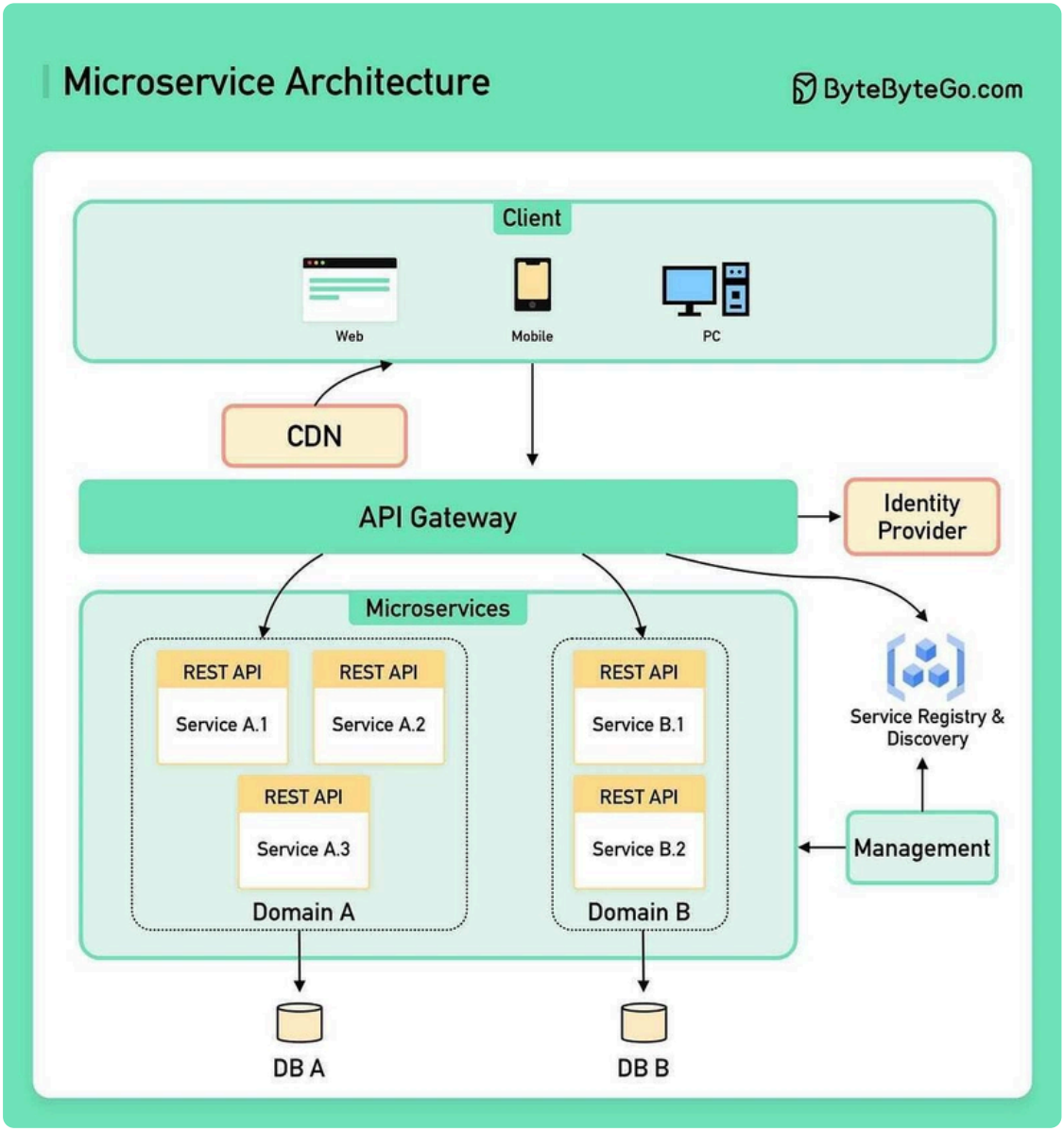
Soma
Posted on Jul 14, 2024 • Edited on Aug 1, 2024

19 Microservices Patterns for System Design Interviews

#microservices #softwaredevelopment #systemdesign #programming

Disclosure: This post includes affiliate links; I may receive compensation if you purchase products or services from the different links provided in this article.



image_credit - [ByteByteGo]

Hello friends, if you are preparing for system design interviews then it make sense to to prepare for Microservices design patterns as well, not just to do well on interviews or make your architecture more robust but also to understand existing projects.

Microservices patterns like Cicuit Breaker, API Gateway, Saga, Event Sourcing are tried and tested solution of common Microservices Problems.

These patterns address common challenges in microservices architectures like scalability, fault tolerance, and data consistency.

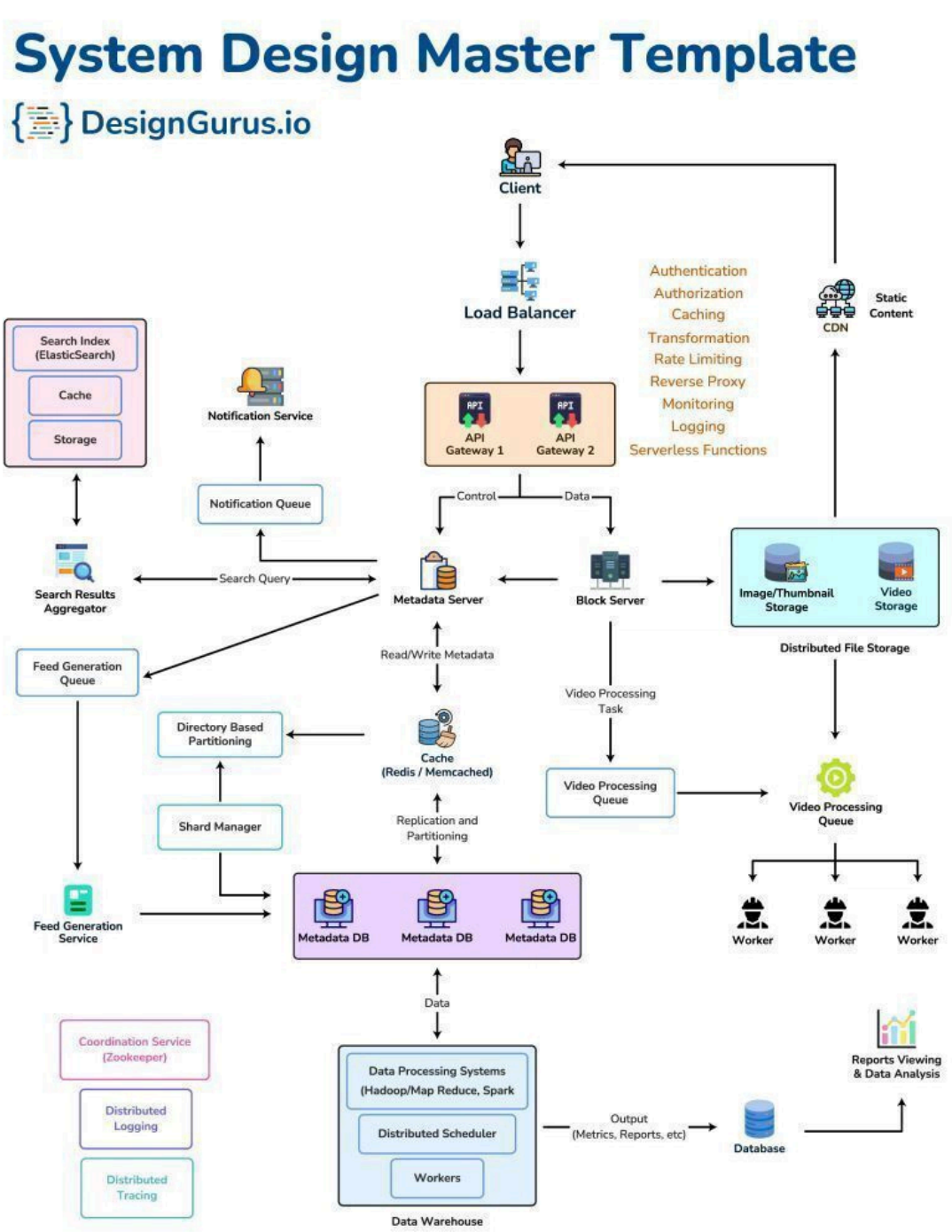
In the past, I have talked about common system design questions like API Gateway vs Load Balancer and Horizontal vs Vertical Scaling, Forward proxy vs reverse proxy as well common System Design problems and in this article I am going to share 24 key Microservices design patterns that are essential knowledge for technical interviews.

They are also one of the essential System design topics for interview and you must prepare it well.

Many companies use microservices, so understanding these patterns shows you're up-to-date with current trends. Knowing when and how to apply these patterns also demonstrates your ability to solve complex distributed system problems.

These patterns often involve trade-offs, allowing you to showcase your analytical thinking and Interviewers often present scenarios where these patterns are relevant solutions.

By the way, if you are preparing for System design interviews and want to learn System Design in depth then you can also checkout sites like **ByteByteGo**, **Design Guru**, **Exponent**, **Educative**, **Codemia.io**, and **Udemy** which have many great System design courses and a System design interview template like this which you can use to answer any System Design question.



If you need more choices, you can also see this list of best System Design courses, books, and websites

P.S. Keep reading until the end. I have a free bonus for you.

So, what are we waiting for, let's jump right into it

19 Microservices Design Patterns for System Design Interviews

Microservices architecture is a design approach that structures an application as a collection of loosely coupled services.

To build scalable, maintainable, and resilient microservices-based systems, various patterns have emerged.

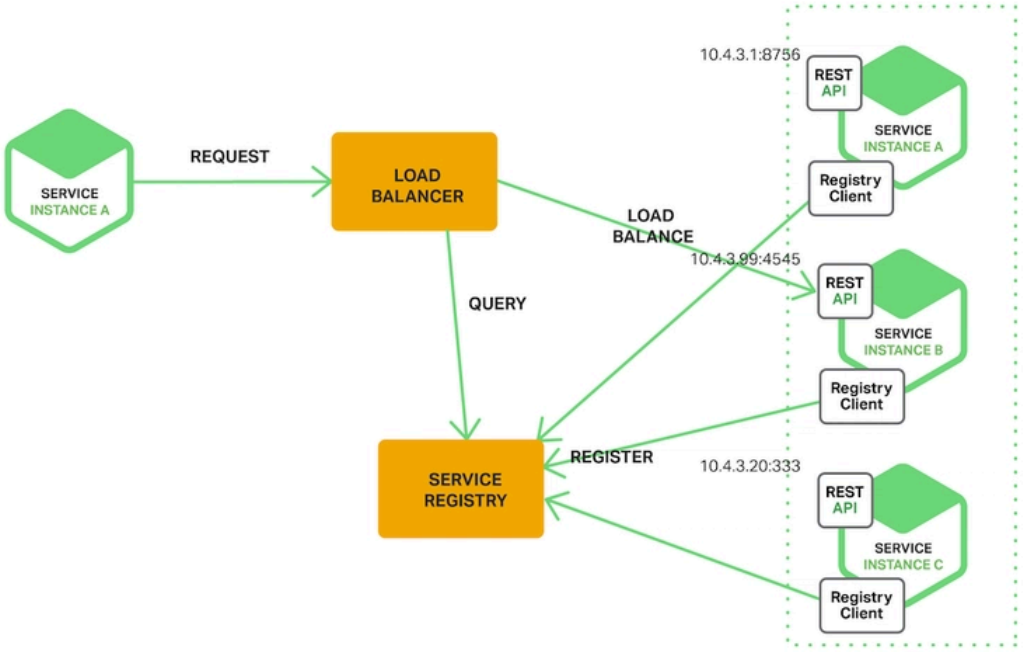
Here are essential microservices patterns you can use in your project and also remember for system design interviews.

1. Service Registry

Since there are many microservices in Microservice architecture they need to discover and communicate with each other.

A Service Registry, such as Netflix Eureka or Consul, acts as a centralized directory where services can register themselves and discover others.

Here is how it looks like:

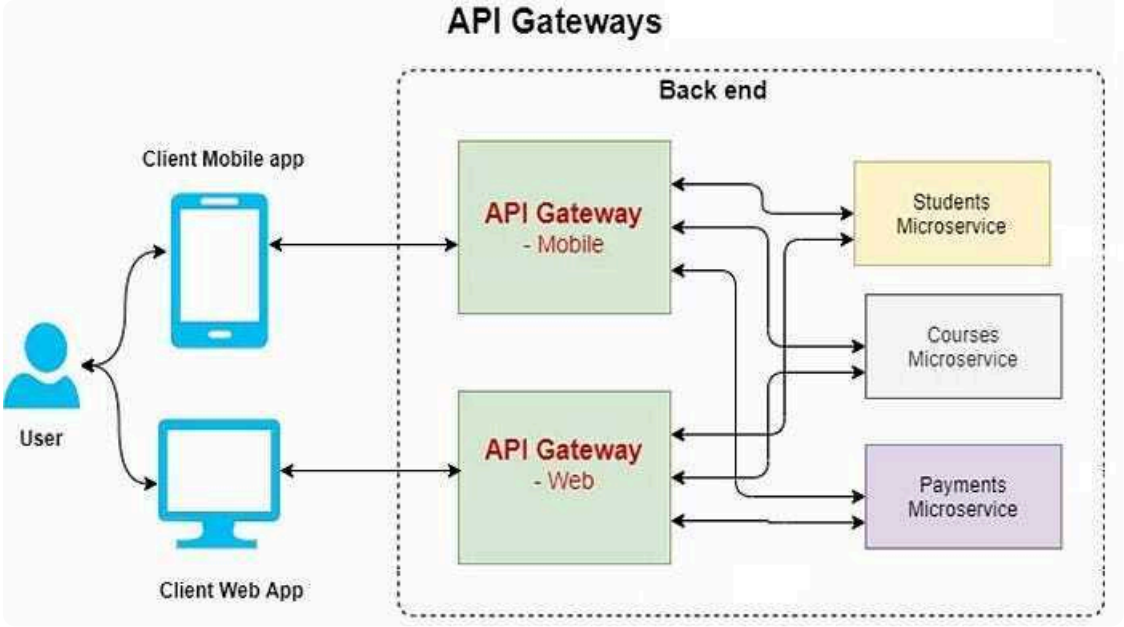


2. API Gateway

An API Gateway serves as a single entry point for client applications, aggregating multiple microservices into a unified API.

It handles requests, routing them to the appropriate services, and may perform tasks like authentication, authorization, and load balancing.

Here is how API Gateway looks like:

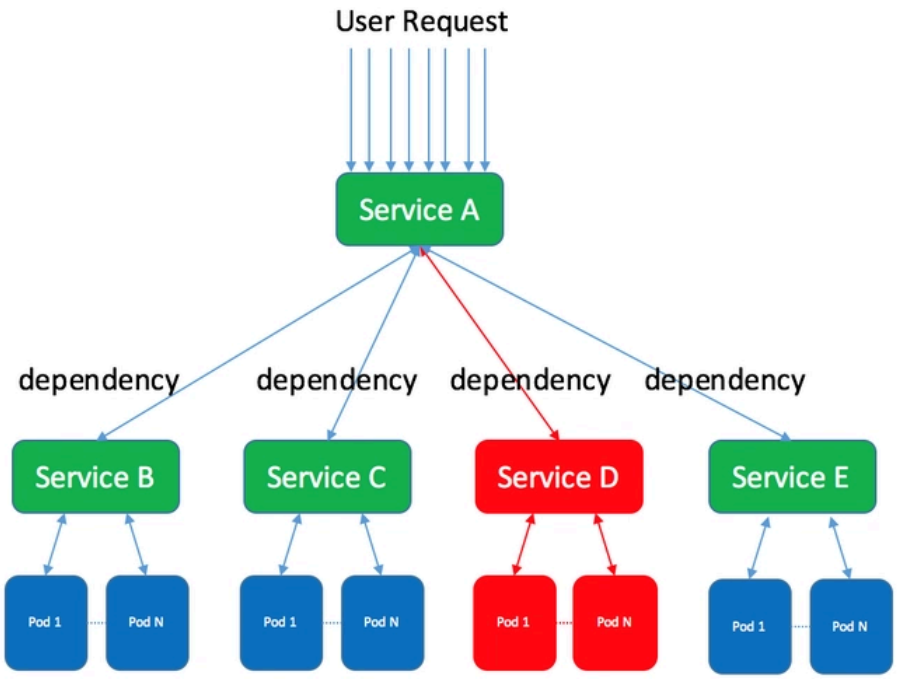


3. Circuit Breaker

Inspired by electrical circuit breakers, this pattern prevents a microservices failure from cascading to other services. Circuit breaker pattern monitors for failures, and if a threshold is crossed, it opens the circuit, preventing further requests.

This helps in graceful degradation and fault tolerance and its absolutely must in a Microservice architecture to prevent total shutdown of your services.

Here is an example of Netflix Hysrix as Circuit breaker:

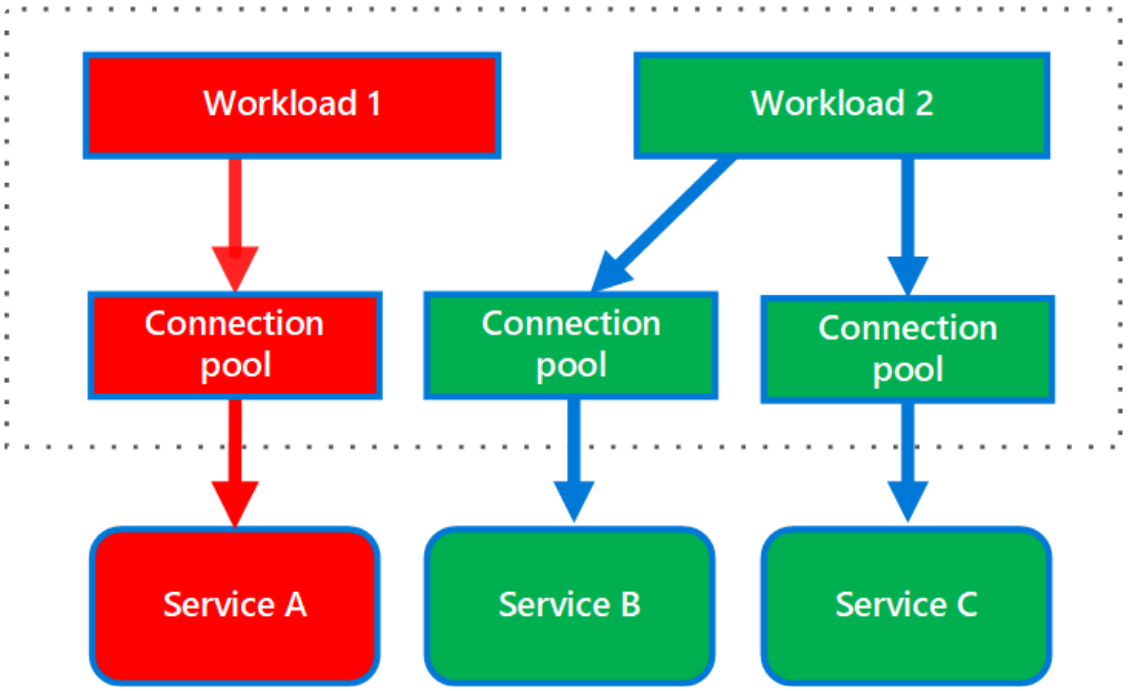


4. Bulkhead

In a microservices system, isolating failures is crucial. The Bulkhead pattern involves separating components or services to contain failures.

For example, thread pools or separate databases for different services can be used to prevent a failure in one part of the system from affecting others.

Here is a diagram showing Bulkhead pattern in Microservices architecture:

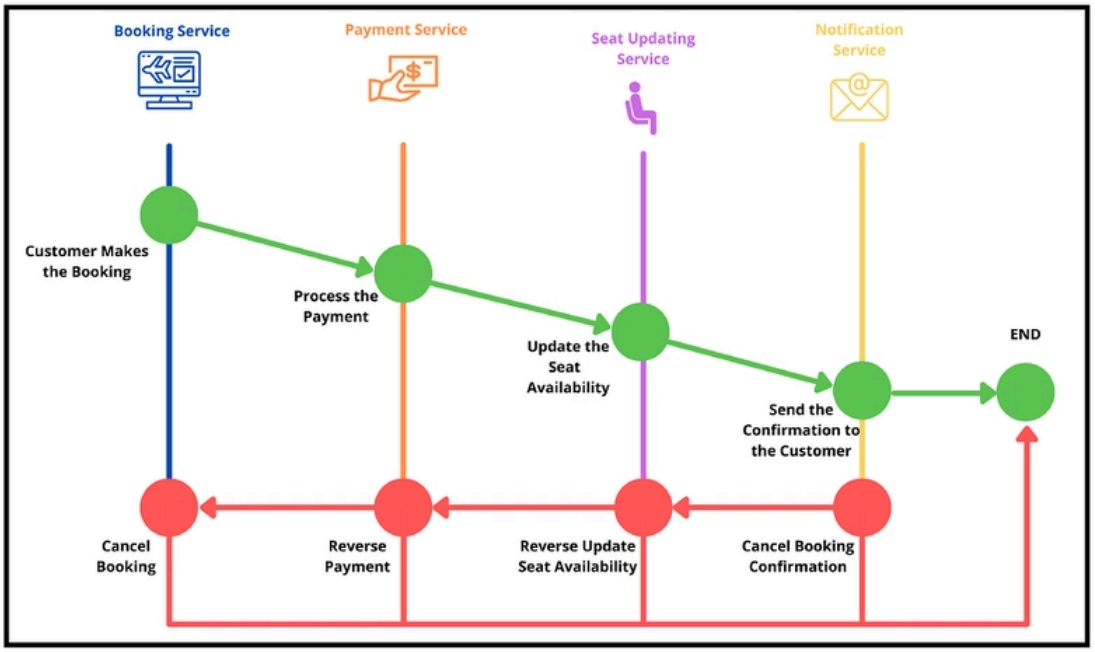


5. Saga Pattern

This pattern is used for managing distributed transactions. The Saga pattern breaks down a long-running business transaction into a series of smaller, independent transactions.

Each microservice involved in the saga handles its own transaction and publishes events to trigger subsequent actions.

Here is how Saga Pattern looks in action:



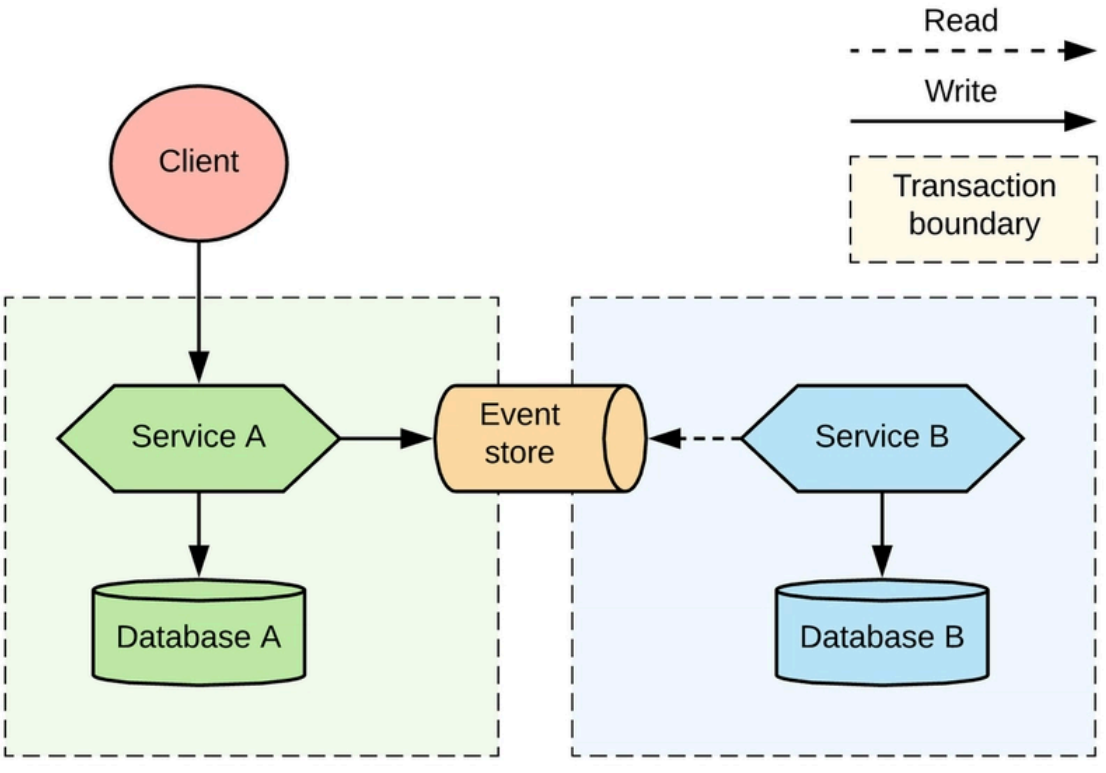
6. Event Sourcing

This is another popular pattern which is used heavily in high frequently low latency applications.

In this pattern, instead of storing only the current state, Event Sourcing involves storing a sequence of events that led to the current state.

This pattern provides a reliable audit trail and allows for rebuilding the system state at any point in time.

Here is how Event Sourcing looks in action:

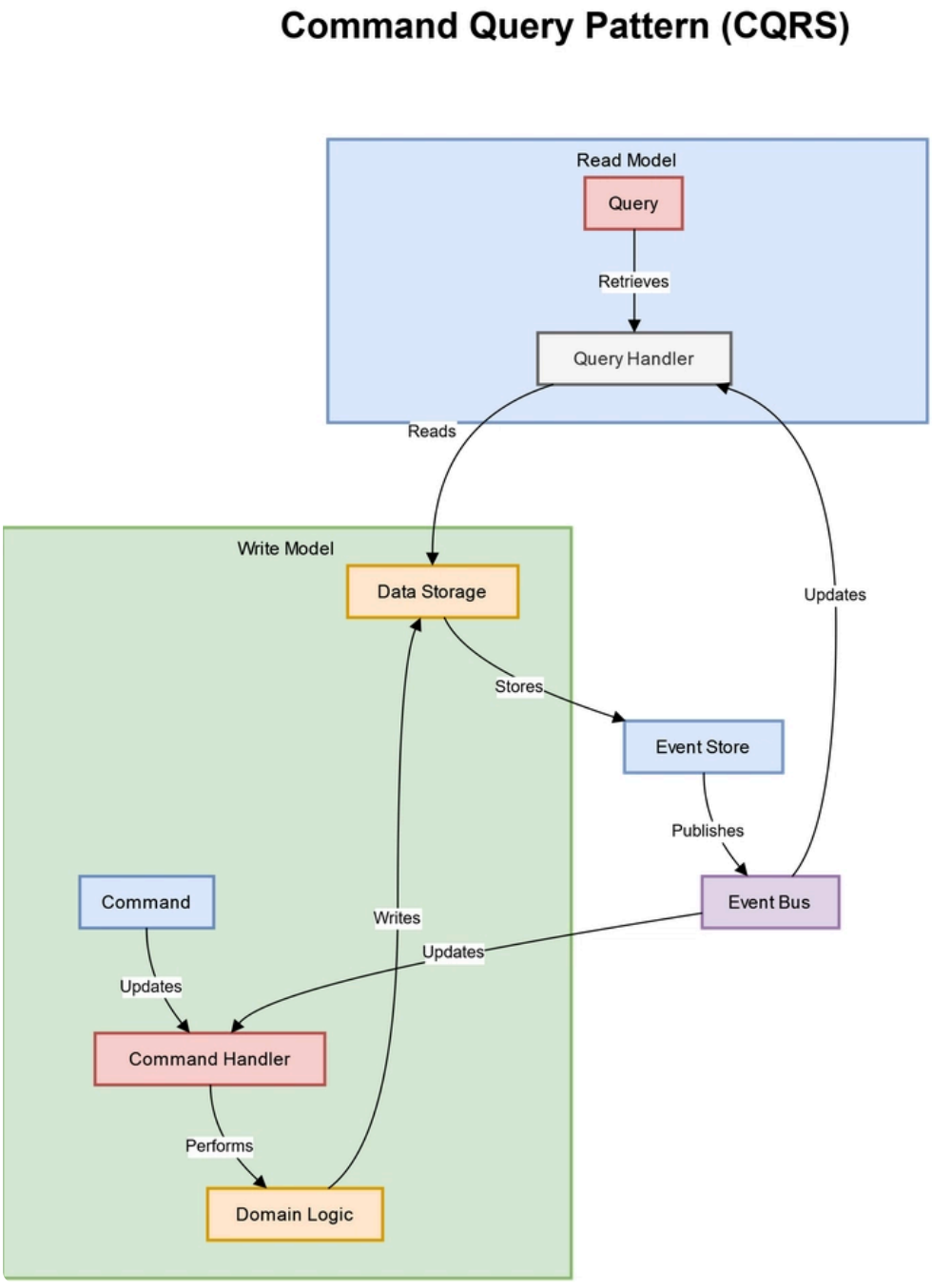


7. Command Query Responsibility Segregation (CQRS)

CQRS Pattern separates the read and write sides of an application. It uses different models for updating information (commands) and reading information (queries).

This pattern can improve scalability, as read and write operations have different optimization requirements.

Here is a nice diagram which shows CQRS pattern:

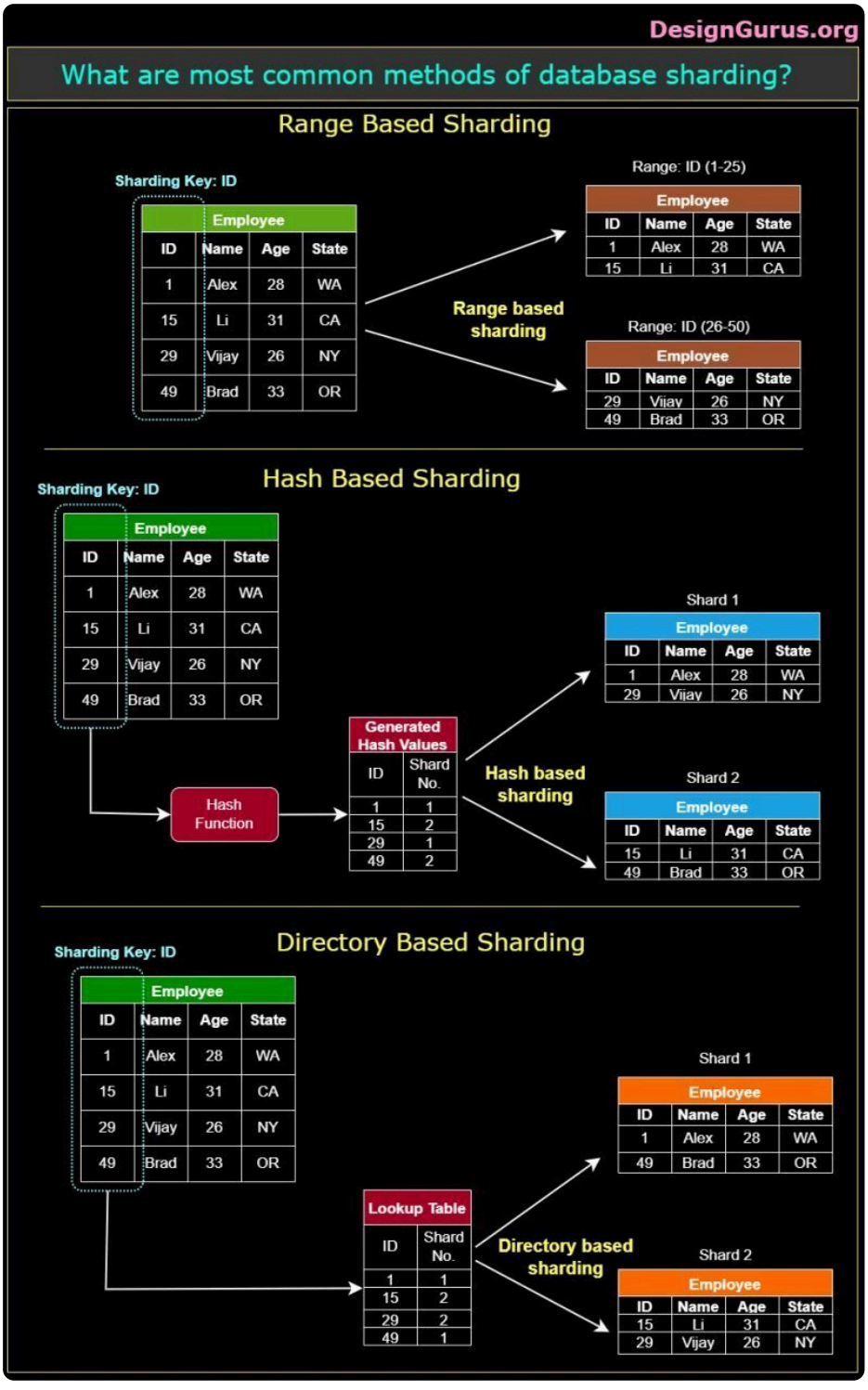


8. Data Sharding

Database sharing pattern is used to distribute the database load and avoid bottlenecks, Data Sharding involves partitioning data across multiple databases or database instances.

In this pattern, each microservice may handle a subset of data or specific types of requests.

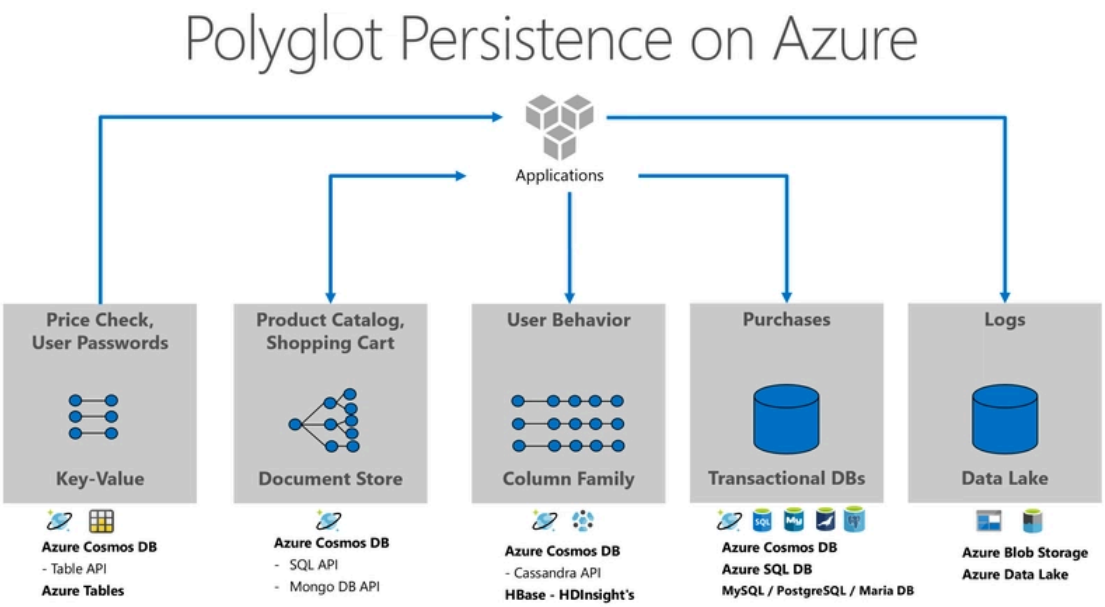
Here is how database sharding looks like, credit - Design Guru



9. Polyglot Persistence

Different microservices may have different data storage needs. Polyglot Persistence allows using multiple database technologies based on the requirements of each microservice, optimizing for data storage, retrieval, and query capabilities.

Here is a nice diagram which shows Polyglot persistence in Azure :

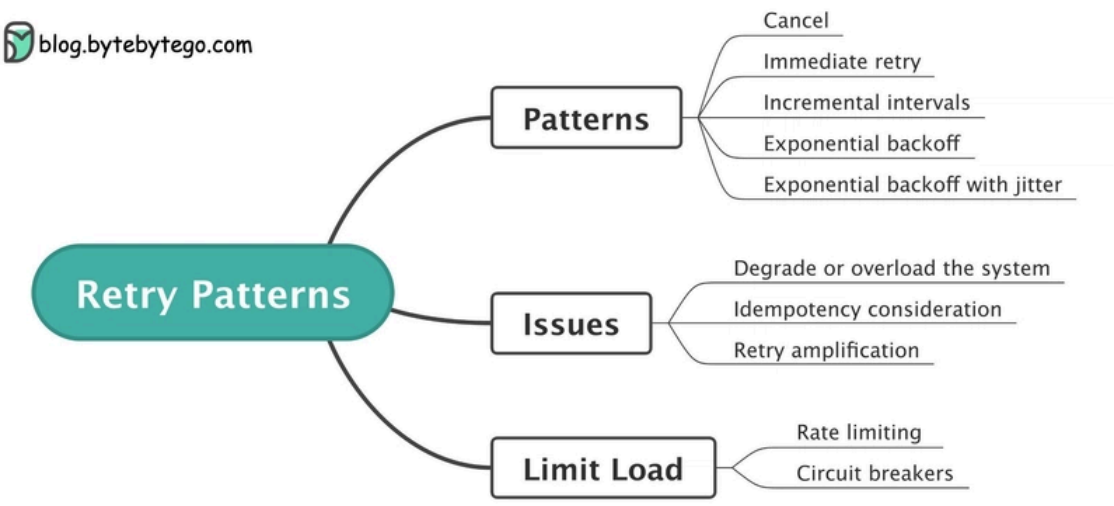


10. Retry

In Microservice architecture, when a transient failure occurs, the Retry pattern involves retrying the operation instead of immediately failing.

It can be applied at various levels, such as service-to-service communication or database interactions.

Here is a nice diagram form ByteByteGo, a great place for system design learning which shows Retry pattern in Microservices:

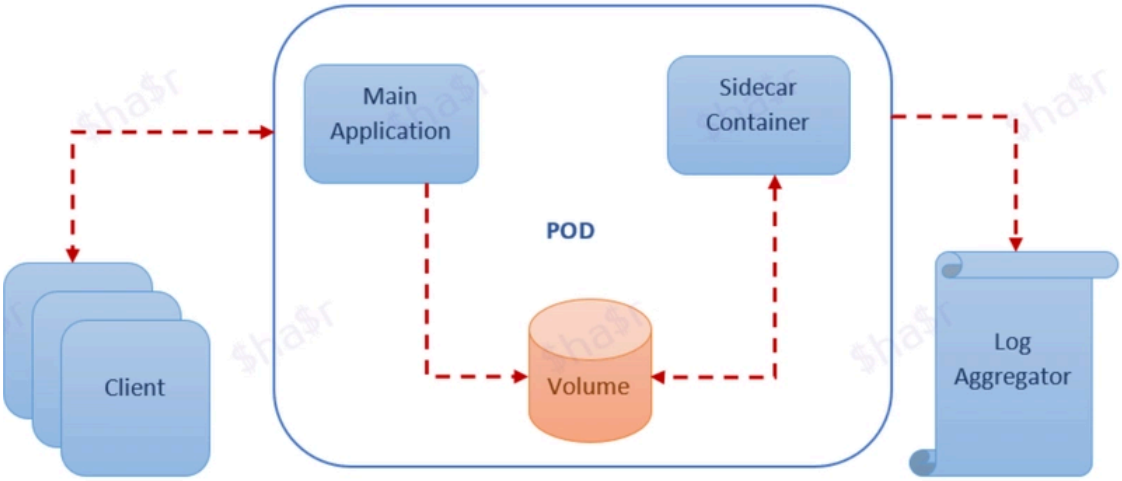


12. Sidecar

The Sidecar pattern involves attaching a helper service (the sidecar) to the main microservice to provide additional functionalities such as logging, security, or communication with external services.

This allows the main service to focus on its core functionality.

Here is how a Sidecar pattern looks like:

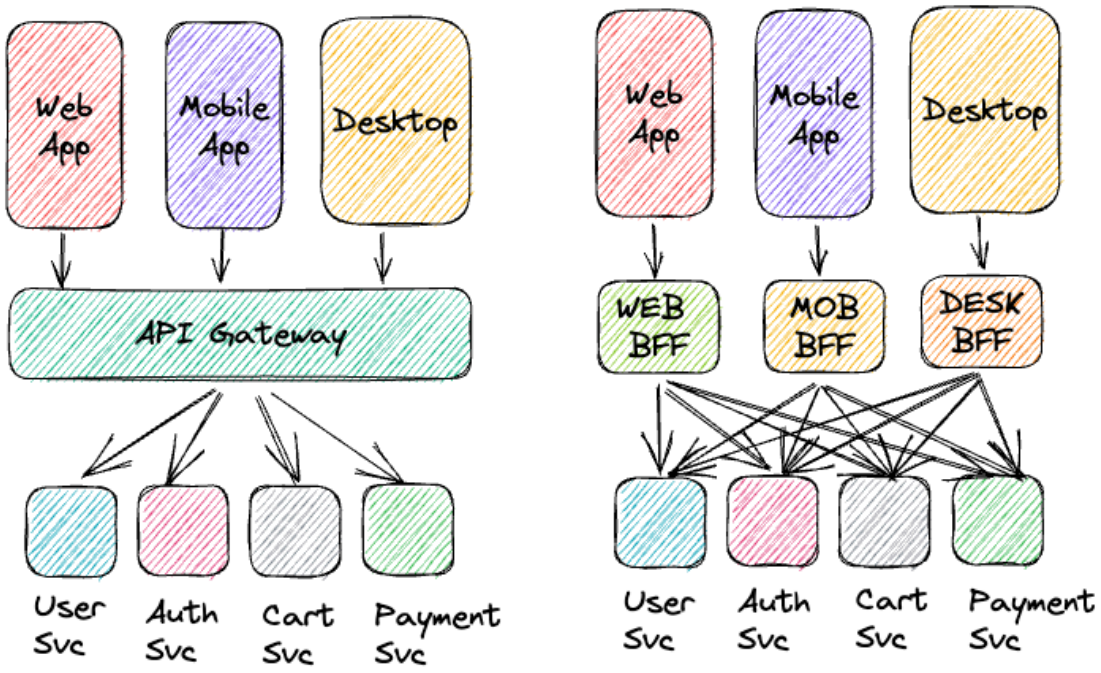


13. Backends for Frontends (BFF)

Also known as BFF this pattern is useful when dealing with multiple client types (e.g., web, mobile), the BFF pattern involves creating separate backend services tailored for each type of client.

This allows for optimized and specialized APIs for each client.

Here is how a Backends for Frontends (BFF) pattern looks like:

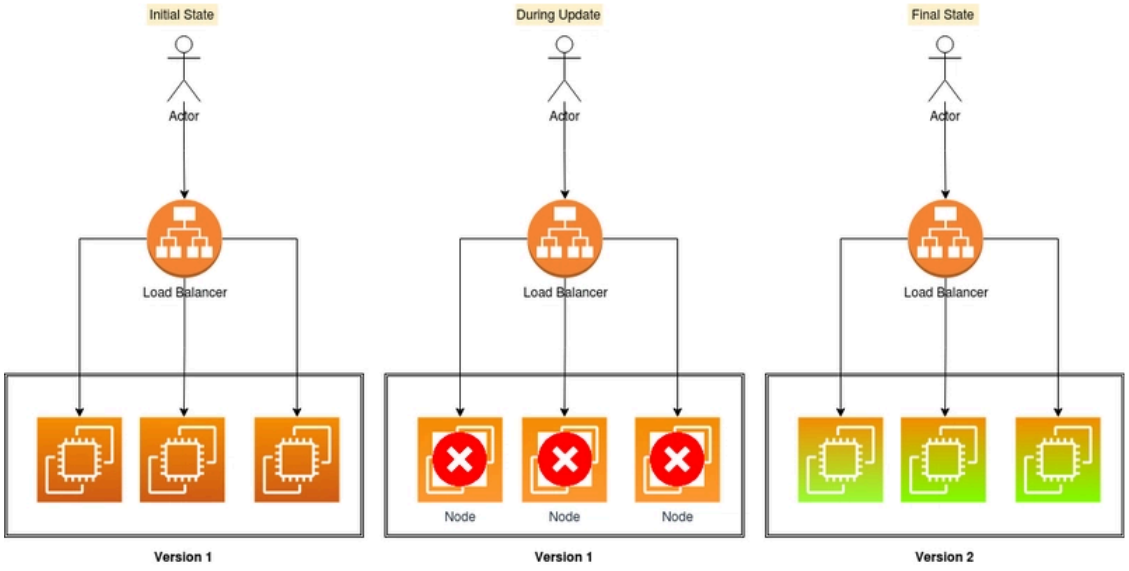


14. Shadow Deployment

The Shadow Deployment pattern involves routing a copy (shadow) of production traffic to a new microservice version without affecting the actual user experience.

This is one of the popular deployment strategy and it helps validate the new version's performance and correctness.

Here is how shadow deployment looks like

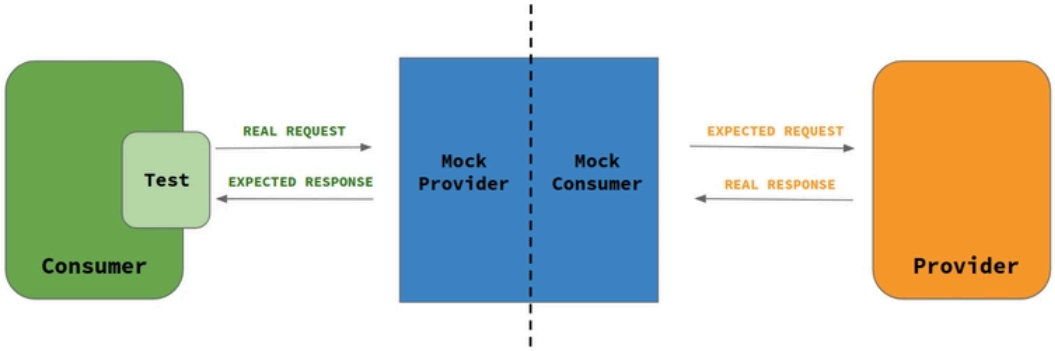


15. Consumer-Driven Contracts

In a microservices ecosystem, multiple services often interact with one another. The Consumer-Driven Contracts pattern involves

consumers specifying their expectations from producers, allowing for more robust and coordinated changes.

Here is a nice diagram which explains Consumer Driven contracts



16. Smart Endpoints, Dumb Pipes

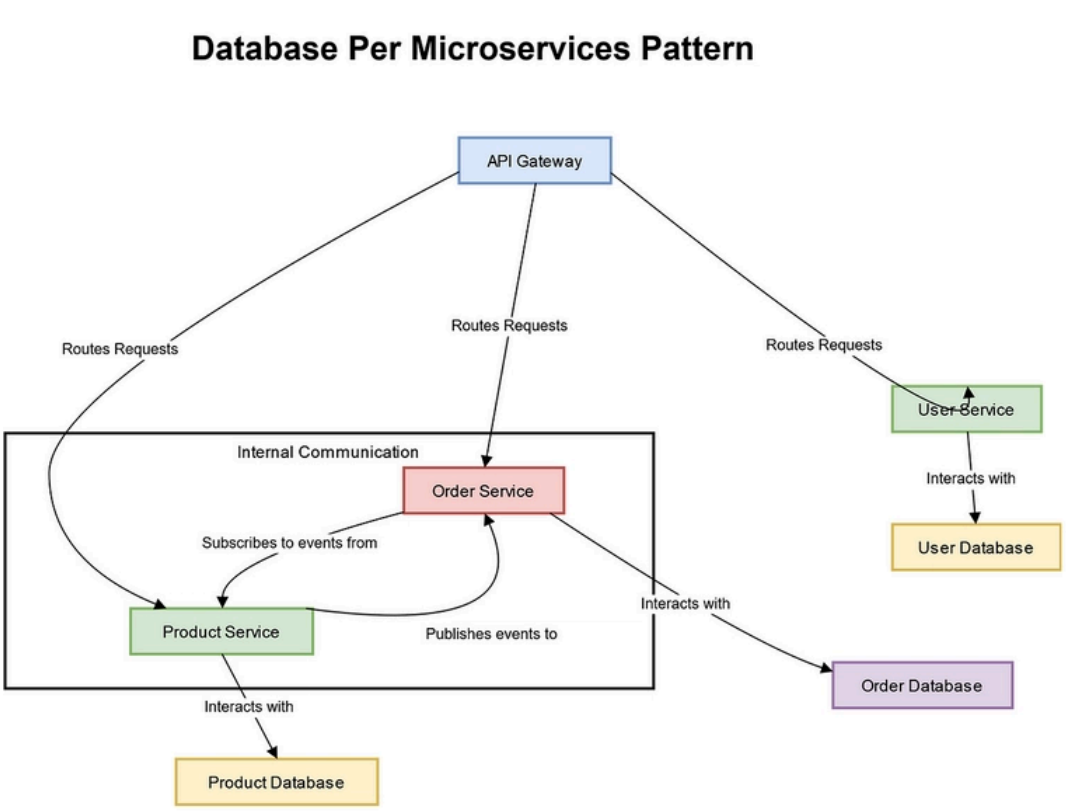
This pattern advocates for placing business logic in microservices (smart endpoints) rather than relying on complex middleware. The communication infrastructure (pipes) should be simple and handle only message routing.

17. Database per Service

This is another popular Microservices pattern where each microservice has its own database, and services communicate through well-defined APIs.

Database per Service pattern provides isolation but also requires careful consideration of data consistency and integrity.

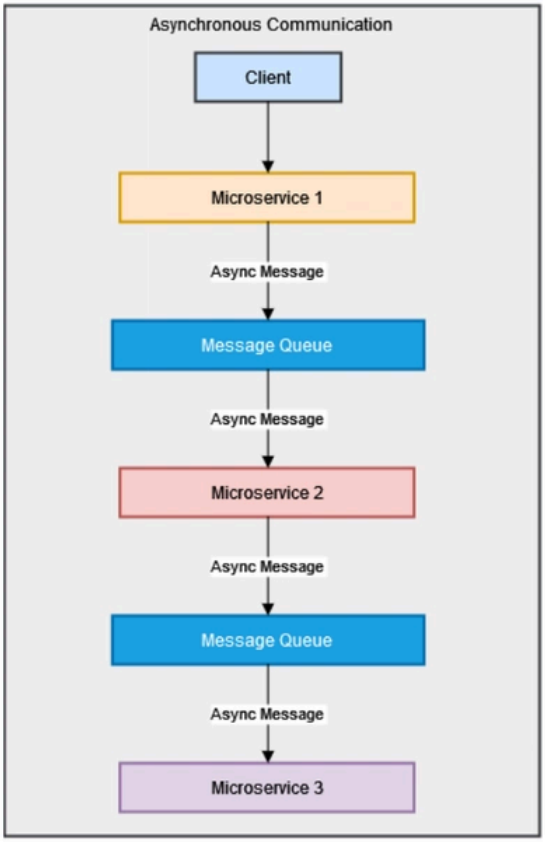
Here is how this pattern looks like:

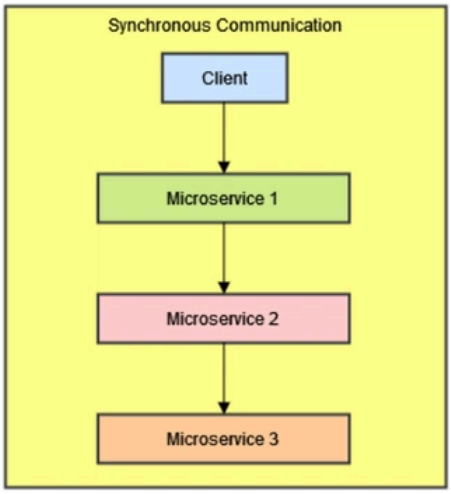


18. Async Messaging

Instead of synchronous communication between microservices, the Async Messaging pattern involves using message queues to facilitate asynchronous communication. This can improve system responsiveness and scalability.

Here is a nice diagram which shows difference between sync and async messaging

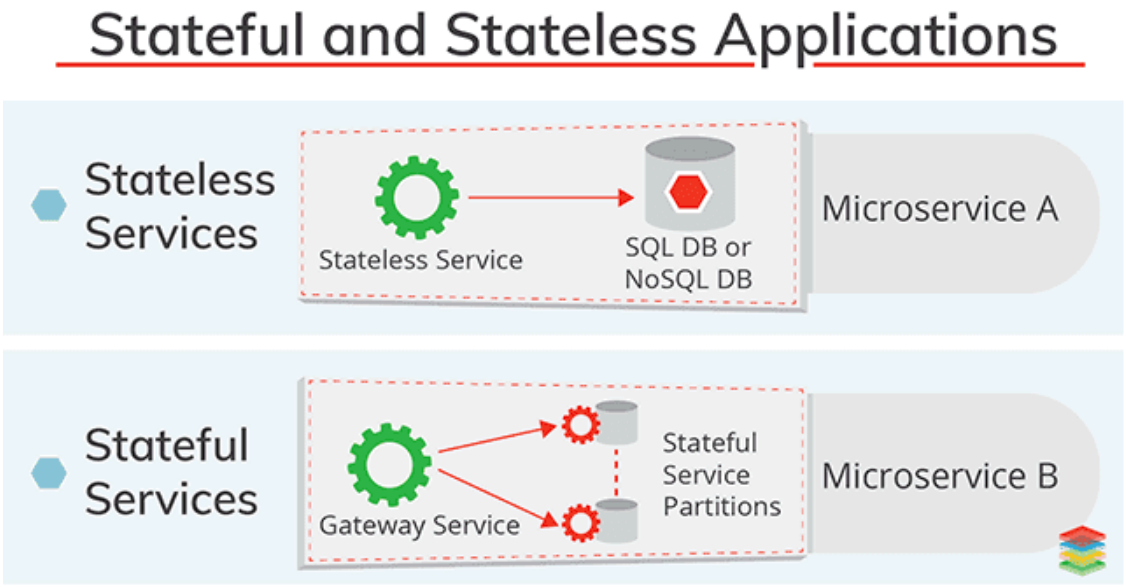




19. Stateless Services

Designing microservices to be stateless simplifies scalability and resilience. Each service processes a request independently, without relying on stored state, making it easier to scale horizontally.

Here is a nice diagram which shows the difference Stateless Services and Stateful Services



System Design Interviews Resources

And, here is my curated list of best system design books, online courses, and practice websites which you can check to better prepare for System design interviews. Most of these courses also answer questions I have shared here.

1. **DesignGuru's Grokking System Design Course**: An interactive learning platform with hands-on exercises and real-world scenarios to strengthen your system design skills.

2. **Codemia.io** : This is one of the best place to practice System design problems for interviews online. It got more than 120+ System design problems, many of them are free and also a proper structure to solve them.

3. **Exponent**: A specialized site for interview prep especially for FAANG companies like Amazon and Google, They also have a great system design course and many other material which can help you crack FAANG interviews.

4. **"Designing Data-Intensive Applications"** by Martin Kleppmann: A comprehensive guide that covers the principles and practices for designing scalable and reliable systems.

5. LeetCode System Design Tag: LeetCode is a popular platform for technical interview preparation. The System Design tag on LeetCode includes a variety of questions to practice.

6. **"System Design Primer"** on GitHub: A curated list of resources, including articles, books, and videos, to help you prepare for system design interviews.

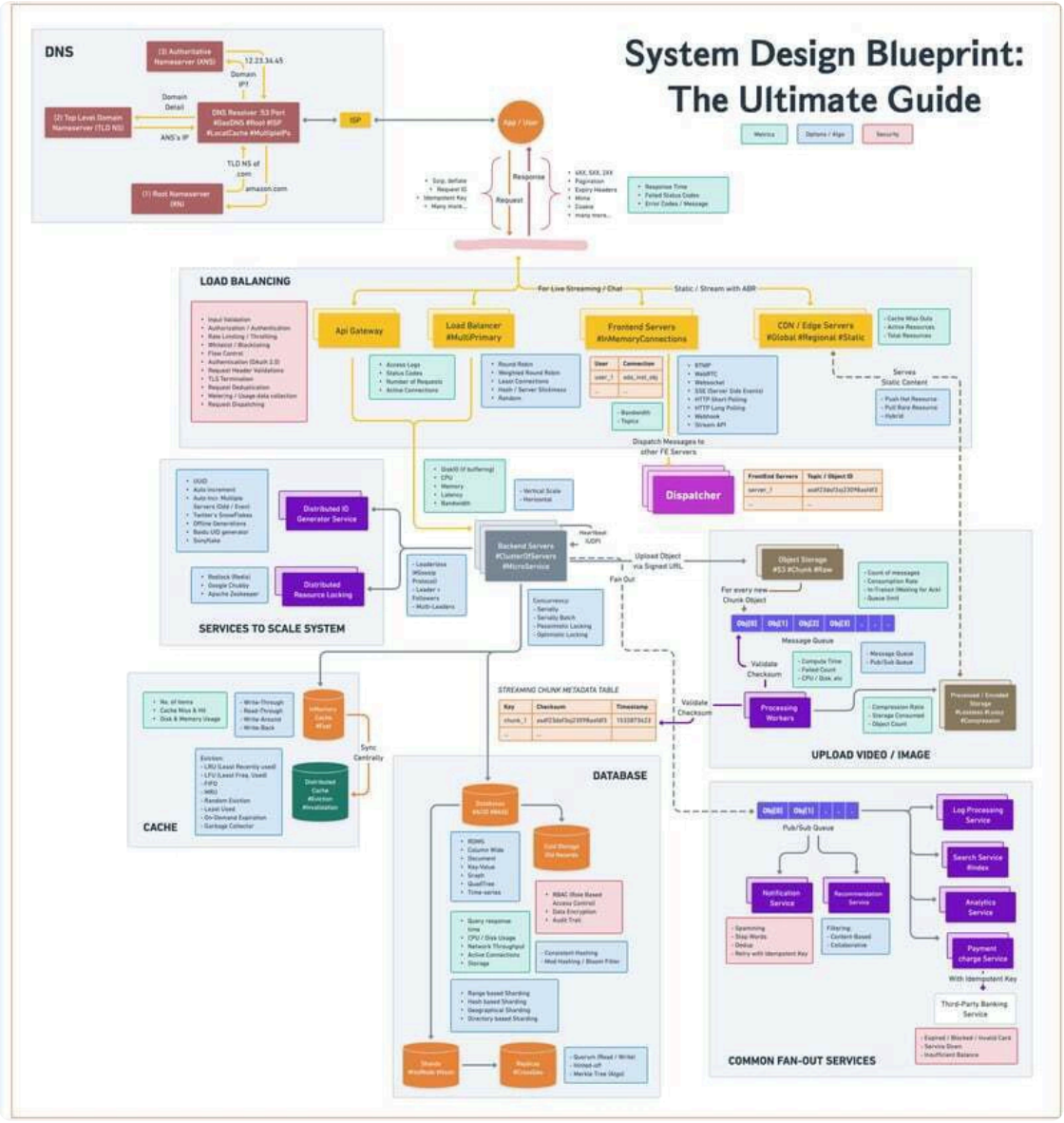
7. **Educative's System Design Course**: An interactive learning platform with hands-on exercises and real-world scenarios to strengthen your system design skills.

8. **High Scalability Blog**: A blog that features articles and case studies on the architecture of high-traffic websites and scalable systems.

9. **YouTube Channels**: Check out channels like "Gaurav Sen" and "Tech Dummies" for insightful videos on system design concepts and interview preparation.

10. [ByteByteGo](): A live book and course by Alex Xu for System design interview preparation. It contains all the content of System Design Interview book volume 1 and 2 and will be updated with volume 3 which is coming soon.

11. ["System Design Interview" by Alex Xu](): This book provides an in-depth exploration of system design concepts, strategies, and interview preparation tips.



image_credit - [ByteByteGo]()

That's all about the **common Microservice patterns and concepts a developer should know**. These microservices patterns help address various challenges associated with building and maintaining distributed systems, providing solutions for communication, fault tolerance, data management, and scalability.

When designing microservices architectures, combining these patterns judiciously can lead to a robust and resilient system.

These additional microservices patterns, when applied thoughtfully, contribute to building resilient, scalable, and maintainable distributed systems.

The choice of patterns depends on the specific requirements and challenges faced during the design and implementation of microservices architectures.

Bonus

As promised, here is the bonus for you, a free book. I just found a new free book to learn Distributed System Design, you can also read it here on Microsoft --- https://info.microsoft.com/rs/157-GQE-382/images/EN-CNTNT-eBook-DesigningDistributedSystems.pdf

Thank you

Top comments (7)

Keanu Dirkse • Jul 20 '24

Hi Soma.

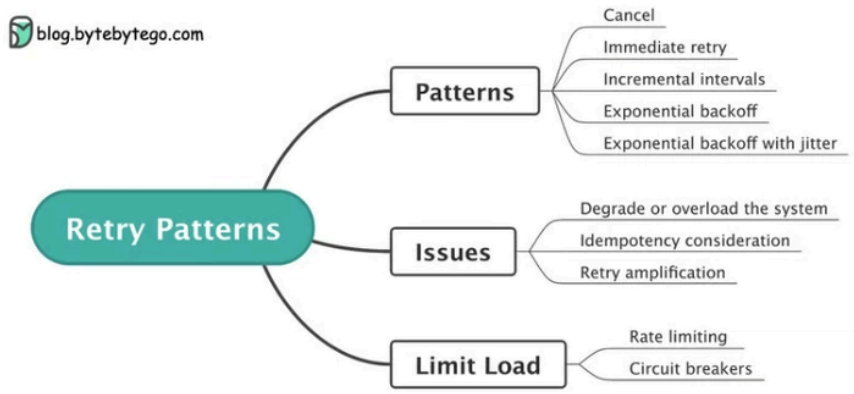
It seems you skipped number 11

10. Retry

In Microservice architecture, when a transient failure occurs, the Retry pattern involves retrying the operation instead of immediately failing.

It can be applied at various levels, such as service-to-service communication or database interactions.

Here is a nice diagram form ByteByteGo, a great place for system design learning which shows Retry pattern in Microservices:



12. Sidecar

The Sidecar pattern involves attaching a helper service (the sidecar) to the main microservice to provide additional functionalities such as logging, security, or communication with external services.

Soma • Jul 20 '24

good catch, thanks, I will update the numbers.

Mehdi Hafid • Jul 15 '24

For a ready to use BFF open source implementation check out Nidam

---

João Angelo • Jul 16 '24

Hi Soma,
Top, very nice and helpful !
Thanks for sharing.

Soma • Jul 16 '24

Thanks for your kind words.

Tung Leo • Aug 14 '24

Awesome post! Thanks @somadevtoo for your efforts!

Sagid Kamilov • Mar 7 • Edited on Mar 7

Very usefull! Thank you much!

Code of Conduct • Report abuse

More from Soma

ByteByteGo vs DesignGurus.io? Which is better for System Design Interview Preparation?

#systemdeisgn #architecture #programming #development

System Design Interviews were HARD, until I learned these Concepts

#systemdesign #programming #architecture #softwaredevelopment

I Tried 20+ Python Courses on Udemy: Here are 5 Courses I recommend

#python #programming #courses #development

DEV

gg

DUMB DEV

ViBE

+

...