Class #12

04. Software Architecture Patterns

Software Architectures

Master in Informatics Engineering

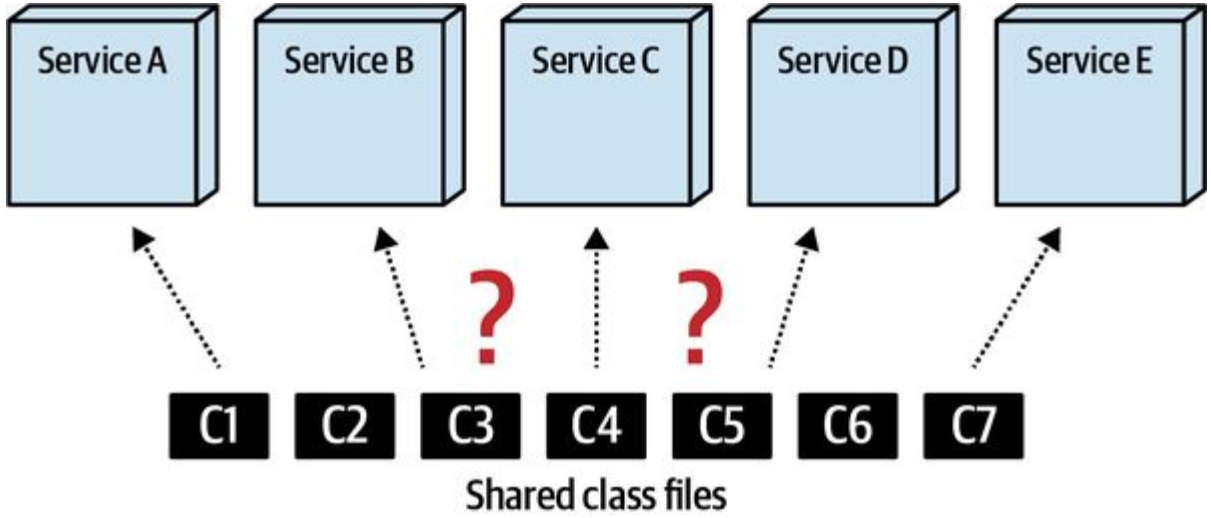
Cláudio Teixeira (claudio@ua.pt)

Agenda

- Reuse Patterns
  - Sidecars and Service Mesh
- Saga pattern

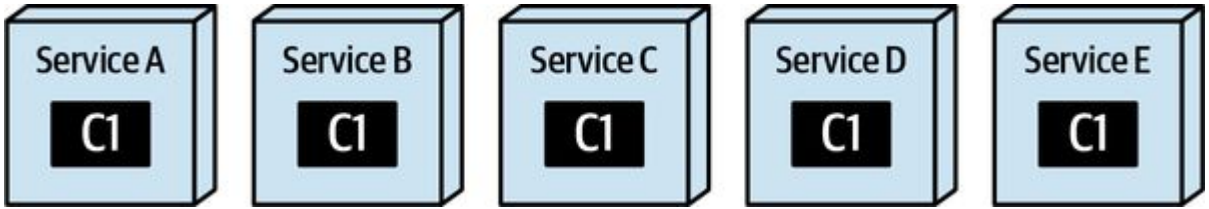
To reuse or not to reuse?

https://learning.oreilly.com/library/view/software-architecture-the/9781492086888/ch08.html#idm45978844034768



- Code Replication
- Shared Library
- Shared Service

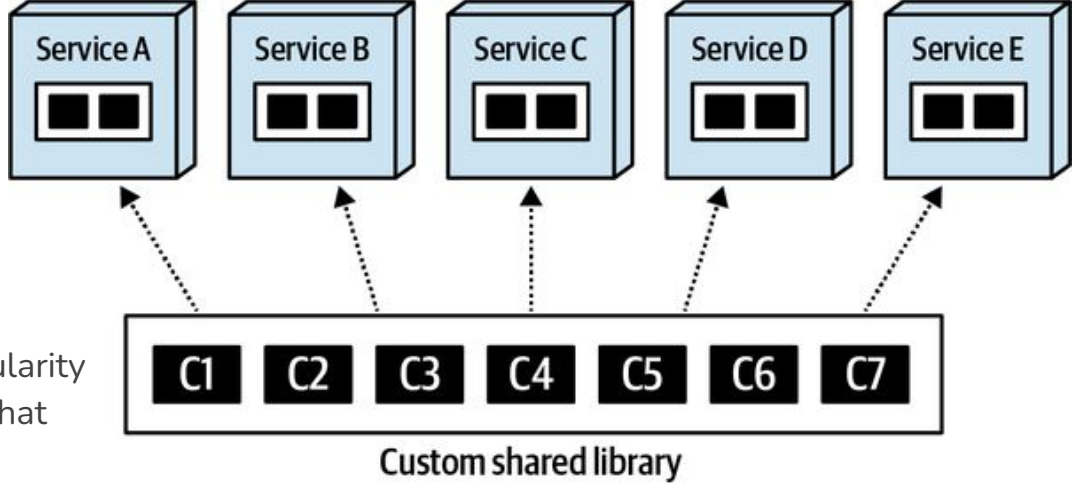
Code replication



- In code replication, shared code is copied into each service (or more specifically, each service source code repository)
- While code replication isn't used much today, it nevertheless is still a valid technique for addressing code reuse across multiple distributed services.

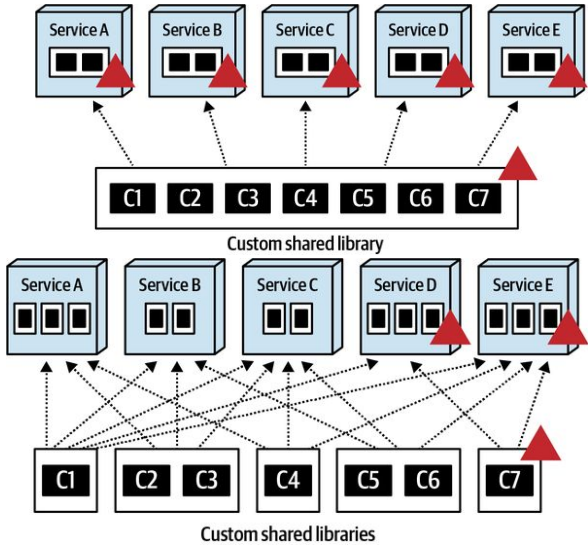
Advantages	Disadvantages
Preserves the bounded context	Difficult to apply code changes
No code sharing	Code inconsistency across services
	No versioning capabilities across services

Shared Library

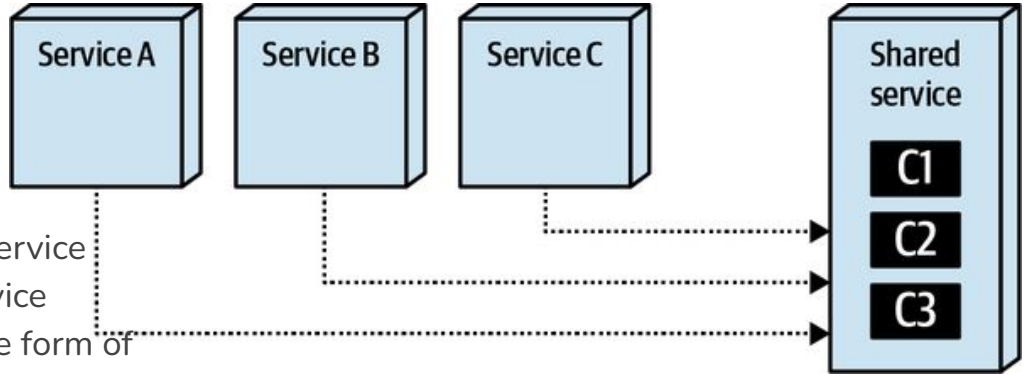


- there are trade-offs associated with the granularity of a shared library. The two opposing forces that form trade-offs with shared libraries are dependency management and change control.
- a change in shared class C7 impacts only Service D and Service E

Advantages	Disadvantages
Ability to version changes	Dependencies can be difficult to manage
Shared code is compile-based, reducing runtime errors	Code duplication in heterogeneous codebases
Good agility for code shared code changes	Version deprecation can be difficult
	Version communication can be difficult



Shared Service

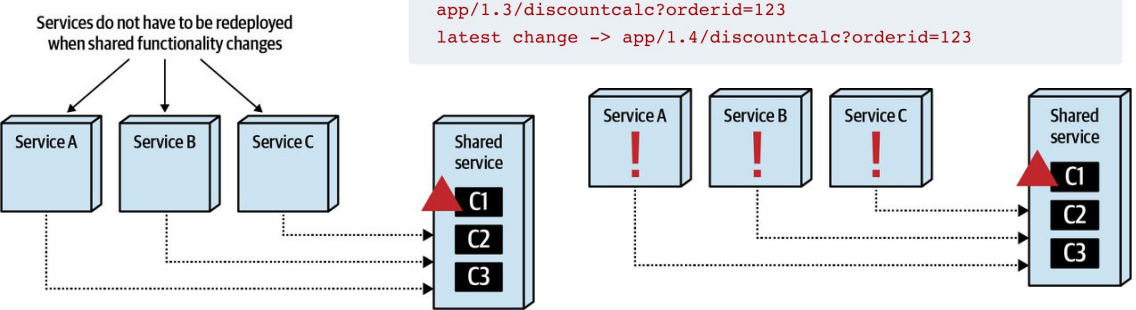


- Shared functionality in a separately deployed service
- One distinguishing factor about the shared service technique is that the shared code must be in the form of composition, not inheritance.
- a change to a shared service is a runtime change, as opposed to a compile-based change with the shared library technique. As a result, a “simple” change in a shared service can effectively bring down an entire system

Advantages	Disadvantages
Good for high code volatility	Versioning changes can be difficult
No code duplication in heterogeneous codebases	Performance is impacted due to latency
Preserves the bounded context	Fault tolerance and availability issues due to service dependency
No static code sharing	Scalability and throughput issues due to service dependency
	Increased risk due to runtime changes

Versioning to the rescue!

```
app/1.0/discountcalc?orderid=123
app/1.1/discountcalc?orderid=123
app/1.2/discountcalc?orderid=123
app/1.3/discountcalc?orderid=123
latest change -> app/1.4/discountcalc?orderid=123
```



Sidecars and Service Mesh

- https://learning.oreilly.com/library/view/software-architecture-the/9781492086888/ch08.html#sec-sidecar-pattern

Deep into sidecar & Service Mesh

Group assignment
45 min

- https://learning.oreilly.com/library/view/software-architecture-the/9781492086888/ch08.html#sec-sidecar-pattern
- https://learn.microsoft.com/en-us/azure/architecture/patterns/sidecar

Saga Pattern

Group assignment
90 min

- https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga
- https://github.com/Azure-Samples/saga-orchestration-serverless

What points should be followed for implementing Saga?

Bibliography

- https://learn.microsoft.com/en-us/azure/architecture/browse/
- https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga