Database per service, shared instance or shared database?

4# Tech and Beer, 18th October 2023

Francesca Motisi

Software Architect, WTM Ambassador

Francesca Motisi

@francesca_mts

mts88

Microservices Architecture

Is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices architecture diagrams and services independently.

Google Cloud Doc

In **2021** 75% of companies already uses microservices and 10% plans to migrate to.

Netflix and other bigs was “guilty”

Cloud friendly

Pinterest hits 1000 microservices in 2016

The most **misunderstood** architecture

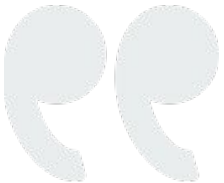
Everyone want to use microservices

The misunderstandings have become urban legends

It seems that databases topic, in microservices, is a kind of boogeyman for the most of architects or programmers — *it's scary and gives you nightmares*.

Let's start to busting a few myths…

Myth #1



Each microservice must use their own database, otherwise isn’t a microservice architecture.

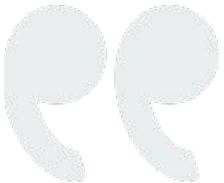
Remember: you should use different instance of DB and each instance should have a **replica** and **backup.**

It’s very **expensive** but is the **best** solution

if your client can’t sustain this infrastructure, **you shouldn’t use it**, instead is preferred a shared instance (still with replica and backups), or a shared database

Doesn’t matter if you use this solution or another, it’s still a microservice architecture, the important thing is **how to use it.**

Myth #2



Shared instance means share all data across microservices

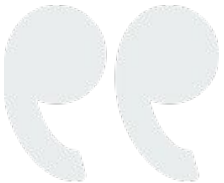
No!

Absolutely no!

Each microservice have a single responsibility and must access only to its part of data.

Shared instance doesn’t means shared database, are very different patterns.

Myth #3



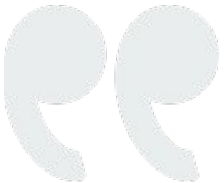
You can use only one kind of database

Nope. The microservices architecture is an agnostic technology solution.

You can use each kind of database you need to accomplish requested feature

As well as you can use each kind of programming language o framework you need.

Myth #4



You can’t mix database per service and shared instance solution

Technically you can mix these solutions, especially if you are going to change your structure.

Why you should?

If you are going to change your structure you should prepare the environment and release it, avoiding middle states.

How to handle database in microservices?

Now that some myths are busted we can talk about different patterns to persist data in microservices, with pros and cons.

What pattern?

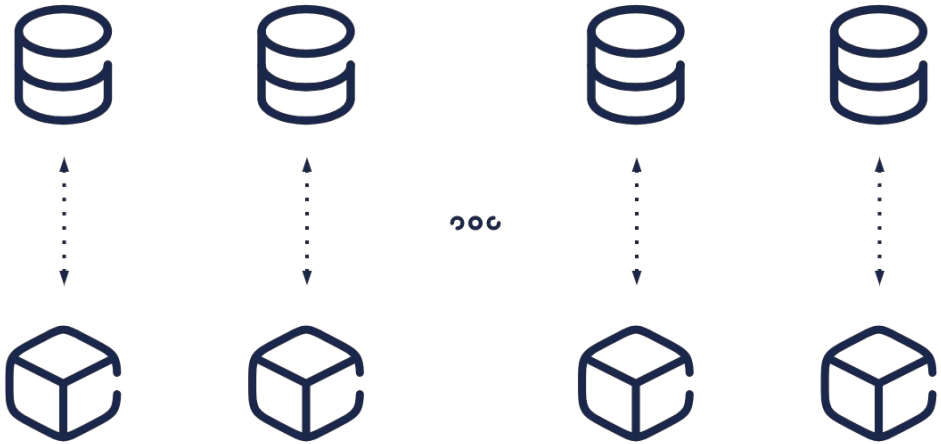
Database per service

Shared instance

Shared database

Database per service

Each microservice uses his own database instance to persists the data and, each database should have a replica and backups to grant high fault tolerance.



Database per service

It's a very expensive solution, but it's ***the solution***.

In practical example: we have five microservices with their constraints and boundaries, with this pattern you should have five databases instances, one for each microservice.

Using this pattern microservices can't crossing their boundaries and can access to **data** of others microservice using **API** or **CQRS**.

When you use a domain decomposition it's easy handle data but, in case of functional decomposition, it's a little bit more complicated because if a microservice needs data from multiple domains should maintains his local copy, having as results a different versions of the same data or, in worst case, a multiple variants of the same data.

Database per service

Pros

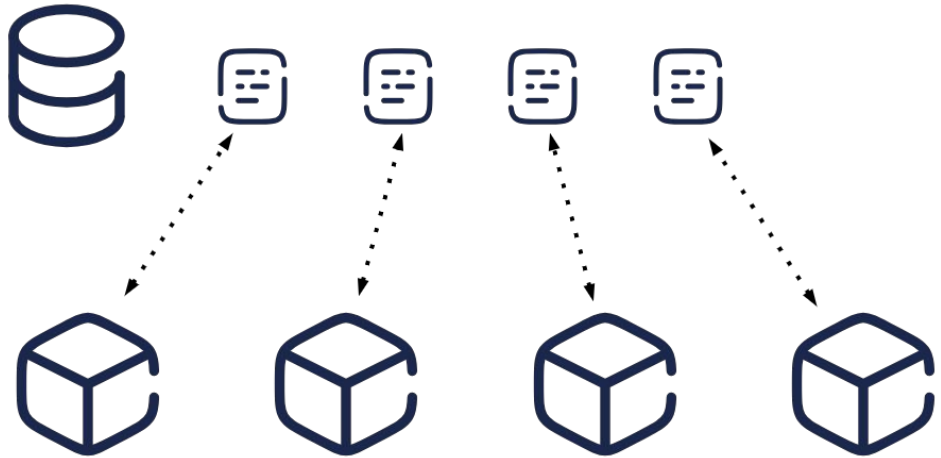
- High fault tolerance;
- Very flexible;
- Helps to reduce breaking changes;

Cons

- Expensive;
- Potentially problematic in a functional decomposition without help of message broker or CQRS;

Shared instance

This solution is based on database-per-service and microservices constraints are still mandatory, however each service doesn't have his database instance, it uses a shared instance.



Shared instance

Using the previous example, with this solution, we don’t have five instances of database, we are going to use one single instance and microservices still can’t access to others data directly, their still have to use internal API or others pattern

This is a low cost solution and in this way you can switch easily to the database-per-service pattern, if you need.

Shared instance

Pros

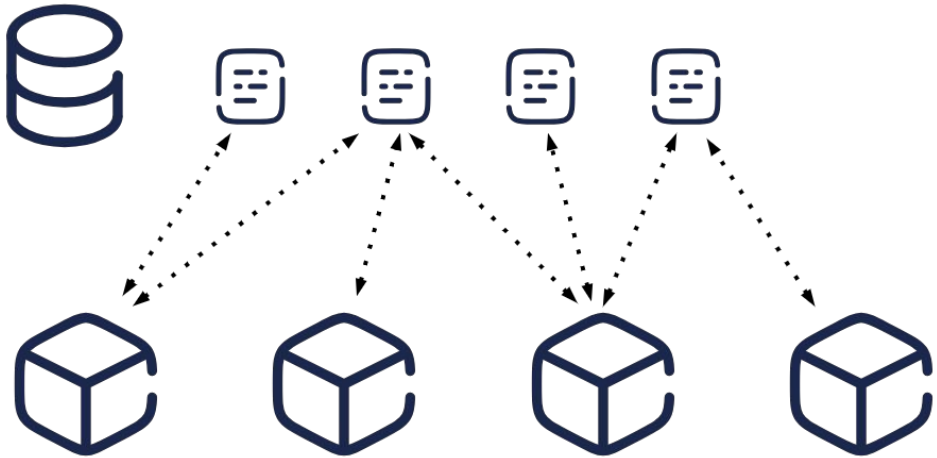
- Great for low budget project;
- Medium fault tolerance;
- Helps to reduce breaking changes;

Cons

- Not good for high fault tolerance;
- Sometimes you need to use multiple instance for multiple kind of DB (NoSQL, RDBMS, graphDB, etc…)

Shared database

I don’t like very much this approach, but in a few words this solution is used when you need to share data s between microservices, it’s without boundaries and constraints.



Shared database

Each microservice can access to the data directly and perform his transformation.

You have to handle concurrency and versioning on data.

This solution is very dangerous. For example if you need to change a table in the database you must check all microservices that use it, and it’s very easy making mistakes and create a lot of side effects.

Shared database

Shared database

Pros

- Easy to develop a small number of microservices;
- You can use ACID transactions, instead of Saga Pattern;

Cons

- High development time coupling;
- Not good for high fault tolerance;
- High chance of breaking changes and side effects;
- You have to handle lock, versioning and concurrency on data;
- It’s an anti-pattern;

Summary

Shared instance is a good trade-off

There isn't a golden rule: solutions are different by cases

Please… don't use **shared database** :D

- https://medium.com/@mts88/database-per-service-or-shared-database-e73cfb756aa1
- https://microservices.io/patterns/data/database-per-service.html
- https://microservices.io/patterns/data/shared-database.html

Happy **beer**

Thank you

That's all folks!

Francesca Motisi

@francesca_mts

mts88