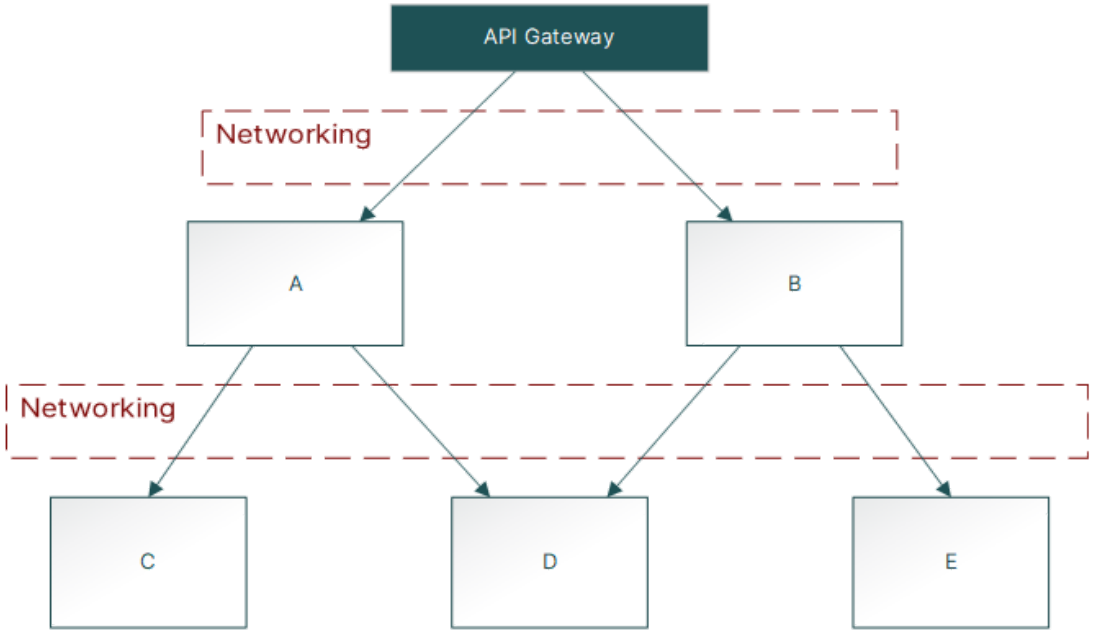
Introduction to Service Mesh on Kubernetes

Tran Trung Nguyen
Software Engineer
VNPAY Cloud

Microservices

are essentially distributed systems, often needing:

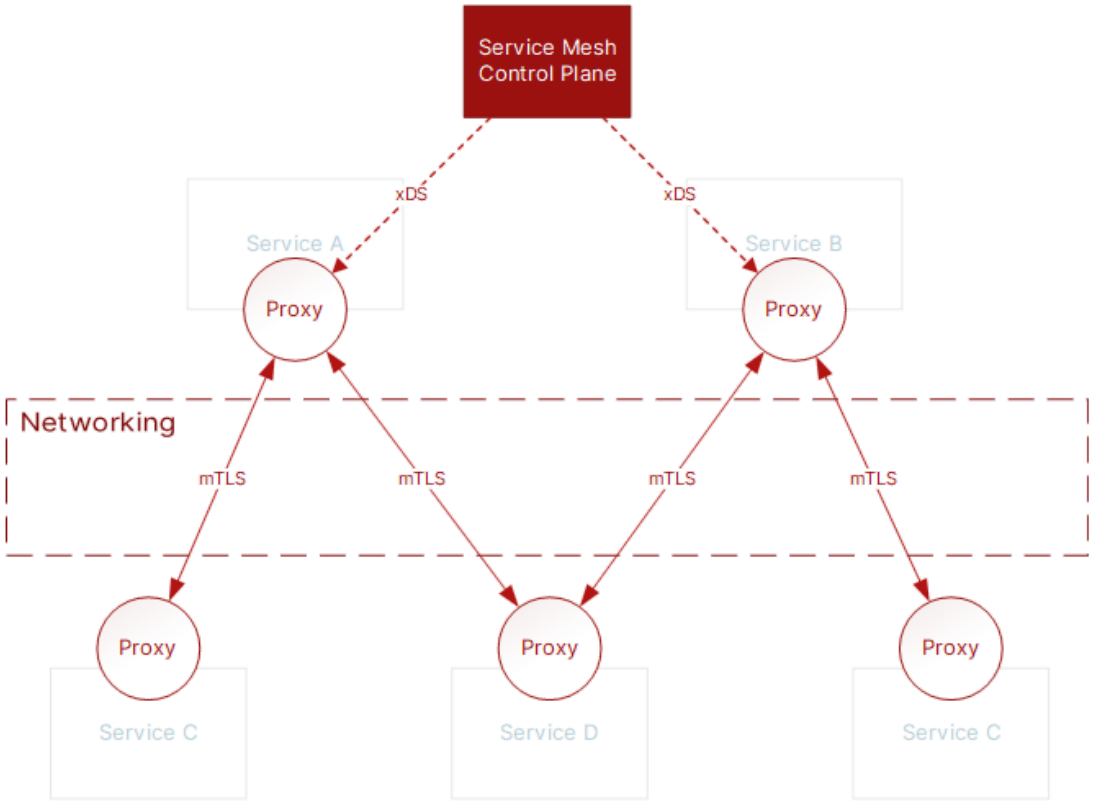
- Routing and Discovery
- Failure Handling *(Retries, Circuit Breaking, Timeout)*
- Encryption
- Observability *(Logging, Tracing, Metrics)*



Service Mesh

Networking between microservices is considered a dedicated infrastructure layer, called *service mesh*

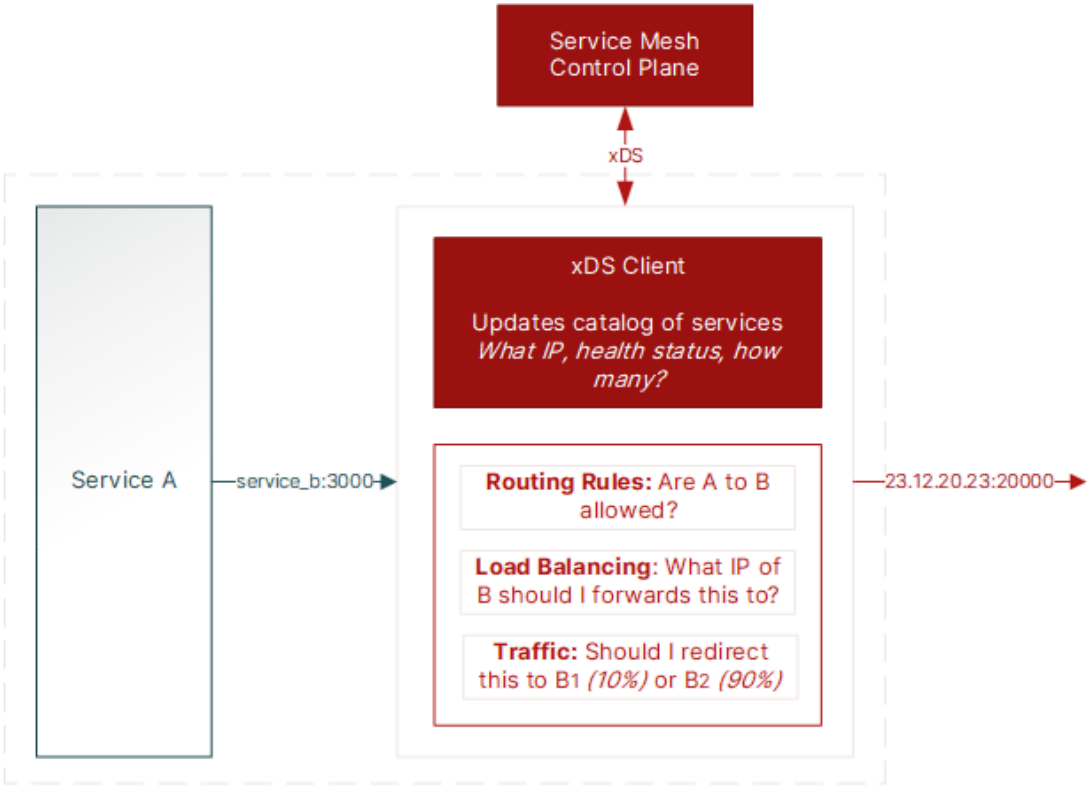
- *Mesh*: Discover and connects services
- Uses sidecar proxies to intercept and forwards communication



Routing and Discovery

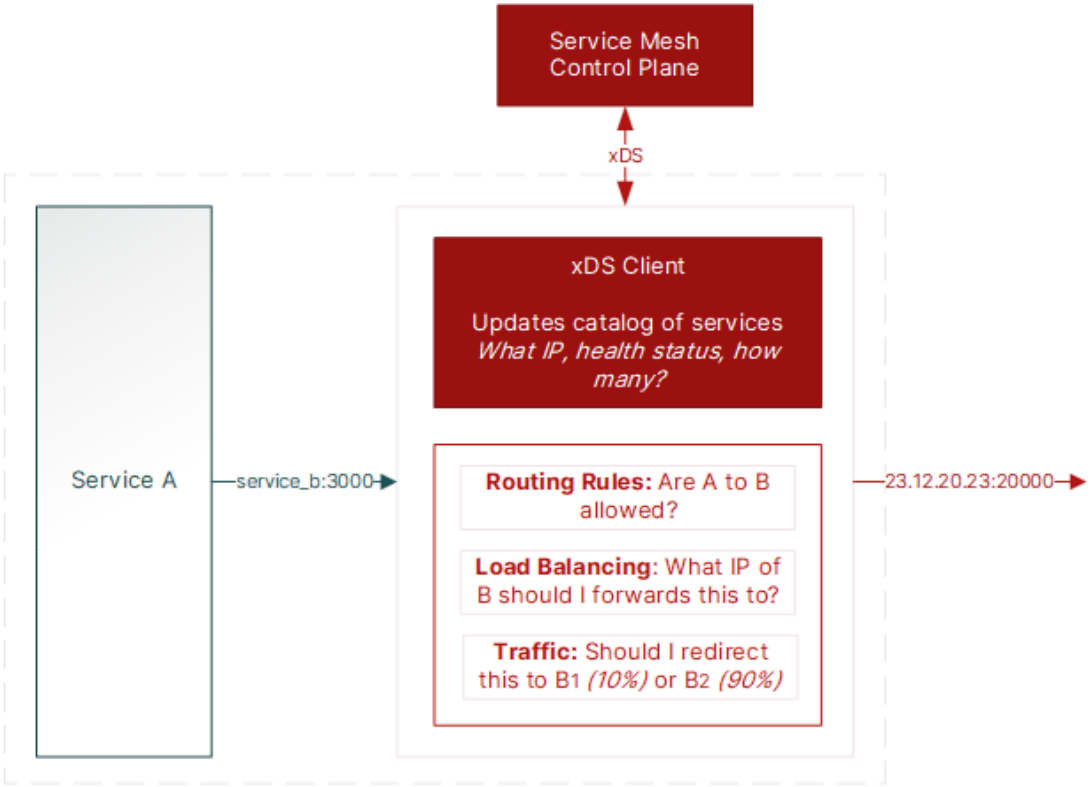
The application need only the name of other services. The *service mesh* do:

- *Discovery*: Updates from and to the control plane the status, IP address, etc. of the services
- *Routing Rules*: Service A is allowed to talk to B through gRPC, HTTP GETs



Routing and Discovery

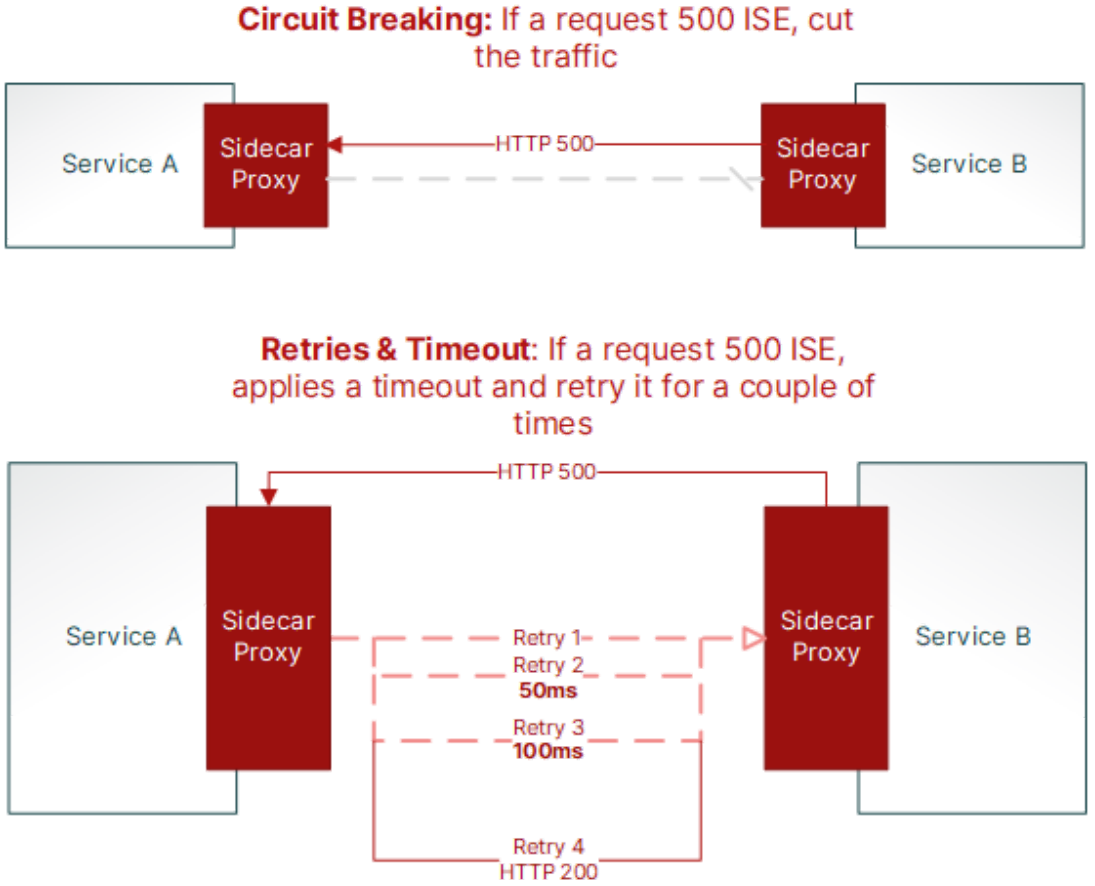
- *Load Balancing*: Kubernetes Service API randomly selects a Pod. Proxies often provides **round-robin, random, weighted, least request,...**
- *Traffic Splitting*: Great for A/B Testing. Can use probability or HTTP headers, paths, etc.



Failure Handling

Often these features are built into application code. The *service mesh* can **automatically** do:

- *Circuit Breaking*: If a request to a service fails, stop the connection and gradually open it again until that service is back
- *Retries & Timeout*

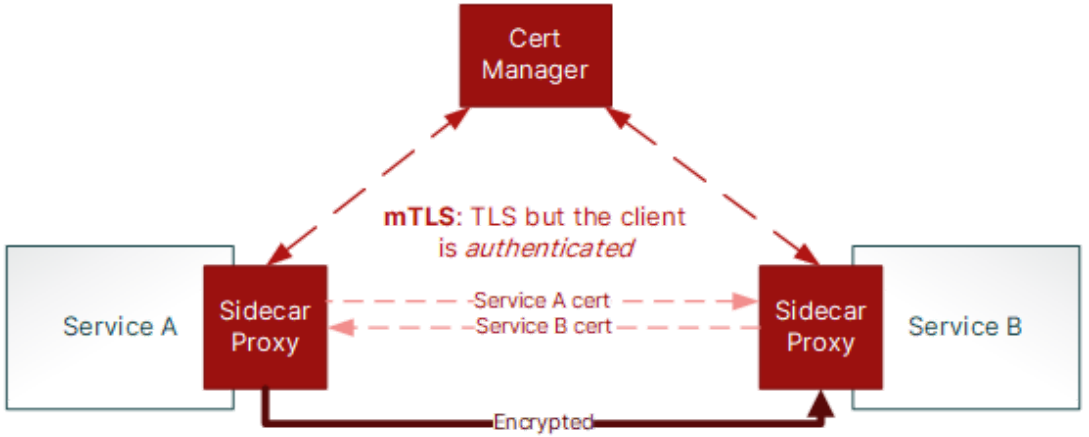


Encryption

Either implement encryption from application-level code with **TLS** or use service mesh

- Service meshes often use **mTLS**, a form of TLS where the client also sends a client certificate
- Server checks the certificate identity and decide to accept handshake or not

> Why not just use regular TLS?

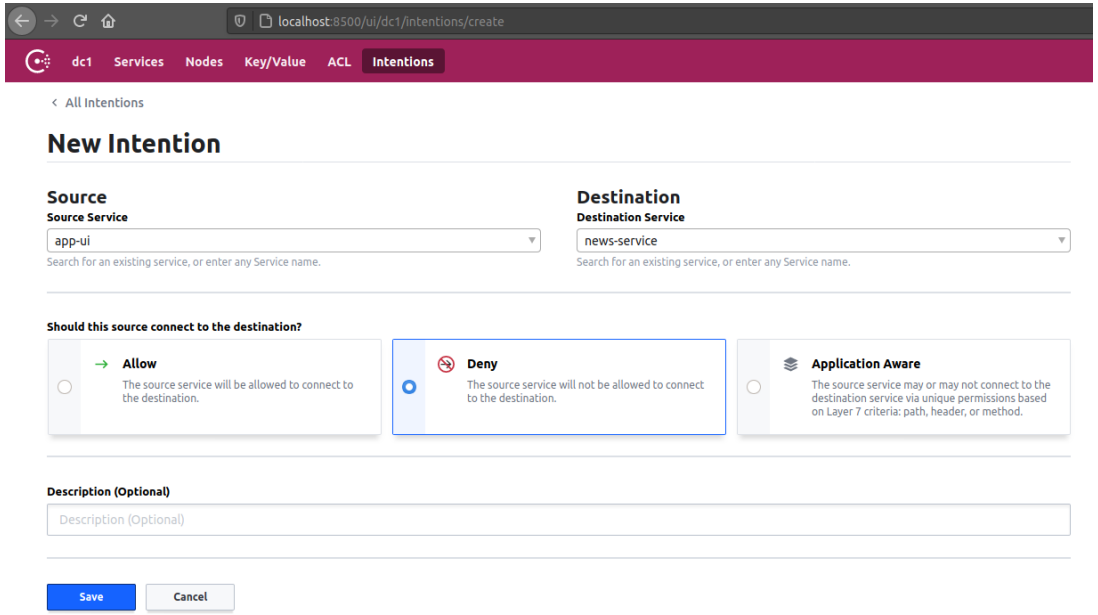


Control

> **mTLS**: Sometimes it is useful to know who the client is

Knowing the client can help service mesh:

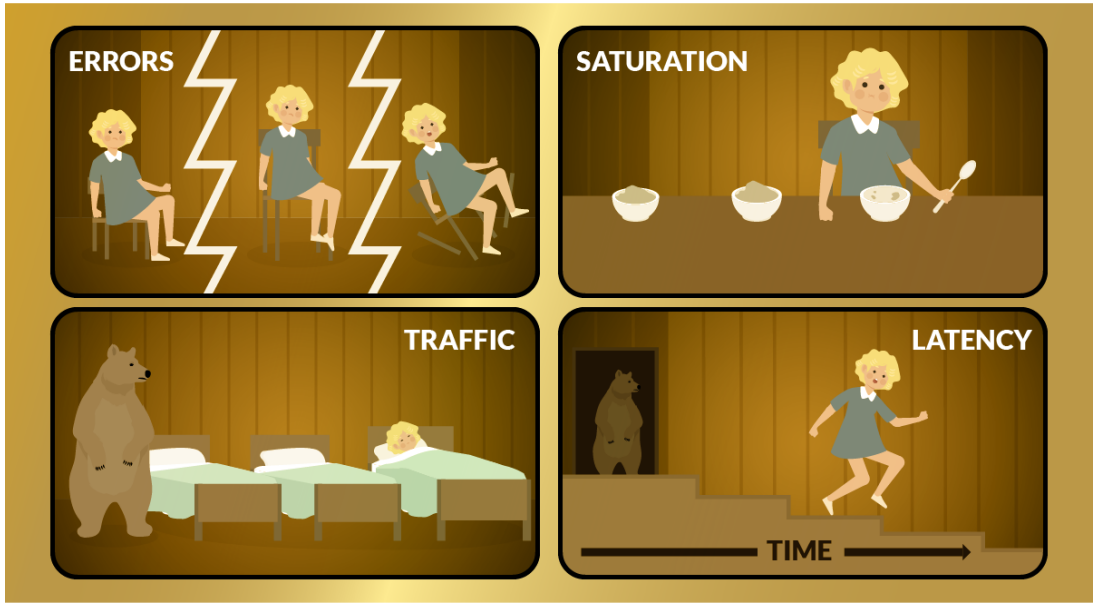
- *Control*: Is client A allowed to communicate with server B?
  Example: Prometheus should only use the `/metrics` API
- *Tracing and Metrics*: We know that client A is communicating with server B



Observability

Collects the 4 **Golden Signals** with ease:

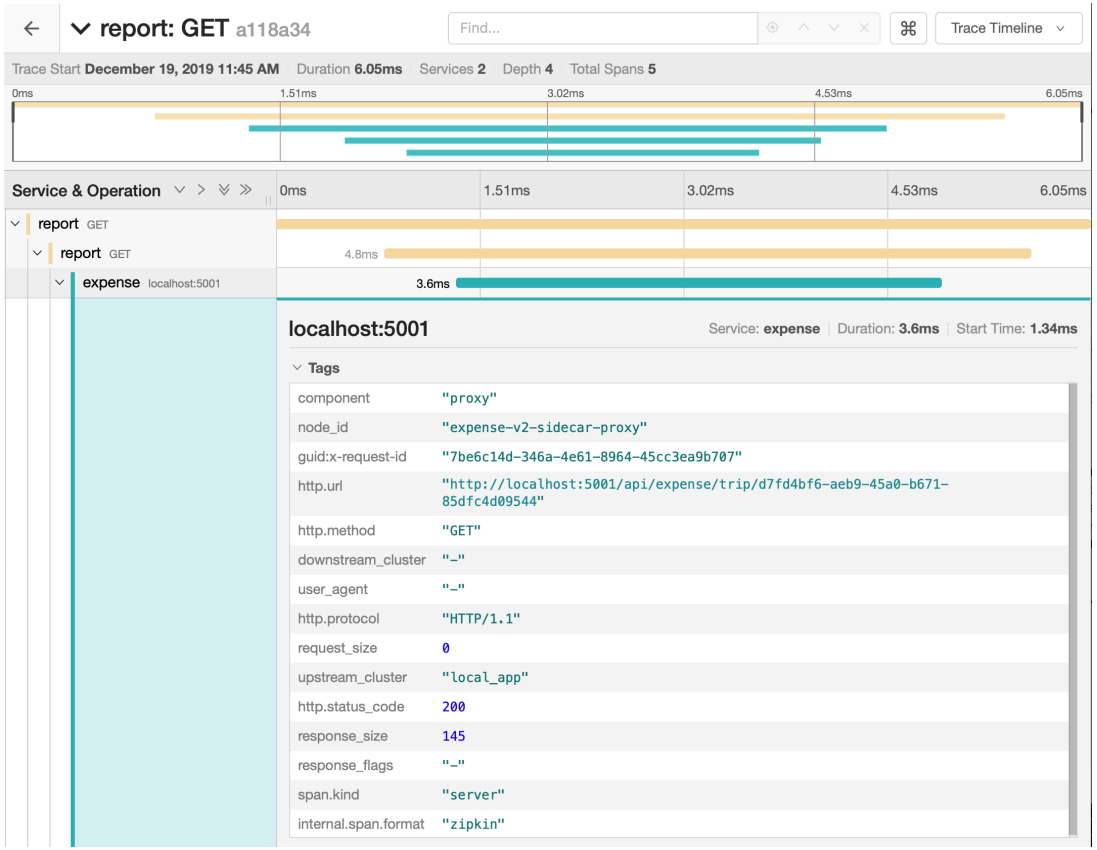
1. *Latency*: The time it takes to service a request
2. *Traffic*: How much demand is being placed on the system
3. *Errors*: How often the requests fail
4. *Saturation*: How much more traffic can the services handle?



Observability

Most service meshes provide means to:

- *Collect metrics*: Envoy proxies can record latency, request rates and error rates. Use other ways to collect resource usage.
- *Export metrics*: Most exports to Prometheus and provide a built-in dashboard to displays the metrics. Some supports exporting metrics to distributed tracing platforms



Options

There are plenty of Service Mesh implementations to choose from:

- **CNCF Projects**: Istio, Linkerd, Kuma, Open Service Mesh,...
- **HashiCorp**: Consul
- **Others**: Traefik Mesh
- **AWS**: App Mesh

> For more, visit: **servicemesh.es**

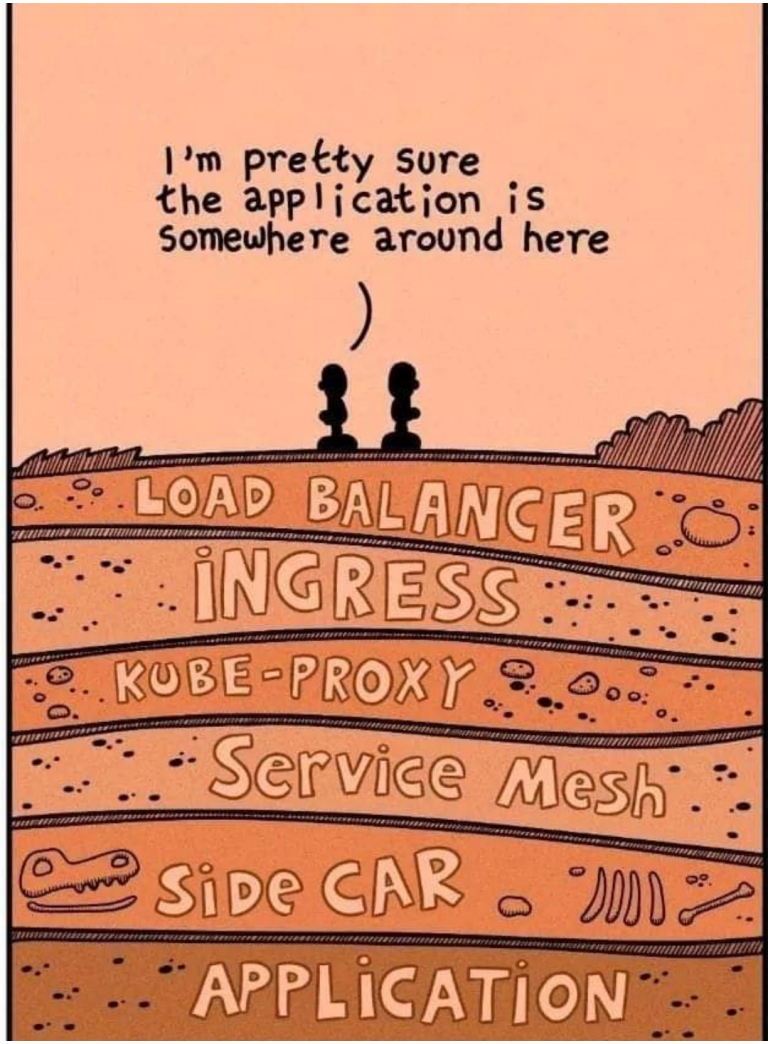
Demo

Service Mesh with Consul on Kubernetes

The problem with sidecar proxies

Sidecar proxies costs:

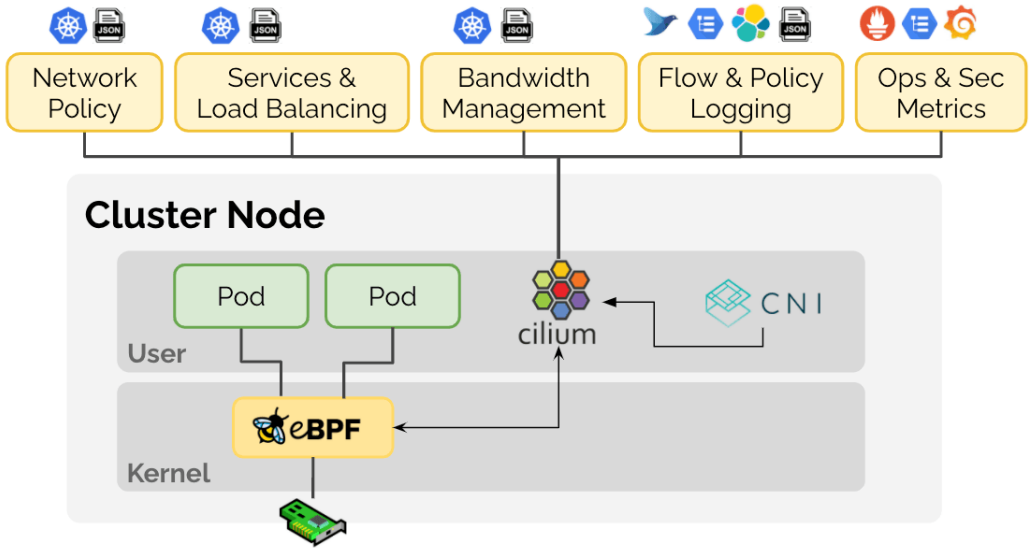
- *Resources*: Each pod requires an Envoy proxy, consuming ~200 CPU units with ~40-50MB of RAM
- *Performance*: Extra network hop means extra delay; slower pod start up time
- *Effort*: Maintaining the connections to the control plane requires scaling; debugging becomes harder; another dependency



Cilium

Started out as a CNI *(Container Network Interface)* for Kubernetes

- *Connects services*: IPv4, IPv6, etc.
- *Enhance security*: Kubernetes Network Policy, Transparent Encryption
- *Load balancing*: Replacement for *kube-proxy*
- *Observability*: Network flow logs, metrics, L3/4 & L7 *(not really)* monitoring



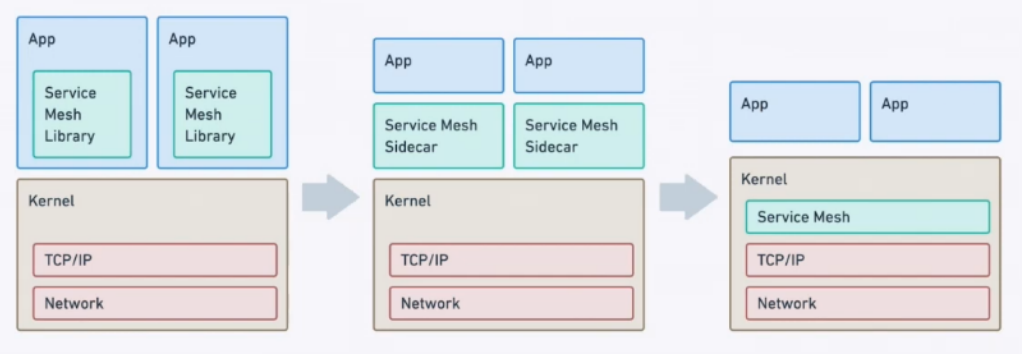
eBPF

Makes kernel **extensible**.

It allows access to kernel-level APIs, bringing better **networking visibility**

- Better networking visibility allows more **control**
- Sits between Pods and forwards network packets to Pod network namespaces based on policies, encrypting it along the way

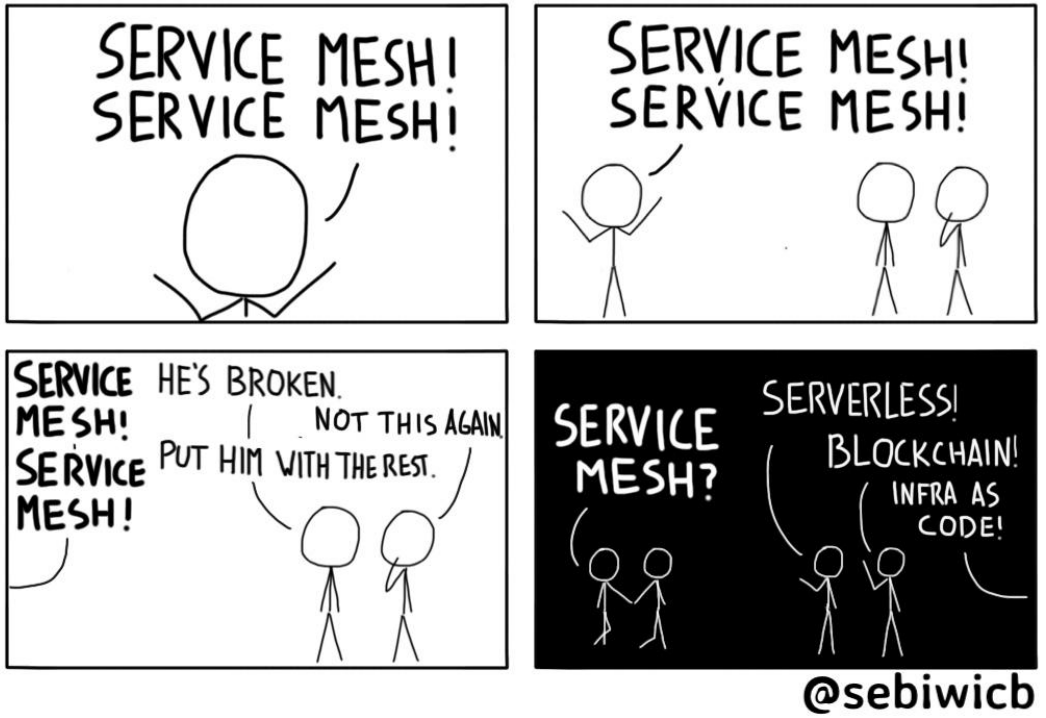
> **YouTube**: Effortless Mutual Authentication with Cilium



Do you need service mesh?

It depends, but if you need...

- **Need pure L3/L4 networking administration**: Use Kubernetes CNIs *(Calico, Cilium)*
- **Need application-aware network policies *(L7 like HTTP and gRPC)* & mutual encryption**: Proxy-based Service Mesh *(Consul, Istio, Linkerd)*



- **Pure service discovery**: Application libraries *(Consul, Netflix Eureka)*; CNIs; `kube-proxy` or Service Mesh
- **Need observability**: CNIs or Service Mesh
- **Need retries, circuit breaking, timeout**: Application libraries *(better customizability)*

... some of the more exotic use cases:

- **Cross-datacenter, cross-cluster network administration**: Cilium or Proxy-based Service Mesh

How about reliability?

- **Proxy-based Service Mesh**: Better support, easier to debug, easier to understand, battle-tested
- **Cilium and others**: Higher performance, good support, but quite new


200
OK

GitHub to the demo application, slides and other materials