Service Mesh:

Why we need Service Mesh?

While building microservices or converting monolithic application to microservices, developers must take care of service discovery, load balancing, fault tolerance, distributed tracing, telemetry, and security while inter service communication.

What is Service Mesh?

A service mesh is a dedicated infrastructure layer built into an application that controls service-to-service communication in a microservices architecture. Provides a whole bunch of capabilities to relieve the developers from handling delivery of service requests to other services, performs load balancing, encrypts data, and discovers other services.

Service Mesh Architectures:

Following are the type of service mesh architectures

**Library**:

Each of the microservices carries a copy of the library that contains all the desired service mesh functions

The library architecture approaches were

1. The first architecture to be adopted
2. Simplest method to implement
3. Performance is low
4. More difficult to maintain

Example: Netflix Hystrix, Netflix Ribbon, Twitter Finagle

Advantage:

1. Resource are locally accounted for each and every service
2. Self-service adoption for developers

Disadvantages:

1. Strong coupling is a significant drawback
2. Non-uniform upgrades are challenging in large environment

**Node Agent**:

The Node agent architecture is easier to manage and maintain than the library architecture, it distributes one copy of the configuration to each node, rather than one copy of the configuration to each pod on each node

In the Node Agent service mesh architecture, a separate agent is running on every worker node of cluster:

1. Node Agent, usually running in user process space service the heterogenous mix of workloads, hosted on worker node
2. The node agent architectural model emphasizes work resource sharing, this method can provide more efficiency than the library model.

Example of Node Architecture: Linkerd, Consul

Advantages:

1. Less overhead for things that could be shared across a node
2. Easier to scale distribution of configuration information that it is with sidecar proxies
3. This model is useful for deployments that are primarily physical or virtual server based.

Disadvantage:

1. Coarse support for encryption of service-to-service communication, instead host-to-host encryption, and authentication policies
2. Blast radius of a proxy failure includes all applications on the node which is essentially equivalent to losing the node itself
3. Not a transparent entity services must be aware of its existence

**Sidecar**:

Sidecar is the latest method developed for service meshing; the sidecar service mesh deploys one adjacent container for every application container.

1. The sidecar handles all the network traffic in and out of the application container
2. To eliminate the potential for a network-based attack, the sidecar has the same privilege as the application to which it is attached
3. Most Sidecar implications founded on security best practices, limit the scope of the authorities necessary to complete the required intercommunication and then end their own process
4. The sidecar acts in closely secured proximity to the application almost like a function call from a library rather than having to traverse the network to an external Node Agent for each intercommunication

Examples of Sidecar Architecture: Istio, Aspen Mesh

Advantages:

1. Granular encryption of service-to-service communication
2. Can be gradually added to an existing cluster without central coordination
3. App-to-sidecar communication is easier to secure than app-to-node proxy
4. Resource consumed for a service are attributed to that service
5. Blast radius of a proxy failure is limited to the sidecar app

Disadvantages:

1. Lack of central coordination. Difficult to scale operationally

Features of Service Mesh:

Traffic Management:

Fundamental feature of service mesh is traffic management; this includes dynamic service discovery and routing. It also enables some interesting use cases like traffic shadowing and traffic splitting. Service mesh can provide retries, timeouts, rate-limiting and circuit breaking. This out of box failure recovery features make the communication more reliable.

Security:

Service mesh handles the security aspect of the service-to-service communication, this includes enforcing traffic encryption through mutual TLS, provides authentication through certificate validation and ensuring authorization through access policies.

Observability:

Robust observability is the under-pinning requirement for handling the complexity of distributed system. A service mesh can generate a lot of metrics like latency, traffic, errors, and saturation. Service mesh can also generate access logs providing full record for each request

Limitations of Service Mesh:

1. The introduction of proxies, sidecars and other components into an already sophisticated environment increases the complexity of development and operations
2. Service meshes are an invasive and intricate technology that can add significant slowness to an architecture
3. Adding a service mesh such as Istio on top of an orchestrator such as Kubernetes often requires operators to become experts in both technologies
4. The invasiveness of service meshes force both developers and operators to adapt to a highly opinionated platform and conform to its rules

Following are the service mesh available in market

Open Source:

1. Envoy
2. Istio
3. Linkerd
4. Linkerd2

Commercial:

1. Consul
2. Aspen Mesh
3. Kong Enterprise Mesh
4. AWS App Mesh