Microservices Architecture

Data consistency techniques

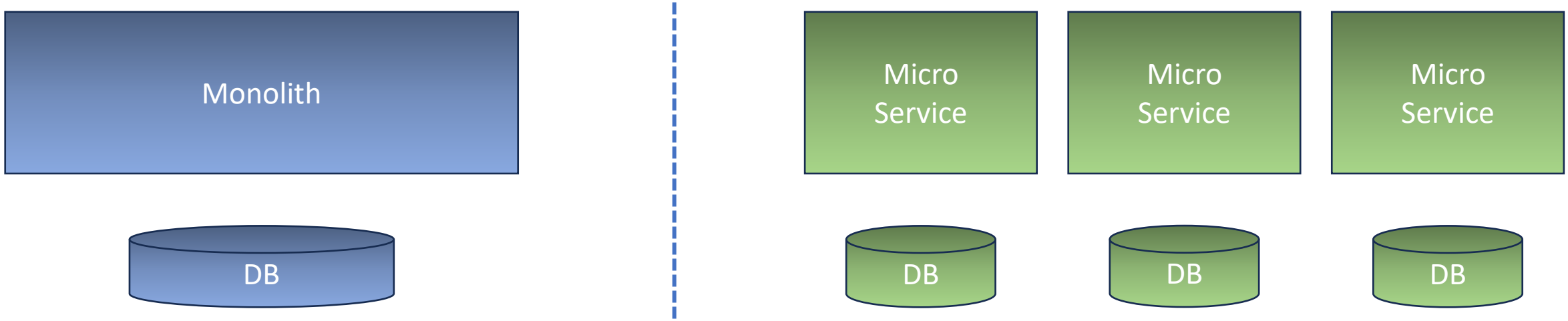
Contents

- Introduction
- Microservice vs Monolith Architecture
- Advantages for using Microservices
- Disadvantages of using Microservices
- Achieving Data Consistency
  - Direct Rest Calls
  - Event-based Architecture
  - Advanced Consistency Techniques

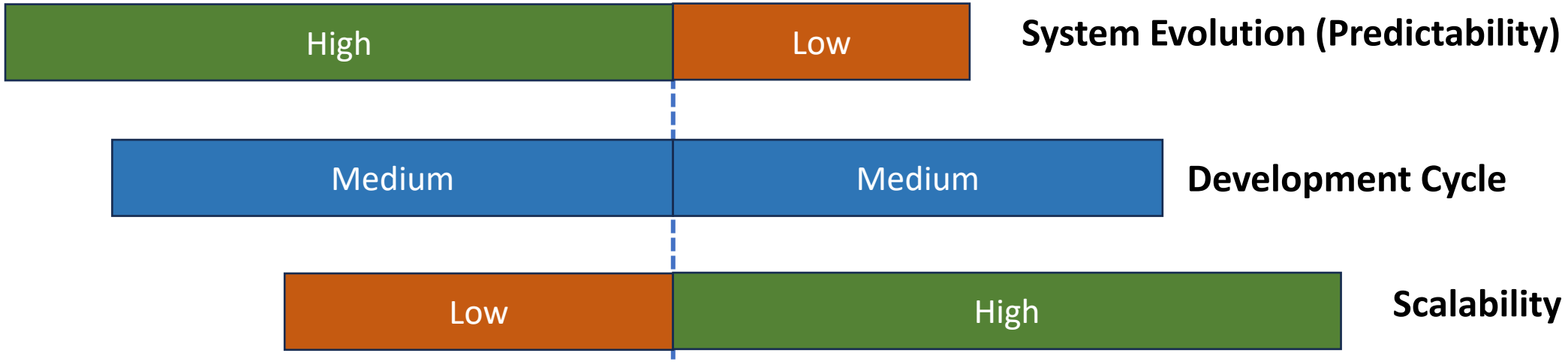
Introduction

- Microservice architecture is a variant of the Service-Oriented-Architecture
- Aims to decouple application functionality and map them to smaller applications (i.e. microservices) in order to decrease coupling, and allow fast changes, individual testing and deploys
- Becomes more and more used recently due to large scale development of cloud services and CI/CD automation pipelines that allow managing large number of applications. This was not feasible with manual deployment and bare-metal servers.
- Each microservice is an individual application that exposes an API (usually REST) and its own encapsulated database.
- Usually, microservices are build around business capabilities of the application (based on use-cases, actors, scenarios, etc.)

Microservice vs Monolith



- Choosing a system architecture is a **trade-off**

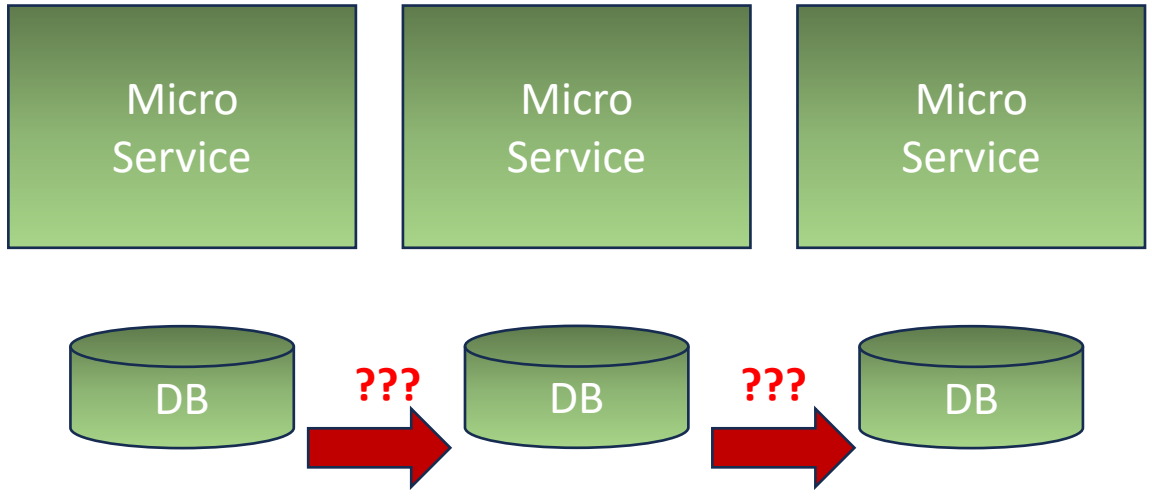


Advantages for using Microservices

- Finer-grained scalability: each service can be scaled-out according to the traffic on specific endpoints
- Increased testability: each service can be tested individually.
- Fast changes on functionality: Changes on one service have little impact on other service
- Fast deployment: each service can be deployed individually without impacting other service deployments.
- Maintenance and reliability: each service can be maintained individually. If one service crashes, it should not affect other running service.

Disadvantages for using Microservices

- **Achieving Data Consistency**: each microservice has its own database, **BUT** often one microservice needs data from other services (otherwise they would be completely different applications)
- How can consistency be achieved? What kind of consistency can we achieve?
- What about **CAP theorem**?



How can data changes be propagated between various databases?

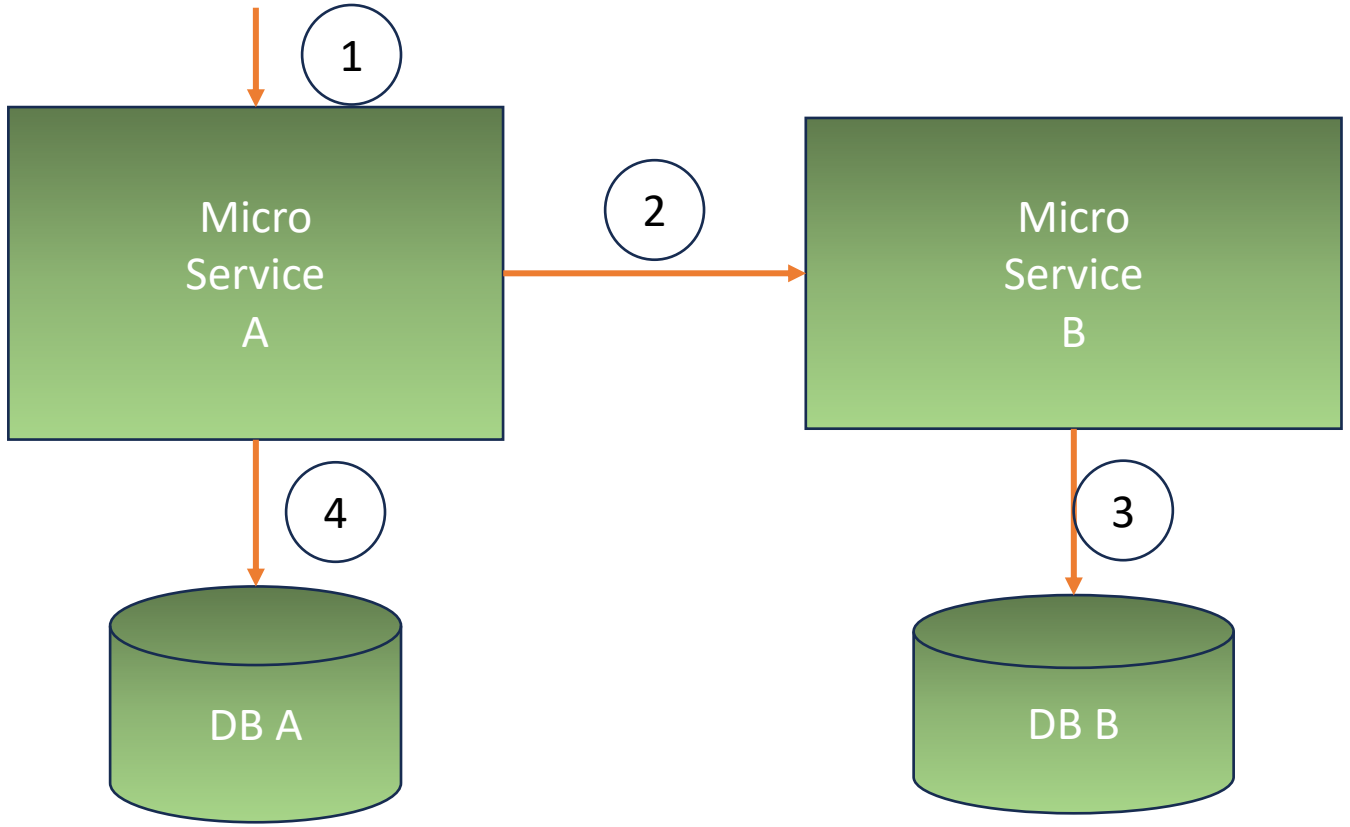
Achieving Data Consistency

- ~~Use one database for all microservices~~ => NOT MICROSERVICE ARCHITECTURE!
- Duplicate some of the data and expose APIs for each microservice to be called by other services when data is changes (synchronous change in two databases)
- Event-driven architecture: publish changes in Message-Oriented-Middleware (e.g. message queues, topics). Microservice Components listen to changes and update data accordingly.

- Advanced techniques:
  - Distributed transactions (SAGA)
  - CDC Systems (change data capture systems)

Achieving Data Consistency

- Duplicate some of the data and expose APIs for each microservice to be called by other services when data is changes (synchronous change in two databases)



- Achieves strong consistency at the cost of high coupling

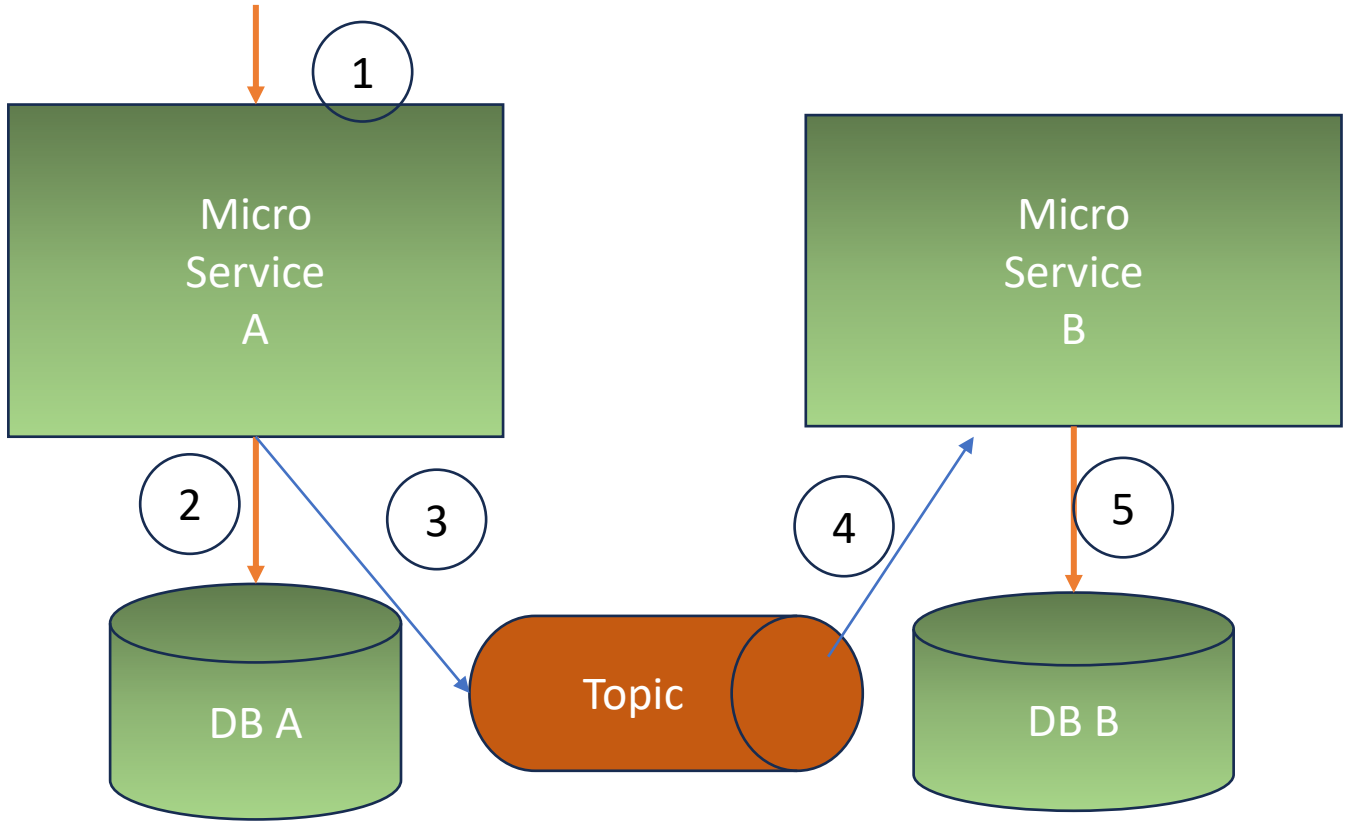
1 Microservice A receives a request to update a field from DB A. The value is also duplicated in DB B

Microservice A implements as a transaction the following steps:

- 2 call API in Microservice B
- 3 Microservice B updates DB B
- 4 On success, Microservice A updates DB A
- If any of the steps 2,3, or 4 fails, the changes to either DB A or DB B are rolled back

Achieving Data Consistency

- Event-driven architecture: publish changes in Message-Oriented-Middleware (e.g. message queues, topics). Microservice Components listen to changes and update data accordingly.



- Achieves weak/eventual consistency BUT has low coupling between components

1 Microservice A receives a request to update a field from DB A. The value is also duplicated in DB B

Microservice A implements as a transaction the following steps:

- 2 Update DB A
- 3 Publish change to Topic

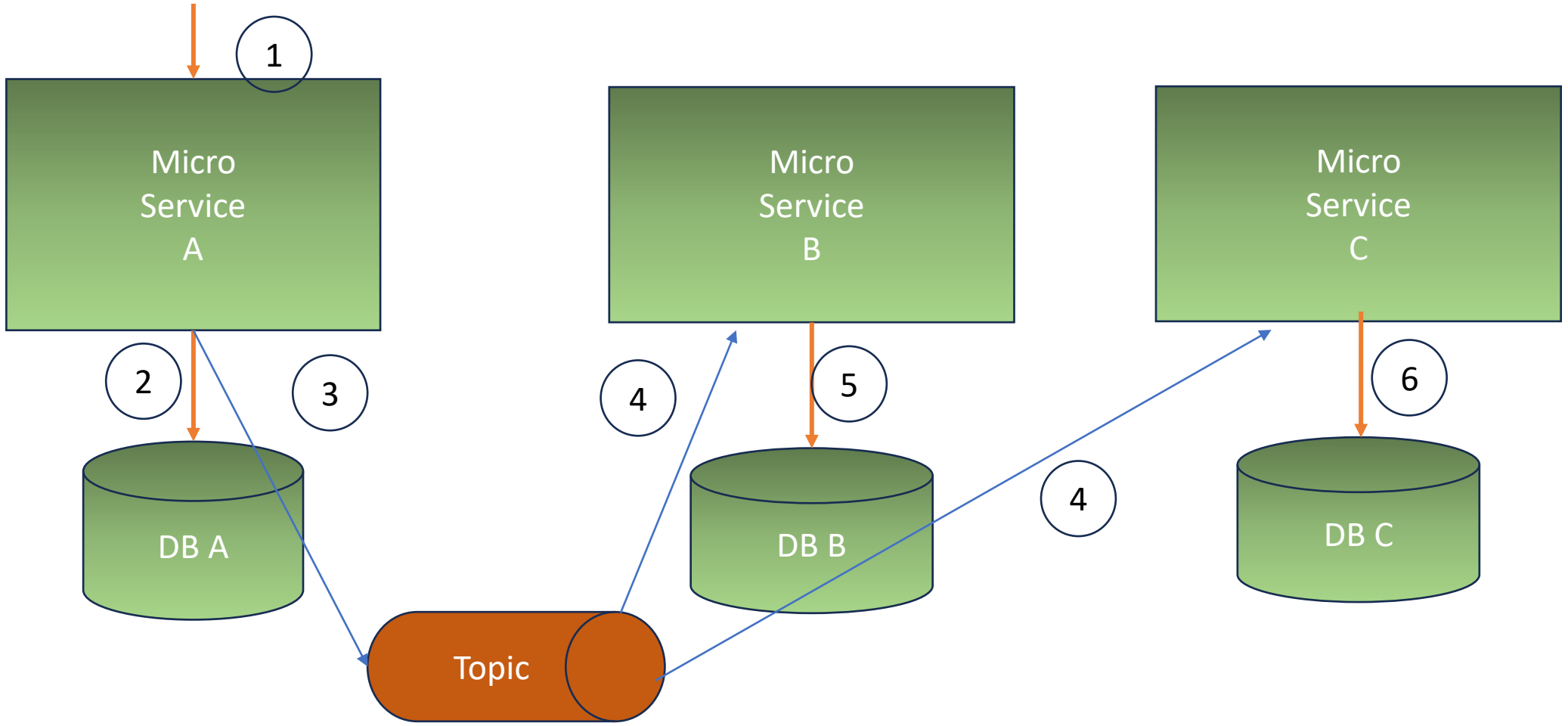
Microservice B listens to the topic:

- 4 Message consumer takes message with change
- 5 Update DB B

If any of the steps 3,4 or 5 fails, then DB A and DB B will not be consistent

Achieving Data Consistency

- Event – based architecture can work with:
  - Point-to-Point Communication (Queue-based) – changes sent to one service
  - Publish-Subscribe (Topic-Based) – changes sent to multiple services



Achieving Data Consistency

- Advanced techniques:
  - Distributed transactions (SAGA) – https://microservices.io/patterns/data/saga.html
    - A saga is a sequence of local transactions that updates the DBs and publishes events to start the subsequent local transaction in the saga.
    - If a transaction fails => the saga executes compensating transactions => undo the changes that were made by the preceding local transactions.
  - CDC Systems (change data capture systems)
    - Trail the databases log files (e.g. the binlog in MySQL) and publish corresponding events for each INSERT, UPDATE and DELETE.
    - E.g. Debezium – has connectors for MySQL, Postgres, MongoDB etc. It can be used with Apache Kafka

References

- https://en.wikipedia.org/wiki/Microservices
- https://microservices.io/
- https://softwareengineering.stackexchange.com/questions/373927/what-is-the-proper-way-to-synchronize-data-across-microservices
- https://microservices.io/patterns/data/saga.html
- https://stackoverflow.com/questions/37915326/how-to-keep-db-in-sync-when-using-microservices-architecture
- https://medium.com/@semrush_official/how-to-sync-data-between-different-databases-35c460f4ee63
- https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/distributed-data-management
- https://github.com/dotnet-architecture/eShopOnContainers
- https://dev.to/koenighotze/dealing-with-data-in-microservice-architectures-part-3-replication-4h7b