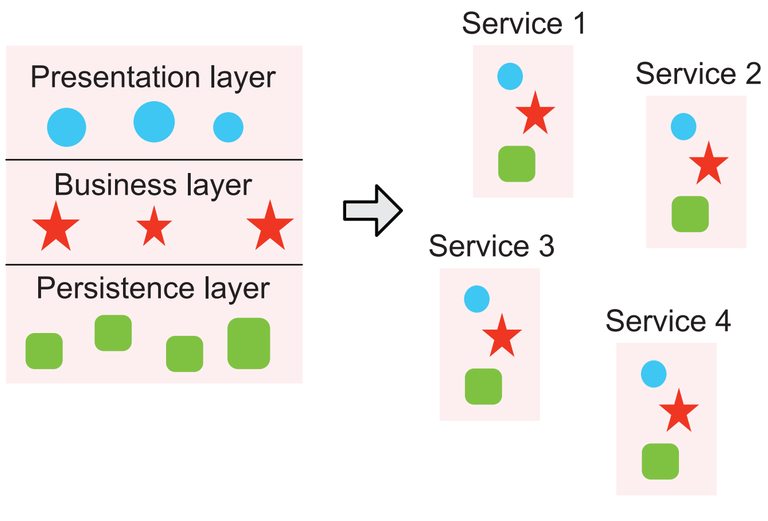
State of the art in Stateful Microservices

Danish AMJAD, Edoardo CONTE

European Commission
ERASMUS MUNDUS

Monolith to Microservices



Stateful Microservices

Each microservice that operates over a state is called a stateful microservice.

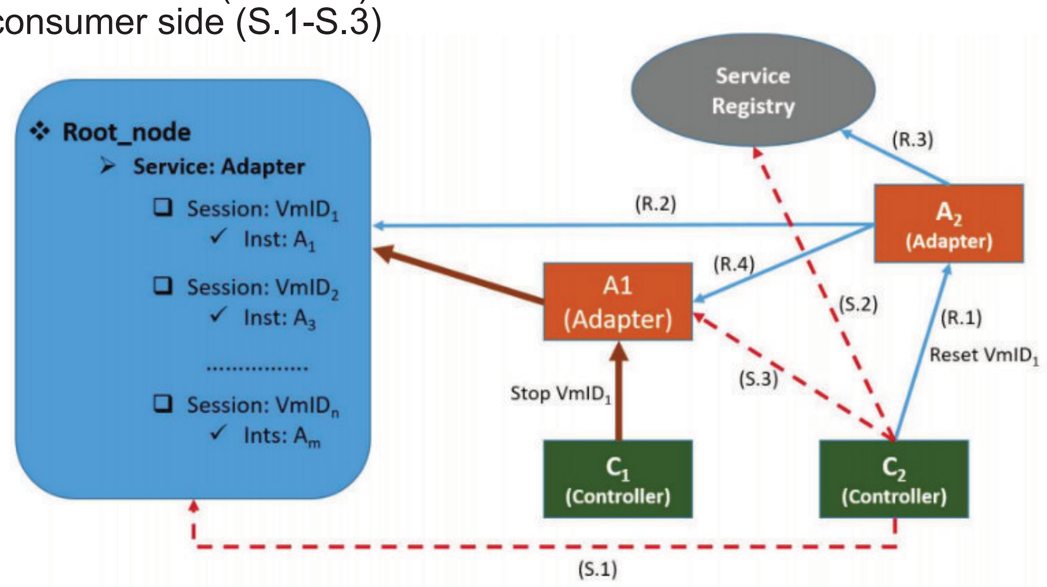
In their deployment, they need to fulfill specific requirements:

1. Fault-tolerance	2. Scaling out
3. Transactions	4. State locality
5. Global state view	6. Loose coupling
7. Debugging and Auditing	8. Dynamic Reconfiguration

Specific Requirements

Routing for stateful microservices

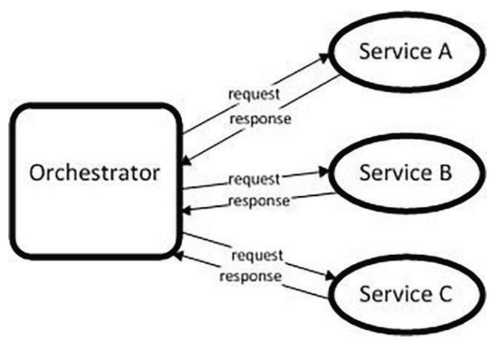
This solution guarantees consistent state locality with dynamic service discovery. It provides a mechanism to maintain session data shared among services. Every time a producer service needs to start an operation it checks if a session is already associated to it. In negative case, it creates a new session, being the owner. Only process owner can modify the session, which is protected by a locking system. The picture explains how the services can be discovered:
- producer side (R.1-R.4)
- consumer side (S.1-S.3)



Road Ahead

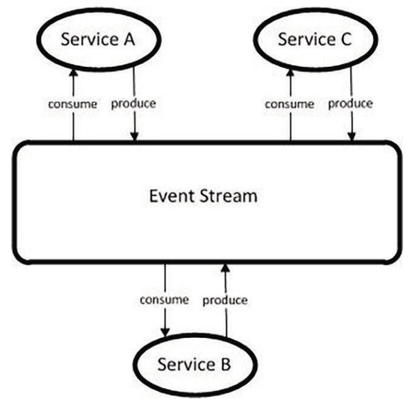
In this research, we focused on defining the main features of microservices, exploring possible solutions in the market. However, we discovered that most of them are custom solutions implemented by the developer. However, we believe that modern stream processors are really promising solutions. They provide distributed state management with automatic service scalability. Even if the technology is not mature enough to cover all the functionalities of stateful microservices (transactions, global state view), we believe that the research will move towards that result. Apache Flink, and other similar solutions, may be the tool that integrate completely all the characteristics of stateful microservices. The goal will be to have all those features by default in one or multiple tools.

Architectures



Orchestrator

- − High coupling.
  Orchestrator overloading.
- + Easy flow management.
  Easy debug.



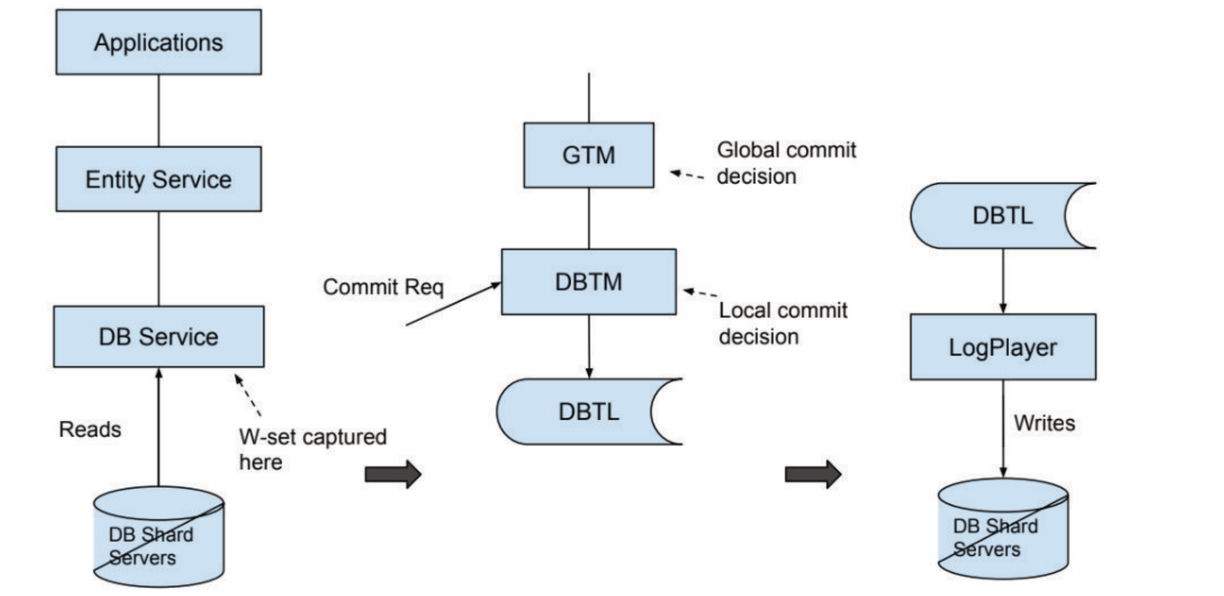
Coreography

- − Low control on flow of execution.
  Hard to debug.
- + Asynchronous processing.
  Distributed control.

GRIT
Consistent distributed transactions

This solution solves the problem of distributed transactions over multiple services. It makes use of a global transaction manager (GTM) and a local one (DBTM), for each database instance. Each manager has then a log, in which it stores the commit sequence.

Here they are explained the three phases responsible for maintining the consistency:



(A) Optimistic execution phase (B) Logical commit phase (C) Physical materialization phase

Stream processors for stateful microservices

- + Modern stream processors as Apache Flink have the capability to maintain a consistent distributed state. Each service can now hold a local state.
- − However, Flink, specifically, does not provide an efficient way to query the global state of the system. Moreover, it presents some downside in the flexibility of the business logic implementation.