Microservices

Agenda

- What is microservices?
- From monolith to microservices
- Benefits and Challenges
- Patterns
- Containers

What is microservices?

What is microservices?

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage.

- James Lewis & Martin Fowler

What is microservices?

Approach to Application Development based on a set of modular services

What is microservices?

Each service is fully independent, fine-grained and self-contained

What is microservices?

Each service supports a specific business goal with each own logic and data

What is microservices?

exposes a well-defined interface to communicate with other services and external actors

What is microservices?

Embracing cross-platform, each can be written with a different programming platform and encapsulates its own data

What is microservices?

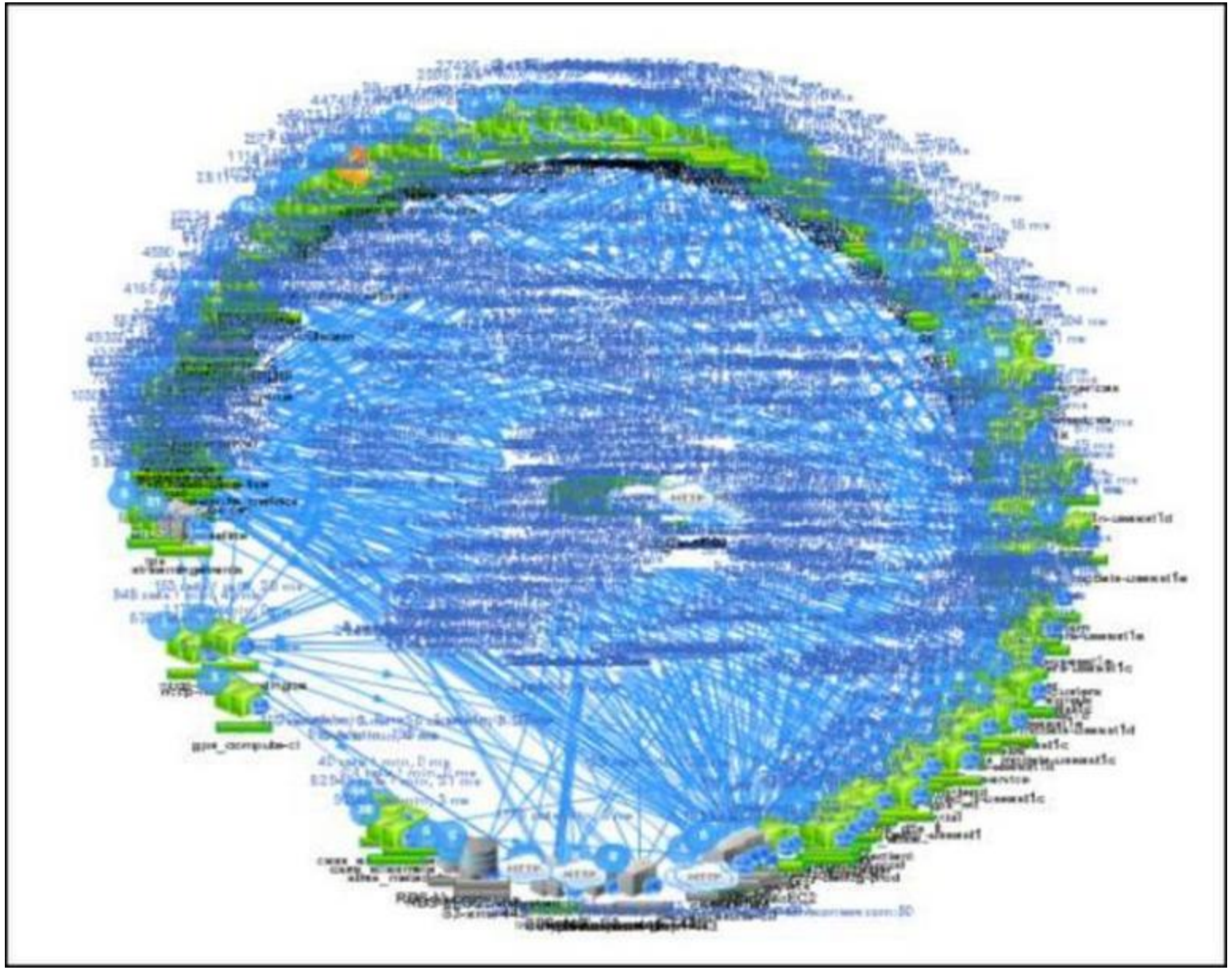
Each service can deploy frequently and evolve independently maintain its interface

Microservices != Containers

- Microservice: Isolated service with a single purpose
- Container: Deployment unit
- Microservices tends to use containers as unit of Deployment but is not mandatory
- Pros:
  - Higher degree of isolation and portability
  - Faster to start-up and execute
  - Usage of orchestrators (like Kubernetes) makes the process of managing Applications based on microservices more agile

Example of Microservices Architecture

Netflix

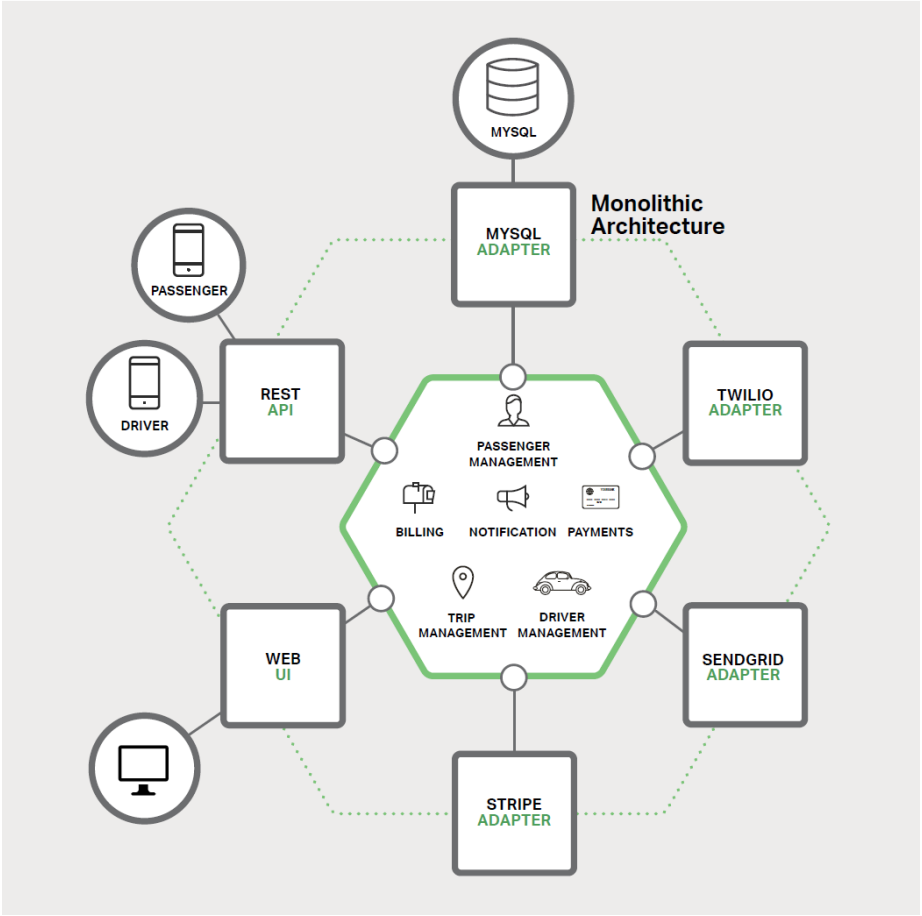


Mastering Chaos - A Netflix Guide to Microservices: https://www.youtube.com/watch?v=CZ3wIuvmHeM

From monolith to microservices

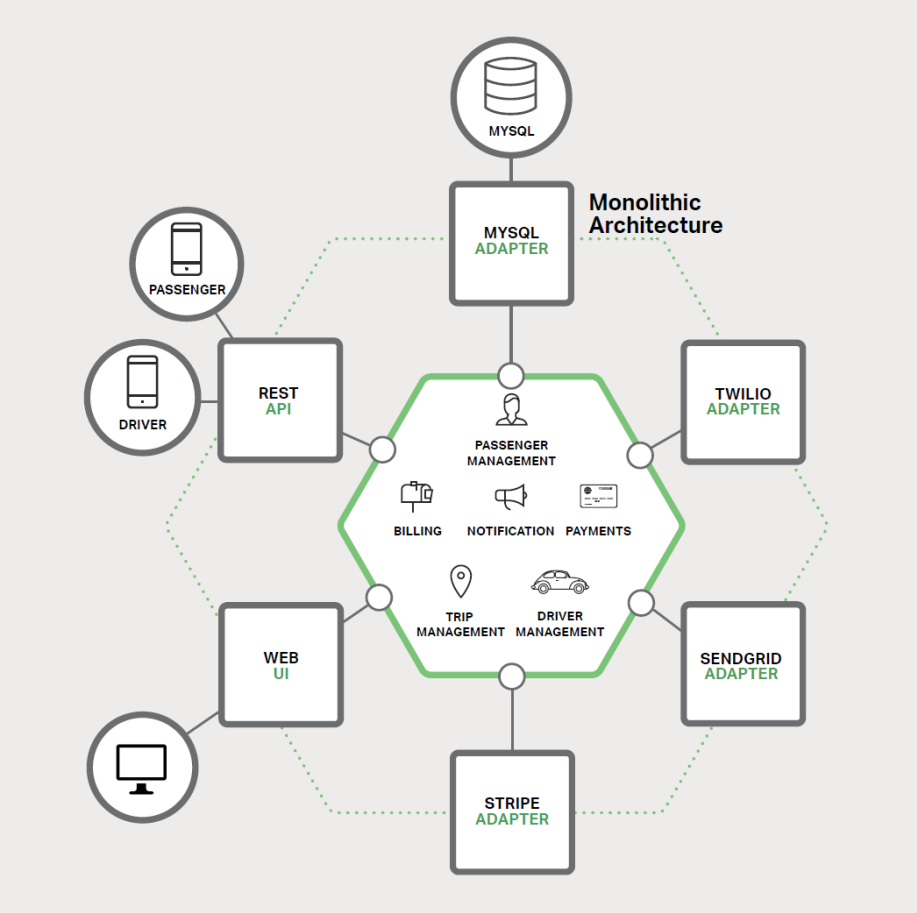
Application Architecture

- Using traditional guidelines and best practices
- Application with **single core component** including all business logic and workflows
- **Layer of adapters** to integrate with outsider world
  - Data access
  - Messaging
  - UI
- All components on a **single package** and deploy of all applications
- Takeaways
  - Layered application but no tiered
  - Runs in single process



Application Architecture

- We've created a monolith!
- Can be straightforward to
  - Build
  - Test
  - Deploy
  - Troubleshoot
- Even performance can be good (single process)
- Many large and successful applications that exists today started as monolith (Netflix, SoundCloud, …)
- But as soon as the Applications needs to evolve…

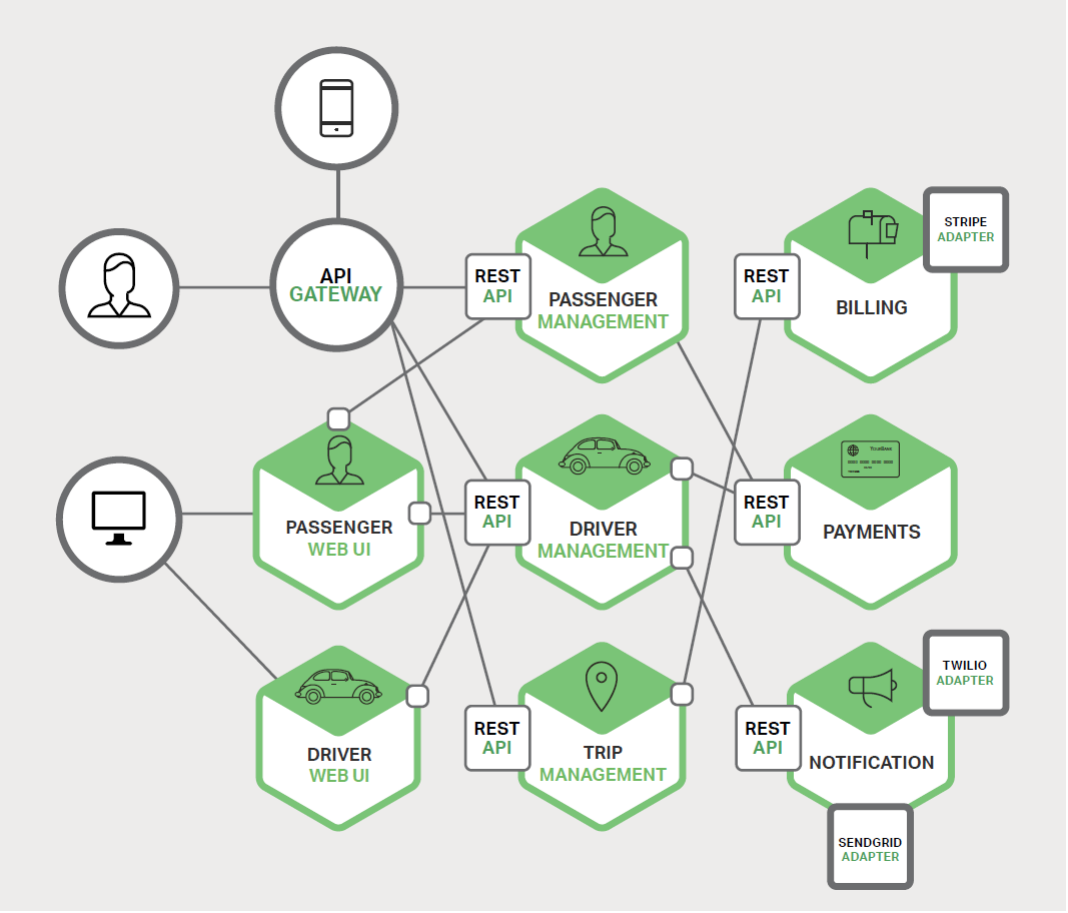


When application starts to evolve…

- Evolution process shows lack of agility, entering on a “Fear Cycle”
  - The app has become overwhelmingly complicated
  - You fear making changes - small changes have unpredictable and costly results
  - Every change is as small as possible
  - New features/fixes become tricky, time-consuming and expensive to implement
  - Each requires a full deployment of the entire application
  - One unstable component can crash the entire application
  - Implementing new technologies and frameworks become difficult, at best

Microservices architecture

- Split app across a set of small isolated microservices
- Each is self-contained and includes its own code, data and state
- Each exposes a RESTful service and consumed by front-end application and/or other services
- Orchestration for microservices using patterns as API Gateway

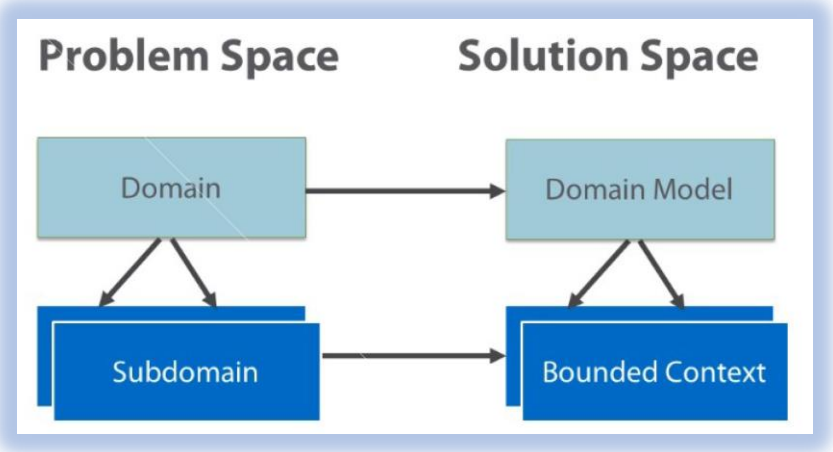


Microservices != SOA

Microservices	SOA
Small, independent processes communicating with each outer using language-agnostic APIs	Often leverages Enterprise Service Bus architecture for point-to-point communication
Services are small and independent (scoped to Bounded Contexts)	Large services being only an abstraction from the monolith
Services are independently deployable	Services are tightly-coupled and deployed as large units
Embrace decentralized data storage	Centralize data storage

Bounded Contexts

- How to split applications as Microservices?
- Defining solution domain allow to split in several sub-domains
- Each sub-domain defines it own entities, rules and flows
- The boundary is defined using Bounded Context strategy (based on human cells)
  - Entities, rules and flows are maintained internally (applicational logic)
  - Boundary is “opened” only for who and what we want to let came in (APIs)



Benefits and Challanges

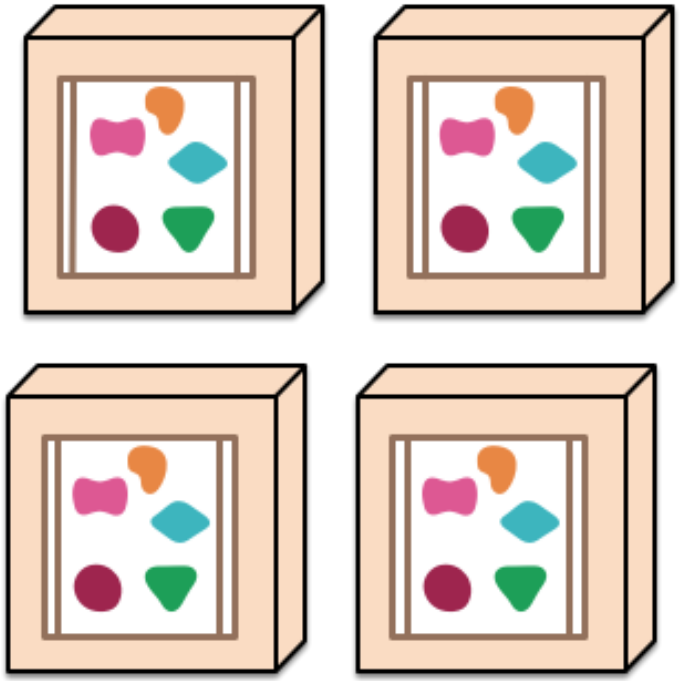
Benefits

- Cross-Platform Development
  - Tecnology, planning
- Deploy, update and maintenance completely independent
- Smaller and more focused teams
- Selecting technology stacks are not tied to a global decision
- Fault isolation
- Granular scalability

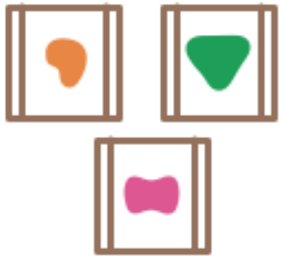
Scalability

A monolithic application puts all its functionality into a single process...

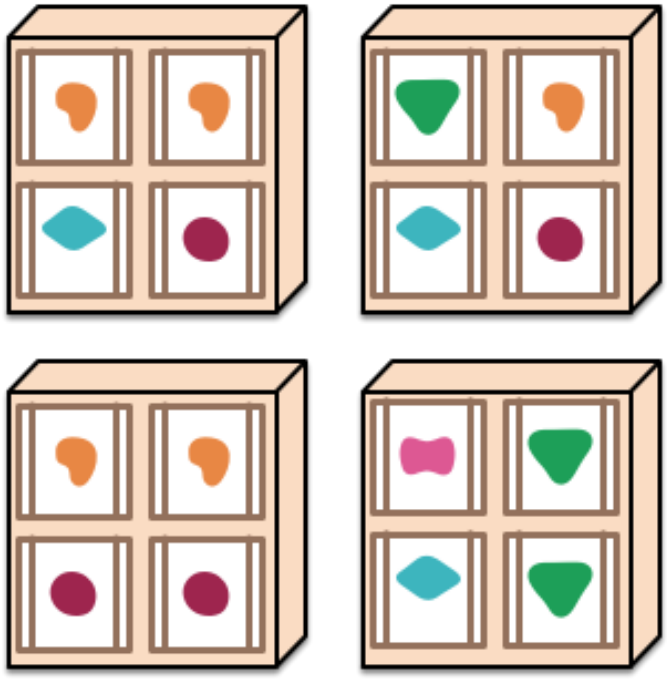
... and scales by replicating the monolith on multiple servers



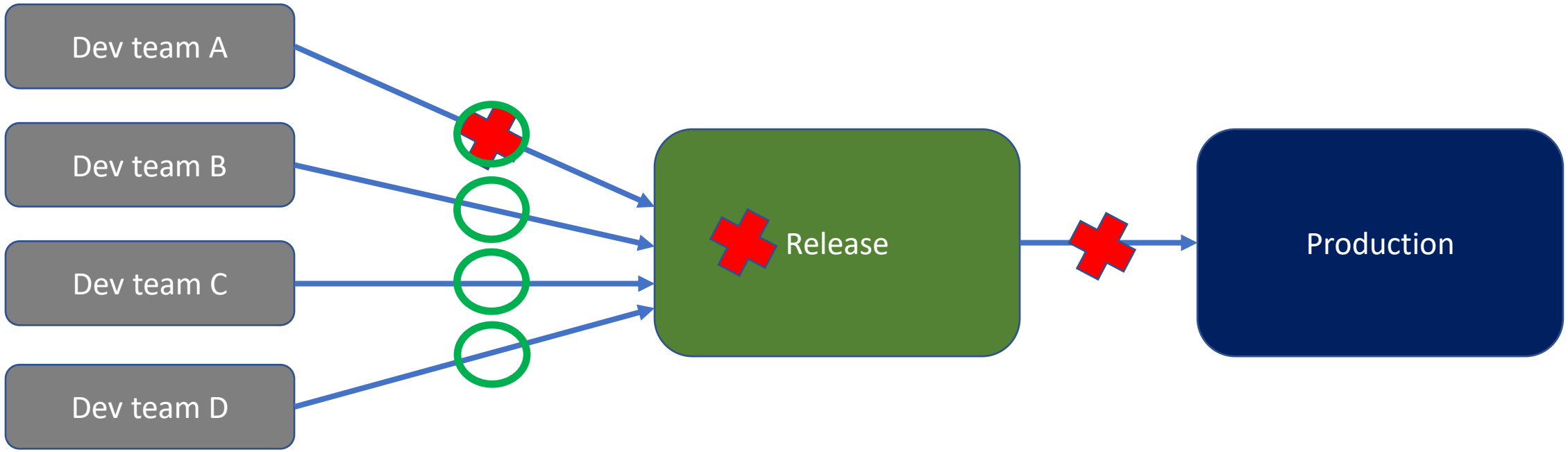
A microservices architecture puts each element of functionality into a separate service...



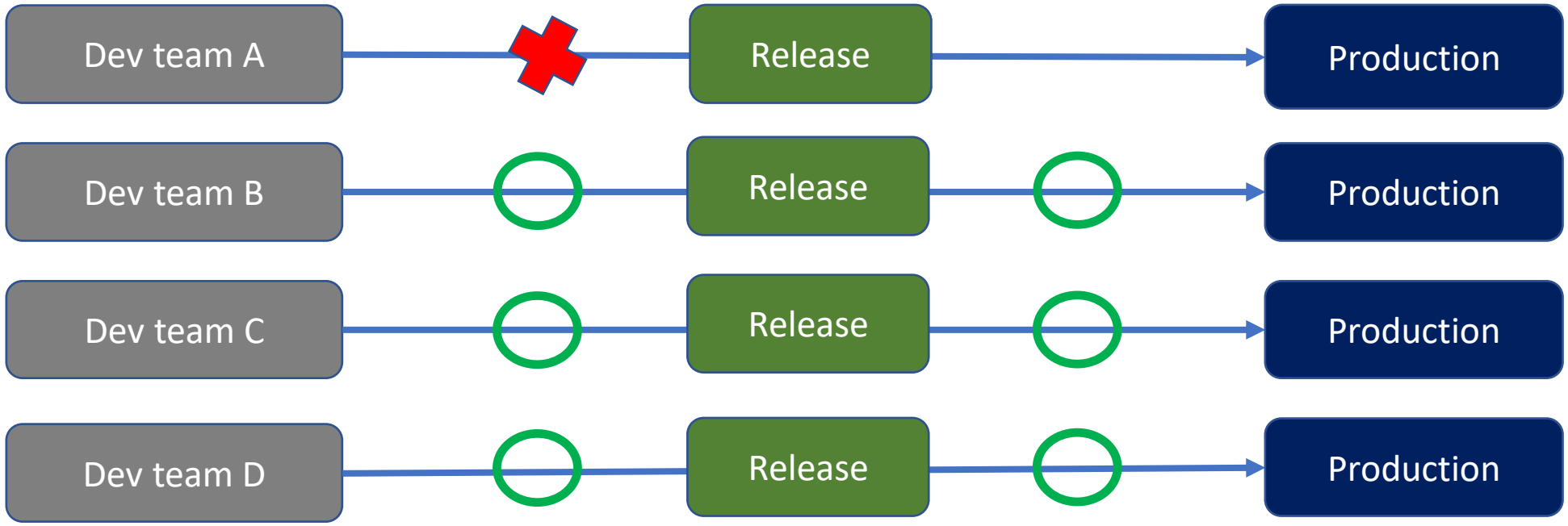
... and scales by distributing these services across servers, replicating as needed.



Fault Isolation: Monolith



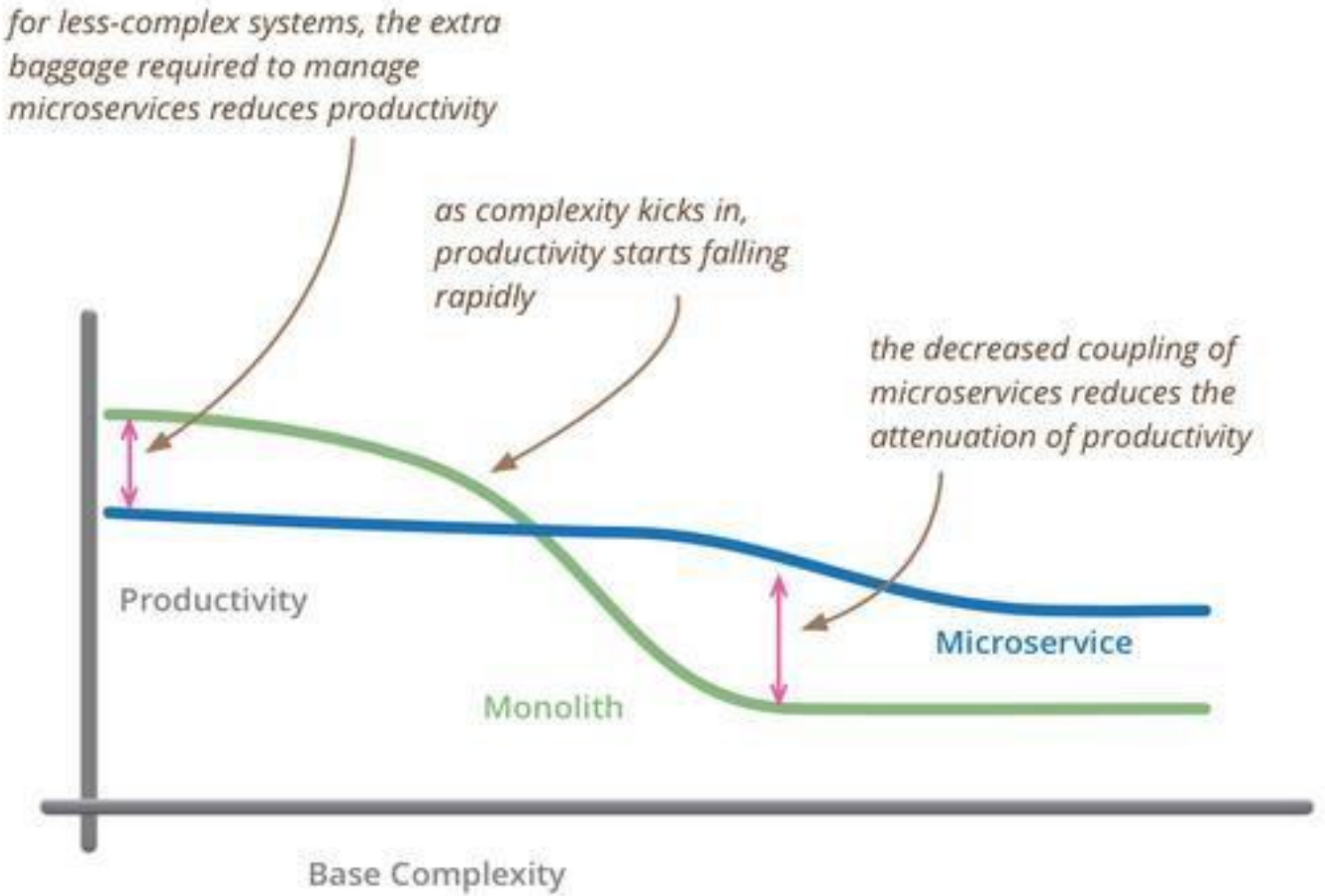
Fault Isolation: Microservices



Challenges

- Operational and Development complexity
  - Testing, integration, ….
- Definition of services boundaries
- Data consistency and integrity
- Service Discovery
- Communication
  - Probably the biggest challenge of these architectures
  - Internal and external Communications
  - Congestion and latency
- DevOps
  - Deployments
  - Evolution and versioning

Challenges



but remember the skill of the team will outweigh any monolith/microservice choice

One-size-fits-all solution?

- Several Solutions don’t fit well with microservices architecture
- Solution with proven records
  - Large Enterprise Systems with big business alignment needs
  - Systems with high evolution velocity
  - Application developed by heterogenous teams
  - Applications with clear different component needs in terms of scalability

Usage of Microservices

120+ services to provide a single page

Changes in production every 11.6 seconds

600+ services in production

Hundreds of deployments each day

1000+ services supported by 8000+ GIT repos

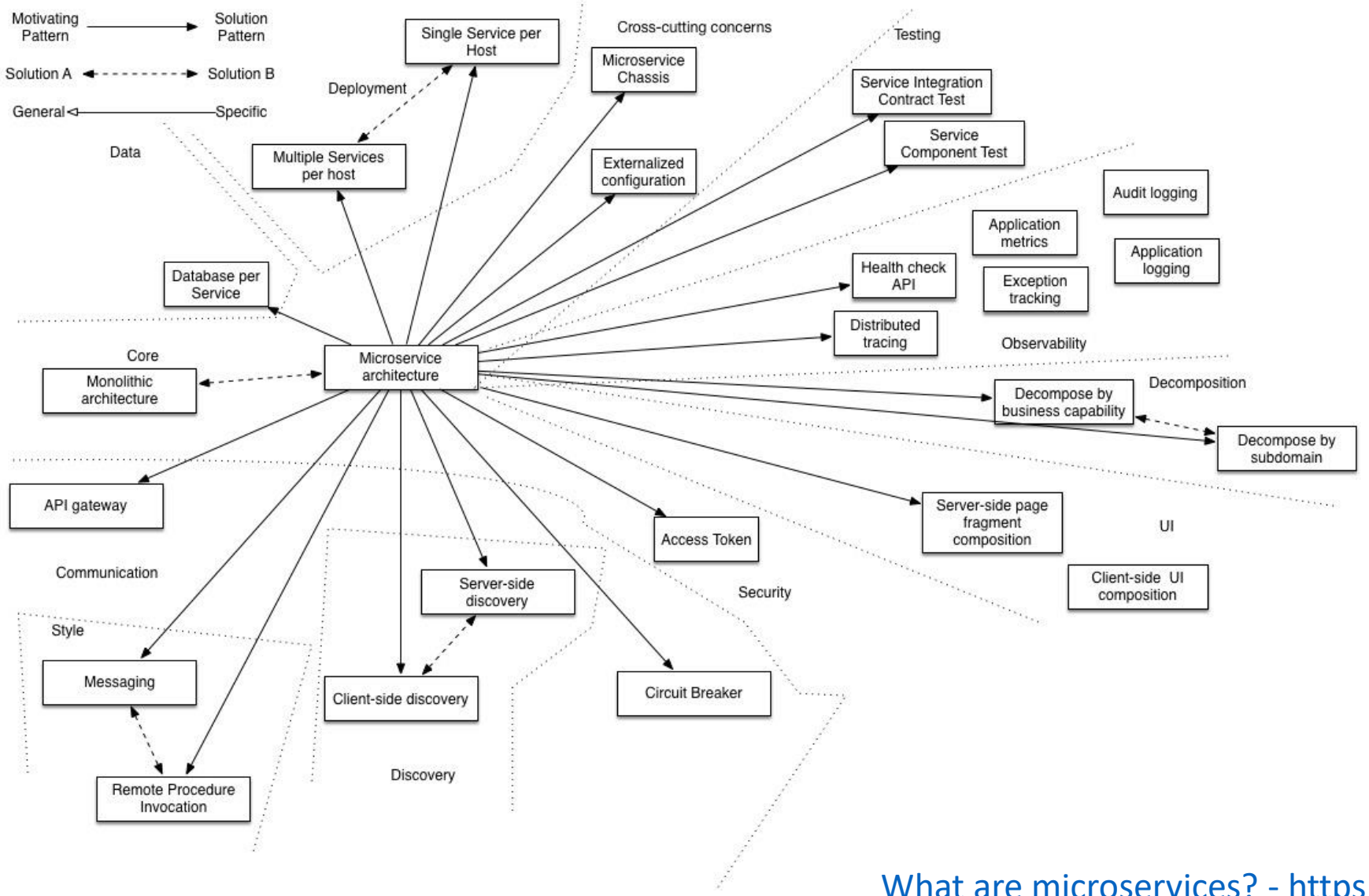
Deploys each minute during business hours

Patterns

Why Patterns?

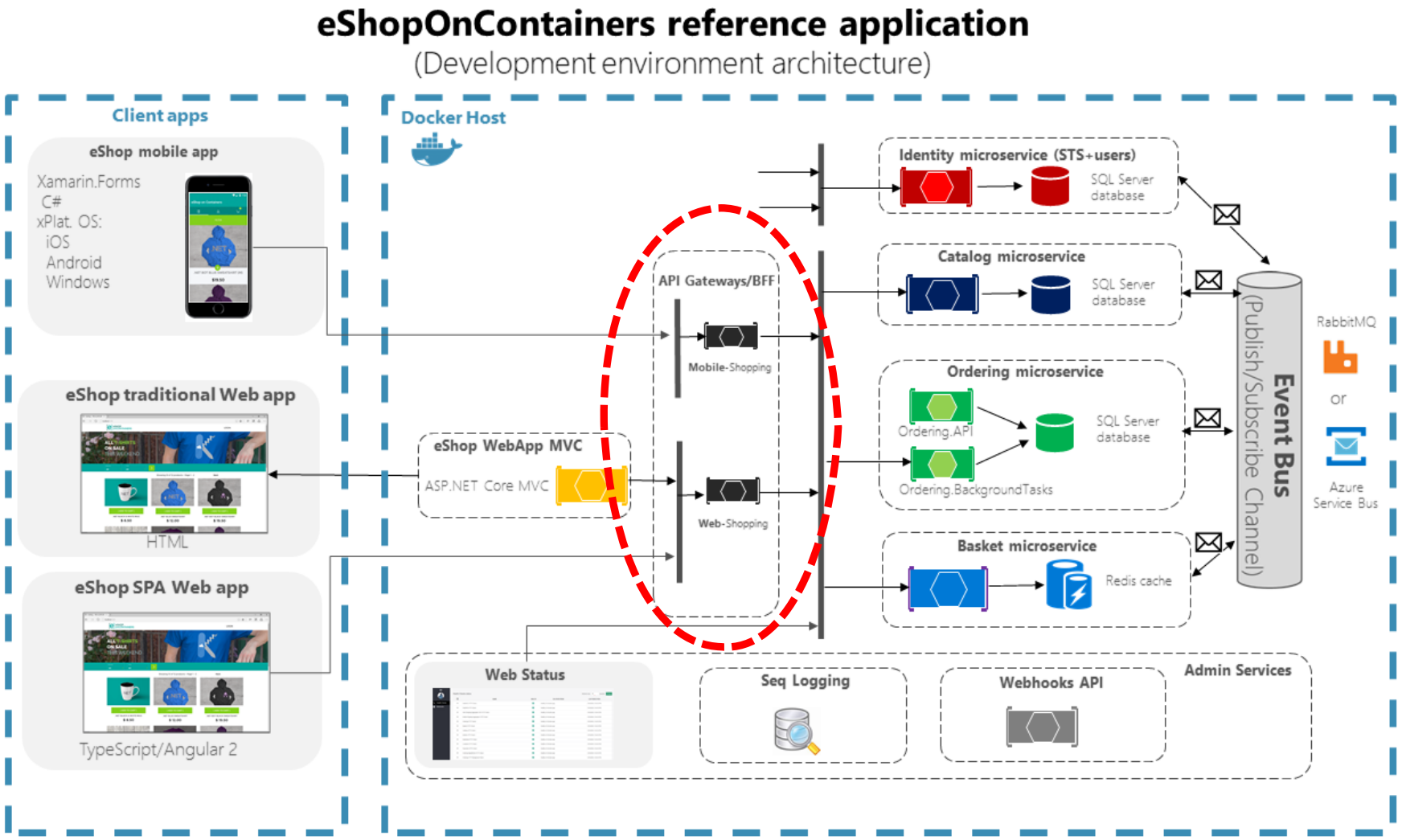
- The problem you have probably someone else already had it
- A great community works on this common issues using microservices
- Using a pattern is using something that already have proven records of working well to tackle your problem
- A lot of libraries/frameworks already have this pattern implemented OOB and easy to use

How many Patterns?



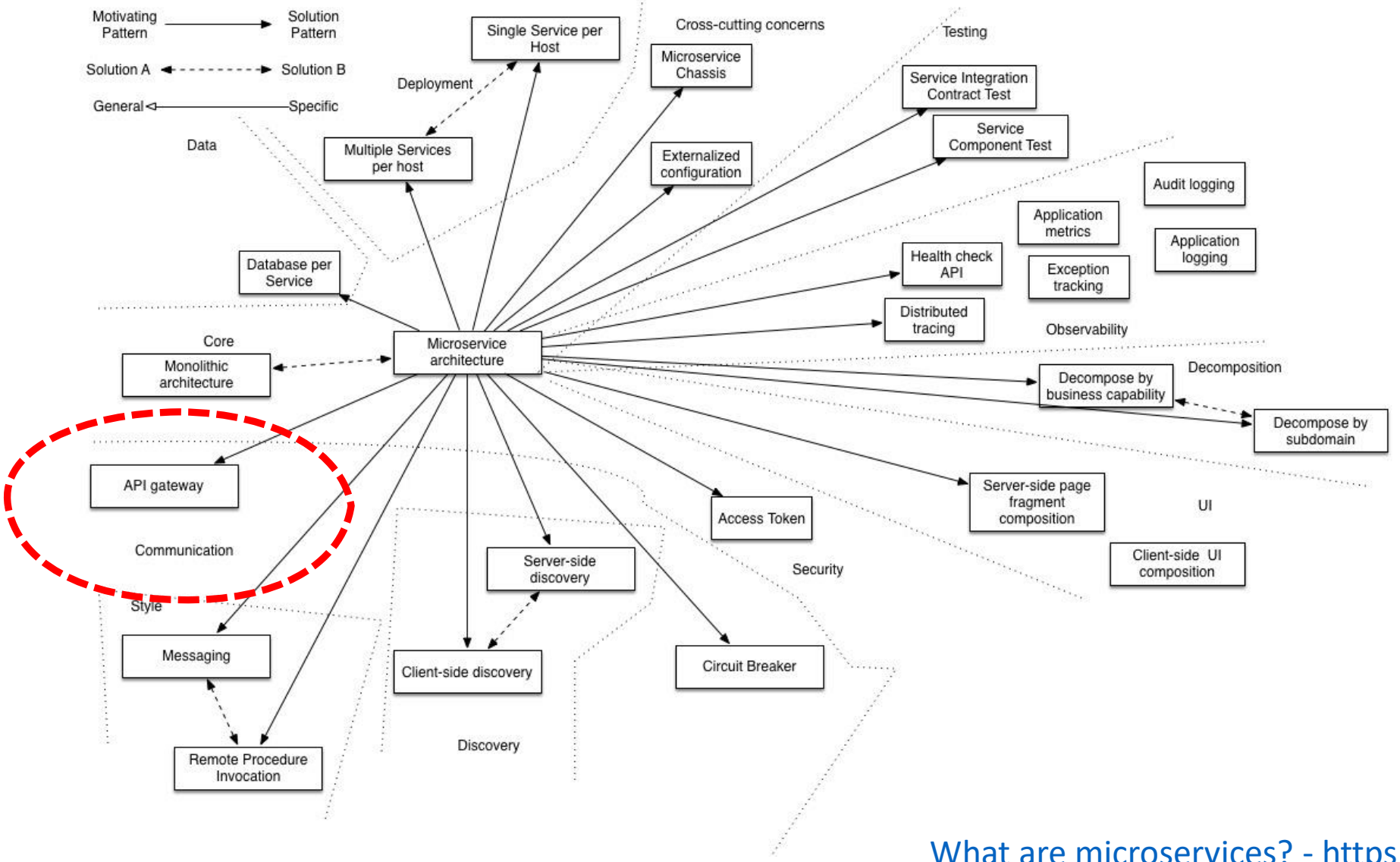
[What are microservices? - https://microservices.io/](https://microservices.io/)

Example of an architecure



https://github.com/dotnet-architecture/eShopOnContainers

Example of a pattern



What are microservices? - https://microservices.io/

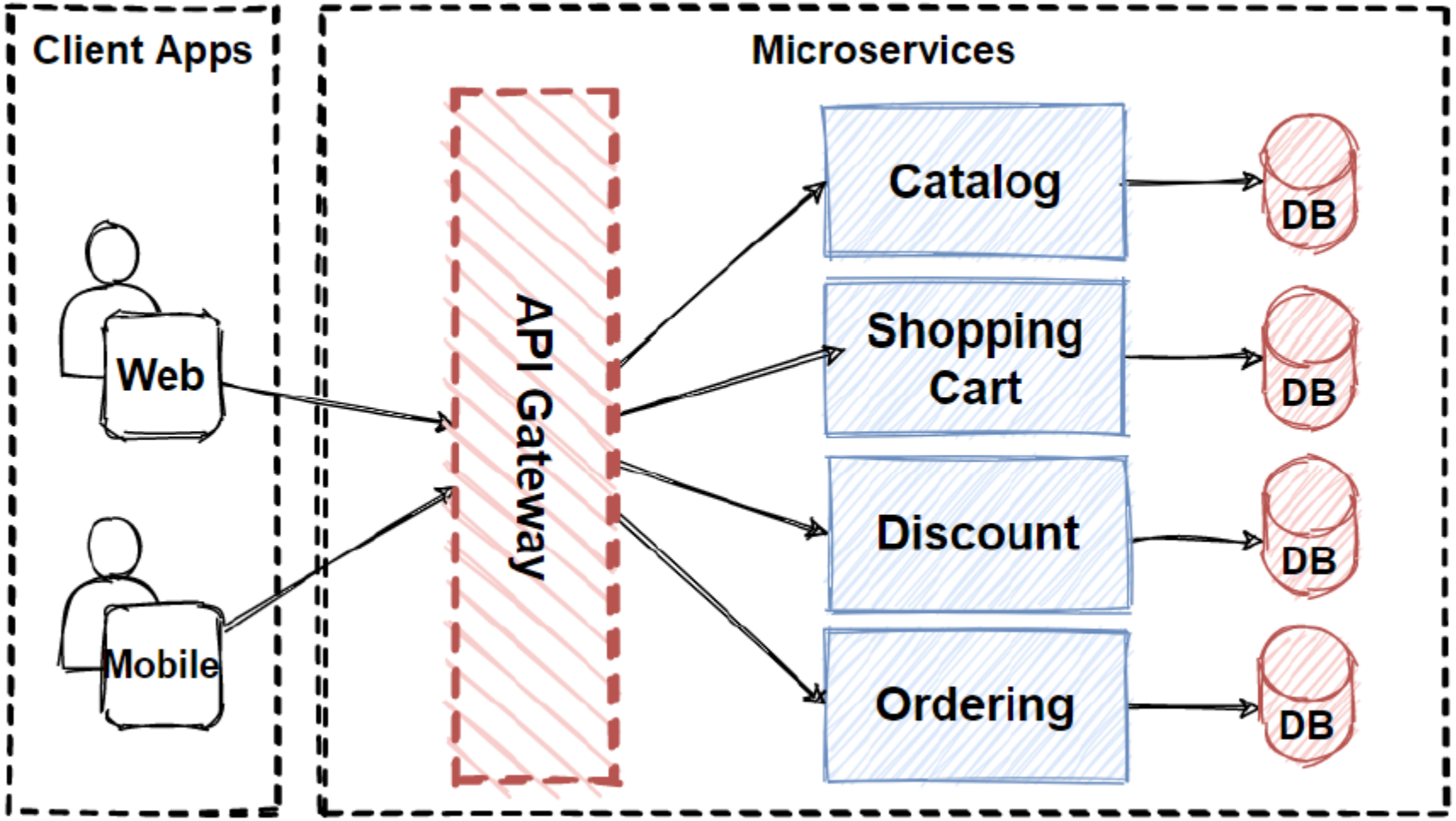
API Gateway: Problem

- How do the clients of a Microservices-based application access the individual services?

API Gateway: Forces

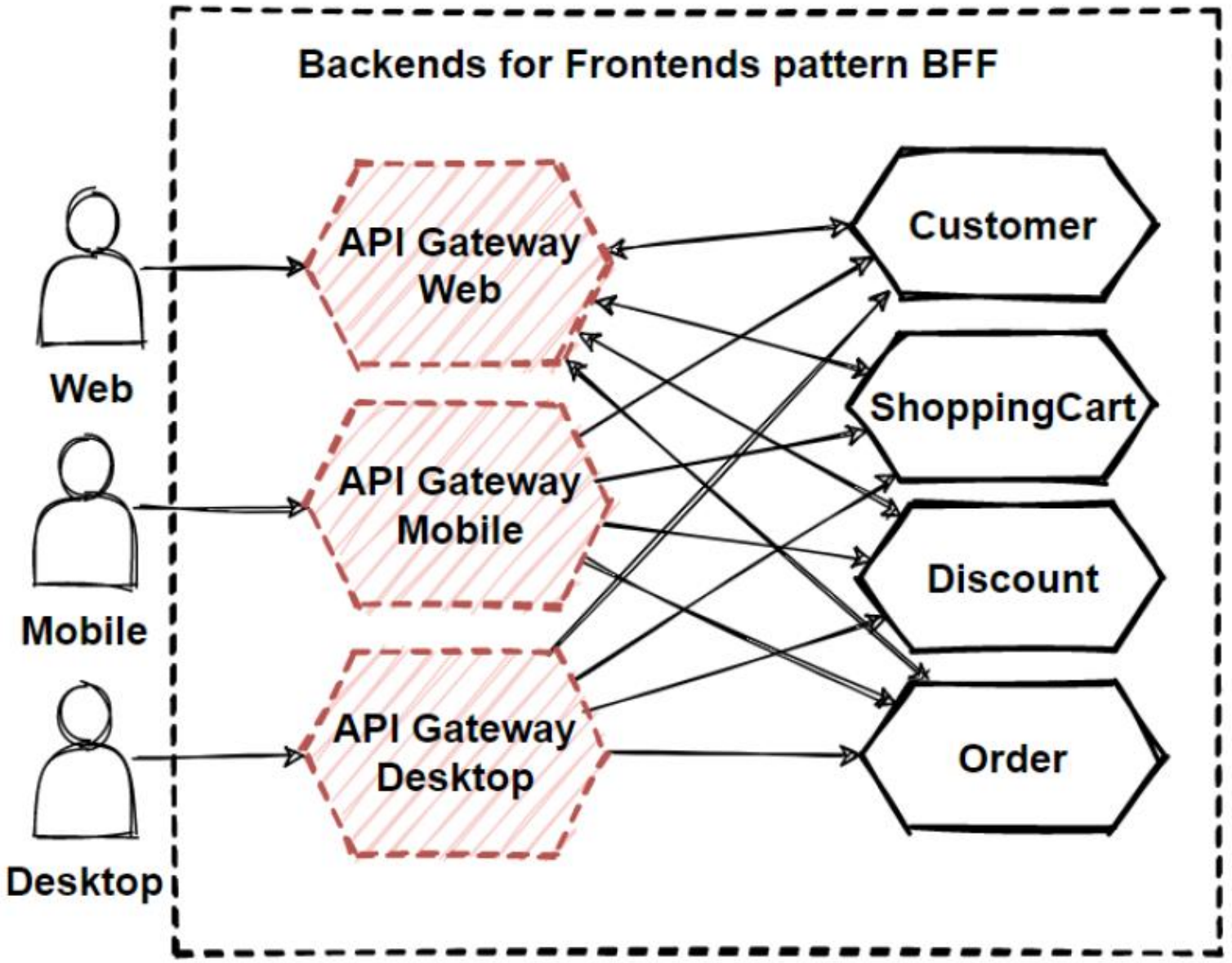
- The **granularity** of APIs provided by microservices is often different than what a client needs. Microservices typically provide fine-grained APIs, which means that clients need to interact with multiple services.
- **Different clients** need different data
- **Network performance** is different for different types of clients
- The **number of service instances** and their locations (host+port) changes dynamically
- **Partitioning** into services can change over time and should be hidden from clients
- Services might use a **diverse set of protocols**, some of which might not be web friendly

API Gateway: Solution



API Gateway Variation

Backends for Frontends



Q&A