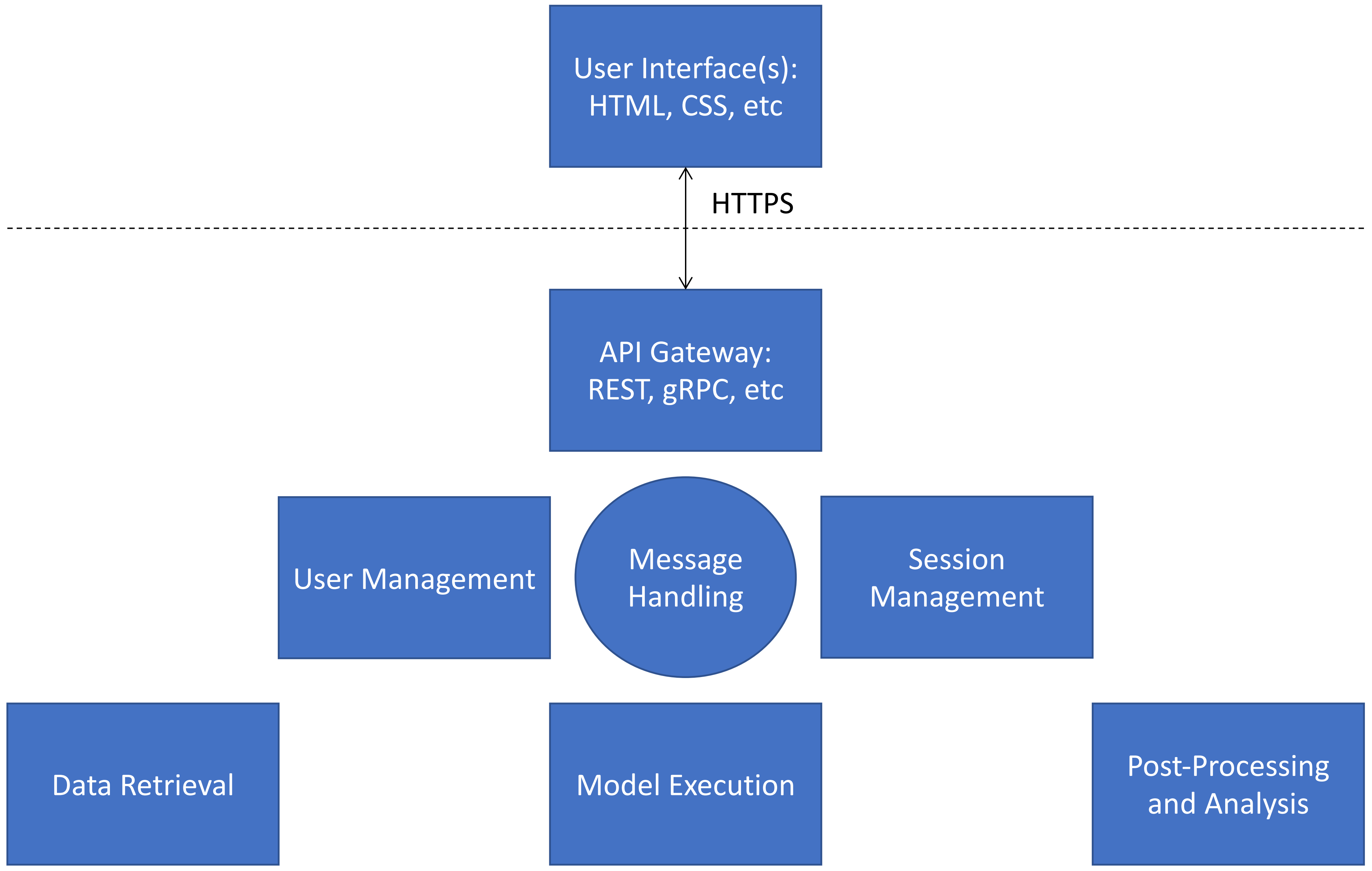
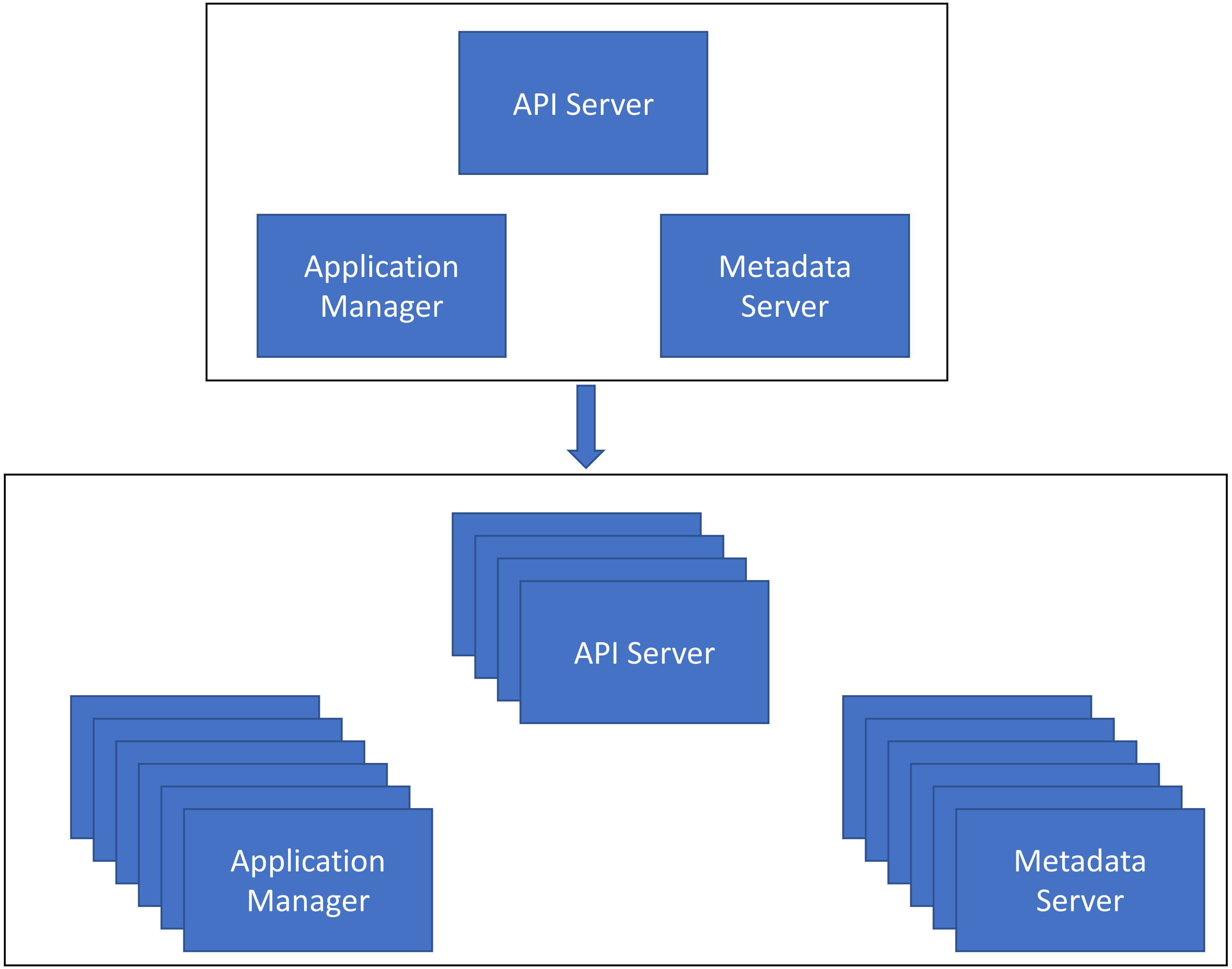
Sidecars and Service Meshes

Next level scaling for microservices


User Interface(s): HTML, CSS, etc
HTTPS
API Gateway: REST, gRPC, etc
User Management
Message Handling
Session Management
Data Retrieval
Model Execution
Post-Processing and Analysis


API Server
Application Manager
Metadata Server
API Server
Application Manager
Metadata Server

So Far, You've Learned

- Messaging systems: RabbitMQ, Kafka
- Containers: Docker
- Container management: Kubernetes
- Continuous Integration and Deployment: Jenkins
- Information management: Consul, Zookeeper

Two Familiar Patterns

Inversion of Control

- This is how Java servlet containers work.
- You deploy your application (servlet) into the container
- You don’t need to worry about adding a lot of dependencies or libraries to your application
- Client programs don’t directly invoke your code
  - The container routes the traffic
- The 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,
- Instead separately specifying which code is modified via a "pointcut" specification, such as "log all function calls when the function's name begins with 'set'.
- This allows behaviors that are not central to the business logic (such as logging) to be added to a program without cluttering the code, core to the functionality.'

https://en.wikipedia.org/wiki/Aspect-oriented_programming

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

Sidecar Proxies and Related Approaches

Sidecars solve some of the same problems as IOC and AOP.

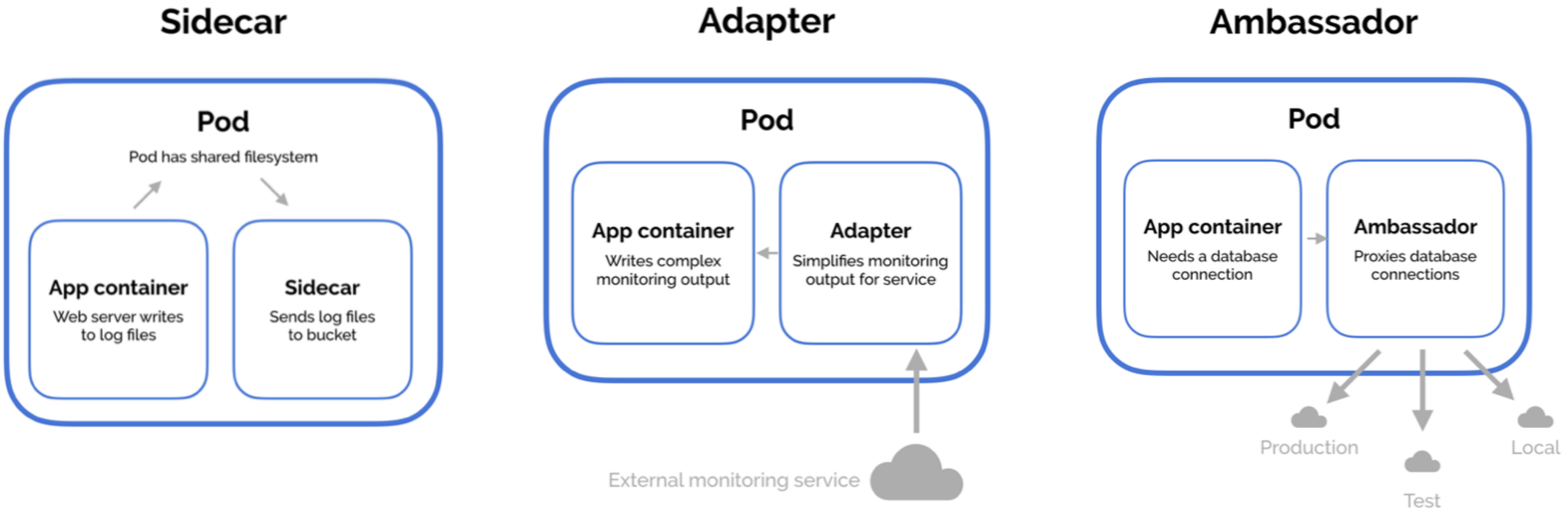
They act as network proxies so you don’t have to change your code.

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
Pod
Pod has shared filesystem
App container
Web server writes to log files
Sidecar
Sends log files to bucket
Adapter
Pod
App container
Writes complex monitoring output
Adapter
Simplifies monitoring output for service
External monitoring service
Ambassador
Pod
App container
Needs a database connection
Ambassador
Proxies database connections
Production
Test
Local

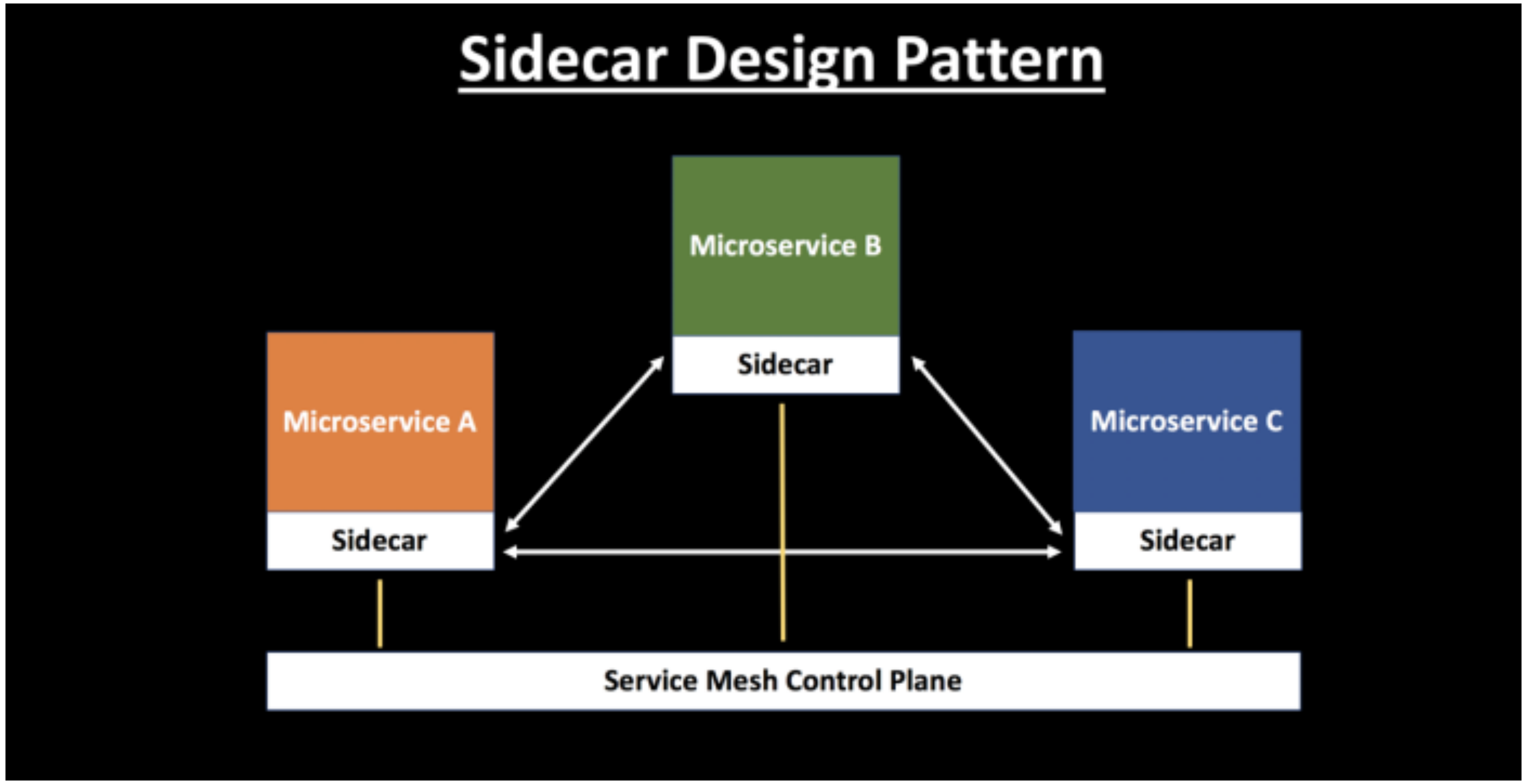
https://matthewpalmer.net/kubernetes-app-developer/articles/multi-container-pod-design-patterns.html

Sidecar-Like Patterns

- **Sidecar Proxy:** Provides essential, general services for your application that don’t need to be part of the application itself.
  - Logging utilities, sync services, watchers, and monitoring agents.
- **Adapter Proxy:** standardize and normalize application input connections, output, or monitoring data for aggregation.
- **Ambassador Proxy:** a way to connect containers with the outside world.
  - An ambassador container allows other containers to connect to a port on localhost while the ambassador container can proxy these connections to different environments depending on the cluster's needs.

Proxies in the Data Layer

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



Another Nagging Issue

- What if I really want to use REST, Thrift, gRPC, or something similar to build my microservice system?
  - I may be trying to scale up a Web application.
  - Or you may need to integrate legacy applications
- Request-response approaches have several advantages
  - Lots of existing standards: HTTP/HTTPS, TLS, OAuth2
  - They let you define language independent APIs and data models
- And you can even build hybrid systems
  - Messaging as a notification and coordination mechanism
  - But REST or RPC for actual service-service communication

More Nagging Questions

- How can I build a microservice-based system if I want to use REST or RPC-based services?
- Messaging systems like Kafka, information systems like Consul, and container management systems like Kubernetes can do a lot of what I need to implement a microservice architecture, but not everything.
  - For example, what do I do about security?
  - And logging, monitoring and other “observability” issues
- Can I avoid incremental approaches?

Service Mesh Systems

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

Fundamental Service Mesh Features

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.

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 the paper)

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 states 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.
  - Think about how you would do this in a request-response based system
- Typically, service instances are discovered by looking up a registry underneath which keeps records of new instances as well as instances that are removed from the network.
  - See previous lecture on Zookeeper

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 backend instances.
- 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.

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

- In case of accessing an overloaded service that has high latency already, the capability of circuit breaking will automatically back off the requests rather than completely failing the service with excessive load and resulting in propagated unavailability.
- 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 services, and what type of traffic is unauthorized and should be denied.
- What is the security model of RabbitMQ? Kafka? gRPC?
- In an upcoming lecture, we will look at OAuth2

Two Open Questions for Gateways

- How well will service meshes work across multiple clouds and in hybrid cloud-edge computing systems?
  - Edge-cloud hybrids are very important for Apache Airavata
  - HPC systems are on the edge
  - Scientific instruments are on the edge
  - Scientists still like to buy their own computers
- What is the best way to express, execute, and trace multi-service transactions that you need to fulfill the business logic of your application?
  - Workflows

Some Open Questions for Apache Airavata

- How can we choose between LinkerD, Istio, and maybe other approaches (Consul+Envoy)?
  - You can look at MFT also
- How well do these approaches work on "edge-to-cloud" and "multi-cloud" deployments?
- How can Airavata take advantage of Service Mesh features mentioned in the lecture, like
  - Blue-green, canary, and other continuous deployment features
  - Monitoring
  - Logging

Project 3, part 1 will be to pick the right problem