Open in app ↗

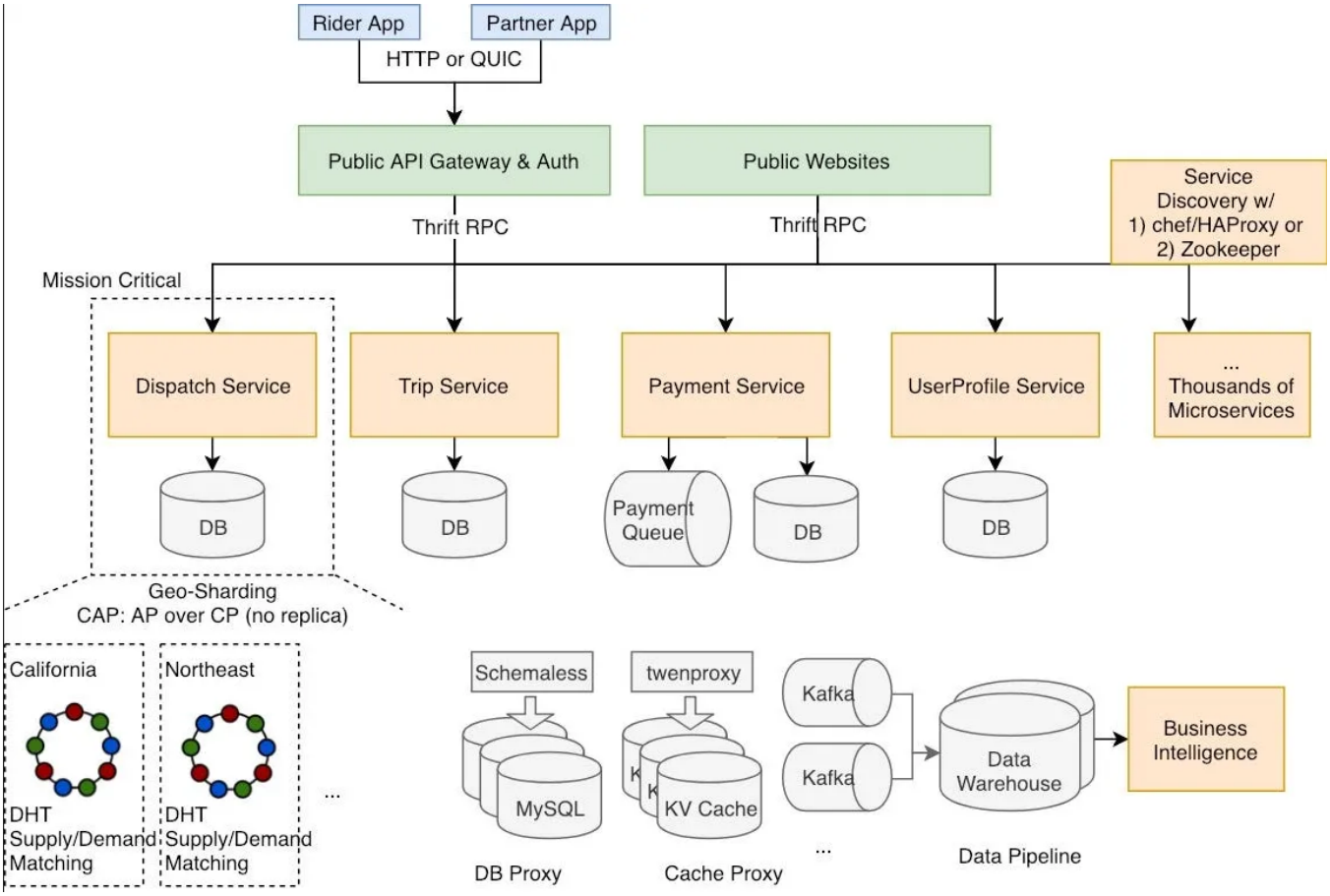
Member-only story

Scaling Microservices: Strategies for Handling Increased Demand with 99% Efficiency

Crafting Resilient Microservices Infrastructures: Proven Strategies for Achieving 99% Efficiency in Handling Skyrocketing Demand

Soma · Following
Published in Stackademic
8 min read · 3 days ago

Listen Share ••• More



image_credit — uber.com

Hello guys, many of you asked me to write about **how to scale microservices in real world**? which give me an idea about this article, where I am going to share what I know about scaling the Microservices and also giving you an example of Scaling Microservices to millions while maintaining 99% uptime from none other than Uber.com, my favorite taxi haling app.

While its one thing to create a Microservices which is working its totally different thing to create a Microservices which is working senselessly with millions of user sending billions of events every day.

Microservices architecture has gained significant popularity in recent years due to its ability to create modular, independent, and highly scalable software systems. However, as an application's user base grows and traffic increases, effectively scaling microservices becomes a critical challenge.

In past few articles I have been sharing my experience on Microservices and Spring Cloud like Microservices best practices, Microservices design patterns, 50 Microservices Interview questions and 10 Spring Cloud Features for Microservices, which I shared earlier as well my article about SAGA Design Pattern and Monolitihic vs Microservices architecture.

In this article, we will delve into strategies for handling increased traffic and demand in Microservices-based applications, focusing on load balancing, auto-scaling, and performance optimization. We will also explore a real-world example to illustrate these concepts in action.

By the way, if you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium here**

Join Medium with my referral link - Soma

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

medium.com

1. The Challenge of Scaling Microservices

Microservices architecture breaks down a monolithic application into smaller, interconnected services that communicate with each other over the network.

This design allows for flexibility, better resource utilization, and independent development and deployment of services. However, as user traffic grows, individual services might experience bottlenecks and performance issues.

Scaling Microservices presents several challenges that organizations need to address to ensure their systems can handle increased traffic and demand effectively. Some of the main challenges include:

1. **Service Dependencies**: Microservices often rely on other services to perform specific functions. As traffic increases, the interdependencies between services can lead to bottlenecks and performance issues, affecting the overall system's scalability.
2. **Data Management:** Coordinating and managing data across distributed microservices can become complex, especially when dealing with large datasets. Ensuring data consistency, availability, and integrity across services can be a challenge.
3. **Network Communication Overhead:** Microservices communicate over a network, and as the number of services grows, network communication can become a significant overhead. This can lead to latency and reduced response times, impacting user experience.
4. **Operational Complexity:** Operating and monitoring a large number of microservices requires robust infrastructure, deployment pipelines, and monitoring tools. As the system scales, managing these aspects becomes increasingly complex.
5. **Dynamic Scalability:** While auto-scaling is a key strategy, dynamically adding or removing instances based on demand can lead to resource contention, especially if not implemented properly.
6. **Consistency and Transactions:** Maintaining transactional consistency across distributed services can be challenging. Ensuring that updates to multiple

services are either fully completed or fully rolled back in case of failures is complex.

7. **State Management:** Managing the state of microservices, especially in scenarios where state is necessary, can be difficult. Handling failover and replication of stateful services adds complexity to the scaling process.
8. **Security and Authorization:** Ensuring security across microservices while handling increased traffic requires robust authentication and authorization mechanisms. As the number of requests grows, managing security becomes more critical.

Addressing these challenges requires careful planning, architectural considerations, and the adoption of best practices in microservices development, deployment, and operations. Successful scaling involves a combination of technical expertise, monitoring tools, and continuous optimization efforts.

But, don't worry we are going to see best practices you can follow to scale your Microservices to millions of users keeping 99% efficiency.

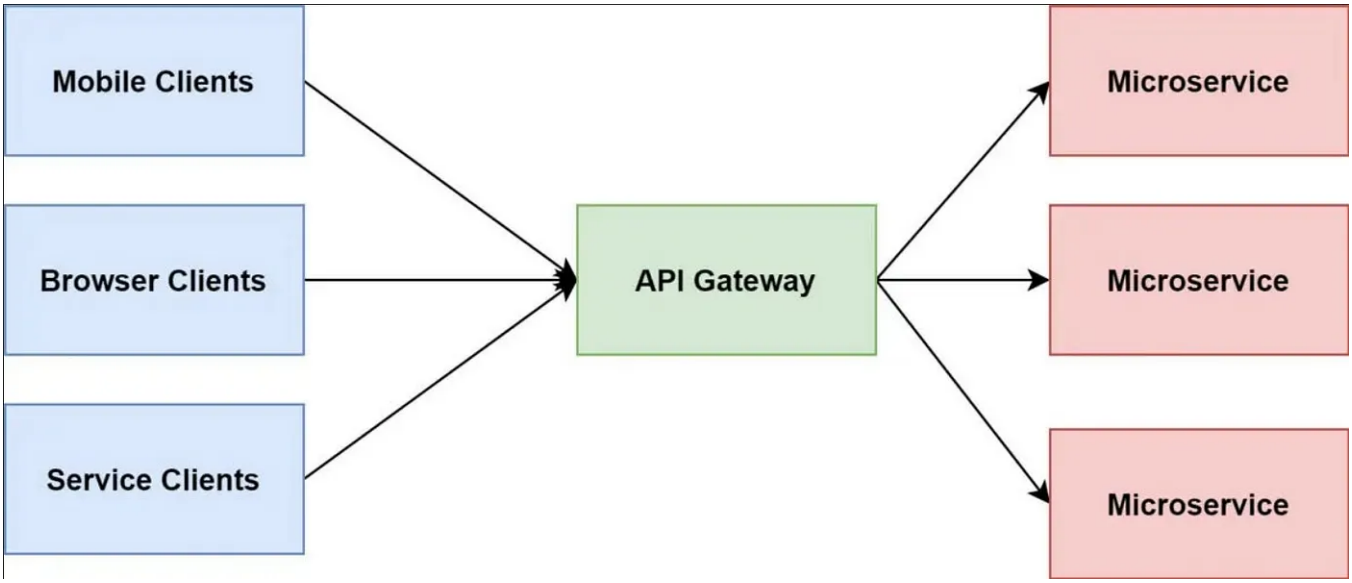
2. Understanding Load Balancing

Load balancing is a critical strategy for distributing incoming network traffic across multiple instances of a service to ensure even resource utilization and prevent overloading any single instance. There are various load balancing algorithms, each with its strengths and use cases:

1. **Round Robin:** Requests are distributed evenly across instances in a circular order.
2. **Least Connections:** Traffic is routed to the server with the fewest active connections.
3. **Weighted Round Robin:** Servers are assigned different weights, reflecting their capacity.

You can also use Patterns like **API Gateway** as a load balancer in a microservices architecture to distribute incoming traffic across multiple instances of your

microservices.



An API Gateway acts as a single entry point for clients to access various microservices, providing benefits such as load balancing, routing, security, and more. I also talked about in depth on my earlier post *how API Gateway works*, you can see that as well

What is API Gateway Pattern in Microservices Architecture? What Problem Does it Solve?

The API Gateway can help in managing authentication, request routing, load balancing, and caching in Microservices...

medium.com

3. Auto-Scaling: Adapting to Demand

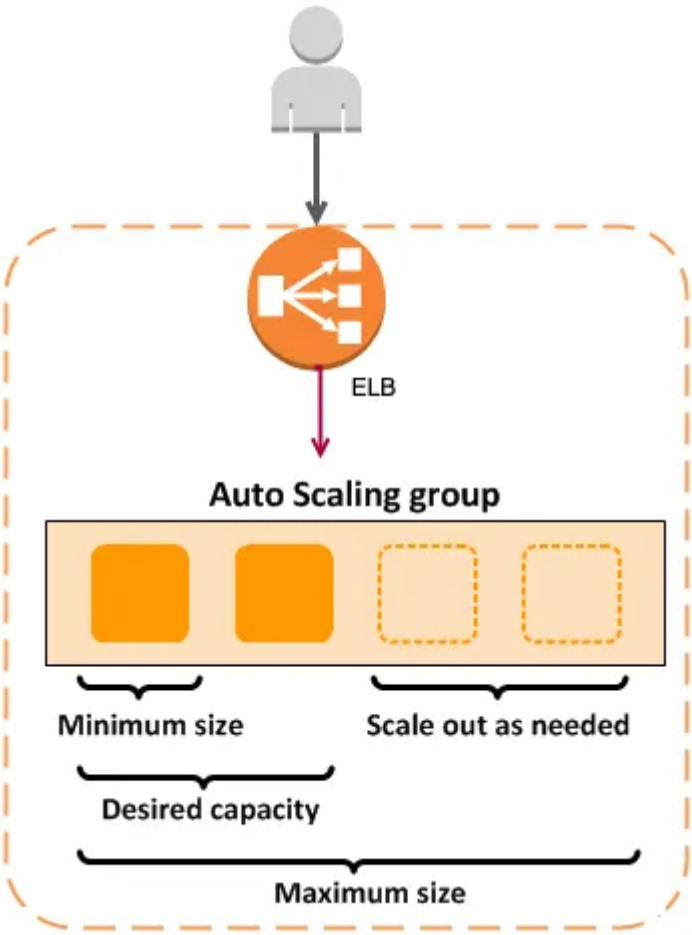
This is another strategy you can use to scale your Microservices to millions. Auto-scaling allows Microservices to dynamically adjust their resources based on traffic fluctuations.

Instead of manually provisioning and de-provisioning instances, an auto-scaling system continuously monitors the load and adds or removes instances as needed.

This ensures optimal performance and cost efficiency.

Key metrics for triggering auto-scaling include CPU utilization, memory consumption, and response time. All major Cloud platforms like AWS, Google Cloud, and Azure provide auto-scaling services that can be integrated with microservices deployments.

Here is an example of setting up Auto Scaling in AWS Cloud:



4. Performance Optimization: Ensuring Efficient Execution

This is probably the oldest strategy to scale your application and also maintain its responsiveness. Performance optimization involves fine-tuning microservices to maximize efficiency and minimize response times. Techniques include:

1. **Caching**: Storing frequently accessed data in memory to reduce database queries.
2. **Asynchronous Processing**: Moving resource-intensive tasks to background workers or queues.

3. **Database Sharding:** Distributing data across multiple database instances to improve read and write performance.

Here is another great example of Caching in Serverless architecture by AWS

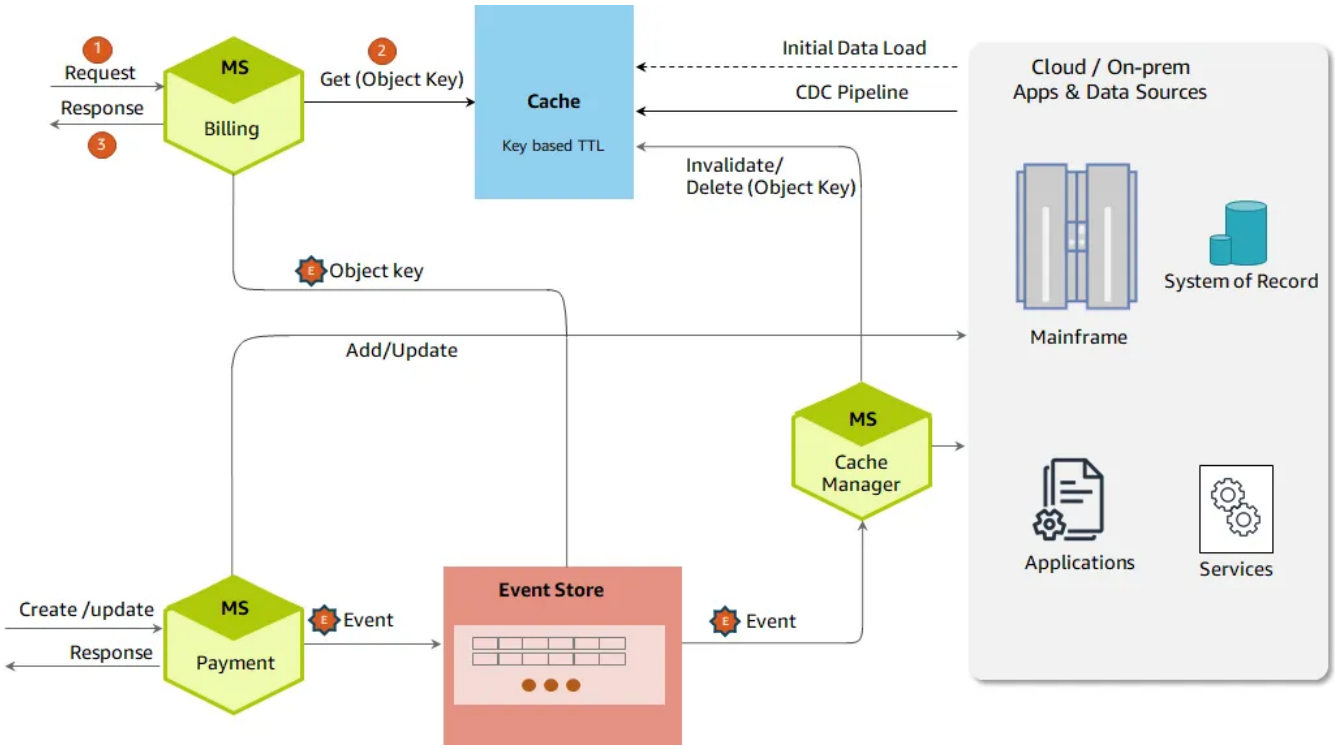


image — https://aws.amazon.com/blogs/architecture/data-caching-across-microservices-in-a-serverless-architecture/

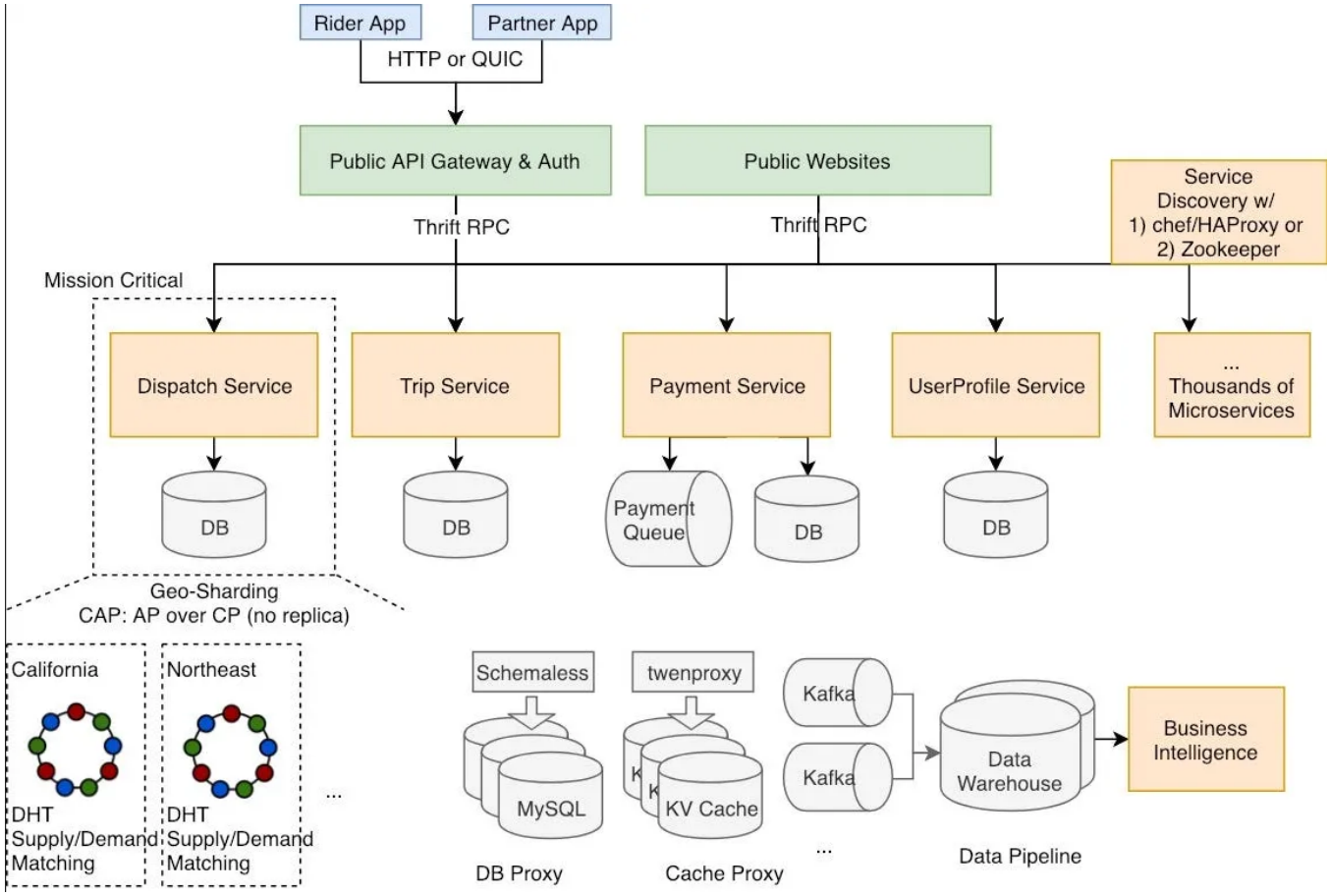
5. Real-World Example: Uber's Journey to Scalability

There is no better way to understand anything then seeing an example. Let's take a look at a real-world example to illustrate these scaling strategies in action: **Uber, the ride-hailing giant.**

Uber's Microservices architecture has enabled it to operate seamlessly in numerous cities worldwide, serving millions of users daily. As Uber's popularity soared, effectively scaling its services became crucial.

Btw, this is another thing I do and suggest fellow developer to read engineering blog from tech giants like Uber, NetFlix etc to learn how they are solving challenges. This will improve your knowledge and understanding .

Here is **how Uber's Microservice architecture** looks like:



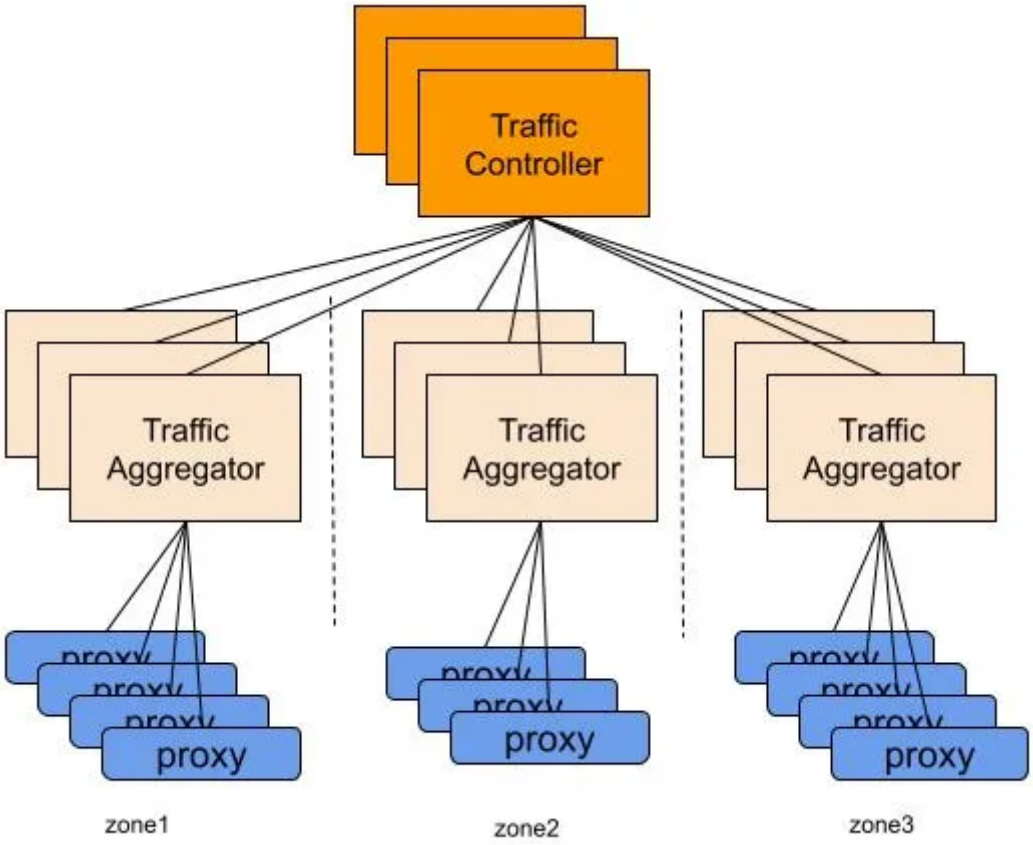
You can see that they are using API Gateway, Cache Proxy and many more things which you can learn to scale your Microservices to millions.

6. Load Balancing in Uber

To manage high traffic, Uber employs a robust load balancing mechanism. When a user opens the Uber app, their request is directed to a nearby data center.

Within the data center, load balancers distribute the request to the least busy microservice instance using the Least Connections algorithm.

This ensures even distribution of user requests, preventing any single microservice instance from becoming overwhelmed. You can read more about Uber's real time load balancing here.

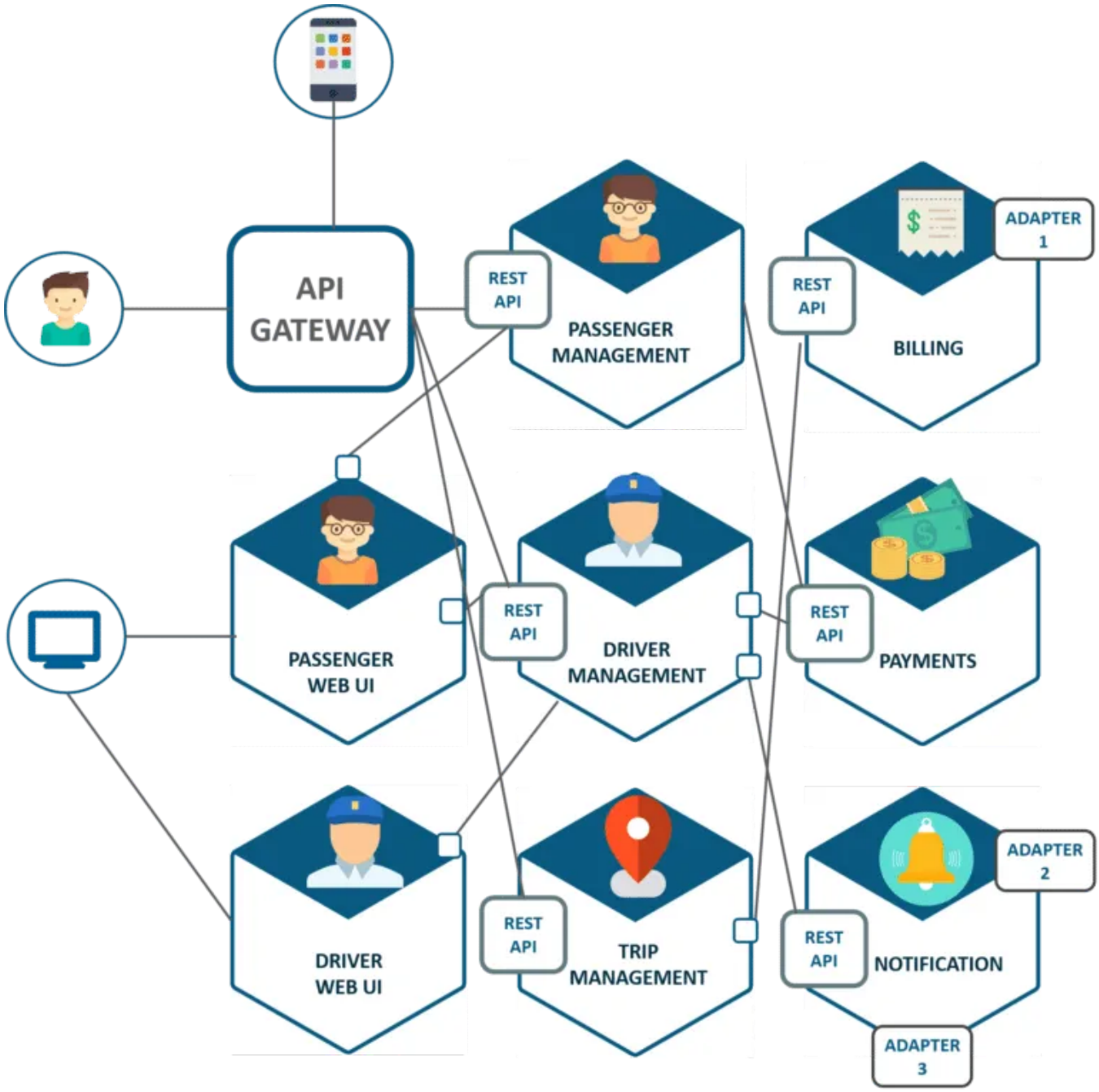


7. Auto-Scaling to Meet Demand

During peak hours or special events, Uber experiences a surge in ride requests. To handle these spikes, Uber utilizes auto-scaling extensively.

For instance, the “Ride Request” Microservice, responsible for matching riders and drivers, dynamically scales its instances based on metrics like the number of incoming requests and average response time.

Uber’s auto-scaling system monitors these metrics and automatically adjusts the number of instances to ensure low response times and quick ride matching. When the traffic subsides, unnecessary instances are scaled down, optimizing resource usage and cost.



8. Performance Optimization for Seamless Experience

Uber's app provides real-time location tracking, which demands high-performance microservices. To optimize performance, Uber uses caching extensively.

Driver and rider data, as well as ride status updates, are cached in memory, reducing the need for frequent database queries.

Furthermore, Uber employs asynchronous processing for tasks like calculating ride fares and sending notifications.

By moving these tasks to background workers and queues, microservices can quickly process ride data without affecting the user experience.

Conclusion

That's all about the essential things you can do to scale your Microservices to millions of users. While this may seems simple when you actually do it you will realize that it require careful planning and execution.

As user demands continue to increase, scaling microservices effectively is a critical aspect of modern application development. Load balancing, auto-scaling, and performance optimization are essential strategies to ensure optimal performance, reliability, and cost-efficiency in microservices-based applications.

Real-world examples like Uber demonstrate the practical implementation of these strategies, showcasing their effectiveness in handling high traffic and demand.

By leveraging load balancing algorithms, auto-scaling mechanisms, and performance optimization techniques, you can create resilient and responsive microservices ecosystems that provide a seamless user experience even during peak usage periods.

In conclusion, mastering the art of scaling microservices is essential for businesses aiming to deliver high-quality digital services in the face of ever-growing user expectations and traffic volumes.

By the way, if you are preparing for Microservices interviews then *you should also prepare questions like difference between API Gateway and Load Balancer, SAGA Pattern, how to manage transactions in Microservices,* and *difference between SAGA and CQRS Pattern,* they are quite popular on interviews.

And, if you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium here**

*Thank you for reading until the end. Please consider following the writer and this publication. Visit **Stackademic** to find out more about how we are democratizing free programming education around the world.*

Join Medium with my referral link - Soma

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

medium.com

Other **Java and Microservices articles** you may like:

Why Microservices are not silver bullet? 10 Reasons for NOT using Microservices

Don't blindly use Microservices for every single application you build

medium.com

Top 10 Microservices Problem Solving Questions for 5 to 10 years experienced Developers

From Service Discovery to Security: 10 Problem-Solving Questions to Test the Microservices Skills of Developers with 5...

medium.com

10 Tips and Best Practices for Creating Microservices with Spring Cloud and Java

Tips and best practices for creating better Microservices in Java using Spring Cloud

medium.com

Microservices Software Engineering Software Development Java Programming

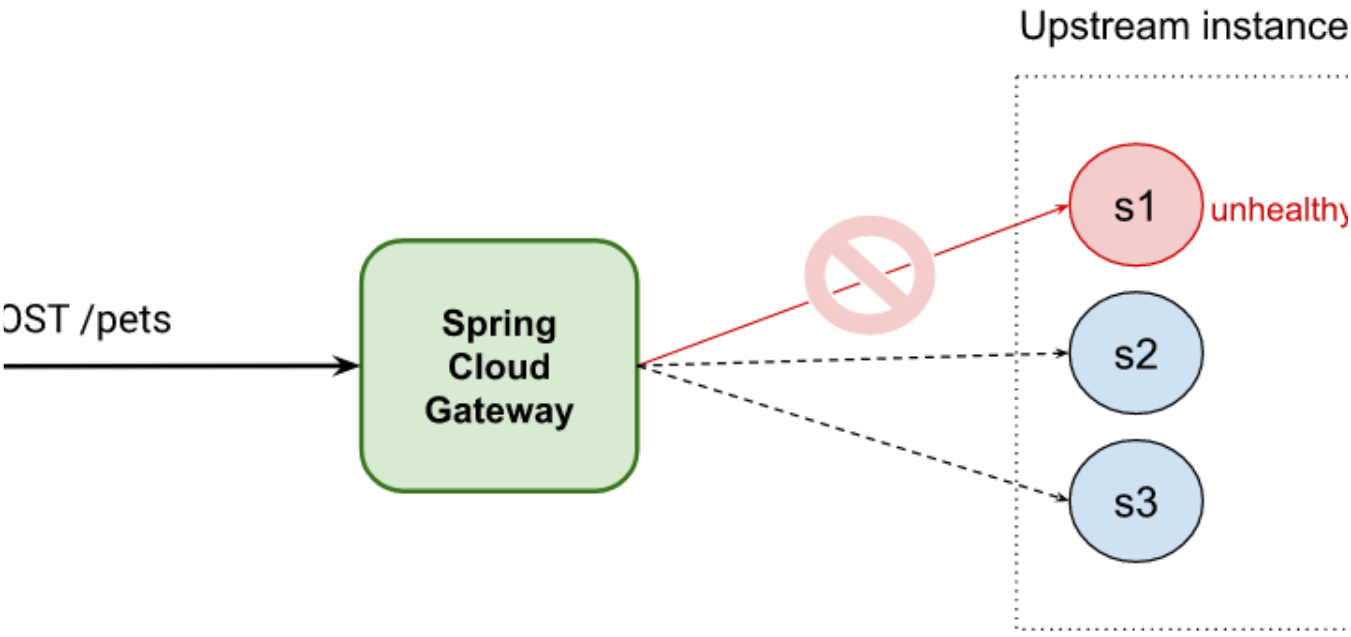
Following

Written by Soma

4.1K Followers · Writer for Stackademic

Java and React developer, Join Medium (my favorite Java subscription) using my link 👉
https://medium.com/@somasharma_81597/membership

More from Soma and Stackademic



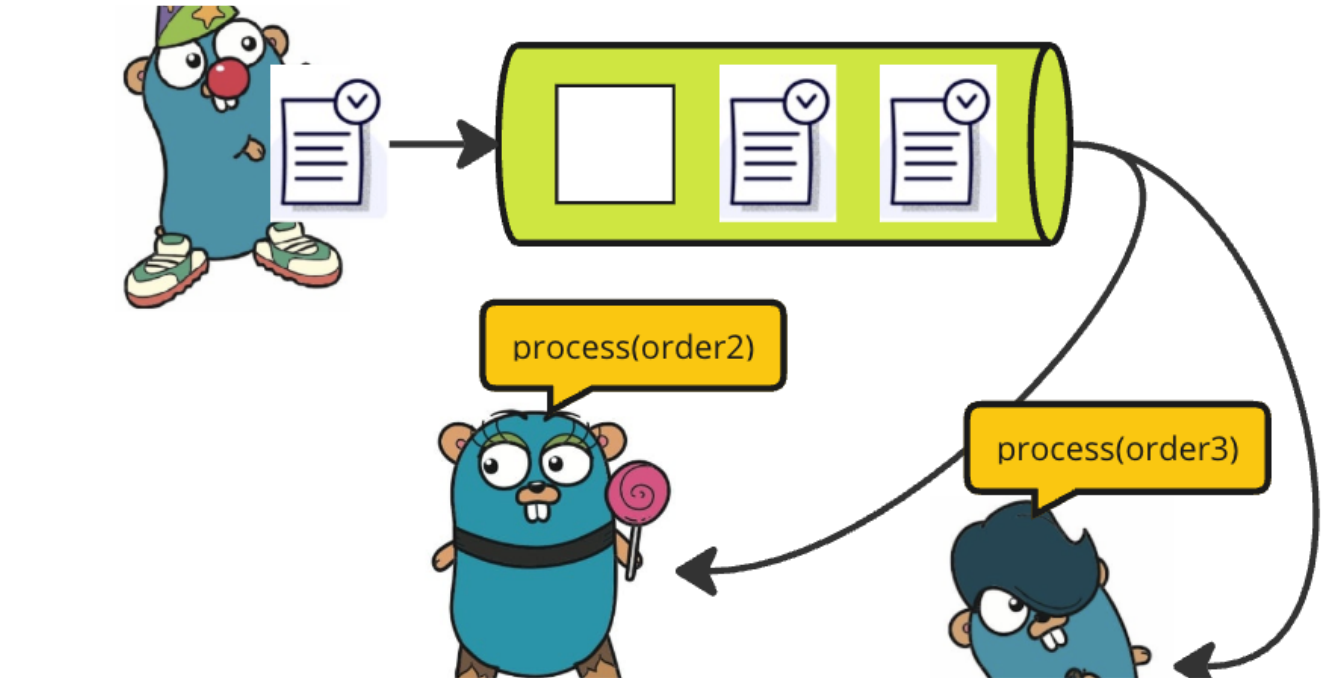
Soma in Javarevisited

Top 20 Spring Cloud Interview Questions with Answers for Java Developers

Common Spring Cloud Questions for Java Interviews with answers

· 9 min read · Jul 23

184



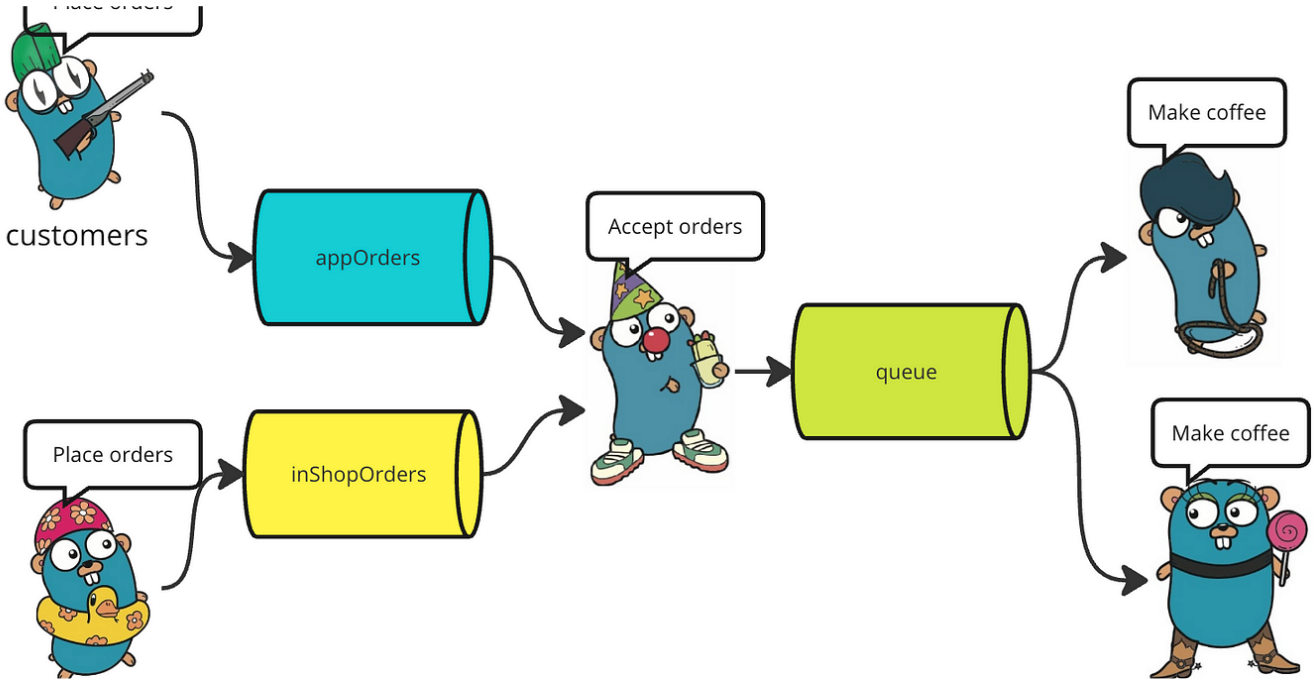
Brian NQC in Stackademic

Go Concurrency Visually Explained—Channel

When it comes to concurrency, many programming languages adopt the Shared Memory/State Model. However, Go distinguishes itself by...

6 min read · Jul 24

264 3



Brian NQC in Stackademic

Go Concurrency Visually Explained – Select statement

5 min read · 6 days ago

122

Purpose	authentication mechanism for transmitting claims	authorization and authentication in web and mobile apps	authentication and authorization data between parties
Centralized Server	None	Authorization server	Identity provider
Used For	Authentication and authorization	Authorization, not authentication	Authentication and authorization
Applications	Single-page apps, mobile apps	Apps that rely on external APIs or services	Enterprise environments
Relationship	Directly between parties	Between third-party apps and resource owners	Between identity providers and service providers
Authentication	Yes	No	Yes

Soma in Javarevisited

Difference between JWT, OAuth, and SAML for Authentication and Authorization in Web Apps?

Understanding the differences between Popular Web Authentication and Authorization Standards: JWT, OAuth, and SAML

· 9 min read · May 9

335 1

See all from Soma

See all from Stackademic

Recommended from Medium

Uptrace

Monitoring Spring Boot with OpenTelemetry

Integrating OpenTelemetry with Spring Boot allows you to capture distributed traces and other telemetry data from your application...

4 min read · Aug 2

11

Ashish Pandey in Excited Developers

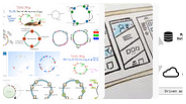
Java Multithreading—All you need to know

The only blog that you will ever need. Let's get started

13 min read · Jun 1

70

Lists

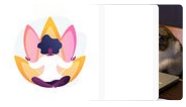


General Coding Knowledge

20 stories · 192 saves

It's never too late or early to start something

13 stories · 66 saves



Stories to Help You Grow as a Software Developer

19 stories · 261 saves

Coding & Development

11 stories · 97 saves

arctic_fox in Stackademic

Leaderboard using Spring Boot and Relational Database

Design for scalable and configurable leaderboard in spring boot and relational database

6 min read · Jul 31

2

Naveen

Unleashing the Power of Java Modularization (Java 9+ Feature)

Introduction:

3 min read · Jun 15

Mahdi Razavi

Spring Autowiring: Strategies for Resolving Bean Conflicts

In case of multiple beans of the same type, Spring finds itself in a quandary over which bean to autowire.

3 min read · Aug 2

3

Rupert Waldron

Create a non-blocking REST Api using Spring @Async and Polling

Get the great asynchronous, non-blocking experience you deserve with Spring's basic Rest Api, polling and @Aysnc annotation.

12 min read · Feb 26

16

See more recommendations