Microservices

Migrating cloud-native applications from monolithic to microservices architectural pattern

What are Microservices?

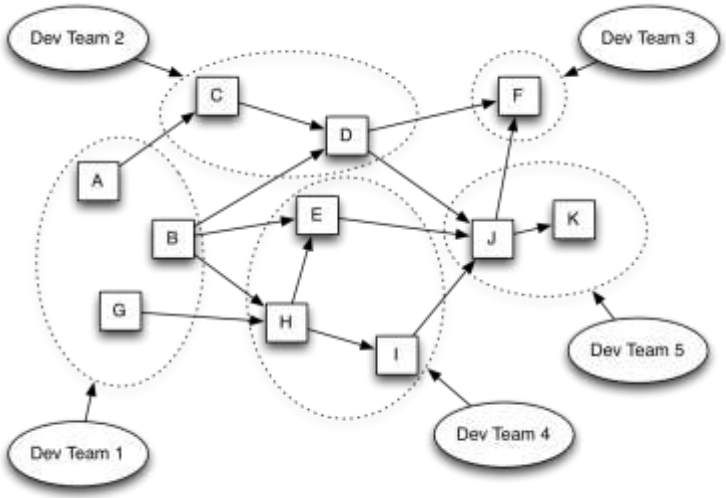
Underlying principle

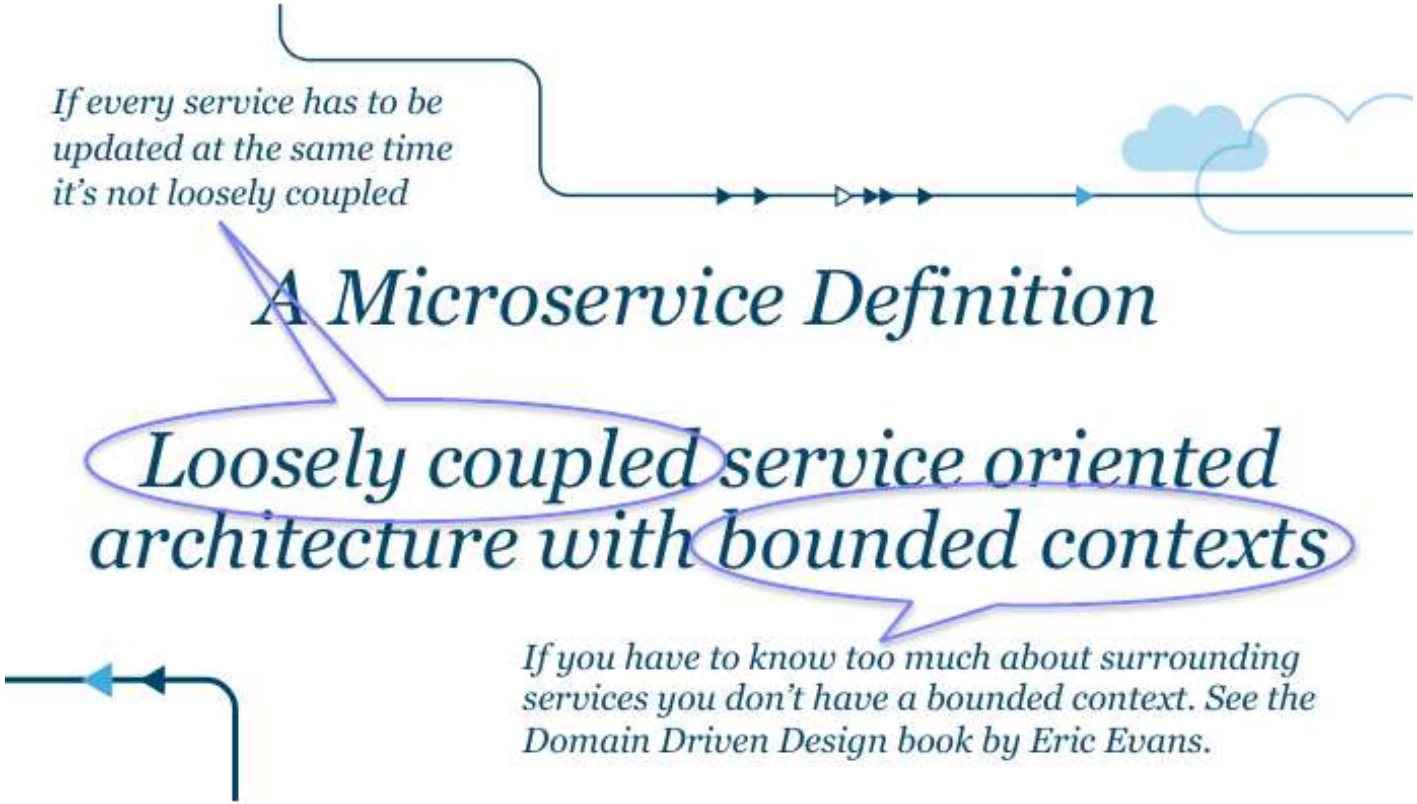
"functional system decomposition into manageable and independently deployable components"

Microservices usually managed by a single development team (5-7 members).

- Functional system decomposition means vertical slicing (in contrast to horizontal slicing through layers)

- Independent deployability implies no shared state and inter-process communication (often via HTTP REST-ish interfaces)

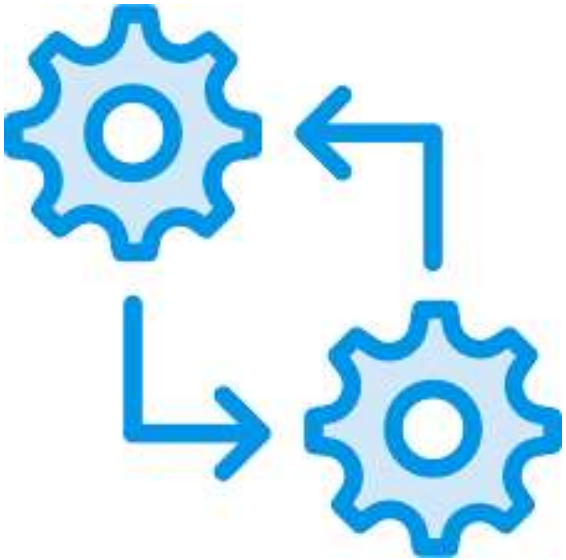



If every service has to be updated at the same time it's not loosely coupled
A Microservice Definition
Loosely coupled service oriented architecture with bounded contexts
If you have to know too much about surrounding services you don't have a bounded context. See the Domain Driven Design book by Eric Evans.

More specifically

Each microservice is functionally complete with

- Resource allocation
- Database consistency
- Each microservice handles one resource (or verb), e.g.
  - Clients
  - Sale items
  - Checkouts
- Checkout Microservices are fun-sized services, as in “still fun to develop and deploy”



Independant deployability

It enables separation and independent evolution of

- code base
- technology stacks
- scaling
- and features, too

Each service has its own software repository

- Codebase is maintainable for developers – it fits into their brain
- Tools work fast – building, testing, refactoring code takes seconds
- Service startup only takes seconds
- No accidental cross-dependencies between code bases

Independent technology stacks

Independant services can have independant technology stacks

The technology stack is selected as a best-fit to the task

Teams can also experiment with new technologies within a single microservice

No system-wide standardized technology stack also means

No piggy-pack dependencies to unnecessary technologies or libraries

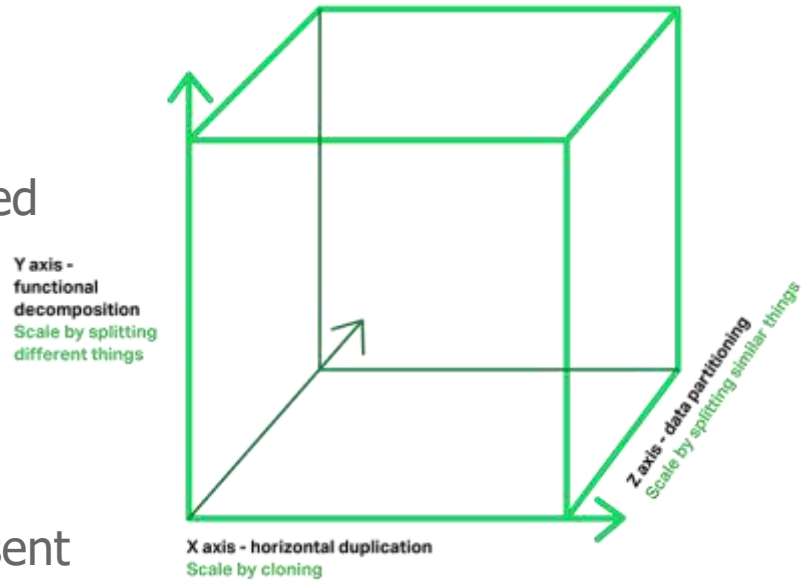
Selected technology stacks are often very lightweight

A microservice is often just a single process that is started via command line, and not code and configuration that is deployed to a container

Independent Scaling

Each microservice can be scaled independently

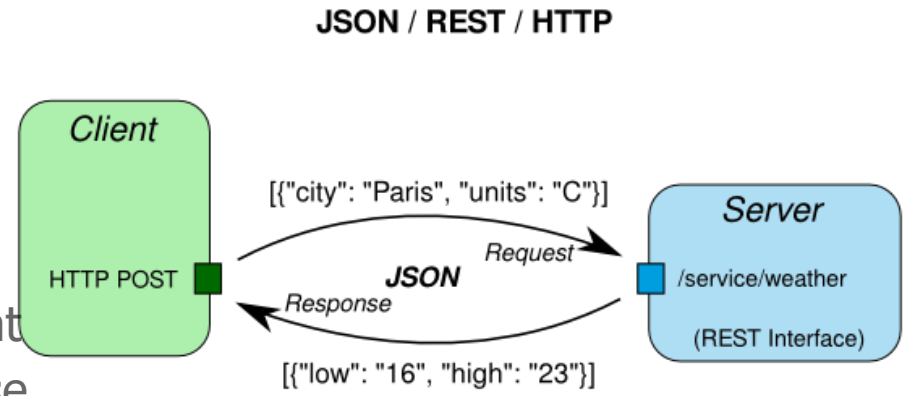
- Identified bottlenecks can be addressed directly
- Data sharding can be applied to microservices as needed
- Parts of the system that do not represent bottlenecks can remain simple and un-scaled



Stable Interfaces – standardized communication

Communication between microservices is often standardized using

- HTTP(S)
- REST
- JSON – simple data representation format
  - REST and JSON are convenient because they simplify interface evolution (more on this later)



Microservices Architecture

Characteristics of Microservices Architecture

Componentization via services

Organized around business capabilties

Products not Projects

Smart endpoints and dump pipes

Decentralized Governance

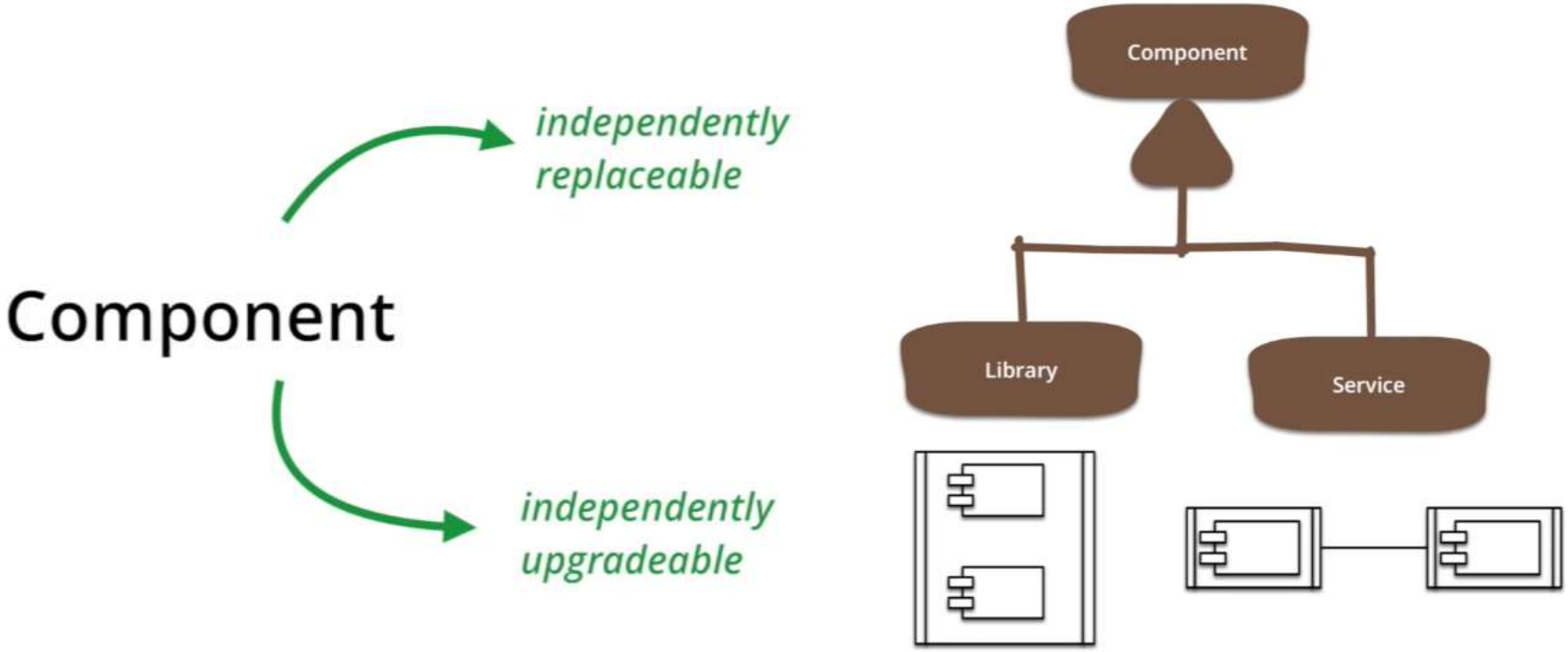
Decentralized Data Management

Infrastructure Automation

Design for Failure

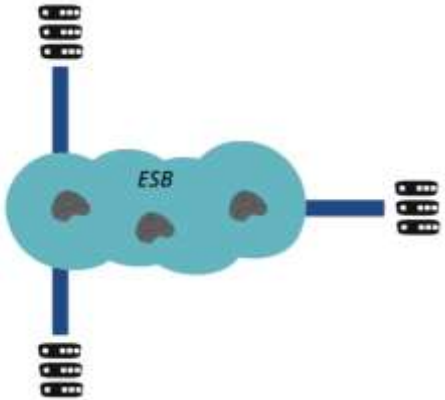
Evolutionary Design

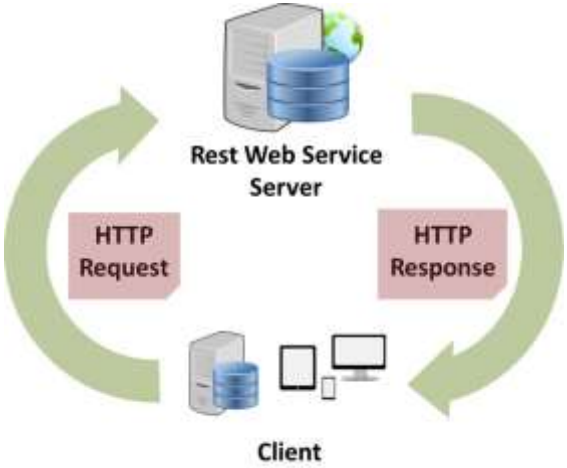
Componentization via services



Organized around business capabilities

Smart endpoints and dumb pipes





Rather than stressing importance on the communication mechanism itself (for eg. Enterprise Service Bus, which produces sophisticated facilties for message routing, choreography, transformation and applying business rules)

Microservices adopt a dumb pipe (ie. only routing mechanism) and request-response mechanisms using REST and HTTP.

Decentralized Governance

Principle: focus on standardizing the relevant parts, and leverage battle-tested standards and infrastructure Treats differently

- What needs to be standardized
  - Communication protocol (HTTP)
  - Message format (JSON)
- What should be standardized
  - Communication patterns (REST)
- What doesn‘t need to be standardized
  - Application technology stack

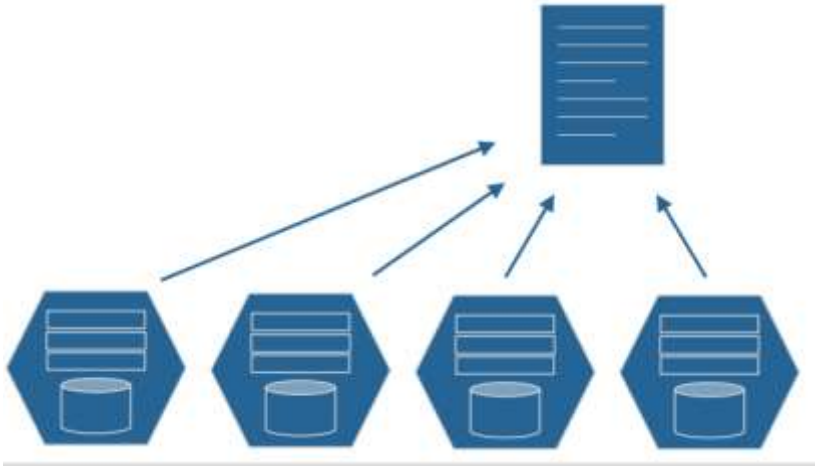
Infrastructure Automation

Having to deploy significant number of services forces operations to automate the infrastructure for

- Deployment (Continuous Delivery)
- Monitoring (Automated failure detection)
- Managing (Automated failure recovery)

Consider that:

- Amazon AWS is primarily an internal service
- Netflix uses Chaos Monkey to further enforce



Are Microservices just SOA ?

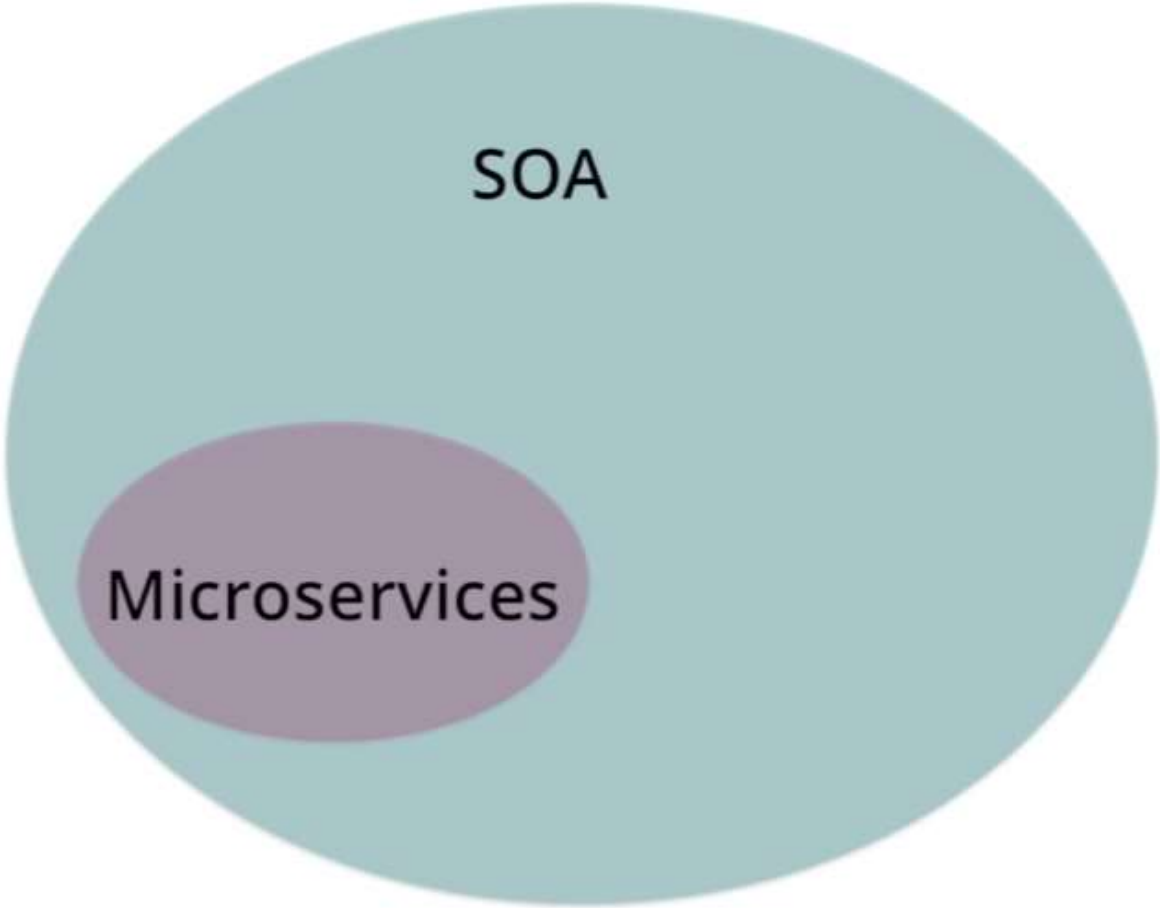
Service Oriented Architecture

Functional decomposition is also adopted by SOA:

- Not usually required to be self contained within Data and UI.
- It is often thought as decomposition within tiers, and introducing another tier – the service orchestration tier

In comparison to microservices

- Focused on enabling business-level programming through business processing engines and languages.
- Does not focus on independent deployment units and its consequences
- Microservices can be seen as the “good-parts” of SOA.


SOA
Microservices

Conclusion

Should we simply adopt it?

No. But they are a possible evolution for web systems due to rise in Cloud Computing.

Community often complains about existing challenges, prerequisites and difficulties.

Just the new fad?

Maybe/maybe not. Microservices is a new term, and an evolution of long-known architectural principles applied in a specific way to a specific type of systems.

Dev-ops-heavy, and less managerial.

Due to open source tech landscape, there is faster growth and development.

That’s all folks!