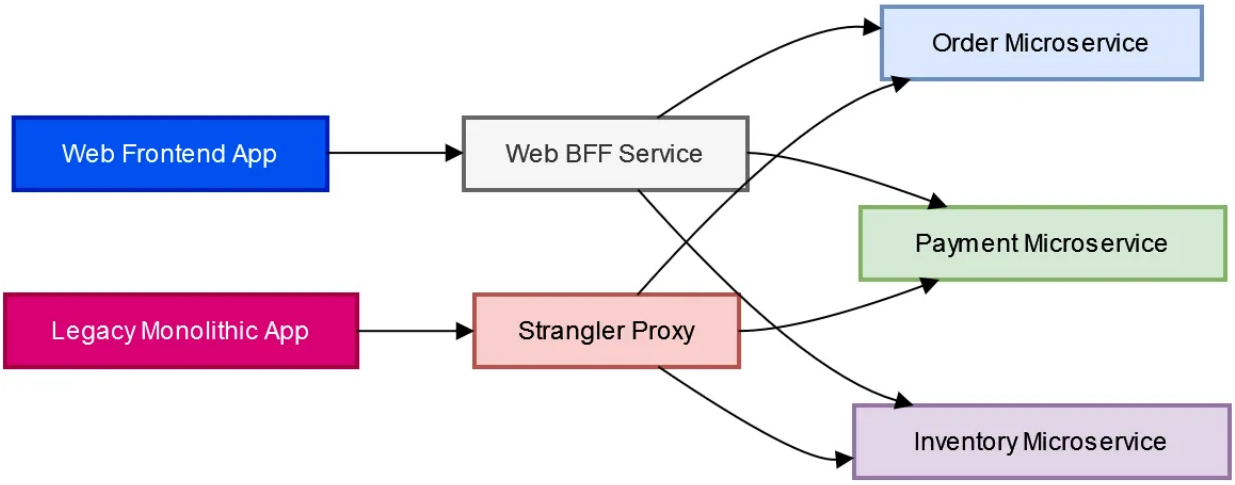
Member-only story

Difference between BFF and Strangler pattern in Microservices?

Key difference between BFF and Strangler pattern in Microservices which every developer should know

Soma · Follow
Published in Javarevisited
16 min read · Apr 18

Listen Share More



Strangler and BFF Pattern in Microservices

Hello folks, in Microservices architecture, there are many design patterns and techniques used to design and implement scalable and resilient systems. Two such commonly used patterns are the *BFF (Backend For Frontend) pattern* and the *Strangler pattern*. While both patterns are used in Microservices architecture, they have different purposes and use cases which we will see in this article.

In the past articles, we have seen the **service discovery**, **Event Sourcing**, **CQRS**, **SAGA**, **Database Per Microservices**, **API Gateway**, **Circuit-Breaker** as well as key **Microservices design principles and best practices** and in this article, we will learn about difference between Strangler and BFF Pattern in Microservices.

The BFF pattern focuses on optimizing the communication between microservices and the frontend, while the Strangler pattern is used to gradually replace a legacy monolithic application with microservices. Understanding the differences between these patterns is crucial for making informed design decisions in microservices architecture.

In this article, we will explore the key *differences between the BFF pattern and the Strangler pattern in microservices architecture*. We will delve into their definitions, use cases, advantages, and limitations, to help you understand when and how to use each pattern in your microservices projects.

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**

What is Strangler Pattern in Microservice Architecture?

The Strangler Pattern is a software development pattern that is commonly used in microservices architecture to **gradually migrate from a monolithic application to a microservices-based system.**The name *"Strangler"* pattern is inspired by the way a vine gradually envelops and replaces an existing tree in the natural world.

In the context of microservices, the Strangler Pattern involves incrementally replacing components or functionalities of a monolithic application with microservices over time. This is done by gradually *"strangling"* the monolithic

application by introducing new microservices around it and gradually redirecting traffic and functionality to these microservices.

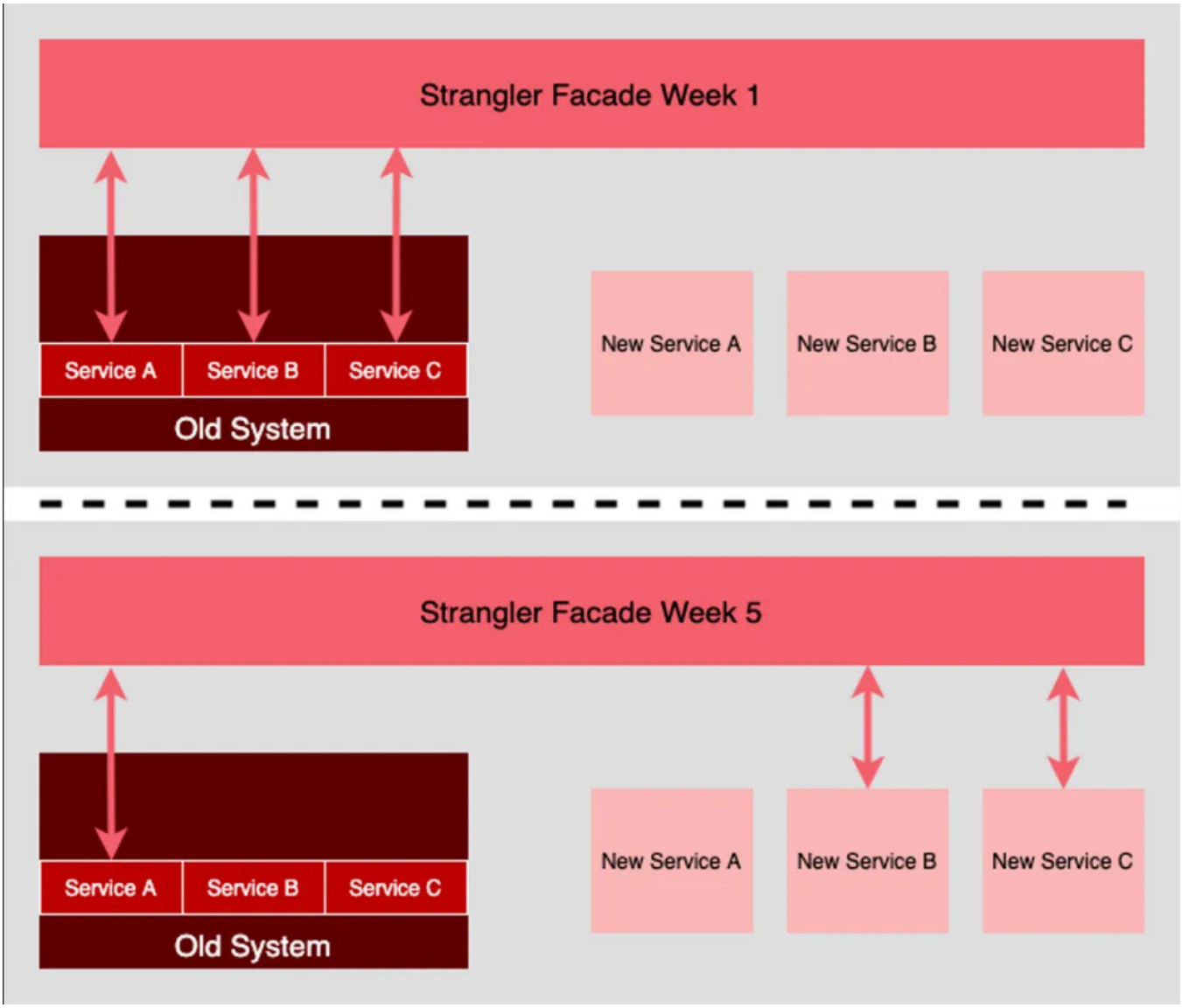
This approach allows for a gradual and controlled transition from a monolithic architecture to a microservices-based architecture, without disrupting the existing functionality of the application.

The **Strangler Pattern typically involves the following steps**:

1. Identifying specific functionalities or components in the monolithic application that can be extracted and refactored into separate microservices.
2. Developing and deploying new microservices that replicate the functionalities of the identified components.
3. Gradually routing traffic from the monolithic application to the new microservices, either through load balancing, API gateways, or other mechanisms.
4. Iteratively repeating the process for other functionalities or components until the entire application is decomposed into microservices.

One of the main advantages of the Strangler Pattern is that **it allows for a gradual migration to microservices without requiring a complete overhaul of the existing monolithic application**. It also provides the flexibility to prioritize functionalities for migration based on business priorities and resource availability.

Here is a nice diagram which shows Strangler pattern in Microservices:



You can se that initially Strangler act as facade for old system (monolith) and gradually the traffic is migrated to new Microservices (new system).

However, implementing the Strangler Pattern requires careful planning, coordination, and thorough testing to ensure smooth transitions and avoid potential disruptions in the application's functionality.

It also requires effective monitoring and management of the evolving microservices landscape to maintain system integrity and performance, so you must pay attention to those details before strangling your monolith application.

What is BFF (Backend for Frotnend) Pattern in Microservices?

Now that you know what is Strangler pattern and what it does, its time to look your BFF (Not Best Friend Forever :-)).

BFF, which **stands for Backend for Frontend**, is a software development pattern used in microservices architecture where a dedicated backend service is designed and optimized to serve the specific needs of a frontend application or client.

The BFF pattern allows for **tailored backend services that cater to the requirements of different frontend applications,** such as web, mobile, or other types of clients, providing optimized APIs and data retrieval mechanisms.

In a Microservices architecture, the BFF pattern is used to decouple the frontend and backend components, allowing for independent development, scalability, and flexibility. The ***BFF acts as an intermediary between the frontend and the backend microservices***, providing a layer of abstraction that enables the frontend to interact with the backend in a more efficient and targeted manner.

The BFF pattern typically involves the following characteristics:

1. **Tailored APIs**
   The BFF is *designed to expose APIs* that are optimized for the specific needs of the frontend application or client. This may include aggregating data from multiple microservices, formatting data in a specific way, or providing a simplified and streamlined API for the frontend to consume.

2. **Aggregation and composition**
   The BFF *may aggregate or compose data from multiple backend microservices to fulfill the requirements of the frontend*. This can help reduce the number of API calls made by the frontend and minimize the amount of data transmitted over the network.

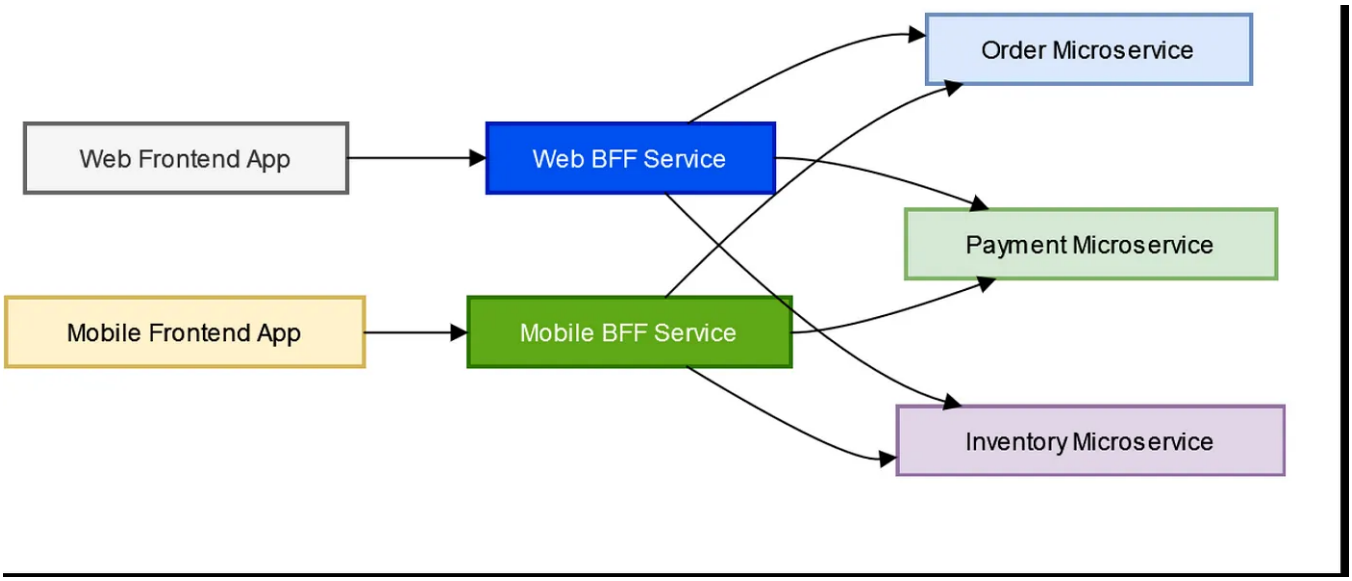
3. **Performance optimization**
   The BFF may *optimize data retrieval mechanisms, caching, and other performance-related aspects* to ensure optimal performance for the frontend application. This may involve caching frequently used data, optimizing queries, or pre-processing data to reduce processing overhead on the frontend.

4. **Security and authentication**
   The *BFF can handle security and authentication-related concerns,* such as authentication and authorization of requests from the frontend, and ensure that only authorized requests are forwarded to the backend microservices.

5. **Frontend-specific functionality**

   The BFF can also provide frontend-specific functionality that is not part of the core backend microservices, such as handling UI-related logic, user session management, or other frontend-specific concerns.

Here is a nice diagram which shows BFF design Pattern in Microservices architecture in action:



In this architecture, both the Web Frontend App and Mobile Frontend App communicate with their respective dedicated BFF Services, which act as intermediaries between the frontends and backend microservices.

The BFF Service for the Web Frontend App (BFF Service (Web)) and the BFF Service for the Mobile Frontend App (BFF Service (Mobile)) may have different implementations optimized for the specific requirements of each frontend platform. Both BFF Services may aggregate data from the same set of backend microservices (Order, Payment, and Inventory Microservices)

In short, the BFF pattern allows for a more fine-grained control and customization of the backend services for different frontend applications, enabling improved performance, flexibility, and maintainability. However, it also requires careful coordination and management to ensure consistency, scalability, and security across the microservices ecosystem.

If you look closely, BFF pattern is very similar to API gateway pattern but not exactly same, don't worry, we will see the difference between BFF and API Gateway in next

article, so stay tuned for that and if you haven't joined Medium yet, **join Medium here**

Difference between BFF and Strangler Patter in Microservices?

Now that we have seen what is BFF and Strangler pattern, its time to look closely and see the differences. As I said, the Backend for Frontend (BFF) pattern and the Strangler pattern are both used in microservices architecture to decouple frontend and backend components and enable independent development and scalability.

Here are key differences between them:

1. Purpose

The BFF pattern is **focused on providing tailored APIs and optimized data retrieval mechanisms for specific frontend applications or clients.** It acts as an intermediary between the frontend and backend microservices, exposing APIs that are optimized for the requirements of the frontend.

On the other hand, t**he Strangler pattern is used to gradually migrate a monolithic application to a microservices architecture** by incrementally replacing parts of the monolith with microservices over time.

2. Scope

The BFF pattern typically applies to a single frontend application or client, providing APIs and functionality that are specific to that application. It may involve aggregating data from multiple microservices, formatting data in a specific way, and providing frontend-specific functionality.

On the other hand, the Strangler pattern is used to gradually replace parts of a monolithic application with microservices, typically starting with small, well-

defined parts of the application and gradually expanding the scope of microservices over time.

3. Implementation Approach

The BFF pattern is typically implemented as a dedicated backend service that is optimized for a specific frontend application or client. It may involve custom API design, data aggregation, caching, and other optimizations to cater to the frontend requirements.

The Strangler pattern, on the other hand, involves gradually replacing parts of a monolithic application with microservices, typically by creating new microservices that replicate the functionality of the monolith and gradually diverting traffic and functionality to the microservices.

4. Migration Strategy

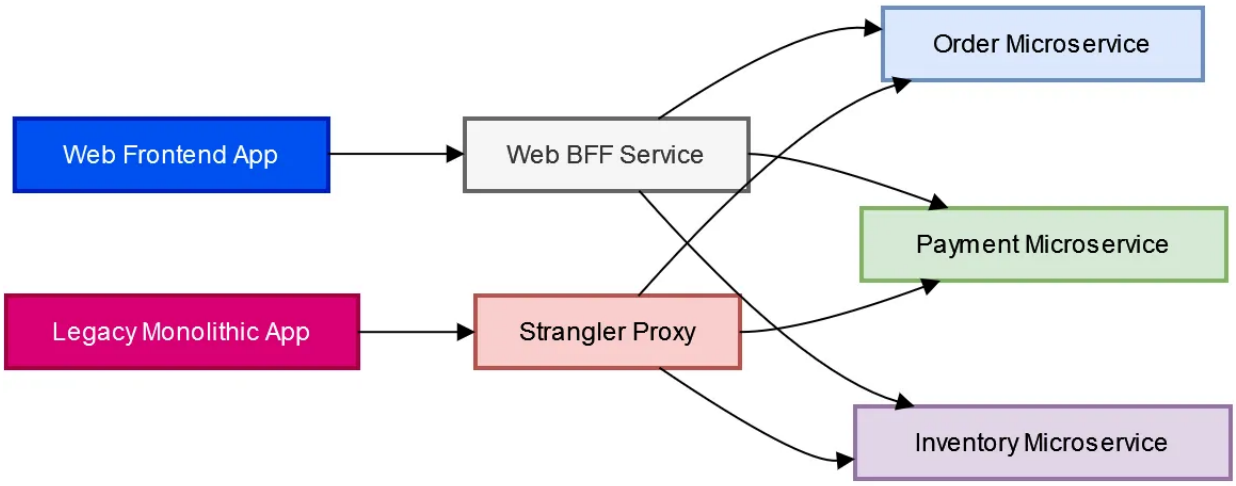
The BFF pattern does not involve migrating from a monolithic application, as it is typically used for new microservices architecture implementations. The Strangler pattern, on the other hand, involves a gradual migration from a monolithic application to a microservices architecture, typically over an extended period of time, with parts of the monolith being replaced with microservices in a phased manner.

5. Flexibility vs. Gradual Migration

The BFF pattern provides more flexibility in terms of tailoring the backend services for specific frontend applications or clients, allowing for customized APIs, data retrieval mechanisms, and frontend-specific functionality. It can be used in new microservices architecture implementations to optimize performance and functionality for different frontends.

The Strangler pattern, on the other hand, is used for gradual migration from a monolithic application to microservices, with a focus on gradually replacing parts of the monolith with microservices in a controlled and phased manner.

Here is a nice *diagram which shows both BFF and Strangler pattern in Microservices architecture in action*:

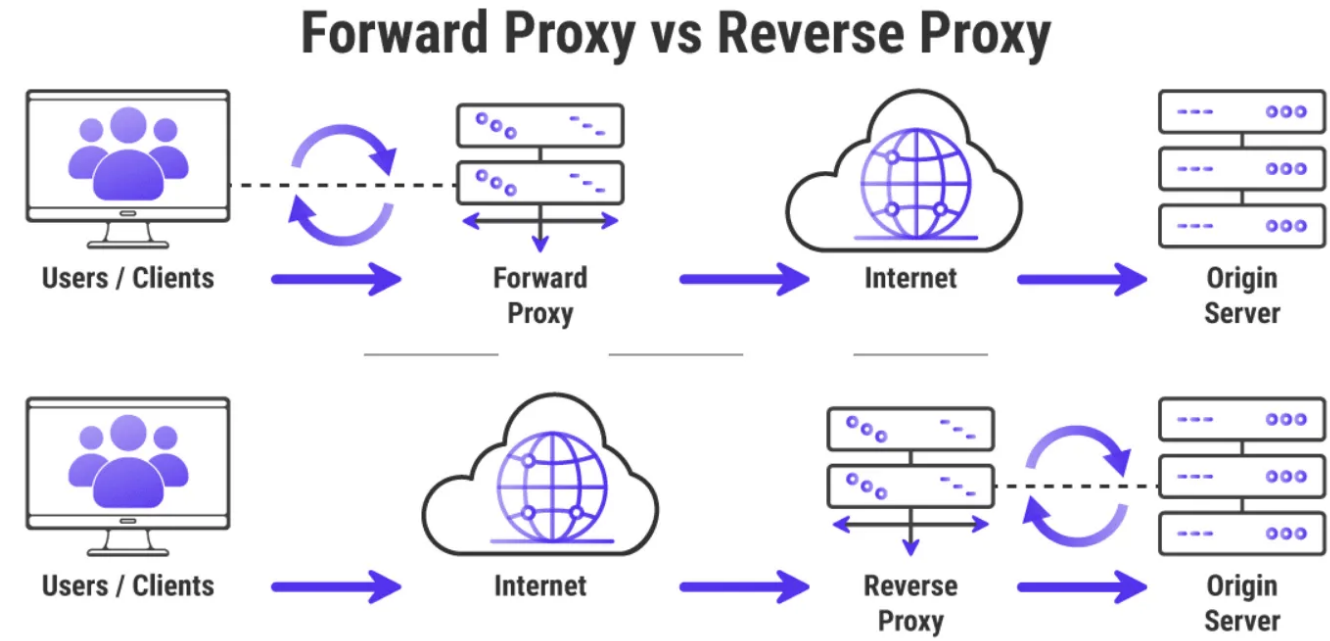


Strangler and BFF Pattern in Microservices

Difference between Forward Proxy and Reverse Proxy in System Design

Understanding the Distinctions between Forward and Reverse Proxies in System Design and when to use them

---



image_credit — https://kinsta.com/wp-content/uploads/2020/08/Forward-Proxy-vs-Reverse-Proxy-Servers.png

Hello folks, if you are preparing for System design interview then knowing the difference between forward proxy and reverse proxy is very important, its one of the most frequently asked question on System Design, along with **difference between API Gateway and Load Balancer,** which we have seen earlier.

When designing complex systems, it's common to use proxy servers to improve performance, security, and reliability. Proxy servers sit between clients and servers and help manage traffic between them.

Two types of proxies that are often used are forward proxies and reverse proxies. While both are designed to **improve the performance and security of a system**, they work in different ways and are used in different contexts. In this article, we'll explore the differences between forward proxies and reverse proxies in system design.

By the way, if you are preparing for senior developer interviews then along with System Design you should also familiar with different architectures like Microservices and various Microservice design patterns like **Event Sourcing, CQRS, SAGA, Database Per Microservices, API Gateway, Circuit-Breaker** they will help you immensely during interview as they are often used to gauge your seniority level.

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**

Now, let's come back to the topic and find out more about what is forward and reverse proxies, how to use them, what are their pros and cons and most importantly difference between forward and reverse proxies.

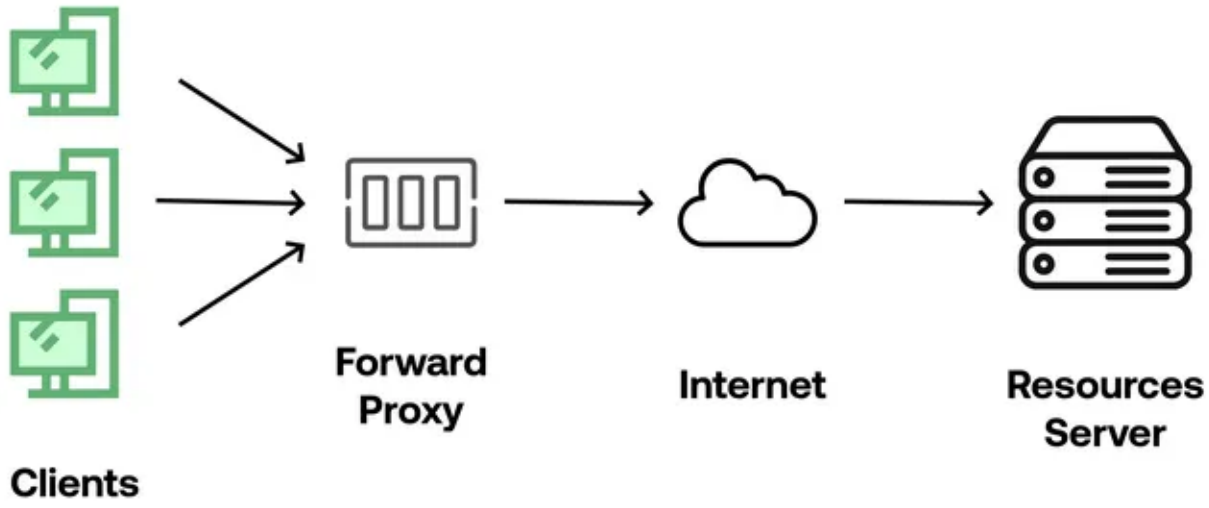
What is Forward Proxy? when to use it?

A forward proxy is a proxy server that sits between a client and the internet. The client requests a resource or service from the internet through the forward proxy,

which acts as an intermediary, forwarding the request on to the internet and then returning the response to the client.

Forward proxies are typically used t**o control access to the internet, filter content, or provide anonymity for the client**. They can also be used to speed up access to resources by caching frequently requested content.

Here is an example of forward proxy:



You can see that clients connect to forward proxy and then it routes request to internet.

What is Pros and Cons of Forward Proxy?

Now that you are familiar with the location of proxy in forward proxy, you can easily find out the pros and cons of it. Here are some pros and cons of using a forward proxy:

Pros:

1. **Enhanced security:** Forward proxy can provide an additional layer of security by hiding the original IP address of the clients accessing the internet.
2. **Improved speed and performance**: Caching frequently requested resources can improve response times for clients accessing the internet.

3. **Access control**: By restricting access to certain resources, organizations can use forward proxies to prevent unauthorized access to sensitive data.
4. **Anonymity:** Users can remain anonymous while browsing the internet, as their IP addresses are hidden.

Cons:

1. **Complex configuration**: Forward proxies require a more complex configuration as they have to be set up on individual devices to be effective.
2. **Single point of failure**: If the forward proxy fails, all the devices that rely on it will also fail to access the internet.
3. **Increased latency**: Forward proxies can increase latency and slow down the overall performance of internet access.
4. **Limited control:** Forward proxies can limit users' ability to access certain resources, leading to frustration and reduced productivity.

Apart from the limitation the access control and security benefit it provides is the main reason for using forward proxy on various architecture. Now, let's see what is Reverse proxy and how does it work.

What is Reverse Proxy? when to use it?

A **reverse proxy is a server that sits between the client and the origin server**, receiving requests from clients and forwarding them to the appropriate server. The response from the server is then returned to the proxy and forwarded to the client. In essence, it helps to protect the origin server from direct access by clients.

Reverse proxies are commonly u**sed as load balancer to balance the load across multiple servers**, improve security by hiding the details of the server infrastructure, and provide other value-added services such as caching and SSL termination.

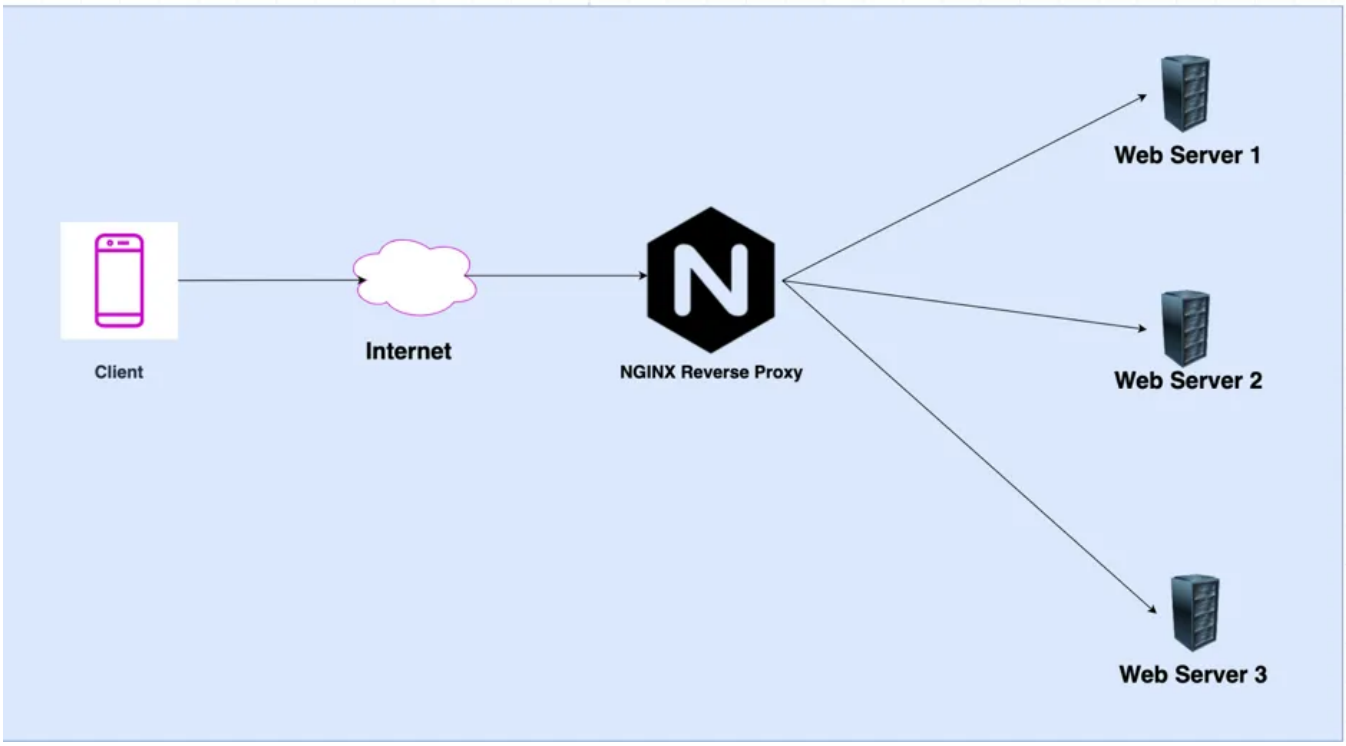
Reverse proxies are often used in the following scenarios:

- **Load balancing:** distributing incoming traffic across multiple servers to improve performance and availability.

- **Security**: protecting the backend servers from direct exposure to the internet and preventing unauthorized access.
- **Scalability**: allowing horizontal scaling of the server infrastructure without affecting the clients.

Reverse proxies **provide a single entry point for clients**, making it easier to manage and monitor the traffic to the backend servers. They also provide a level of abstraction between the clients and servers, allowing the server infrastructure to be modified or upgraded without affecting the clients.

Here is an example of NGINX reverse proxy setup:



You can see that client connects to internet directly but servers are behind proxy so client will never know which server their request are processed, hence security your infrastructure from outside.

What is Pros and Cons of Reverse Proxy Architecture?

Here are some pros and cons of using a reverse proxy architecture:

Pros:

1. **Increased security**: Reverse proxies can provide an additional layer of security by masking the identity and location of backend servers, preventing direct access to them from external clients.

2. **Better scalability:** Reverse proxies can distribute traffic evenly across multiple backend servers, ensuring that no single server becomes overloaded and causing the application to crash.

3. **Improved performance:** By caching and compressing data, reverse proxies can reduce the amount of data that needs to be transferred between clients and servers, leading to faster response times.

4. **Simplified architecture:** Reverse proxies can be used to consolidate multiple backend servers into a single endpoint, simplifying the overall architecture of the application.

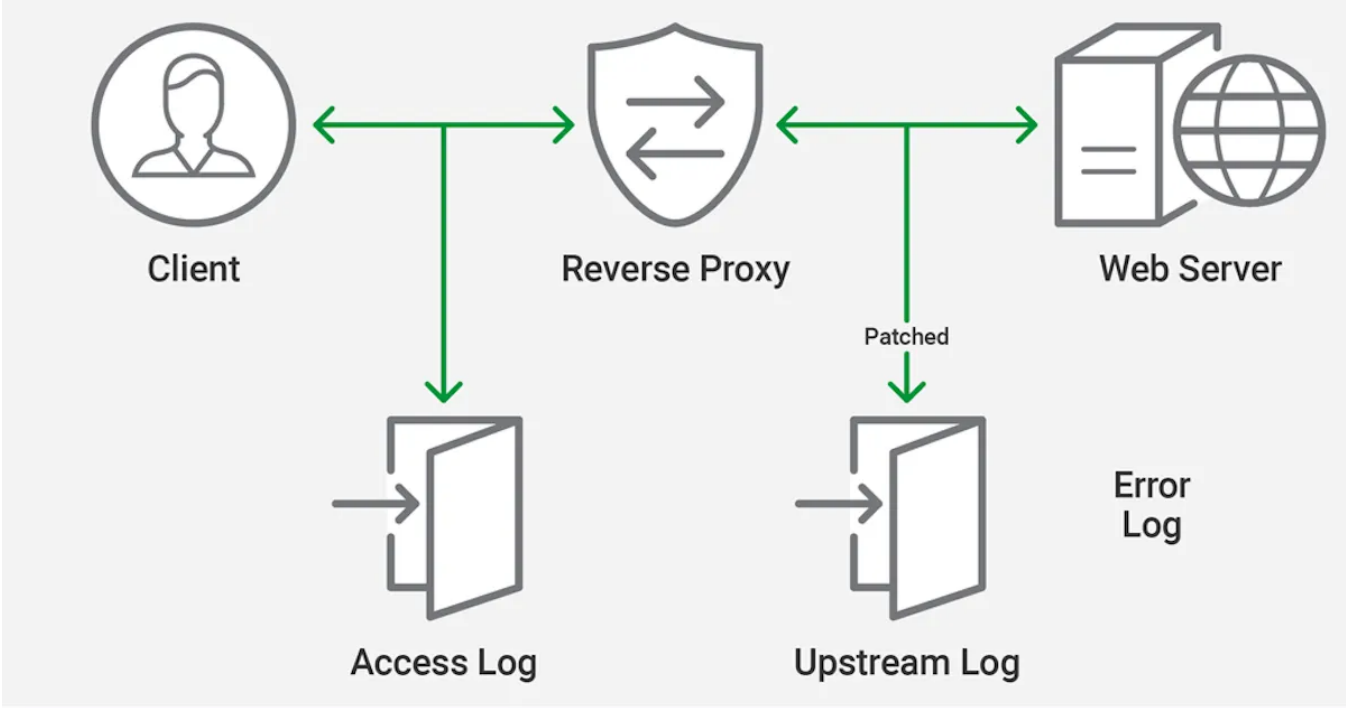
Cons:

1. **Single point of failure:** If the reverse proxy fails, the entire application may become unavailable. This is its biggest drawback.

2. **Increased complexity**: Implementing and maintaining a reverse proxy can be more complex than a simple client-server architecture.

3. **Limited customization**: Reverse proxies may not offer the same level of customization as a direct connection between clients and servers, which could limit the functionality of the application.

4. **Additional cost:** Implementing a reverse proxy can require additional hardware and software, which could increase the cost of the overall system.



What is difference between Forward Proxy and Reverse Proxy?

Now that you have basic idea of what is forward and reverse proxy, their location as well their function, now is the time to look into differences to understand them better:

1. Direction

The main difference between a forward proxy and a reverse proxy is the direction of traffic flow. Forward proxy is used to forward traffic from a client to the internet, while a reverse proxy is used to forward traffic from the internet to a web server.

2. Client Access

With a forward proxy, clients have to be explicitly configured to use the proxy server. In contrast, a reverse proxy is transparent to the client, and clients can access the web server directly without needing to know the proxy server's address.

3. Load Balancing

Reverse proxies can distribute incoming requests across multiple servers to balance the load, while forward proxies cannot.

4. Caching

Forward proxies can cache frequently accessed resources to reduce the load on the web server and speed up the response time for subsequent requests. Reverse proxies can also cache resources, but the caching is typically done closer to the client to improve performance.

5. Security

A forward proxy can be used to protect a client's identity by hiding their IP address from the internet. A reverse proxy can be used to protect a server by hiding its identity and exposing a single IP address to the internet.

6. SSL/TLS Termination

A reverse proxy can terminate SSL/TLS connections on behalf of a web server to reduce the load on the server and simplify certificate management. Forward proxies typically do not terminate SSL/TLS connections.

7. Content Filtering

Forward proxies can be used to filter content, block access to specific websites, and enforce access policies. Reverse proxies can also perform content filtering, but this is typically done closer to the client to reduce the load on the web server.

8. Routing

Forward proxies can be used to route traffic to different servers based on predefined rules. Reverse proxies can also perform routing, but this is typically done based on the requested URL and other criteria.

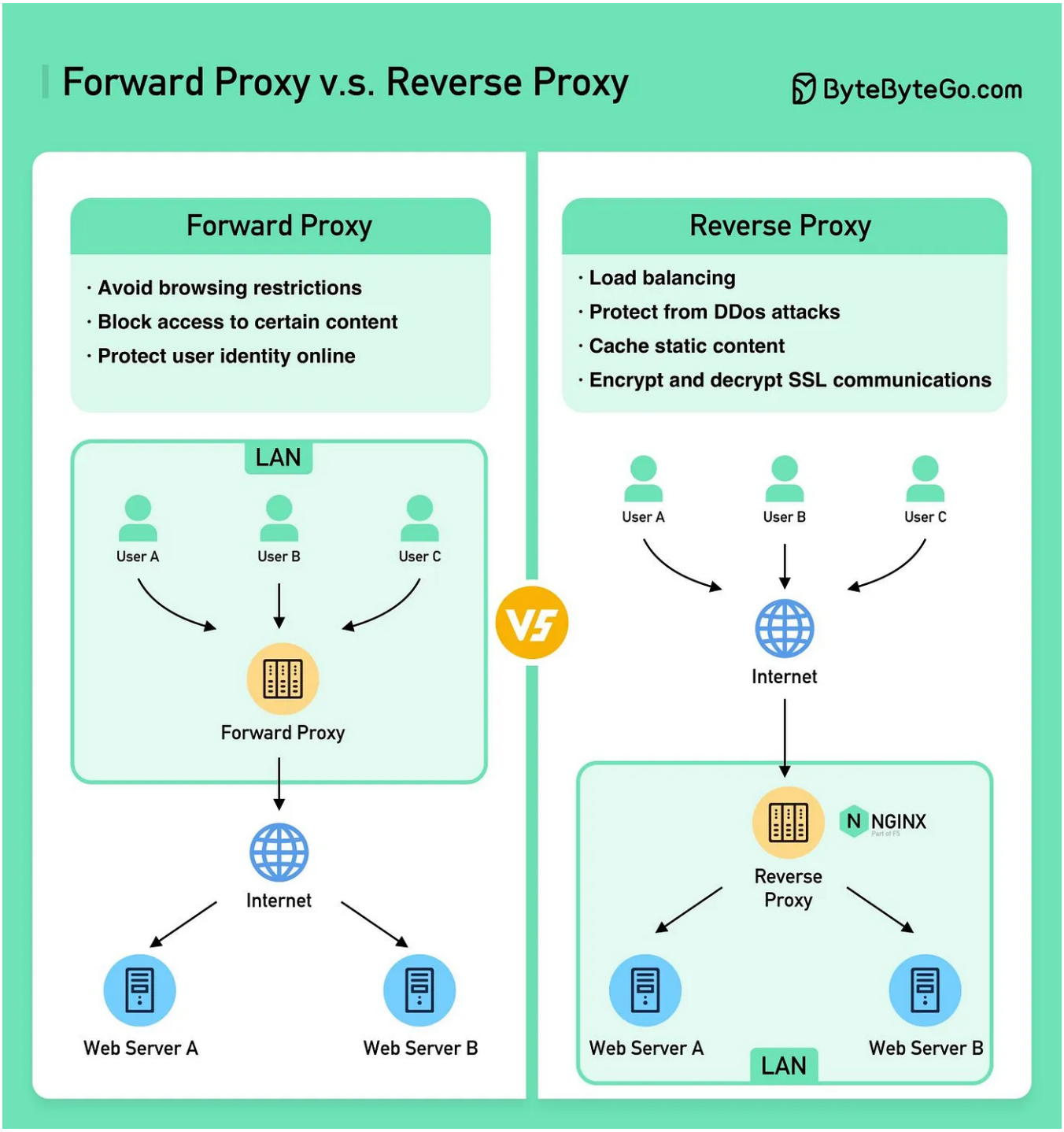
9. Scalability

Reverse proxies can be used to scale web applications horizontally by distributing traffic across multiple servers. Forward proxies do not provide this scalability feature.

10. Network Complexity

Forward proxies are relatively simple to set up and manage, while reverse proxies can be more complex due to their load balancing, SSL/TLS termination, and caching features.

Here is a nice diagram I found online which also highlight the difference between Forward and Reverse Proxy



image_credit — ByteByteGo.com

ByteByteGo is a great place to learn System Design, they also run a newsletter and have a YouTube channel, if you are preparing for System Design interview, you can also check that as a resource. Also, if you want to learn more about reverse proxy and forward proxy you can watch this YouTube video from their channel

Java and Spring Interview Preparation Material

Before any Java and Spring Developer interview, I always read the below two resources

Grokking the Java Interview

Grokking the Java Interview: click here

I have personally bought these books to speed up my preparation.

You can get your sample copy here, check the content of it and go for it

Grokking the Java Interview [Free Sample Copy]: click here

> *If you want to prepare for the Spring Boot interview you follow this consolidated ebook, it also contains microservice questions from spring boot interviews.*

Grokking the Spring Boot Interview

You can get your copy here — **Grokking the Spring Boot Interview**

Conclusion

That's all about difference between Strangler and BFF (Backend for Frontend Pattern) in Microservices. Understanding the nuances between the BFF (Backend for Frontend) and Strangler patterns in microservices architecture is crucial for building robust and scalable systems. While both patterns aim to address the challenges of microservices development, they have different use cases and approaches.

The BFF pattern focuses on providing tailored APIs for specific frontend applications, allowing for optimized communication and user experience. On the other hand, the Strangler pattern focuses on gradually migrating from a monolithic architecture to microservices by incrementally replacing functionality.

Both these patterns are also important from interview point of view and difference between BFF and Strangler is common Microservice interview question along with difference between API Gateway vs Load balancer.

By grasping the differences between BFF and Strangler patterns, software architects and developers can make informed decisions when designing and implementing microservices-based systems. It is important to carefully evaluate the requirements and constraints of the system and choose the appropriate pattern accordingly.

But, 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**

Other **Microservices Articles** you may like:

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

Difference between Synchronous vs Asynchronous Communication in Microservices

Choosing the Right Communication Pattern for Microservices Architecture: Exploring Synchronous and Asynchronous...

medium.com

Microservices Java Programming Development Software Engineering

Follow

Written by Soma

4.1K Followers · Editor for Javarevisited

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

More from Soma and Javarevisited

Implementation	implementation	an implementation	features
Persistence API	Hibernate provides its own API	Defines a set of standard APIs for ORM	Builds on JPA APIs, adds more functionality
Database Support	Supports various databases through dialects	Depends on the JPA implementation used	Depends on the JPA implementation used
Transaction Management	Provides its own transaction management	Depends on the JPA implementation used	Depends on the JPA implementation used
Query Language	Hibernate Query Language (HQL)	JPQL (Java Persistence Query Language)	JPQL (Java Persistence Query Language)
Caching	Provides first-level and second-level caching	Depends on the JPA implementation used	Depends on the JPA implementation used
Configuration	XML, annotations, or Java-based configuration	XML, annotations, or Java-based configuration	XML, annotations, or Java-based configuration
Integration	Can be used independently or with JPA	Works with any JPA-compliant implementation	Works with any JPA-compliant implementation

Soma in Javarevisited

Difference between Hibernate, JPA, and Spring Data JPA?

Hello folks, if you are preparing for Java Developer interviews then part from preparing Core Java, Spring Boot, and Microservices, you...

· 10 min read · May 26

253 1

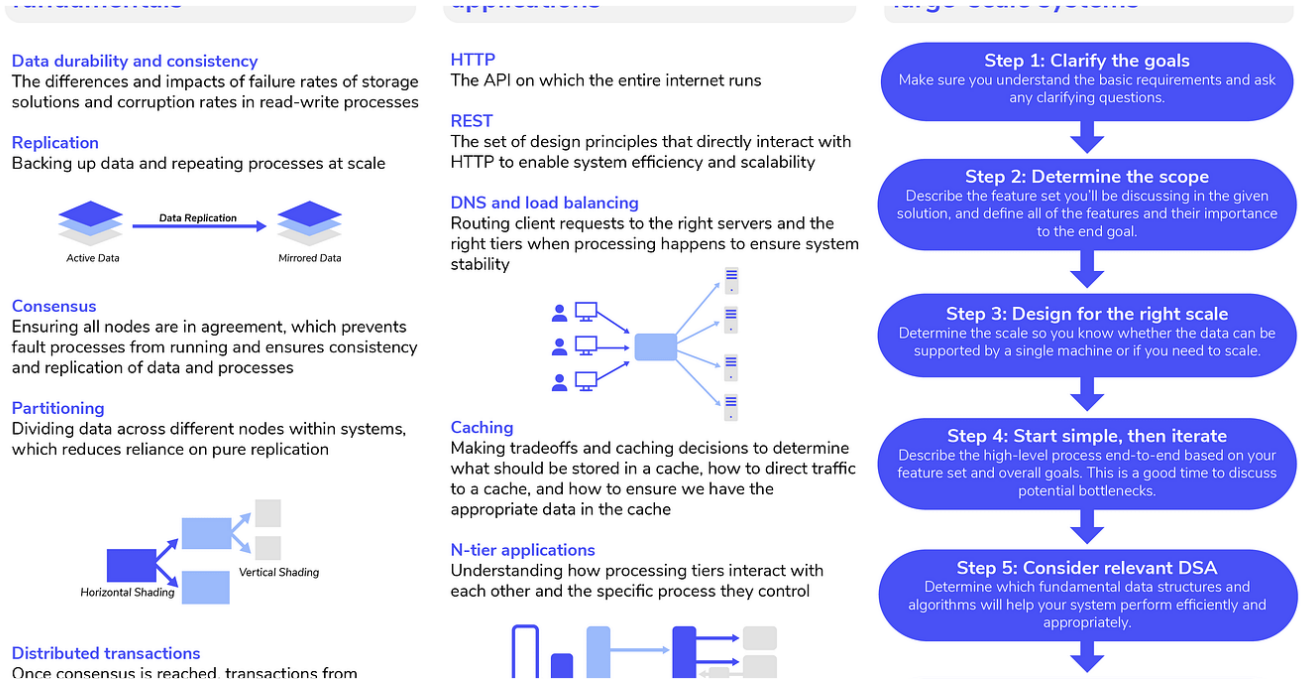
javinpaul in Javarevisited

Top 10 Websites to Learn Python Programming for FREE [2023]

Hello guys, if you are here then let me first congratulate you for making the right decision to learn Python programming language, the...

13 min read · Apr 22, 2020

328 6



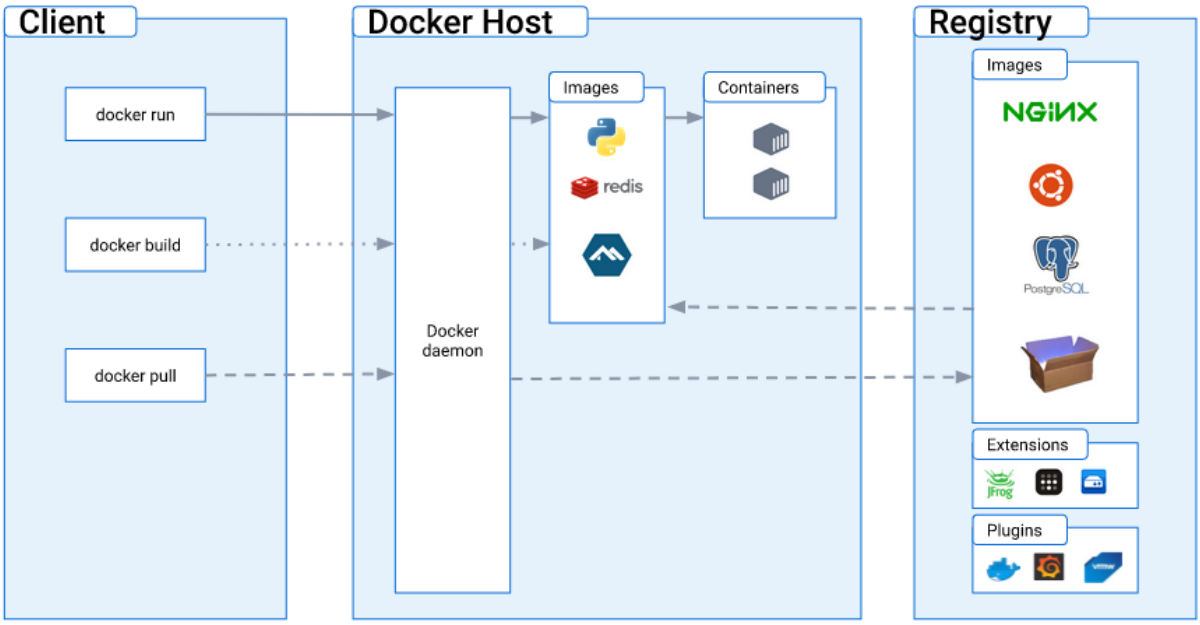
javinpaul in Javarevisited

Top 3 System Design Cheat Sheets for Developers

3 System Design Cheat Sheet you can print and put on your desktop to revise before Tech interviews

6 min read · Jul 19

204 1



Soma in Javarevisited

How Docker works internally? Magic Behind Containerization

Exploring the Inner Workings of Docker: Unveiling the Magic Behind Containerization

· 6 min read · Jun 30

184

See all from Soma

See all from Javarevisited

Recommended from Medium

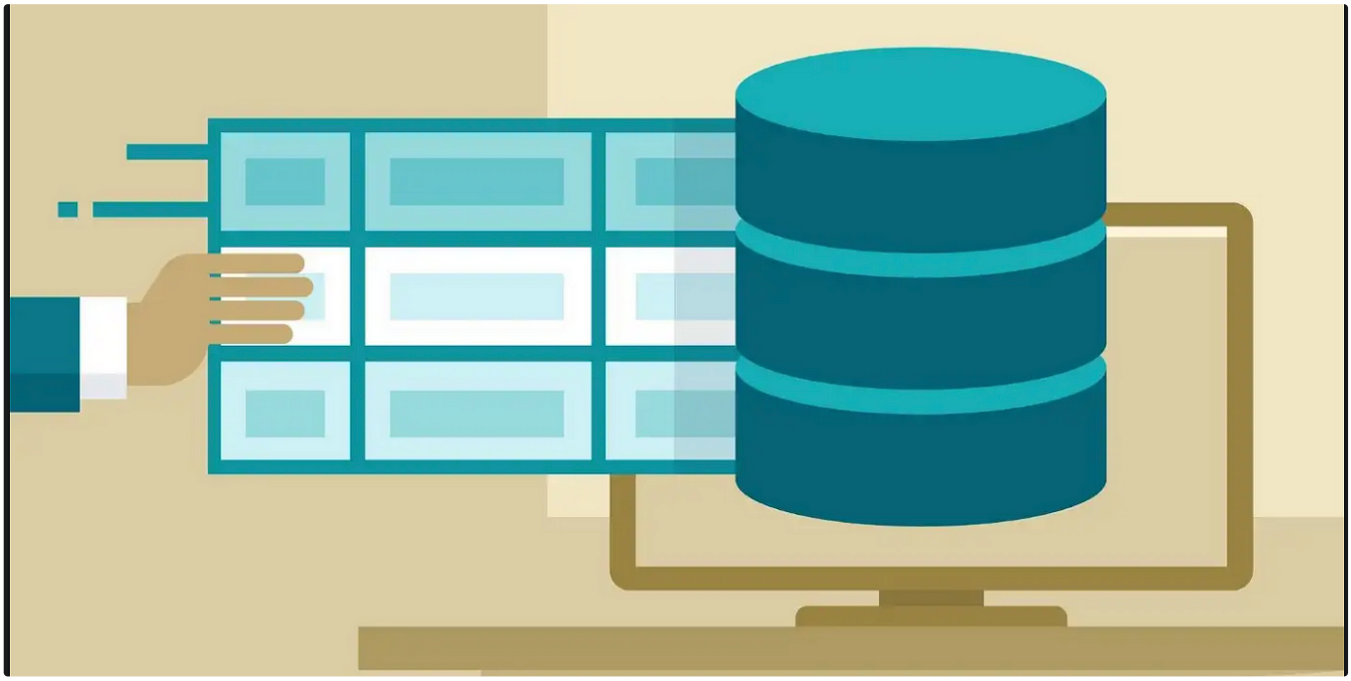
Anto Semeraro in Level Up Coding

Microservices Orchestration Best Practices and Tools

Optimizing microservices communication and coordination through orchestration pattern with an example in C#

· 10 min read · Mar 27

67



Keshav Gupta in Walmart Global Tech Blog

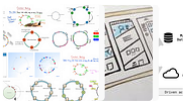
Event Sourcing Design Pattern

This blog will discuss the Event Sourcing design pattern and how this can be used in designing large-scale distributed systems.

5 min read · Mar 7

55

Lists

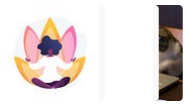


General Coding Knowledge

20 stories · 194 saves

It's never too late or early to start something

13 stories · 67 saves



Stories to Help You Grow as a Software Developer

19 stories · 264 saves

Leadership

35 stories · 91 saves

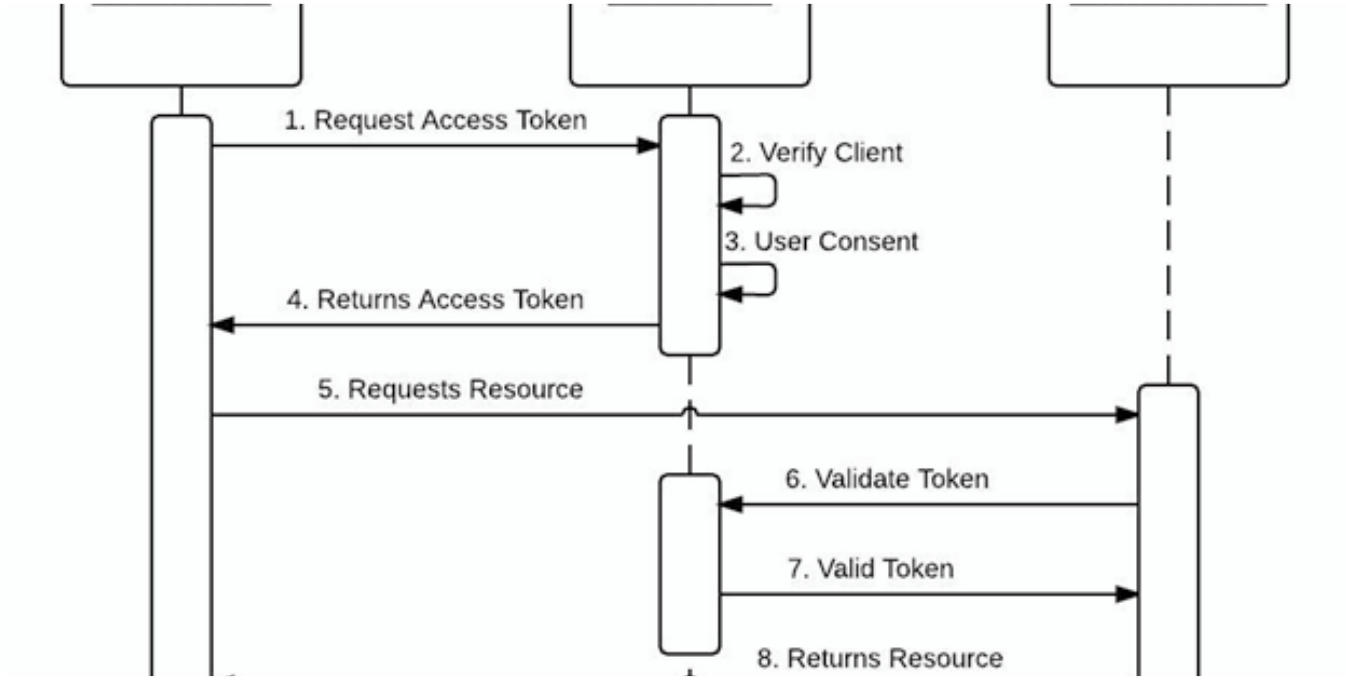
Sowmya.L.R

Docker—Container era (PART-I)

In earlier days people rarely used web apps. But in this digital era, every single task is done by any particular app. Ex grocery buying...

4 min read · 2 days ago

10



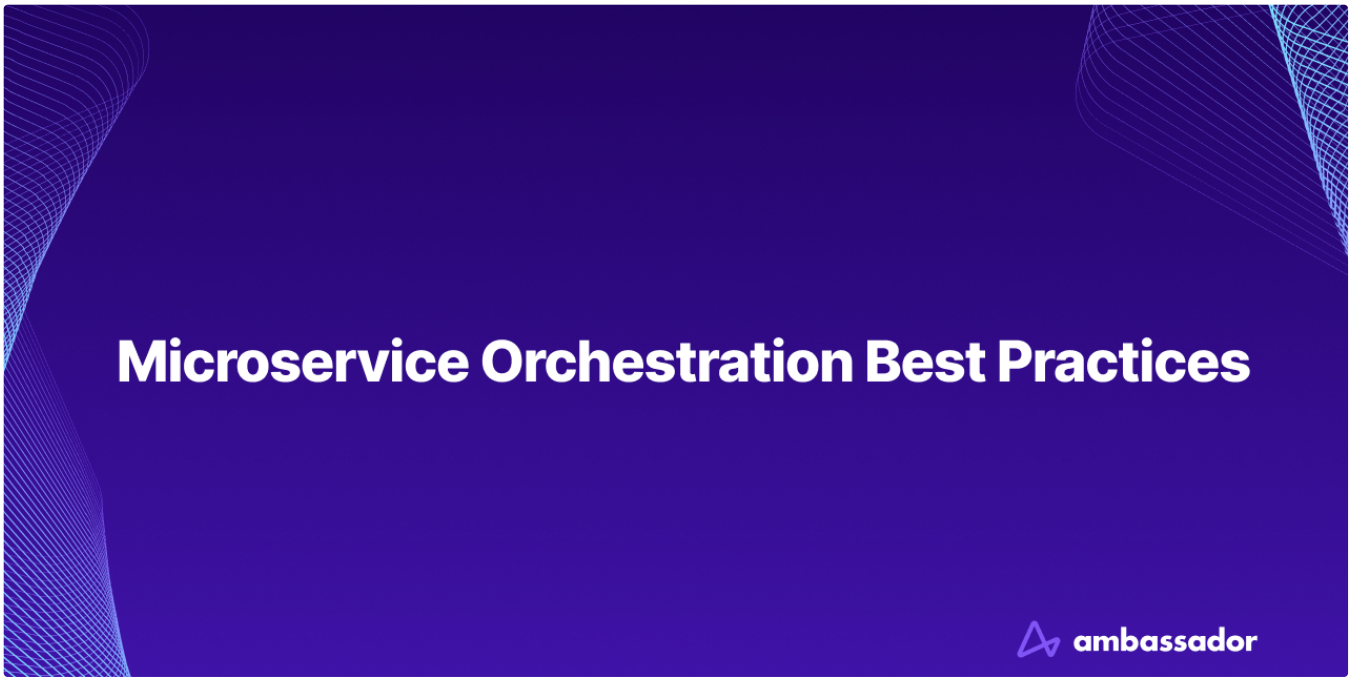
Somal Chakraborty in Dev Genius

Securing a Microservice with OAuth 2.0 (Part 2 : How Token Based Security Works Internally)

Table of content

5 min read · May 1

30 1



Ejiro Onose in Ambassador Labs

Microservice Orchestration Best Practices

In this article, you'll discover nine microservice orchestration best practices, the importance of orchestration, and more ...

8 min read · May 4

265 3

Muhammad Taha

Designing Modern Applications: Chapter 1—Genesis

A guide on how to build software, from understanding requirements to architectural design to clean code

7 min read · Jul 6

 21