Challenges: Inconsistencies & CAP Theorem

In this lesson, we'll discuss some challenges that may arise when using asynchronous microservices.

WE'LL COVER THE FOLLOWING

Old events

If the communication infrastructure for event sourcing has to store old events, it has to handle considerable amounts of data. Consequently, if old events are missing, the state of a microservice can no longer be reconstructed from the events.

As an optimization, it would be possible to **delete events that are no longer relevant**. If a customer has moved to a different address several times, the last address is probably the only relevant one. The others can then be deleted.

In addition, it must also be possible to **continue processing old events**. If the schema of the events changes in the meantime, **old events have to be migrated**. Otherwise, every microservice has to be able to handle events in all old data formats.

This is particularly difficult if new data has to be contained in events that have not yet been saved in old events.

Inconsistency

Due to asynchronous communication, the system is not consistent. Some microservices already have certain information while others do not.

For example, *order process* might already have information about an order, but *invoicing* or *shipping* does not know about the order yet.

This problem cannot be solved. It takes time for asynchronous communication to reach all systems.

CAP theorem

These inconsistencies are not only practical problems but **cannot even be solved in theory**.

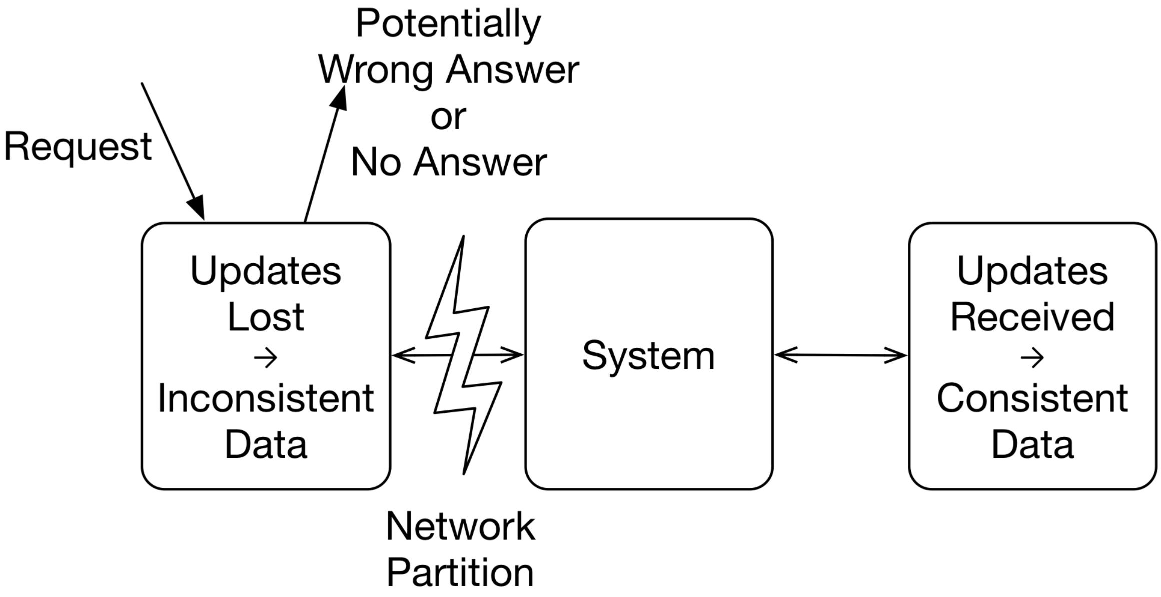
According to the CAP theorem, three characteristics exist in a distributed system:

- **Consistency (C)** means that all components of the system have the same information.
- **Availability (A)** means that no system stops working because another system failed.
- **Partition tolerance (P)** means that a system will continue to work in case of arbitrary package loss in the network.

> The **CAP theorem** states that a system can have a maximum of two features out of these three.

Partition tolerance is a special case. A system must react if the network fails. In fact, not even a complete failure is necessary; the package loss just needs to be high or the response time is very long. A system that responds very slowly is indistinguishable from a system that has failed completely.

Reasons for the CAP Theorem



If Communication Fails, a System Can Either Return a Potentially Wrong Response Upon a Request (AP) or None at All (CP)

There are **only two options** as to how a system can react to a request when the network is partitioned, see the drawing above.

1. **The system provides a response**. In this instance, the response can be wrong because changes have not reached the system; this is the *AP* case. The system is available, however, it might return a different response from systems that have obtained newer information. Thus, inconsistencies exist.

2. Alternatively, **the system returns no response**; this is the *CP* case. On the one hand, the system is not available when there is a problem. On the other hand, all systems always return the same responses and therefore are consistent as long as there is no network partitioning.

Compromises with CAP

Of course, you can make compromises. Let's take a **system with five replicas**:

- When writing, each replica confirms that the data has actually been written.
- When reading, several systems can be called to find out the latest state of the data.

Such a system with five replicas, in which one replica is read and only the

confirmation from one replica is waited for, **focuses on availability**. Up to four nodes can fail without the system failing.
However, it does not guarantee high consistency:

- The data could possibly be written to one node and – due to the time for replication to other nodes to occur – an old value is read from another node.

If the system with five replicas always waits for five nodes to be confirmed and always reads from five nodes, the data is always consistent.

However, the failure of a single node causes the system to become unavailable. A compromise can be to wait for confirmation of writes from three nodes and for reads from three nodes. In this way, inconsistencies can still be ruled out and the failure of up to two nodes can be compensated for.

CAP, events, and data replication

The CAP theorem actually considers data storage like NoSQL databases, which achieve performance and reliability via replication. But similar effects also occur when systems use events or data replication.

Ultimately, an event can be seen as a kind of data replication across multiple microservices. However, unlike the full replication of data between nodes of NoSQL databases, **each microservice can react differently to the event and may use only parts of the data**.

> A microservice that relies on asynchronous communication, events, and data replication corresponds to an **AP system**.

Microservices may not have received some events yet, so the data may be inconsistent. Nevertheless, the system can process requests using local data and is therefore available even if other systems fail.

> The CAP theorem says that the only alternative is a **CP system**. This would be consistent but not available.

For example, it could store the data in a central microservice which is accessible by all. As a result, **all microservices would receive the latest data.** However, **if the central microservice fails, all other microservices would no longer be available**.

QUIZ

1 Which of the following is NOT a part of the CAP theorem?

COMPLETED 0%

1 of 2

In the next lesson, we’ll continue discussing the challenges that involve inconsistencies.