**Service Mesh Architecture in a Nutshell**

Circuit Breaking, Rate Limiting, Conditional Rate Limiting, Traffic Shifting



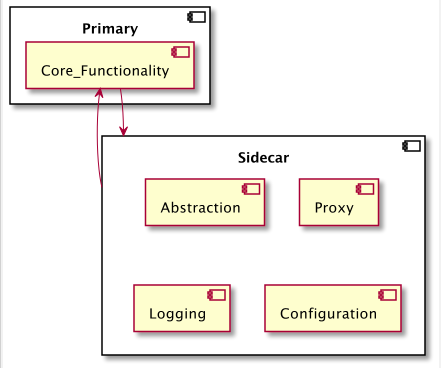
Photo by [Ricardo Gomez Angel](https://unsplash.com/es/@rgaleriacom?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Performance concerns are driving forces for software architects to come up with new techniques for more efficient software performance. The performance is also hot topic for teams applying microservices. The extreme popularity of microservices has led to evaluation of service mesh pattern/architecture (whatever you want to call this…).

*In software architecture, a****service mesh****is a dedicated infrastructure layer for facilitating service-to-service communications between services or microservices, using a proxy.*

The main idea is to isolate application-independent cross-cutting concerns such as communication, monitoring, security, authentication/authorization etc. from core business logic. This kind of dedicated infrastructure layer adds value to low latency, configurability.

Service Mesh is implemented typically through array of proxies, and sidecar pattern. Sidecar/Sidekick pattern uses a same languages and libraries and service is needed that shares the lifecycle but can be independently deployed. The functionalities like logging, configuration etc. can be abstracted away to another microservice, as shown in example below. This pattern has 1:1 relation with Primary Service.



Sidecar/Sidekick Pattern

For more information about cloud-native patterns please check my other blog post:

[Cloud-Native Architecture Patterns](https://azeynalli1990.medium.com/cloud-native-architecture-patterns-60a010d90978)

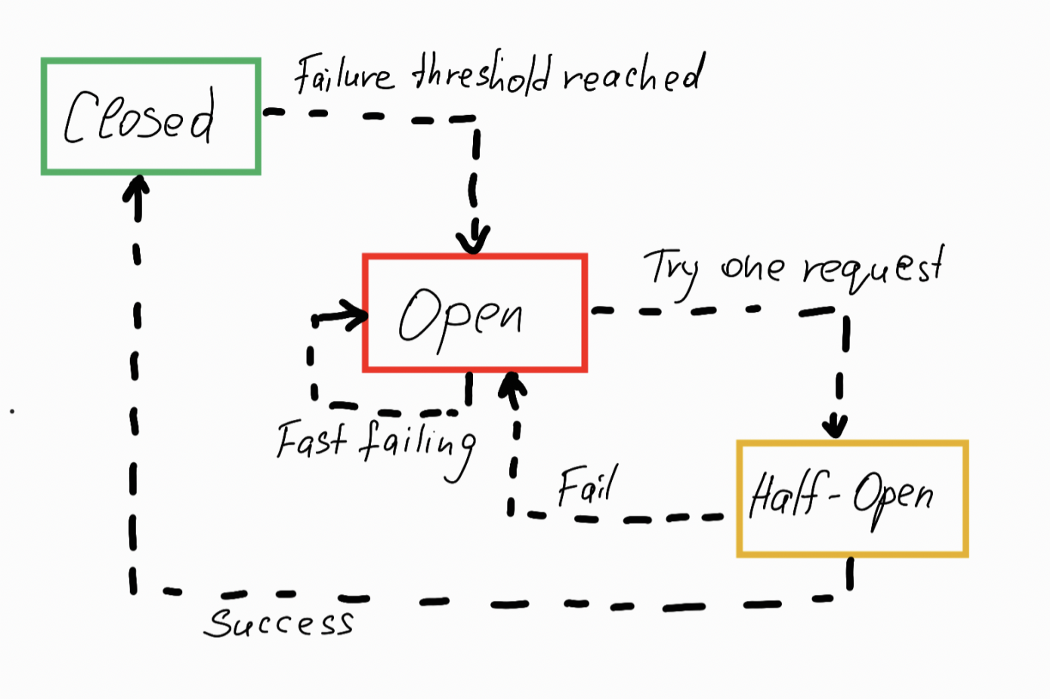
**Basic Tactics**

Beside Authentication/Autorisation, Service Discovery, Service Mesh Pattern provides some other critical capabilities such as:

* Circuit Breaking
* Rate Limiting
* Conditional Rate Limiting
* Traffic Shifting

*Circuit Breaker…*

Circuit Breaker is a popular tactics that prevent service of network failure from cascading. By applying a failure threshold it avoids redundant calls to failed service, and responses back from back up service or default values.



Circuit Breaker

*Rate Limiting…*

This functionality defends service from overuse by limiting requests firstly in given period of time by given user, secondly by limiting concurrent requests and lastly denying further calls if threshold is reached.

*Conditional Rate Limiting…*

This functionality is also rate limiting but on conditions:

* Limiting requests based on business requirement such as enforcing SLA, Contracts.
* Limiting by quote

*Traffic Shifting…*

This functionality is very handy when migrating from old version of microservice to newer one. It gradually shifts traffic from old service call to new service call.

**Pros and Cons**

*Benefits…*

* service mesh should be deployed to same processor as the microservice itself allowing to cut down communication time
* separation of cross-cutting concerns from functionality gives the developers more freedom

*Tradeoff…*

* deployment of service mesh to same operational unit same as the microservice itself might overload the processor being used
* service mesh brings extra layer of architecture which might sometime overhead for small projects