Aalto University
School of Electrical
Engineering

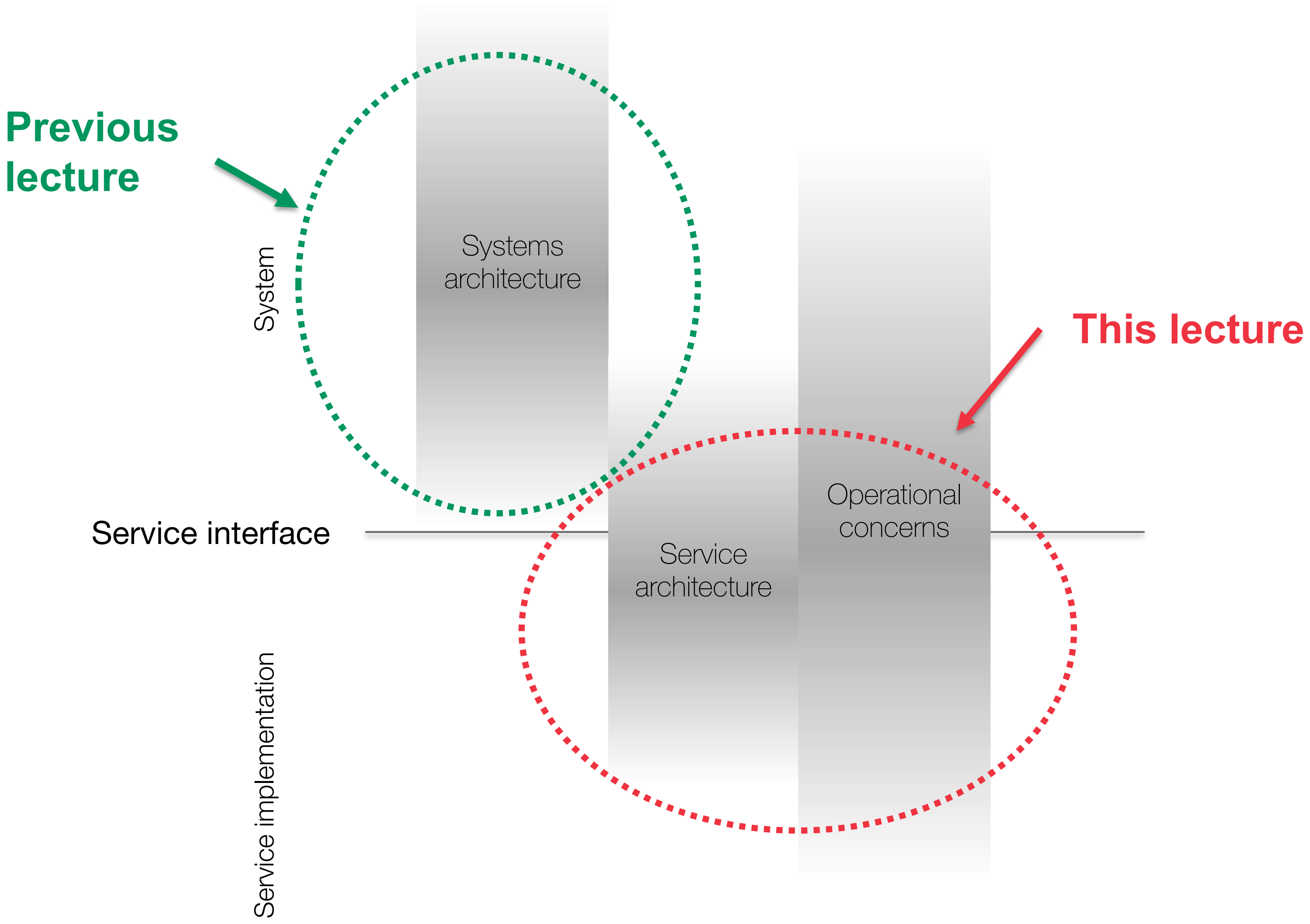
Single-node patterns

14.2.2019
Santeri Paavolainen

Overview

- **“Single node” refers to physically co-located components**
  - Usually part of a single service
  - Internal structure for a service (local decisions)
  - In Kubernetes this would be containers in a *single pod*
    - *Affinity-based scheduling of different pods is a multi-node pattern, discussed later*
- **Why single node?**
  - Microservice architectures are multi-node (distributed) systems … ?

- **Terminology follows *Designing Distributed Systems* (Burns, 2018)**


Previous lecture
This lecture
System
Service implementation
Service interface
Systems architecture
Service architecture
Operational concerns

Nomenclature

- **"Patterns" refers to**
  - "Re-usable form of a solution to a design problem" [Wikipedia]
  - Popularized in CS by GoF's book *Design Patterns* (1994)
    - *Originally very OO-focused, but has been expanded to software architecture*
    - *Anti-patterns are counterproductive patterns (enlightening!)*
- **Pattern is not a template or a code library, nor a component**
  - "Way of understanding and structuring a problem and its solution"

- **Important in establishing common terminology!**

Some architectural patterns

- **Layering**
  - UI-Service-Business-Persistence
- **Client-Server**
- **Master-Slave**
- **Event-bus**
- **Microservices**

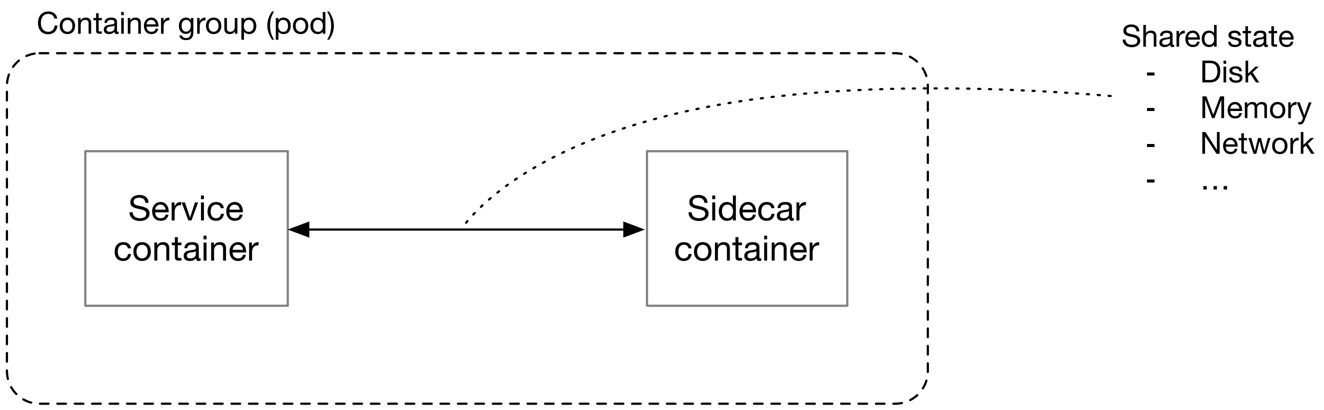
- **Why not MVC?**

Patterns for co-scheduled containers

- **Sidecar**
- **Ambassador**
- **Adapter**

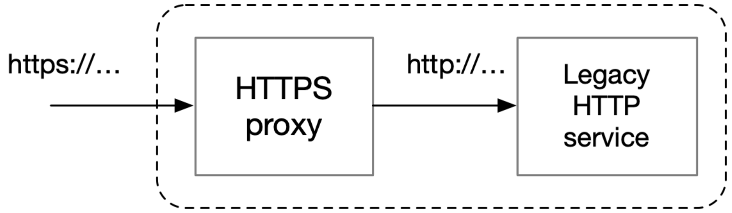
Sidecar pattern

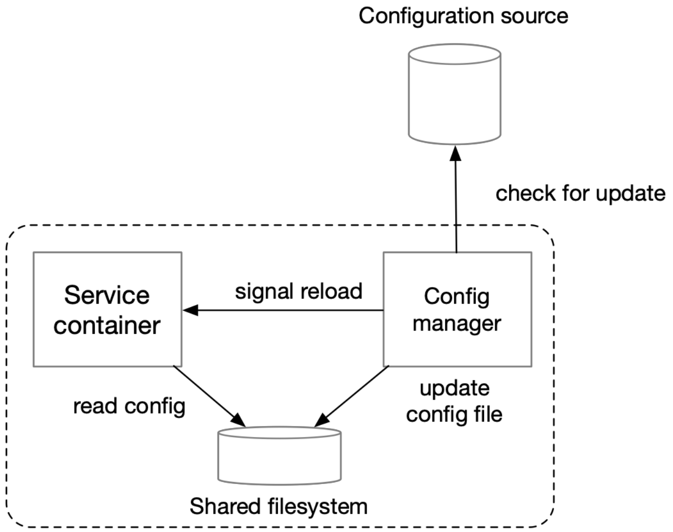
- **Sidecar as in "sidekick"**
  - Adding something the main protagonist does not have
- **Co-scheduling of a container (potentially with shared state)**

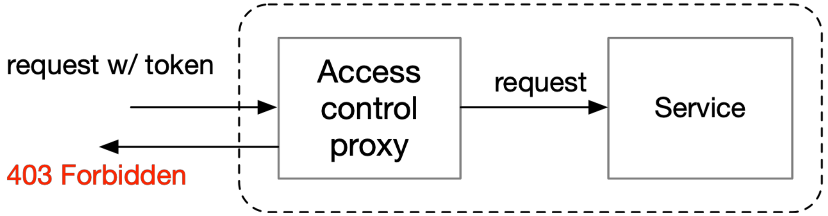


Sidecar examples

- Adding HTTPS to legacy application
- Updating configuration
- Access control

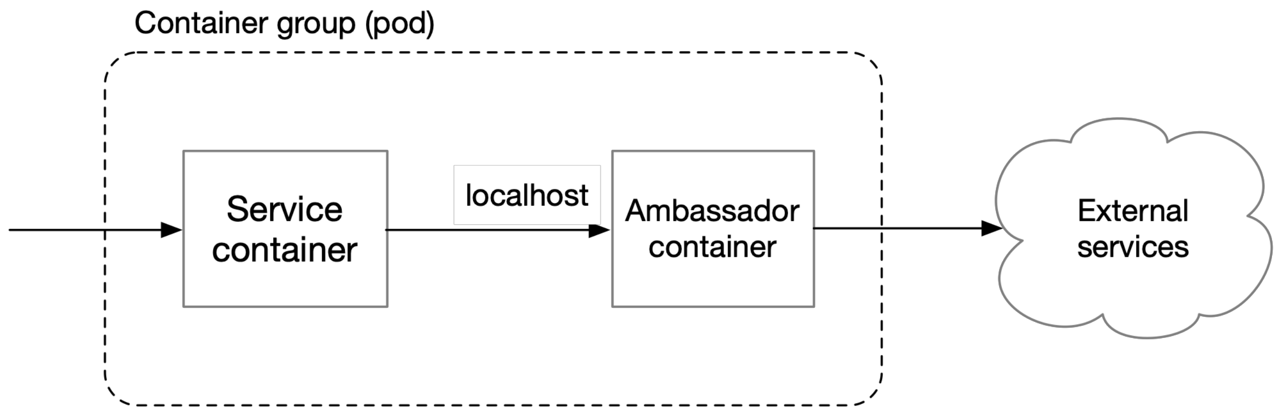






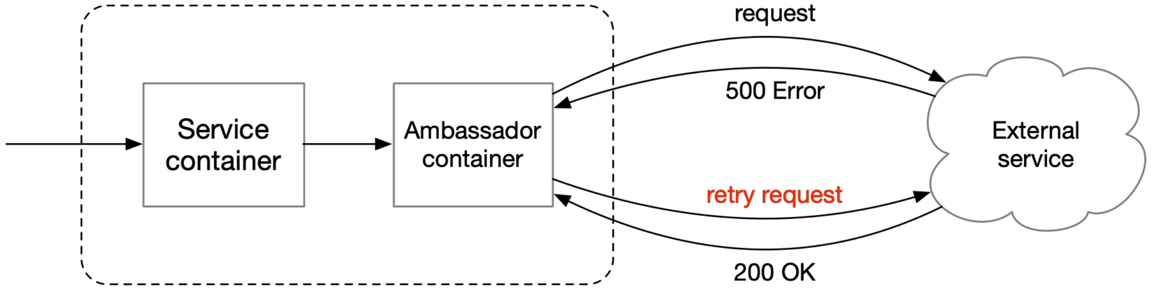
Ambassador

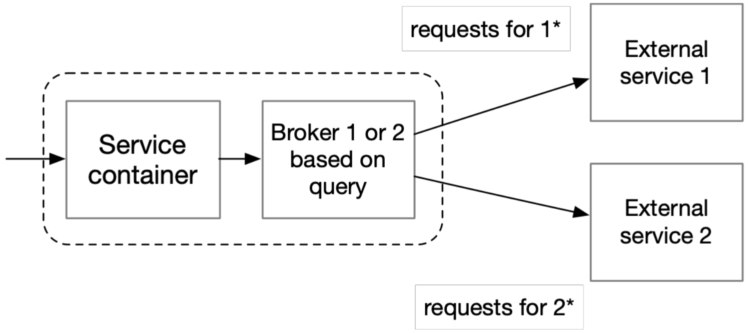
- **Specific type of sidecar**
- **Abstracts and/or brokers external interface for the service**
  - Different ambassador in different environments (dev vs. prod)
  - Provides a constant interface for the service

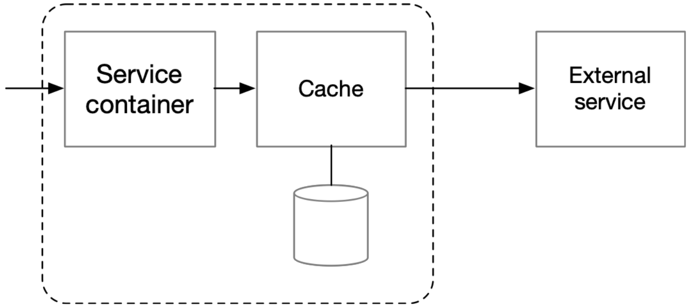


Ambassador examples

- Hiding 500s
- Service brokering
- Local caching

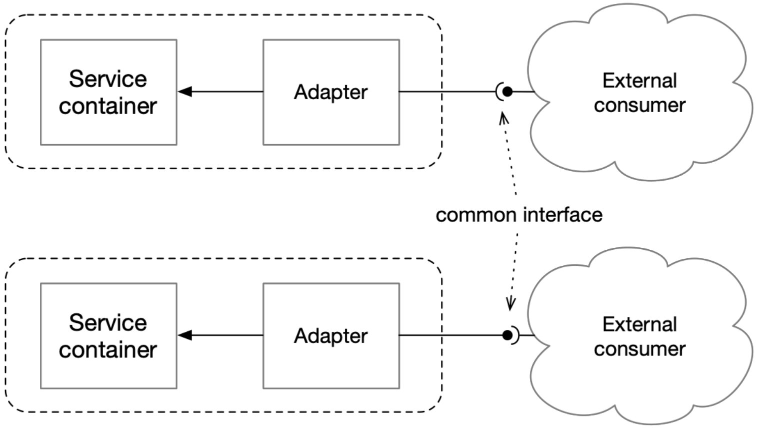






Adapter

- **Sidecar pattern when someone <u>else</u> needs a specific interface**
  - Common interface used across the system such as logging, metrics, service health etc.
  - Not “core” service but supporting interfaces
- **Both push and pull interfaces**



Kubernetes example of sidecar

- **Simple “Hello world!” web server**
- **Using UWSGI to generate a log to /var/log/uwsgi.log**
- **Simple app showing last 40 lines of /var/log/uwsgi.log**
- **Two containers sharing /var/log**
- **Both run on same pod**
  - Both can not bind to the same port

Sidecar vs. ambassador vs. adapter

- **All co-scheduled with a service container**
  - Tight coupling!!!
- **Names are important!**
  - All similar in structure and functionality
  - Difference in what interacts and to/from where

- **Sidecar: augment and improve service**
- **Ambassador: brokers external interface to service core**
- **Adapter: transforms an interface to common interface**

- **Warning: Semantics sometimes a bit murky (consider metrics)**

Why co-scheduled containers?

- **Easy argument for legacy systems**
  - If it ain't broke, don't fix it!
- **Avoid tight coupling at <u>code level</u>**
  - Changing application logging code
  - Move tight coupling to interface level (up the stack)
- **Easier to test and validate**
  - Separate life cycle from core application
- **Shareable across services as containers**
  - Container-level re-use!
- **Reduced variability for service core**