

Sidecars and Service Meshes

Simple approaches to achieve complexity

From the Intro Lecture: Build on Foundations



Network Messaging, Messaging Patterns,
Protocols



Algorithms



Design Patterns



Engineering Practices



Tools

Lecture Source Material

Li, W., Lemieux, Y., Gao, J., Zhao, Z. and Han, Y., 2019, April. Service Mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (pp. 122-1225). IEEE.

Chandramouli, R. and Butcher, Z., 2020. Building secure microservices-based applications using service-mesh architecture. *NIST Special Publication, 800*, p.204A.

The Story So Far



Microservices

Small applications that encapsulate some core functionality of a larger application

Run in containers on clouds

Are accessed via over-the-wire communications

Each service consists of multiple instances that can be scaled up or down as needed

Are collectively but not individually fault tolerant

RabbitMQ and Microservices



Queue-based messaging



Push messaging



Supports many message patterns (RPC, Publish-Subscribe, ...)



Combines "data plane" and "control plane"

Apache Kafka and Microservices



Log-based messaging



Supports event sourcing



Topic-based publish/subscribe with client pull



Combines data plane and control plane services



Control plane implemented with embedded Zookeeper


Microservices with REST or RPC



Separate out the control plane from the messaging layer



Control plane technologies: Consul, ETCD, Zookeeper



Today: Instrumenting Microservices

Observability Issues in Distributed Systems

- Log aggregation
- Application metrics
- Audit logging
- Distributed tracing
- Exception tracking
- Health checks
- Log deployments and changes

Two Familiar Patterns

Inversion of Control

- This is how Java servlet containers work.
- You deploy your application (servlet) into the servlet container
- Client programs don't directly invoke your code
 - The container routes the traffic
- You don't need to worry about adding a lot of dependencies or libraries to your application
- The servlet container provides supplemental services that you don't have to implement
 - Logging
 - Transport level security (HTTPS)
 - Authentication and other security services

Aspect Oriented Programming

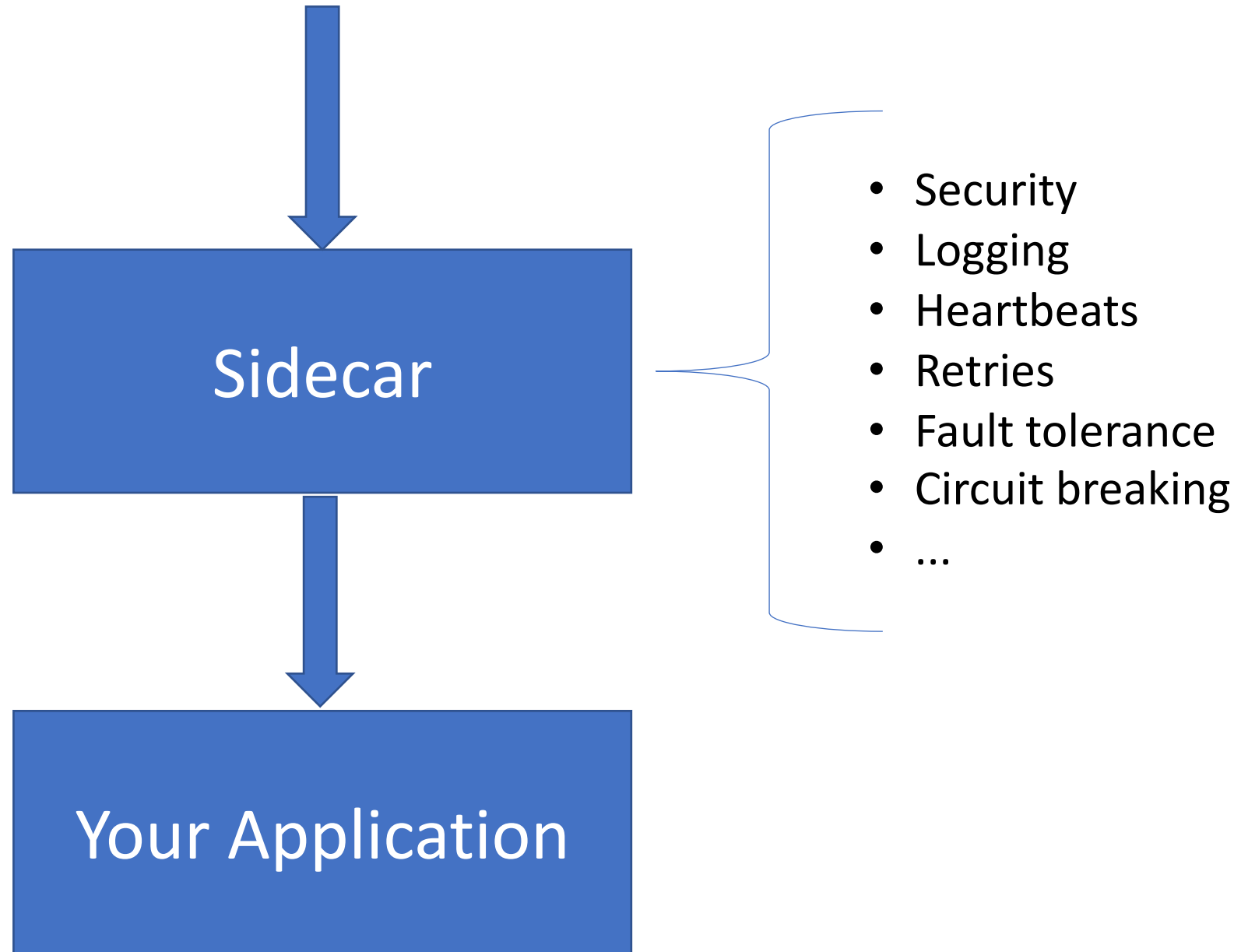
- Increases programming modularity by allowing additional behavior to be added to existing code (an [advice](#)) without modifying the code itself.
- This allows behaviors that are not central to the [business logic](#) (such as logging) to be added to a program without cluttering the code.

Key Idea for Both: Add Useful General
Capabilities without Modifying Your
Applications

The Sidecar Proxy Pattern Does the Same for Microservices

Sidecars solve some of the same problems as IOC and AOP but in distributed systems.

They act as network proxies so you don't have to change your code.



A good sidecar proxy should be lightweight and should not consume a lot of resources

You May Have Noticed...

- Consul, Zookeeper, ETCD manage information but require that you **include their client SDKs in your code**
 - You also need to program your services to use these appropriately for your system
- Kafka, RabbitMQ, and other messaging systems also require you to **embed client SDKs into your application code.**



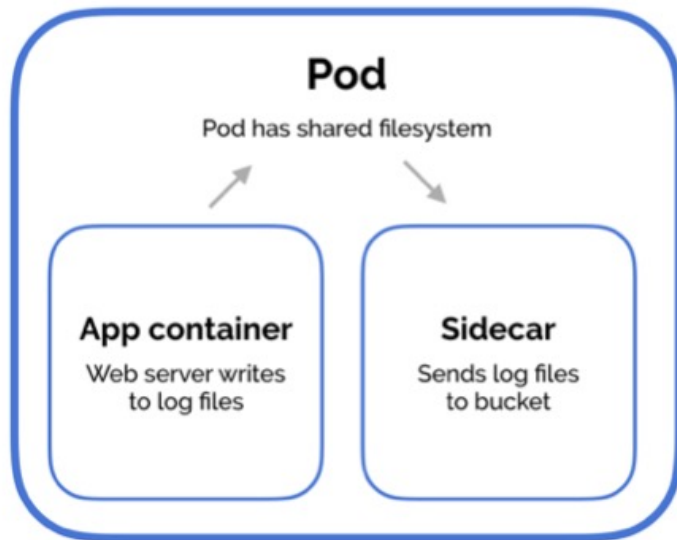
Some Nagging Issues

- What if a client SDK doesn't work well or isn't available in your programming language of choice?
- Library dependencies can be a problem.
- What if you decide to change from Zookeeper to Consul?
- Or RabbitMQ to Kafka?

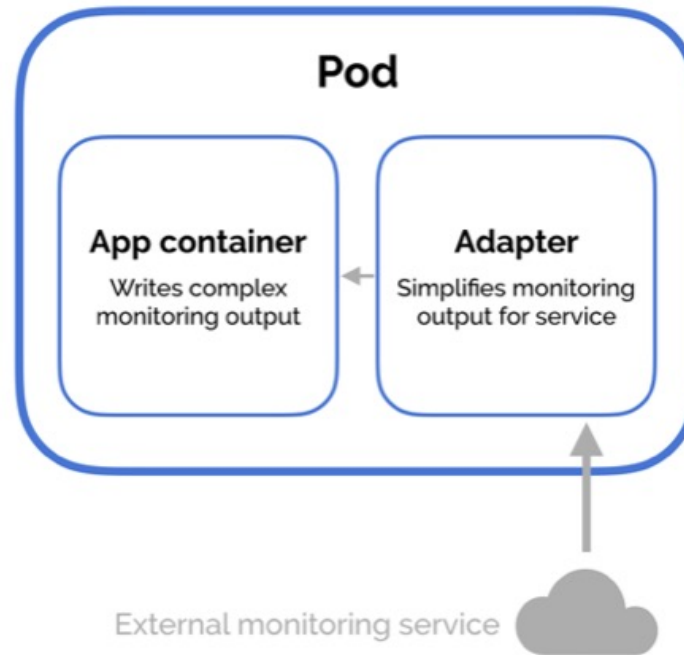


Sidecar proxies can
solve these problems

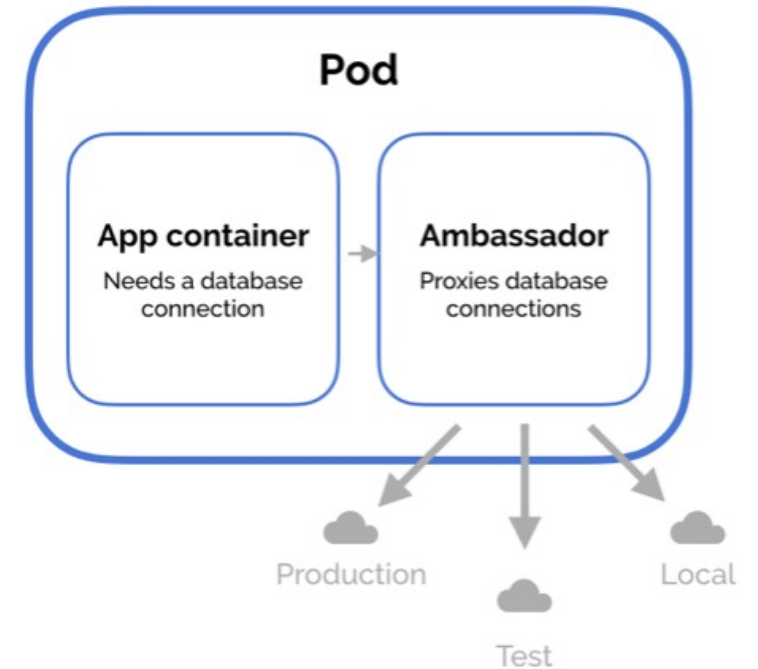
Sidecar



Adapter



Ambassador



Sidecar-Like Patterns: Sidecar Proxy

- **Problem:** see above
- **Sidecar Proxy:** Provides essential, general services for your application that don't need to be part of the application itself.
- **Example Sidecar Proxy Services:** Logging utilities, sync services, watchers, and monitoring agents.

Sidecar-Like Patterns: Adapter Proxy

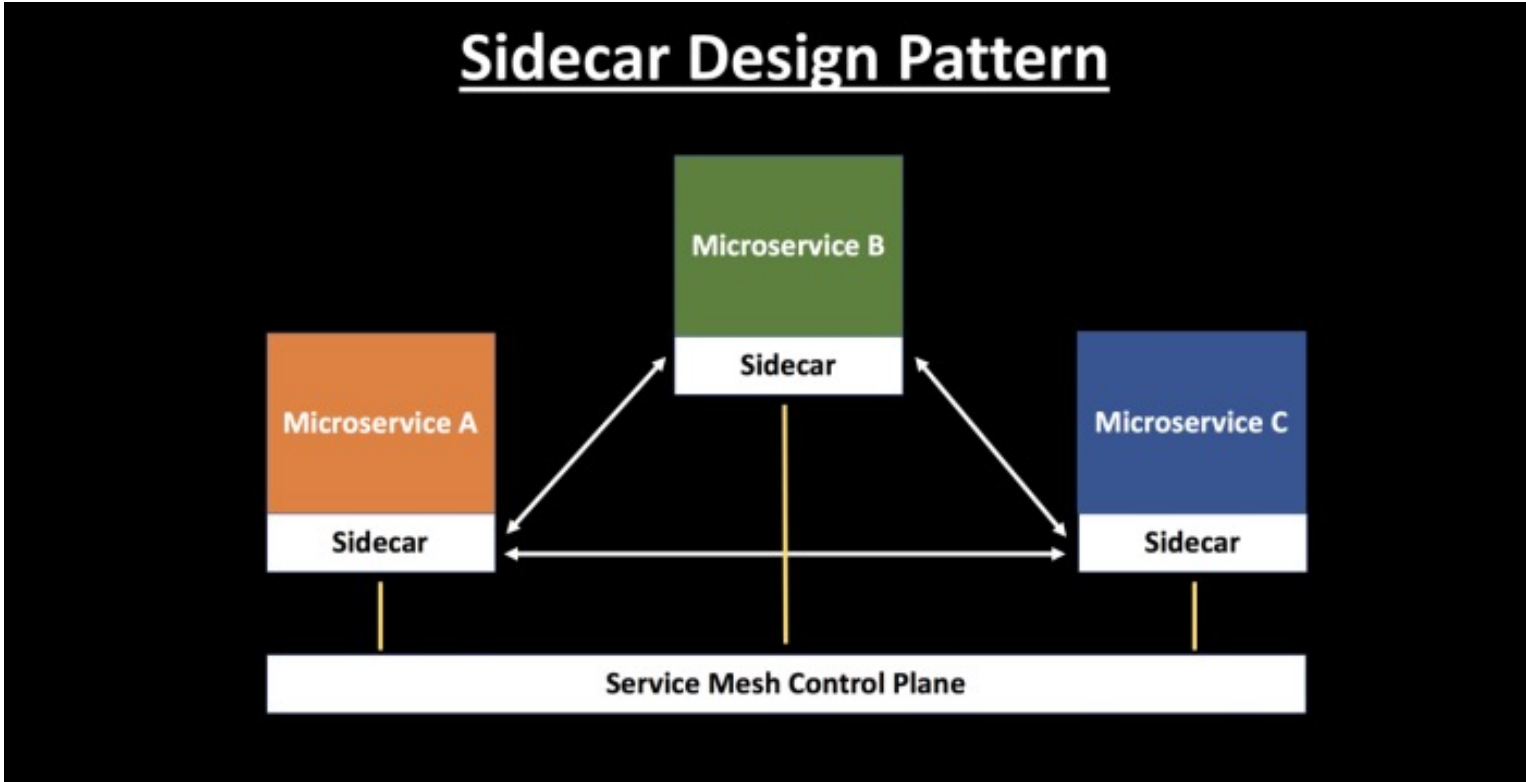
- **Problem:** Your microservices produce logs in different formats
- **Problem:** You need to prepend some metadata to your microservices' output
- **Adapter Proxy:** standardize and normalize application output or monitoring data for aggregation.

Sidecar-Like Patterns: Ambassador Proxy

- **Problem:** you need to dynamically change the DB that a service connects to
 - Switch from “test” to “prod”, for example
- **Ambassador Proxy:** a way to connect containers with the outside world.
- An ambassador container allows its partner to connect to it to a port on localhost
- The ambassador container can proxy these connections to different environments (dev, test, production)

Proxies in the Data Layer

- General Idea: deploy proxies alongside your applications
- The proxies interact with the Control Plane so you don't have to modify your programs

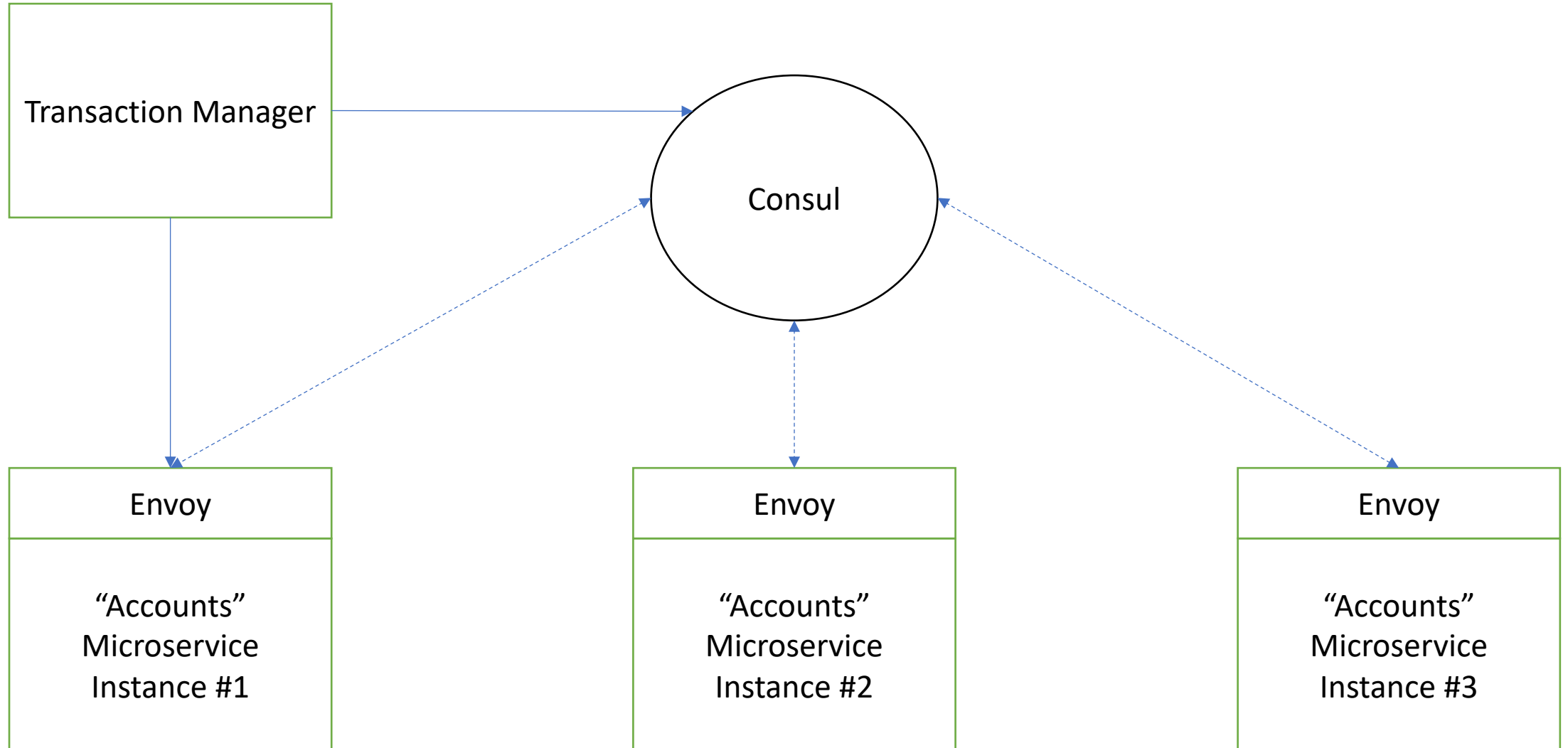


Service Meshes

Service Mesh Systems

- Envoy + Consul
- LinkerD
- Istio
- Commercial cloud vendors have their own

A Do-It-Yourself Service Mesh



A Simple Service Mesh

- Each microservice interacts directly with its co-deployed Envoy proxy.
- Envoy proxies interact with Consul to provide heartbeats, load information, etc.
- Clients ask Consul which service to use
- Clients connect to the Envoy sidecar, which captures additional traffic information as it passes the request to the service

Service Mesh Definition

- “A service mesh is a dedicated infrastructure layer for handling service-to-service communication.
- It’s responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application.
- In practice, the service mesh is typically implemented as **an array of lightweight network proxies that are deployed alongside application code**, without the application needing to be aware.”

(William Morgan, quoted in Li et al, 2019)

The following slides list of fundamental features of a Service Mesh

1. Service Discovery

- In microservices applications, the number of service instances as well as the state and location of a service are changing dynamically over time.
- Service discovery is required to enable service consumers to discover the location and make requests to a dynamically changing set of ephemeral service instances.
- Typically, service instances are discovered by looking up a registry that keeps records of new instances as well as instances that are removed from the network.
 - See previous lecture on control plane technologies

2. Load Balancing

- Provides the capability of traffic routing across the network.
- Compared to simple routing mechanisms (e.g., round robin, and random routing), modern service mesh load-balancing routing can consider **latency** and the **state** (e.g., health status, and current variable load) of the services.
- Compare and contrast messaging-level load balancing with RabbitMQ and Kafka

3. Fault Tolerance

- From a networking model perspective, a service mesh sits at a layer of abstraction above TCP/IP.
- With the assumption that the network and services on the network are unreliable, the service mesh must be capable of handling failures.
- This is typically achieved by redirecting the consumer requests to a service instance with healthy state.
- The service mesh may also implement retries

4. Traffic Monitoring

- All communication among microservices are to be observed, enabling reporting of requests volume per target, latency metrics, success and error rates, etc.
- This doesn't mean the mesh reads every message.
 - You can still encrypt your network traffic

5. Circuit Breaking

- Circuit breaking: automatically backing off the requests to a slowly responding client rather than completely failing the service with excessive load.
- Fundamental problem of networked applications: is an unresponsive service just slow or has it failed?

6. Authentication and Access Control

- Through policy enforcement from the control plane, a service mesh can define which services can access which other services, and what type of traffic is unauthorized and should be denied.
- What is the security model of RabbitMQ? Kafka? gRPC?
- See also future lecture on OAuth2

Summary

- Service meshes are technologies that provide many of the base technologies that you need to operate a microservice-based system.
- Sidecar proxies can help you deploy standalone applications into a service mesh
- Sidecars interact with the control plane to manage connections to their companion services while providing observability, circuit breaking, etc.