Service Mesh for Microservices

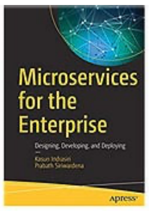
Kasun Indrasiri Follow 6 min read · Sep 15, 2017

Microservices architecture has been evolving a lot during the last couple of years and there are quite a few new concepts and patterns are emerging. 'Service Mesh' concept is getting quite popular. In this post, I'm planning to cover the key concepts related to Service Mesh and how it is used in real-world microservices implementations.

Microservices for the Enterprise: Designing, Developing, and Deploying

Microservices for the Enterprise: Designing, Developing, and Deploying [Kasun Indrasiri, Prabath Siriwardena] on...

www.amazon.com



Why 'Service Mesh'?

As with many emerging technologies, there was a lot of hype around the Microservices Architecture. Most people think that microservices is the answer to all the problems they had with previous architecture such as SOA/ESB. However, when we observe the real world microservices implementations, we can see that most of the functionalities that a centralized bus (ESB) supports are now implemented at microservices level. So, we are more or less solving the same set of fundamental problems, but we are solving them at different dimensions with Microservices.

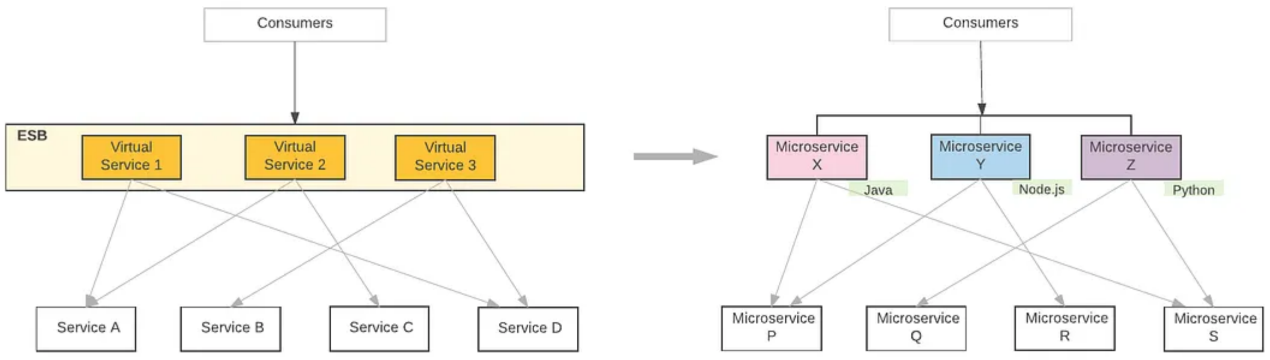


Figure 1: From centralized integration/ESB to Microservices

For example, let's take a scenario where you need to call multiple downstream services in resilient manner and expose the functionality as a another (composite) service. As shown in figure 1, with the ESB architecture, you can easily leverage the inbuilt capabilities of ESB, for building virtual/composite services and functionalities such as circuit breakers, timeouts and service discovery etc., which are useful during inter-service communication.

When you implement the same scenario using Microservices, then you no longer have a centralized integration/ESB layer but a set of (composite and atomic) microservices. So, you have to implement all these functionalities at the microservices level.

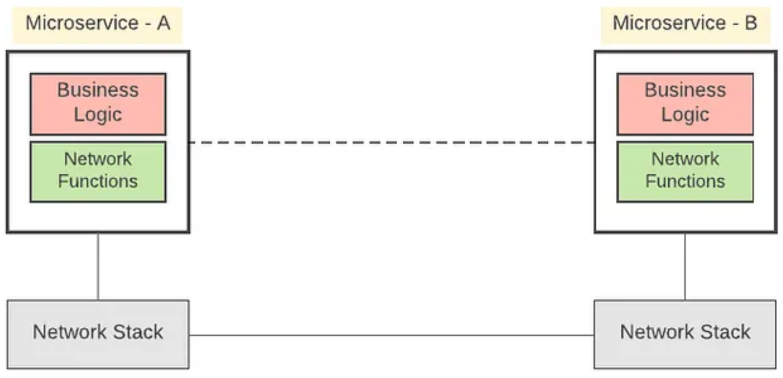


Figure 2: Microservice components and service-to-service communication

Therefore a given microservice which communicates with other services(figure 2), comprises of:

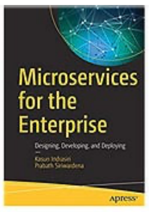
- **Business Logic** that implements the business functionalities, computations and service composition/integration logic.
- **Network Functions** that take care of the inter-service communication mechanisms (basic service invocation through a given protocol, apply resiliency and stability patterns, service discovery etc.) These network functions are built on top of the underlying OS level network stack.

Now think about the effort involved in implementing such microservice. Implementing the functionalities related service-to-service communication from scratch is a nightmare. Rather focusing on the business logic, you will have to spend a lot of time on building service-to-service communication functionalities. And this is even worse if you use multiple technologies to build microservices (such as multiple programming languages as shown in figure 1), because you need to duplicate the same efforts across different languages (e.g. Circuit breaker has to be implemented on Java, Node, Python etc.).

Microservices for the Enterprise: Designing, Developing, and Deploying

Microservices for the Enterprise: Designing, Developing, and Deploying [Kasun Indrasiri, Prabath Siriwardena] on...

www.amazon.com



> The most complex challenge in realizing microservice architecture is not building the services themselves, but the communication between services.

Top highlight

Since most of the inter-service communication requirements are quite generic across all microservices implementations, we can think about offloading all such tasks to a different layer, so that we can keep the service code independent. That's where 'service mesh' comes into the picture.

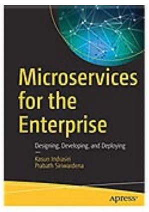
What is a 'Service Mesh'?

In a nutshell, a Service Mesh is an inter-service communication infrastructure.

Microservices for the Enterprise: Designing, Developing, and Deploying

Microservices for the Enterprise: Designing, Developing, and Deploying [Kasun Indrasiri, Prabath Siriwardena] on...

www.amazon.com



With a service mesh,

- A given Microservice won't directly communicate with the other microservices.
- Rather all service-to-service communications will take places on-top of a software component called service mesh (or side-car proxy).
- Service Mesh provides built-in support for some network functions such as resiliency, service discovery etc.
- Therefore, service developers can focus more on the business logic while most of the work related to network communication is offloaded to the service mesh.
- For instance, you don't need to worry about circuit breaking when your microservice call another service anymore. That already comes as part of service mesh.
- Service-mesh is language agnostic: Since the microservice to service mesh proxy communication is always on top to standard protocols such as HTTP1.x/2.x, gRPC etc., you can write your microservice from any technology and they will still work with the service mesh.

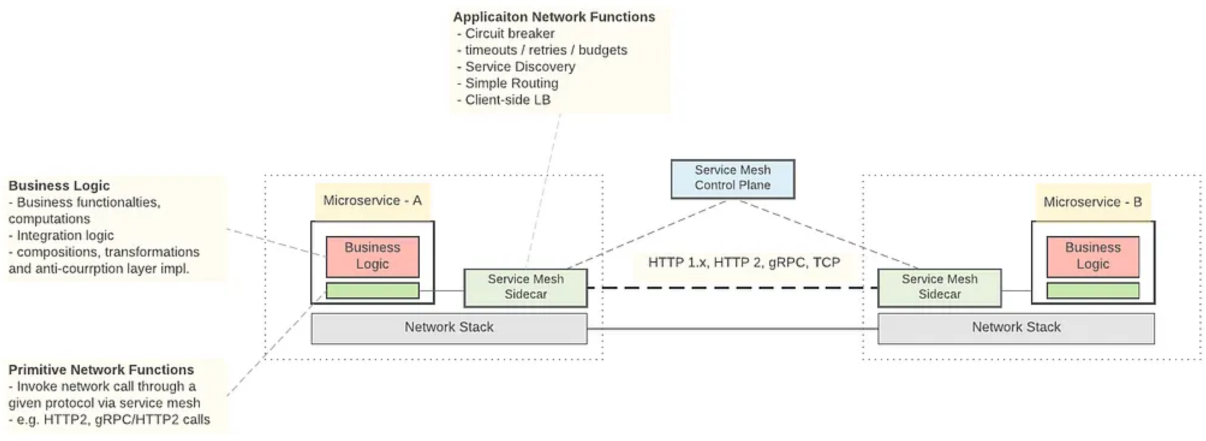

Applicaiton Network Functions
- Circuit breaker
- timeouts / retries / budgets
- Service Discovery
- Simple Routing
- Client-side LB
Business Logic
- Business functionalties, computations
- Integration logic
- compositions, transformations and anti-courrption layer impl.
Primitive Network Functions
- Invoke network call through a given protocol via service mesh
- e.g. HTTP2, gRPC/HTTP2 calls
Microservice - A
Business Logic
Service Mesh Sidecar
Network Stack
Service Mesh Control Plane
HTTP 1.x, HTTP 2, gRPC, TCP
Microservice - B
Business Logic
Service Mesh Sidecar
Network Stack

Figure 3: Service to service communication with Service Mesh

Let’s try to further understand the service interactions and responsibilities which are shown in figure 3.

Business Logic

The service implementation should contain the realization of the business functionalities of a given service. This includes logic related to it’s business functions, computations, integration with other services/systems(including legacy, proprietary and SaaS) or service compositions, complex routing logics, mapping logic between different message types etc.

Primitive Network Functions

Although we offload most of the network functions to service mesh, a given service must contain the basic high-level network interactions to connect with the service mesh/side-car proxy. Hence, a given service implementation will have to use a given network library(unlike the ESB world, where you just have to use a very simple abstraction)to initiate network calls (to service mesh only). In most cases, microservices development framework embed the required network libraries to be used for these functions.

Application Network Functions

There are application functionalities which tightly coupled to the network, such as circuit breaking, timeouts, service discovery etc. Those are explicitly separated from the service code/business logic, and service mesh facilitate those functionalities out of the box.

> *Most of the initial microservices implementations simply ignore the gravity of the network functions offered from a central ESB layer, and they implemented all such functionalities from scratch at each microservice level. Now they have started*

> *realizing the importance of having a similar shared functionality as a distributed mesh.*

Service Mesh Control Plane

All service mesh proxies are centrally managed by a control pane. This is quite useful when supporting service mesh capabilities such as access control, observability, service discovery etc.

Functionalities of a Service Mesh

As we have seen earlier, the service mesh offers a set of application network functions while some (primitive) network functions are still implemented the microservices level itself. There is no hard and fast rule on what functionalities should be offered from a service mesh. These are the most common features offered from a service mesh.

- **Resiliency for inter-service communications**: Circuit-breaking, retries and timeouts, fault injection, fault handling, load balancing and failover.
- **Service Discovery:** Discovery of service endpoints through a dedicated service registry.
- **Routing**: Primitive routing capabilities, but no routing logics related to the business functionality of the service.
- **Observability**: Metrics, monitoring, distributed logging, distributed tracing.
- **Security**: Transport level security (TLS) and key management.
- **Access Control:** Simple blacklist and whitelist based access control.
- **Deployment**: Native support for containers. Docker and Kubernetes.
- **Interservice communication protocols**: HTTP1.x, HTTP2, gRPC

Service Mesh Implementations

Linkerd and Istio are two popular open source service mesh implementations. They both follow a similar architecture, but different implementation mechanisms. You can find a very good comparison between these two service mesh implementations at [1].

Service Mesh — Pros and Cons

Let's have a quick look at the pros and cons of service meshes.

Pros

- Commodity features are implemented outside microservice code and they are reusable.
- Solves most of the problems in Microservices architecture which we used to have ad-hoc solutions: Distributed tracing, logging, security, access control etc.

- More freedom when it comes to selecting a microservices implementation language: You don't need to worry about whether a given language supports or has libraries to build network application functions.

Cons

- *Complexity*: Having a service mesh drastically increase the number of runtime instances that you have in a given microservice implementation.
- *Adding extra hops*: Each service call has to go through an extra hop(through service mesh sidecar proxy).
- *Service Meshes address a subset of problems*: Service mesh only address a subset of inter-service communication problems, but there are a lot of complex problems such as complex routing, transformation/type mapping, integrating with other services and systems, to be solved at your microservice's business logic.
- *Immature*: Service mesh technologies are relatively new to be declared as full production ready for large-scale deployments.

Conclusion

In summary, service mesh addresses some of the key challenges when it comes to the realization of microservice architecture. It gives you more freedom to select a diverse set of microservices implementation technologies as well as to focus more on business logic rather investing more time on network functions between services. However, service mesh won't solve any of the business logic related or service integration/composition related problems.

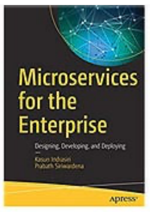
Update: **"Service Mesh vs API Gateway"?**

Please check the articles API Gateways vs Service Mesh for more further information on this.

Microservices for the Enterprise: Designing, Developing, and Deploying

Microservices for the Enterprise: Designing, Developing, and Deploying [Kasun Indrasiri, Prabath Siriwardena] on...

www.amazon.com


Microservices for the Enterprise
Designing, Developing, and Deploying
Kasun Indrasiri
Prabath Siriwardena
Apress

Microservices Circuit Breaker Service Discovery Distributed Tracing Esb

Published in Microservices in Practice

Follow

1.7K followers · Last published Oct 9, 2017

Microservices in Practice

Written by Kasun Indrasiri

Follow

1.7K followers · 82 following

Sr. Product Manager at Confluent, Ex-Product Manager of Azure Event Hubs, O'Reilly Author of gRPC & Cloud Native Patterns, Speaker @KubeCon

Responses (17)

I Ishu

What are your thoughts?

Alex Ivy
Oct 7, 2017

Great article Kasun, however it's not very clear how service mesh differs from API Gateway, it's seems they both have a common functionality, so can we use both or it one way or another ?

11 Hide replies Reply

> Kasun Indrasiri Author
> Oct 10, 2017
>
> Please check https://medium.com/microservices-in-practice/service-mesh-vs-api-gateway-a6d814b9bf56. I tried addressing your concerns.
>
> 5 Reply

Aschwin Wesselius
Aug 1, 2019

Thank you for the article Kasun. How long did it take this pattern to evolve?

If I open up the book "Programming WCF Services" (4th edition, but shouldn't matter from earlier editions), by Juval Löwy, I see on page 4 and 5 similar diagram as what you... more

3 Reply

Arash Rahimi
Mar 8, 2021

great article Kasun

Reply

See all responses

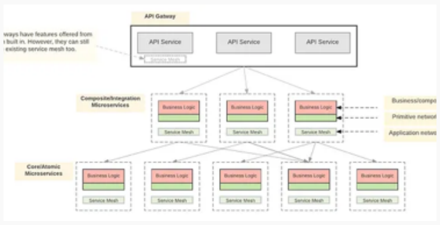
More from Kasun Indrasiri and Microservices in Practice

Kasun Indrasiri

Understanding RAFT Distributed Consensus

Distributed Consensus is one of the hardest problems in the distributed computing...

Apr 5, 2020 421 3

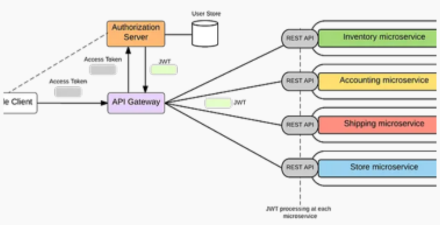


In Microservices in Practice by Kasun Indrasiri

Service Mesh vs API Gateway

In one of my previous articles on service mesh, there were a couple of questions...

Oct 10, 2017 1.7K 11

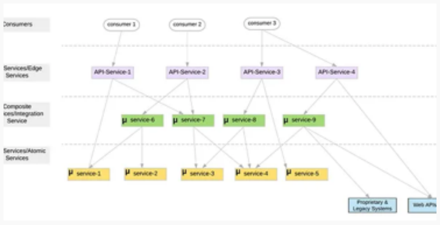


In Microservices in Practice by Kasun Indrasiri

Pragmatic Microservices

Nowadays, Microservices is one of the most popular buzz-words in the field of software...

Aug 29, 2017 2.2K 12



In Microservices in Practice by Kasun Indrasiri

Microservices Layered Architecture

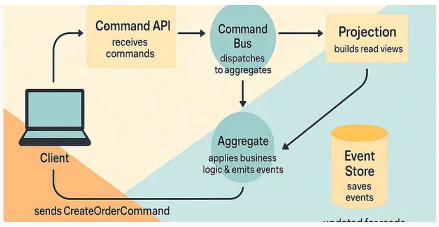
With Microservices architecture, a single software application/functionality is...

Sep 15, 2017 676 6

See all from Kasun Indrasiri

See all from Microservices in Practice

Recommended from Medium


Command API receives commands
Command Bus dispatches to aggregates
Projection builds read views
Aggregate applies business logic & emits events
Event Store saves events
Client
sends CreateOrderCommand

Rahul Kumar

Command Query Responsibility Segregation (CQRS) with Event...

Modern enterprise systems often face challenges in scaling read and write...

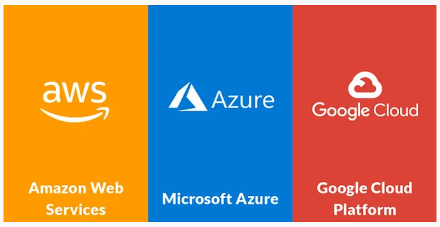
May 21

Indrajit

WebFlux vs Virtual Threads: I Hit Both With 100K Requests—One...

100K requests against WebFlux and Virtual Threads. One dominated, one disappointed....

Aug 17 7 1

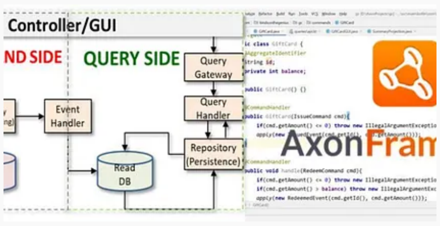


Matías Salinas

GCP Pub/Sub v/s Amazon SQS v/s Azure Service Bus

As cloud computing continues to gain popularity, more and more businesses are...

May 1 38



Anh Trần Tuấn

Event Sourcing in Microservices with Axon Framework

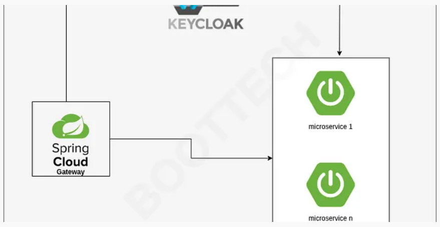
May 6 2

Ujjawal Rohra

5 Spring Boot Patterns That Separate Senior Developers From...

Let me ask you something honest

Jul 5 363 6



Eric Anicet

Spring Cloud Gateway OpenID Connect with Keycloak

In this story, we'll explore how to secure microservices architectures with Spring...

Jun 23 7

See more recommendations