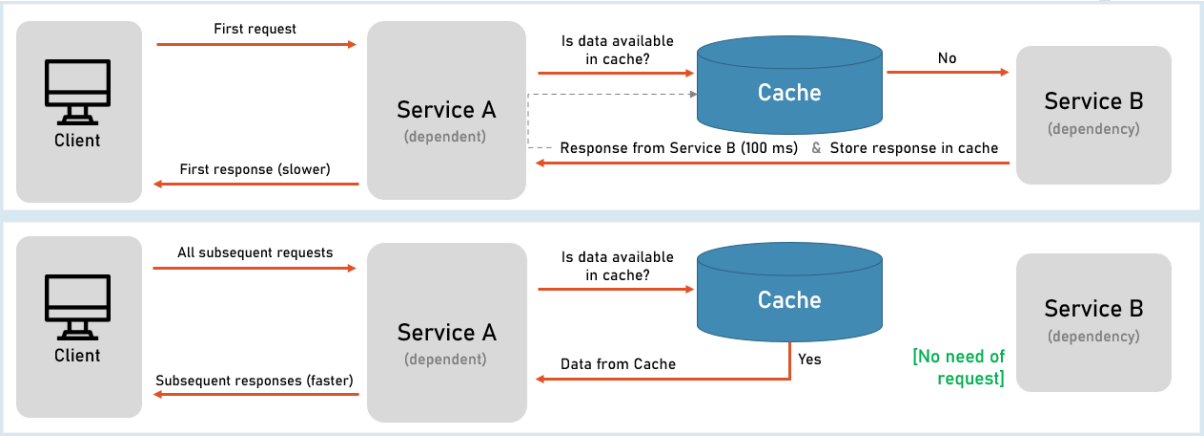
.NET Core Microservices – True Ultimate Guide

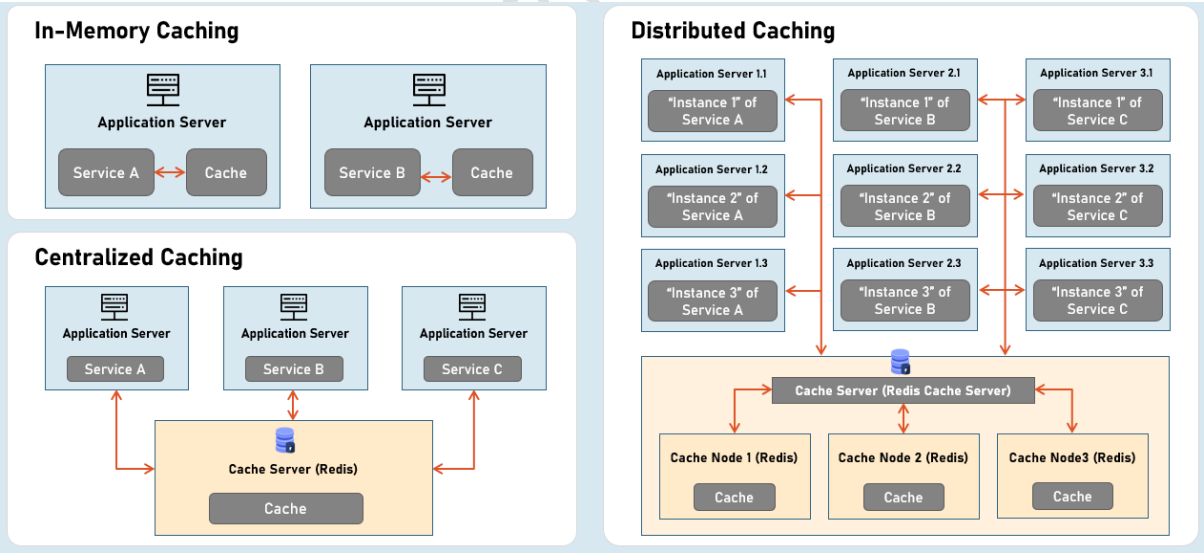
Section 9 – Caching – Cheat Sheet

Introduction to Caching

Caching is a technique of storing frequently accessed data in a temporary location for faster retrieval.



Types of Caching



In-Memory Caching

Storage: In-memory caching stores data in the RAM of the application server or instance.

When-to-use: Suitable for data that is read frequently but doesn't need to persist beyond the lifetime of the application instance.

Advantages:

- Speed: Data is accessed with minimal latency since it resides in memory.
- Simplicity: Easy to implement and manage within a single application instance.

Problems:

- Limited Size: Size constrained by available memory, which may not be sufficient for caching large datasets.
- Persistence: Cache data can be lost if the application restarts.

Centralized Caching

Storage: Centralized caching involves using a single shared cache server that can be accessed by multiple application instances of different microservices.

When-to-use: Useful when multiple instances of different microservices need to share cached data to avoid redundant network calls to database servers and redundant computations.

Advantages: Allows multiple instances of an application to access the same cached data, reducing duplication of effort.

Problems:

- Single Point of Failure: If the centralized cache fails, it can impact all instances relying on it.
- Performance Bottlenecks: A heavily loaded cache server may struggle to keep up with the demand, resulting in increased latency for cache accesses.

Distributed Caching

Storage: Distributed caching extends centralized caching by distributing cached data across multiple cache nodes hosted on different servers.

When-to-use: High Scalability: Suitable for applications with high traffic and scalability requirements.

Geo-distributed Systems: Ideal for applications deployed across multiple regions or data centers.

Advantages:

- Scalability: Scales horizontally by adding more cache nodes.
- Performance: Provides low latency access to data distributed across nodes.

Problems:

- Complexity: Setting up and managing a distributed cache requires more infrastructure and operational expertise.