-memory only (for speed) or with backup (for durability).
Sharding	Tools like Redis Cluster, Ignite, and Hazelcast handle it natively.

👨‍💻 Java & Spring Boot Integration

Use **Spring Cache abstraction** to decouple business logic from caching implementation. Backends can be swapped (Redis, Hazelcast, Ignite, etc.)

without changing code.

Example:

```
@Cacheable("users")
public User getUser(String userId) {
   return userRepo.findById(userId);
}
```

Backends:

- `spring-boot-starter-data-redis`
- `spring-boot-starter-cache` + Apache Ignite or Hazelcast

📌 Final Takeaways

- **Centralized caching** is simple but doesn't scale.
- **Distributed caching** is the industry standard for **high availability**, **horizontal scalability**, and **resilience**.
- **Apache Ignite** is a feature-rich caching + compute platform with **strong consistency** and **ACID transactions**.
- Use **Java Spring Cache abstraction** to plug into any caching backend.
- Choose tools and configs based on your **use case**, **consistency requirements**, and **performance needs**.