HAI704I Architecture
Logicielle Distribuée

Microservice-Oriented Architecture (MSA)

Pascal Zaragoza

Resources

1. Eberhard Wolff. (n.d.). ***Microservices : flexible software architecture***.
2. Richardson, C. (2018). ***Microservices Patterns : With examples in Java***.
   - *https://microservices.io/*
3. Pereira, P. A., & Morgan Bruce. (n.d.). ***Microservices in Action***.
4. Lewis, J. and Fowler, M., 2018. Microservices. [online] martinfowler.com. Available at: <https://martinfowler.com/articles/microservices.html> [Accessed 21 November 2021].
   - ***https://martinfowler.com/articles/microservices.html***

Sommaire

01 Introduction : contexte + microservices

02 Development & Operations Techniques (DevOps)

03 Containers

04 Microservice Design Patterns

05 Interactive Demo

1. Introduction

Context

- Existing Applications
  - 3-part system
  - Server-side application : monolith
- Upside
  - Easy to design & develop
- Downsides:
  - Difficult to maintain and evolve
  - Ill-adapted for the Cloud (e.g., “Scalability”)
  - Technological lock-in

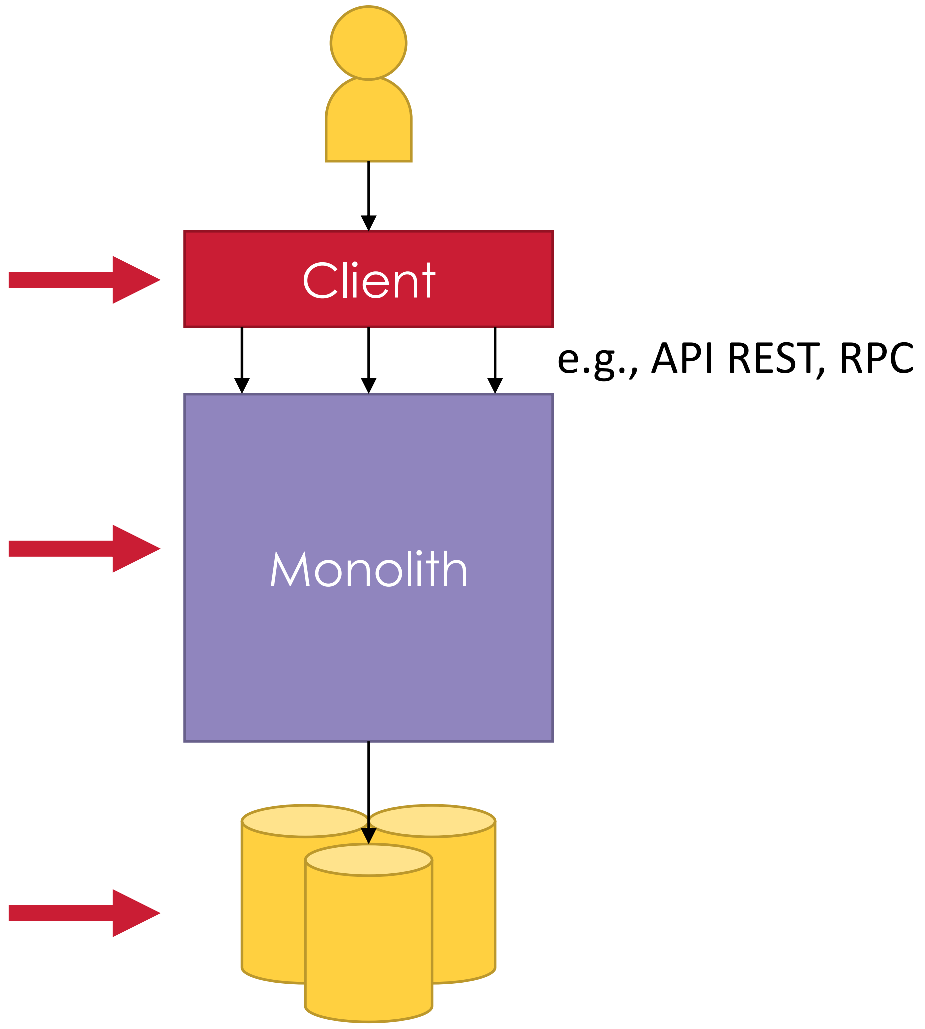


Figure 1a. Typical 3-part monolithic system

Scenario 1 : Correctif urgent d’un monolithe

Scenario 1 : Un correctif doit être fait en urgence au niveau d’un module.

1. Temps d’isoler l’erreur et effectuer le correctif.
2. Temps de compilation du monolithe.
3. Temps d’éteindre et relancer le monolithe.

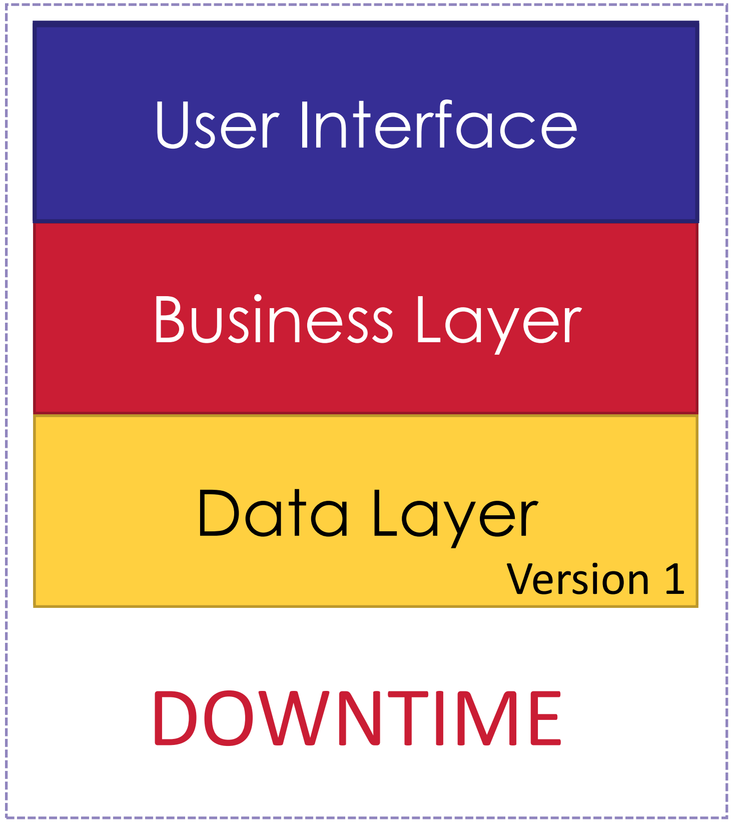


Figure 2. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

Scenario 1 : Correctif urgent d'un monolithe

Scenario 1 : Un correctif doit être fait en urgence au niveau d'un module.

1. Temps d'isoler l'erreur et effectuer le correctif.
2. Temps de compilation du monolithe.
3. Temps d'éteindre et relancer le monolithe.

Conséquence : Temps d'arrêt lors de la maintenance + temps de compilation

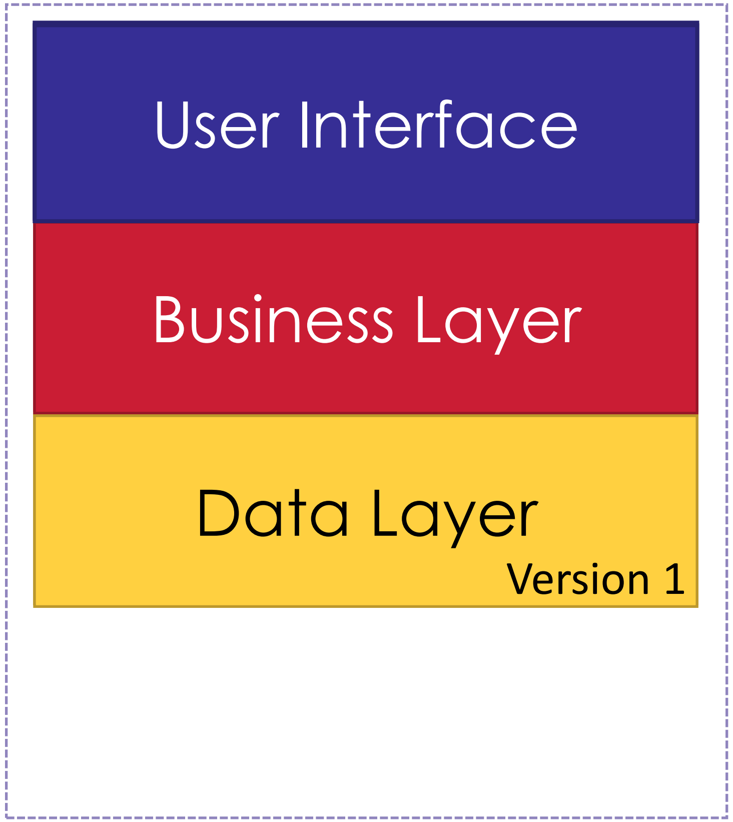


Figure 2. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

Scenario 2 : Monté en charge avec un monolithe

Scenario 2 : Suite à un imprévu l'usage d'un service particulier double temporairement.

Choix 1 : Dédoubler l'instance du monolithe pour prendre en charge les demandes.

Choix 2 : Prendre le risque d'une surcharge du logiciel.

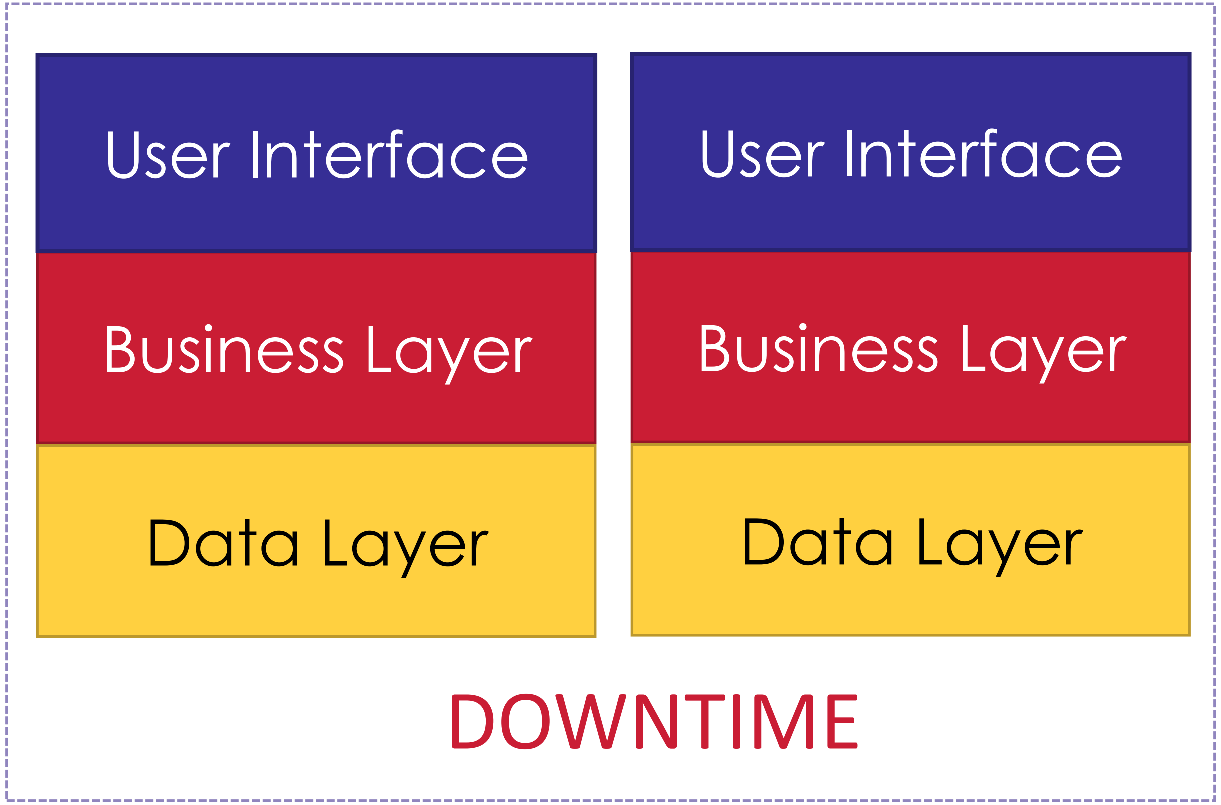


Figure 3. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

Scenario 2 : Monté en charge avec un monolithe

Scenario 2 : Suite à un imprévu l'usage d'un service particulier double temporairement.

Choix 1 : Dédoubler l'instance du monolithe pour prendre en charge les demandes.

Choix 2 : Prendre le risque d'une surcharge du logiciel.

Conséquence : Doubler les ressources requises pour le bon fonctionnement de l'application.

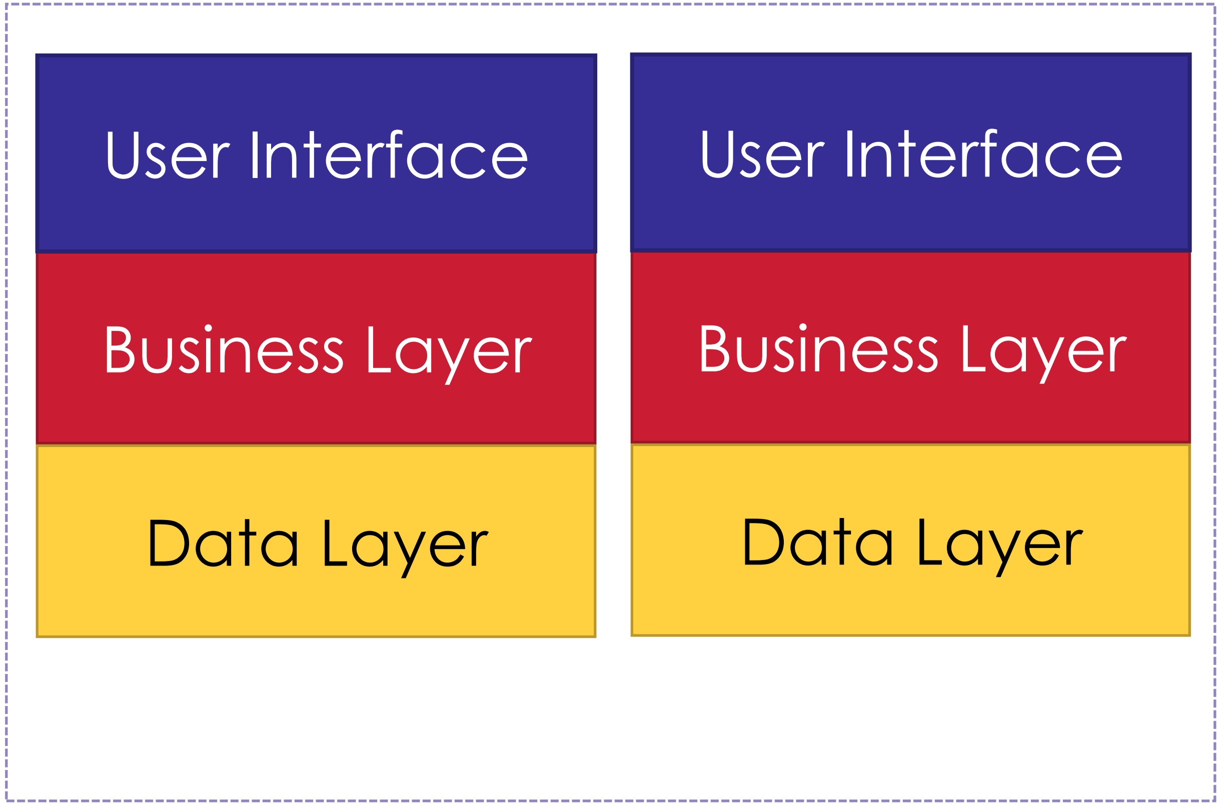


Figure 3. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

What are microservices?

Context

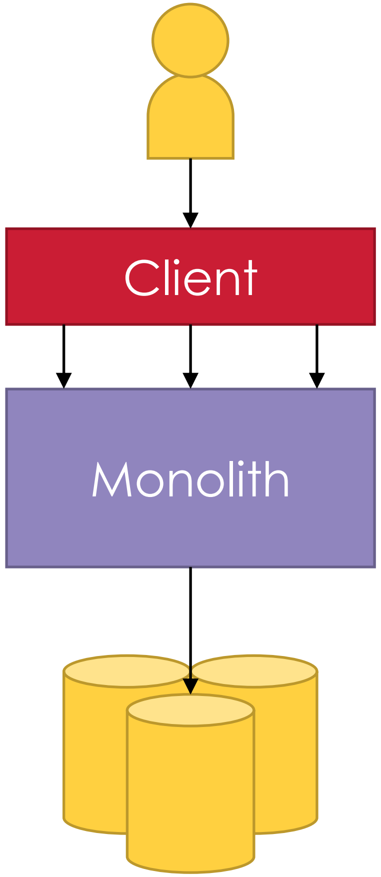


Figure 1a. Typical 3-part monolithic system

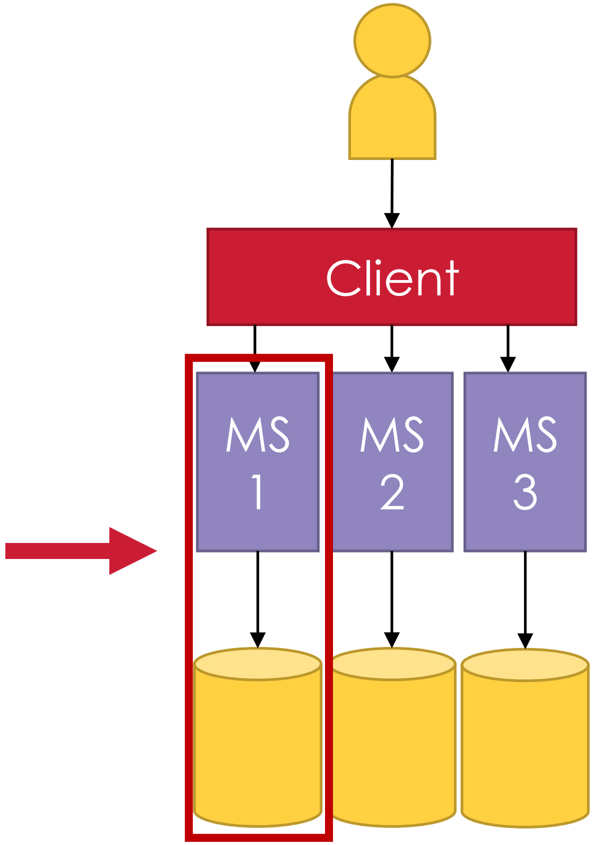


Figure 1b. The microservice-oriented architecture

Microservice Definition

The microservice architectural style is an approach to developing a single application as :

- Set of **small** services,
- running on its own process,
- Communicating via **lightweight** mechanisms (e.g., gRPC, REST api, events),
- Built around **specific business capabilities**,
- Independently deployable (via automated deployment)
- language **agnostic**,
- and data autonomous.

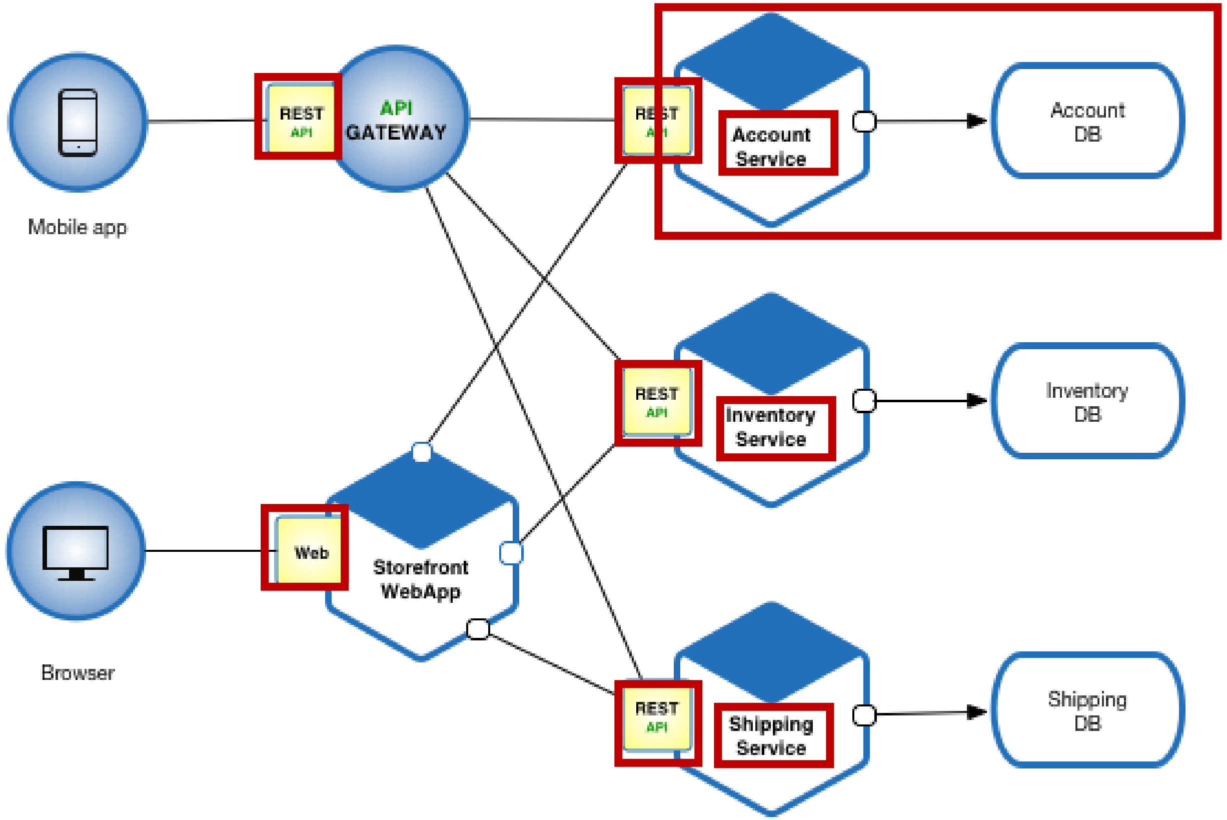
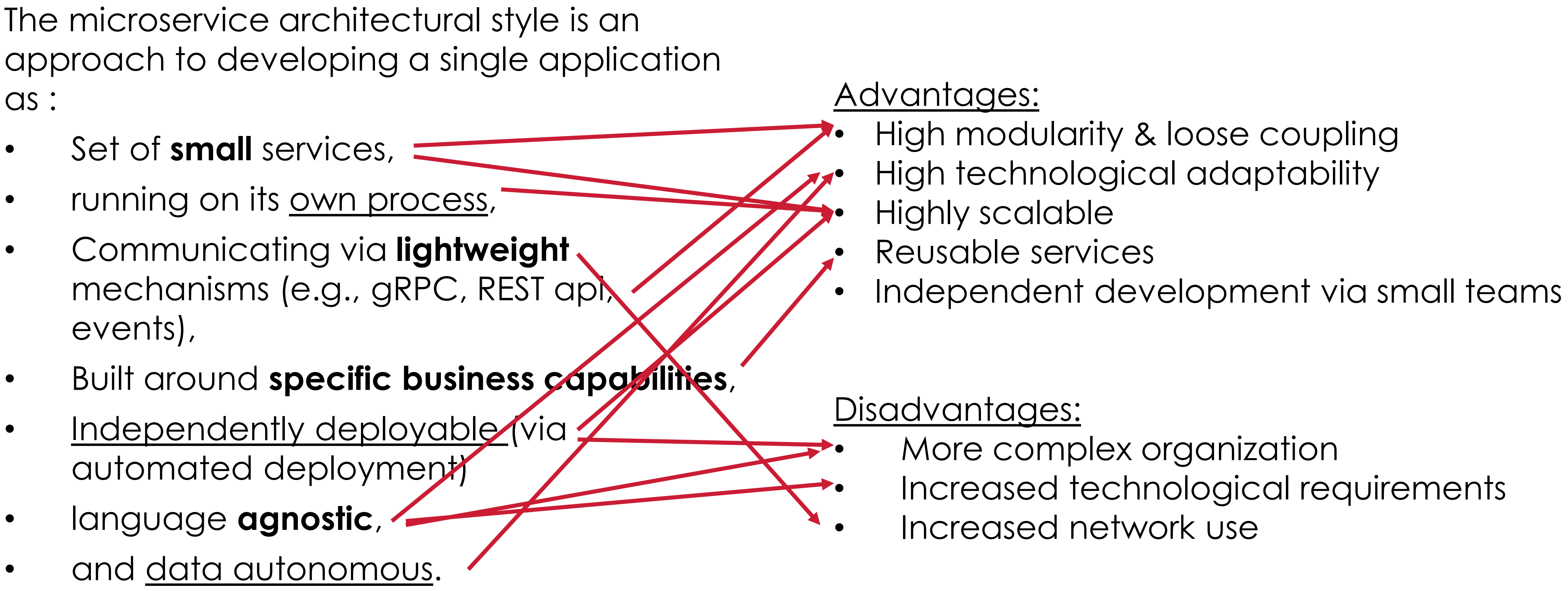


Figure 4. Example of a microservice-oriented Architecture [2]

Microservice Definition



The microservice architectural style is an approach to developing a single application as :

- Set of **small** services,
- running on its own process,
- Communicating via **lightweight** mechanisms (e.g., gRPC, REST api, events),
- Built around **specific business capabilities**,
- Independently deployable (via automated deployment)
- language **agnostic**,
- and data autonomous.

Advantages:

- High modularity & loose coupling
- High technological adaptability
- Highly scalable
- Reusable services
- Independent development via small teams

Disadvantages:

- More complex organization
- Increased technological requirements
- Increased network use

Scenario 1 : Correctif urgent d'un système à base de microservice

Scenario 1 : Un correctif doit être fait en urgence au niveau d'un module.

1. Temps d'isoler l'erreur et effectuer le correctif.
2. Temps de compilation du microservice.
3. Temps de lancer la nouvelle version.

Conséquence : Temps de compilation réduit + temps d'arrêt éliminé + temps de lancement réduit

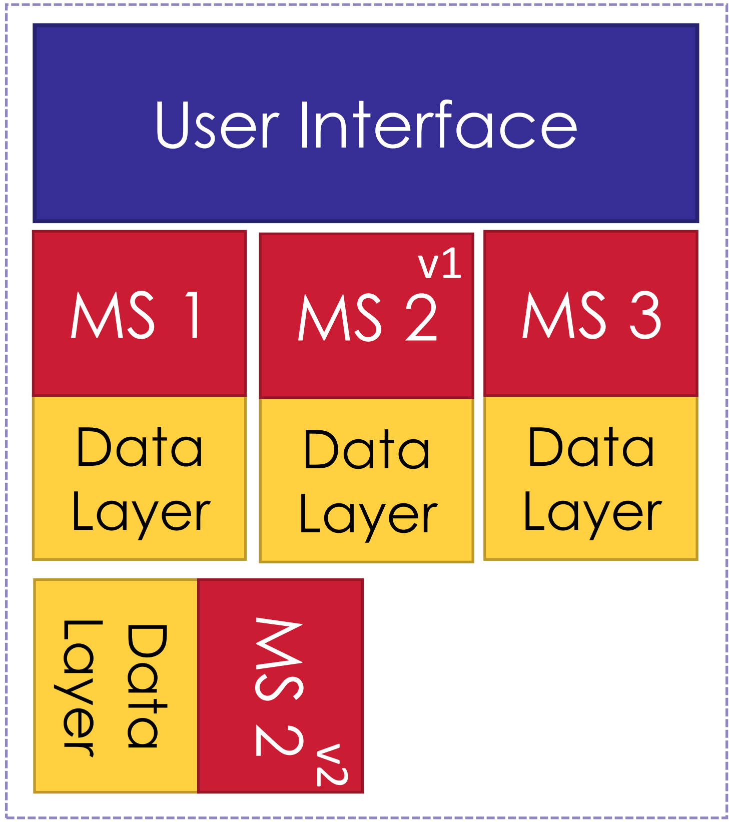


Figure 5. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

Scenario 2 : Monté en charge

Scenario 2 : Suite à un imprévu, l'usage d'un service particulier double temporairement.

Solution : Dédoubler l'instance du service qui est requis pour prendre en charge les demandes augmentées.

Conséquence : Une montée en charge optimisée sans prendre de risque.

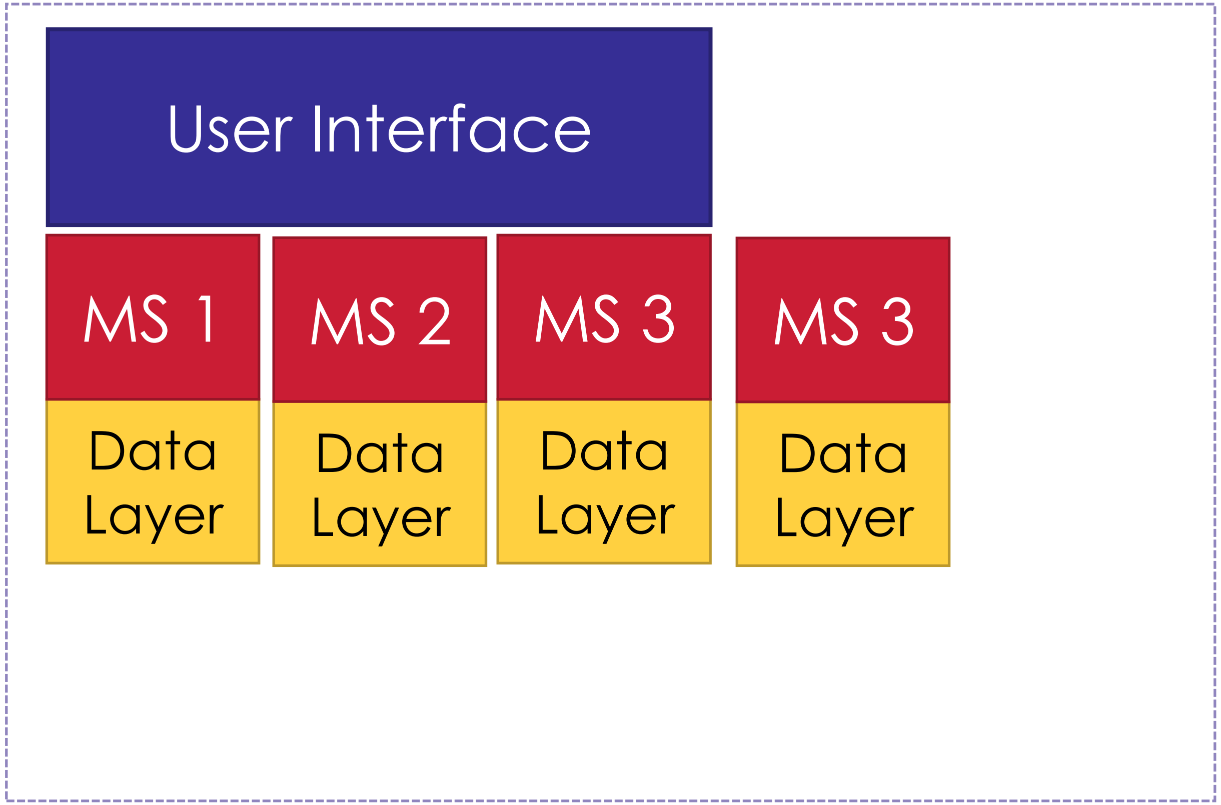
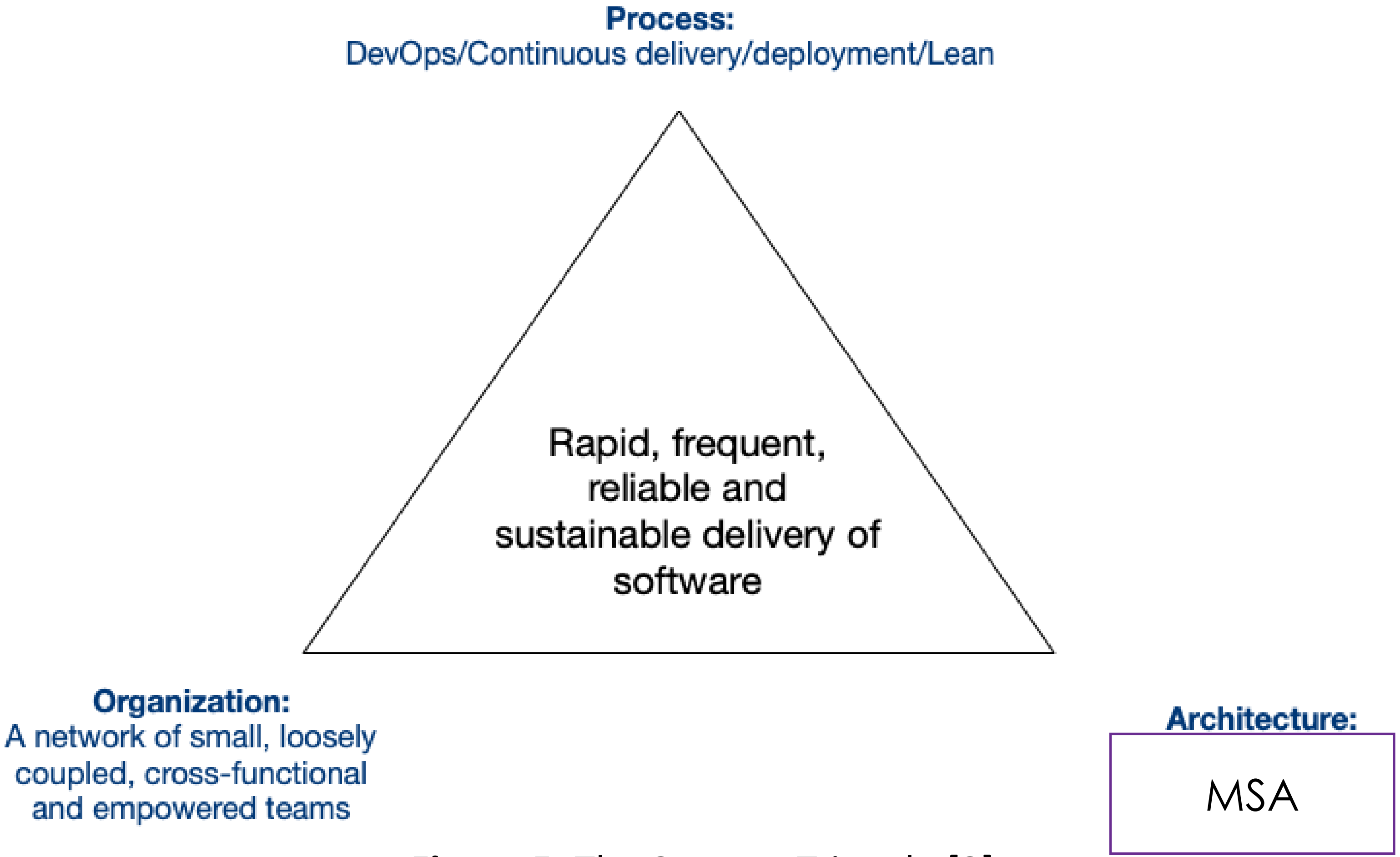


Figure 4. Représentation des ressources disponible du serveur vs les ressources utilisés pour exécuter le monolithe

2. DevOps & Microservices

Motivation



Process:
DevOps/Continuous delivery/deployment/Lean

Rapid, frequent, reliable and sustainable delivery of software

Organization:
A network of small, loosely coupled, cross-functional and empowered teams

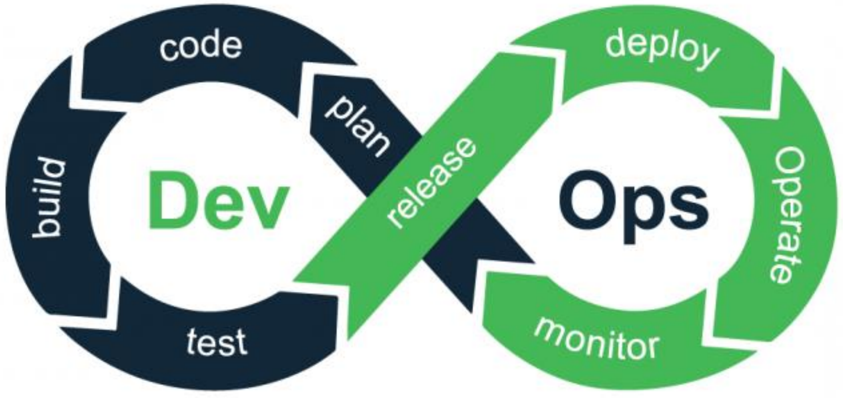
Architecture:
MSA

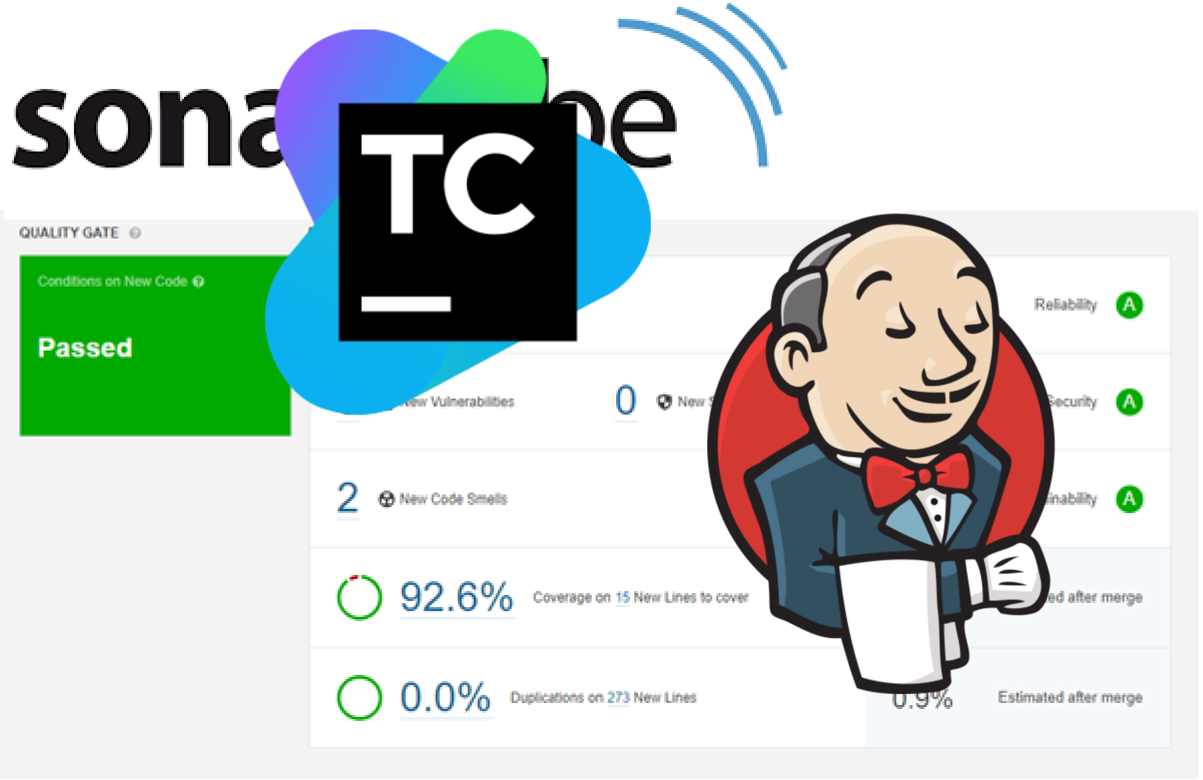
Figure 5. The Success Triangle [2]

DevOps: A definition

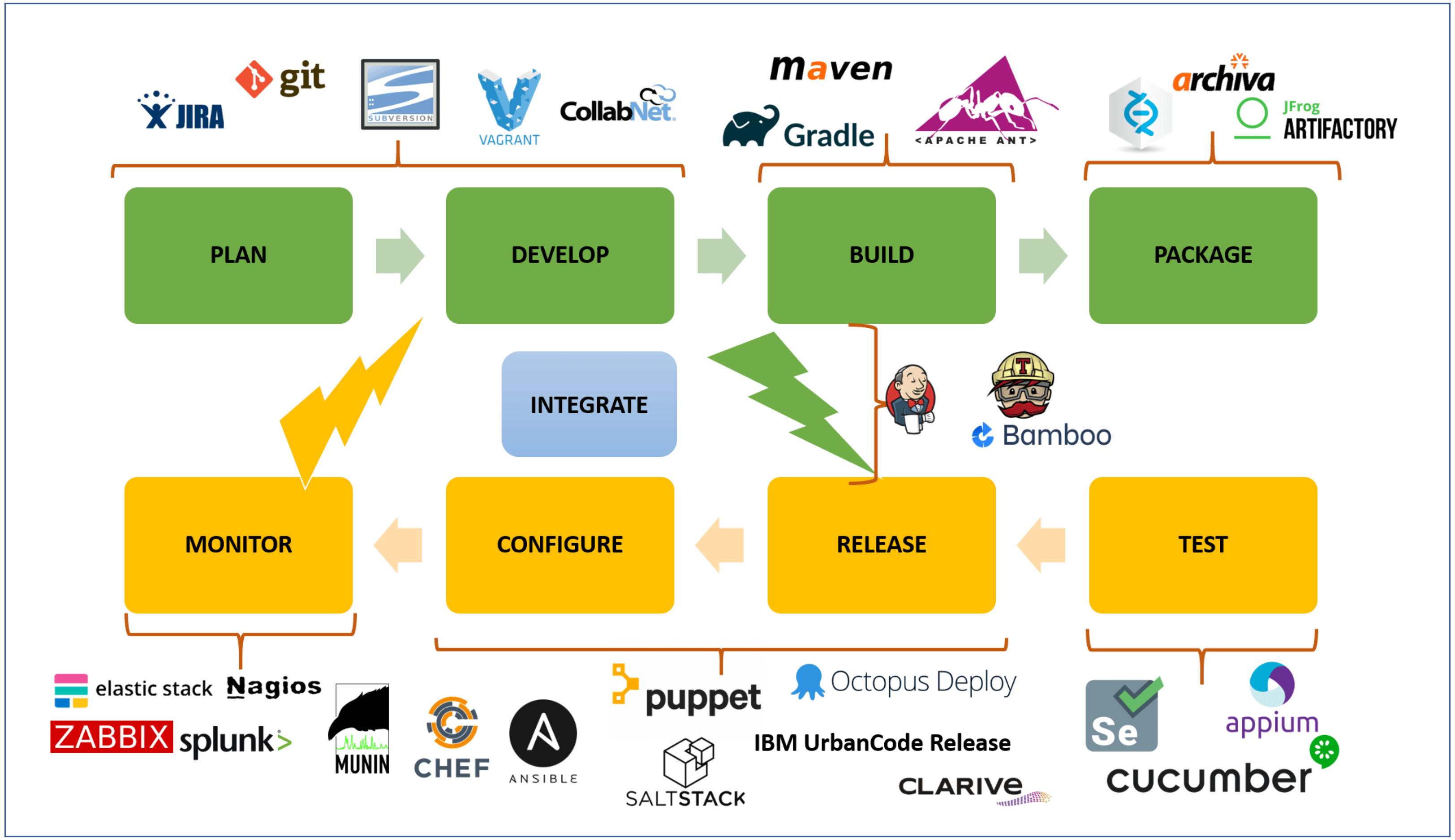
DevOps (Development & Operations) is a set of practices that combines software development and the operation of software in production (either on premise or on the Cloud).

1. Coding – code development and review, source code management tools, code merging.
2. Building – continuous integration tools, build status.
3. Testing – continuous testing tools that provide quick and timely feedback on business risks.
4. Packaging – artifact repository, application pre-deployment staging.
5. Releasing – change management, release approvals, release automation.
6. Configuring – infrastructure configuration and management, infrastructure as code tools.
7. Monitoring – applications performance monitoring, end-user experience.





The whole process



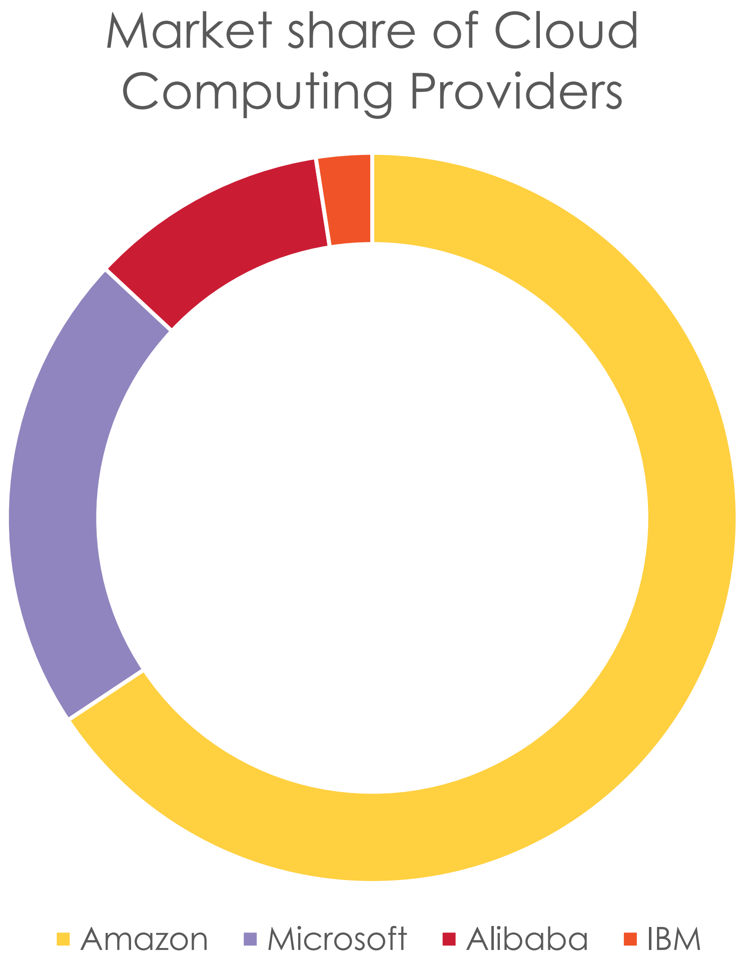
Source: https://digitalvarys.com/tools-for-devops/

3. Cloud Technologies

Cloud Computing

Cloud computing is:

1. A **model for establishing network access to a shared pool of** standard, configurable computing of standard and configurable **computing resources** (e.g., network, servers, storage, applications and services) such that :
   - resources can be quickly mobilized and made available;
   - management effort or contact with the resource provider is minimized
2. A set of hardware, network connections, and software providing services that-services that individuals and communities can use from anywhere in the world.



Why use the Cloud?

Advantages for the consumer:

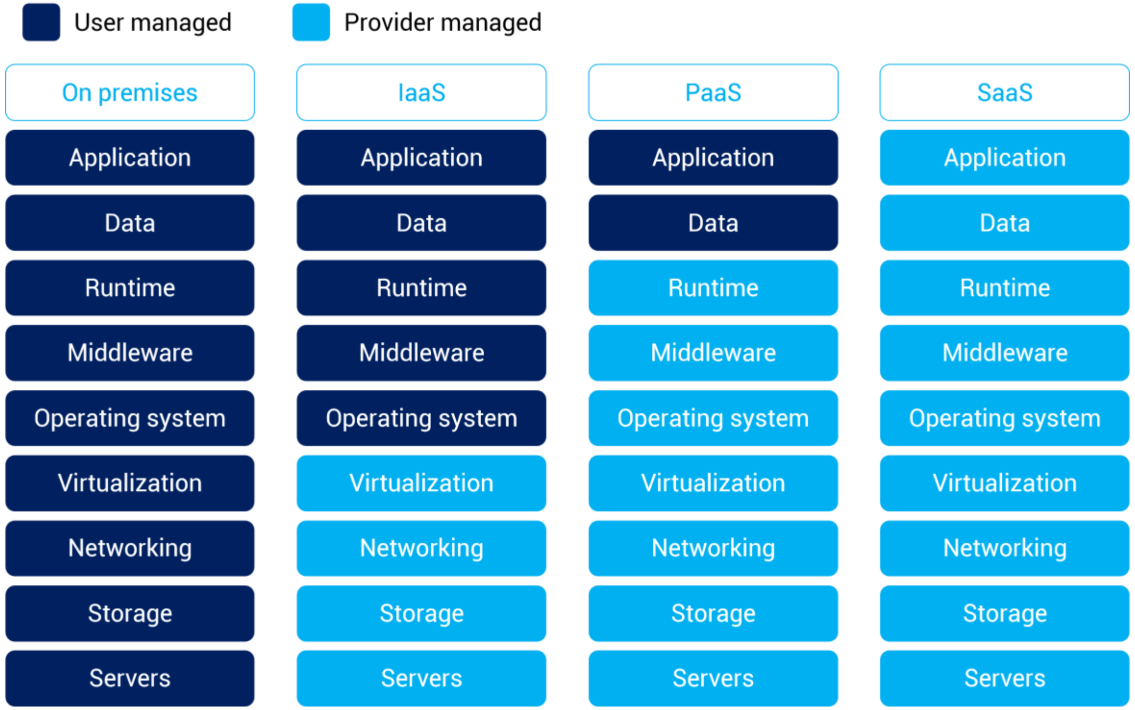
- instead of obtaining computing power and/or storage power by acquiring hardware and/or software, **the consumer uses the power made available to him by a provider** via the Internet.

Advantage for the provider:

- **rental on demand or on a fixed price** according to technical criteria (e.g., computing power, bandwidth, ...).

Advantages for a consumer company :

- reduce the total cost of ownership of IT systems;
- **easily increase/decrease resources**;
- offload the IT teams of the companies, which will thus have more availability for high value-added activities;
- Enable small businesses to access services that were previously reserved for large enterprises because of their cost.



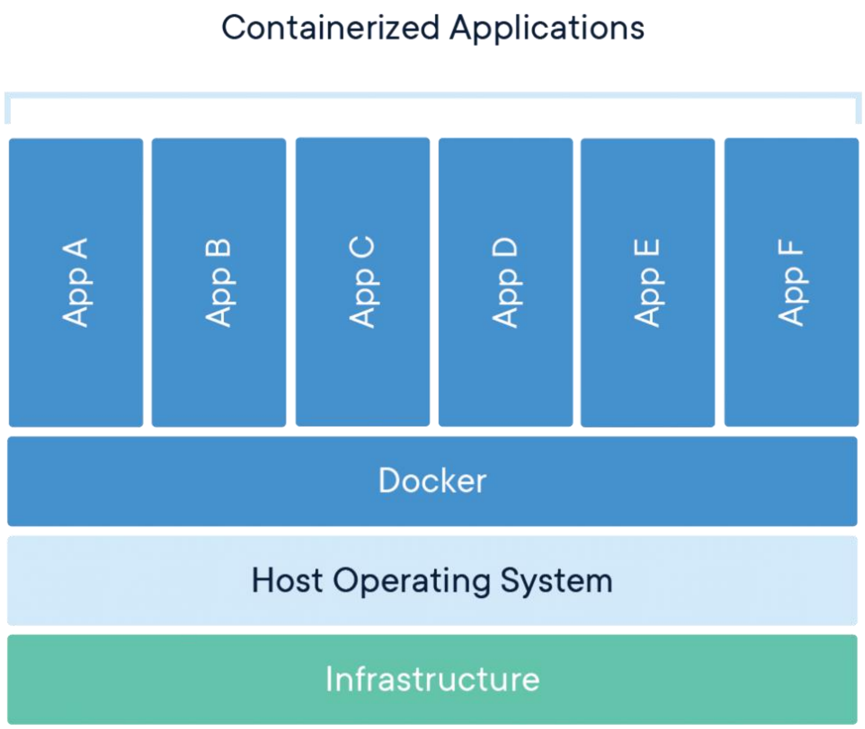
■ User managed □ Provider managed

On premises	IaaS	PaaS	SaaS
■ Application	■ Application	■ Application	□ Application
■ Data	■ Data	■ Data	□ Data
■ Runtime	■ Runtime	□ Runtime	□ Runtime
■ Middleware	■ Middleware	□ Middleware	□ Middleware
■ Operating system	■ Operating system	□ Operating system	□ Operating system
■ Virtualization	□ Virtualization	□ Virtualization	□ Virtualization
■ Networking	□ Networking	□ Networking	□ Networking
■ Storage	□ Storage	□ Storage	□ Storage
■ Servers	□ Servers	□ Servers	□ Servers

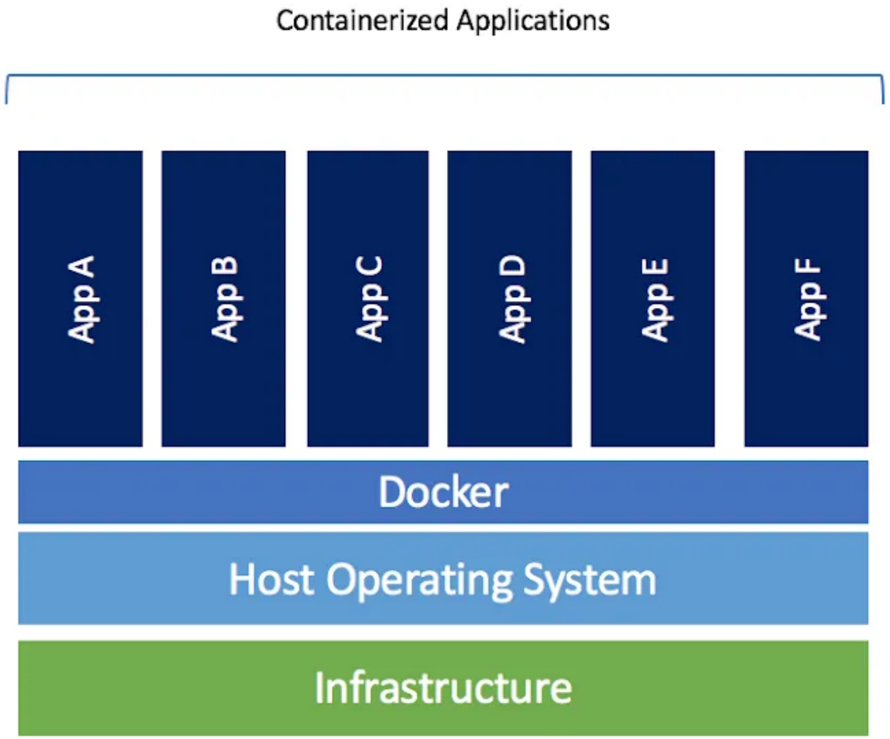
Containers

- Containerization is an approach to software development in which **an application/service, its dependencies, and its configuration (i.e., deployment manifest files) are packaged into a container image**.
- Software containers are a standard unit of software deployment that can contain different code and dependencies.

Advantages : portable, agile, evolving, isolated environment, increased productivity, etc.



Containers versus VMs


Containerized Applications

App A

App B

App C

App D

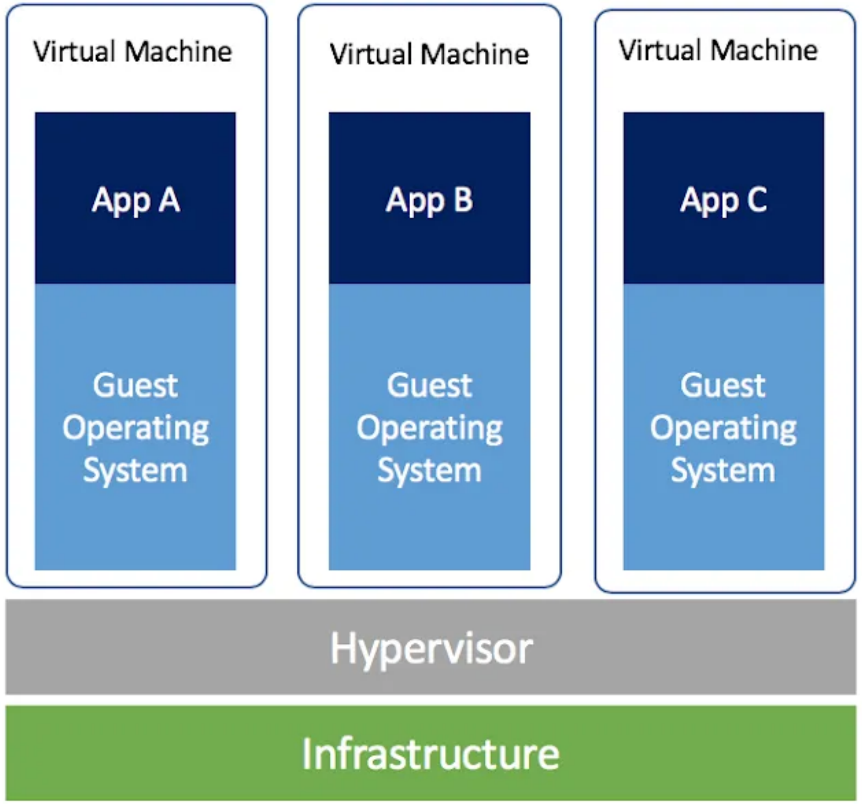
App E

App F

Docker

Host Operating System

Infrastructure


Virtual Machine

App A

Guest Operating System

Virtual Machine

App B

Guest Operating System

Virtual Machine

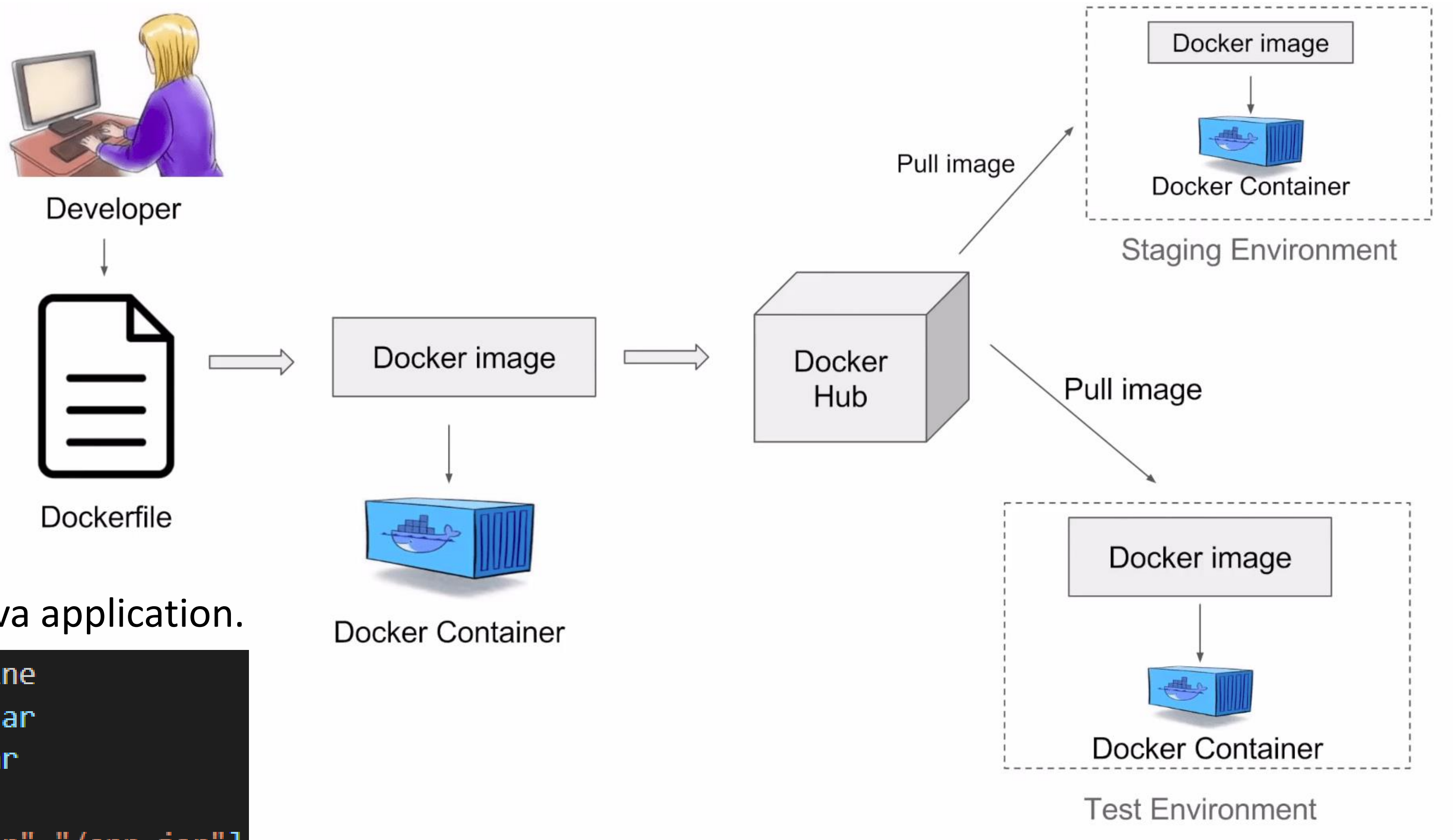
App C

Guest Operating System

Hypervisor

Infrastructure

Docker Workflow

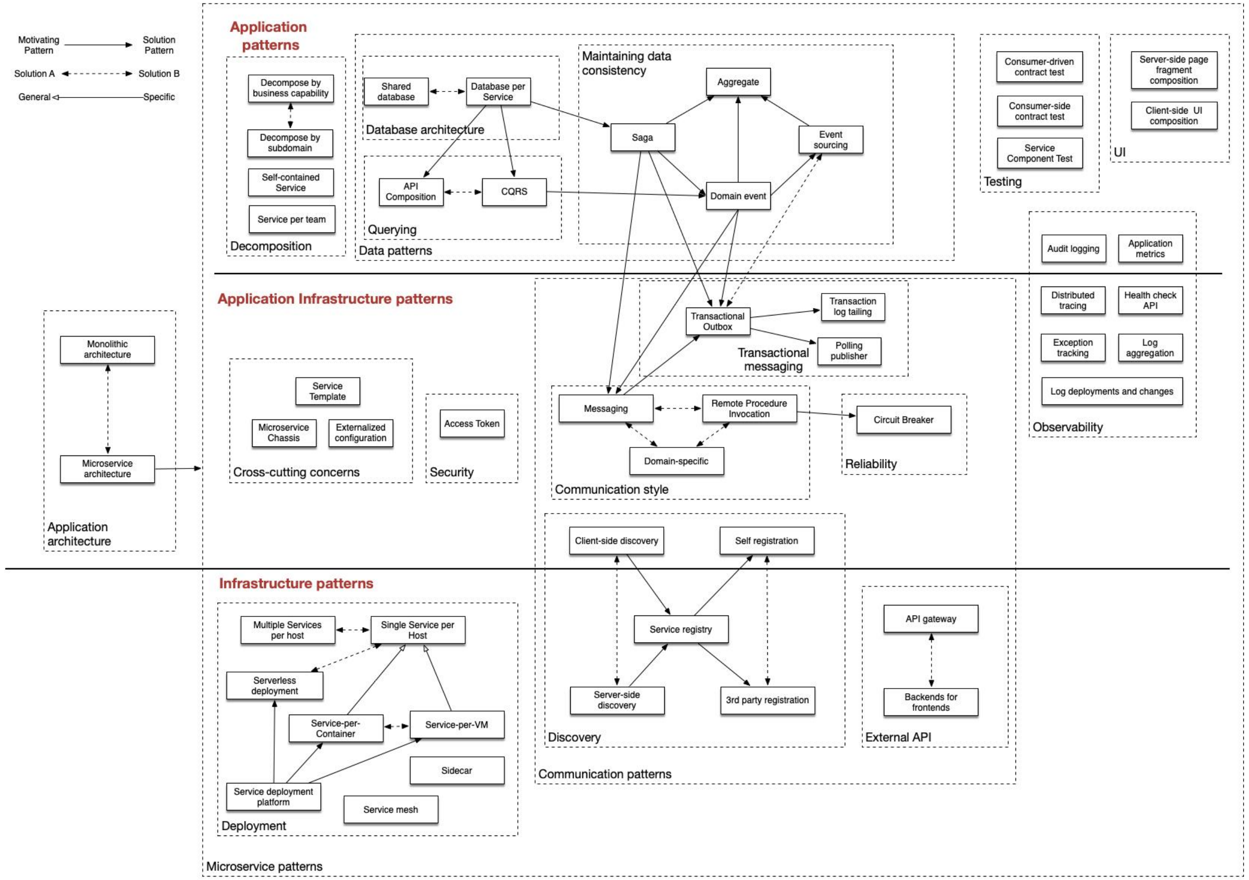


e.g., Dockerfile of a java application.

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8081
ENTRYPOINT ["java","-jar","/app.jar"]
```

4. Microservice Design Patterns

Overview of the many patterns



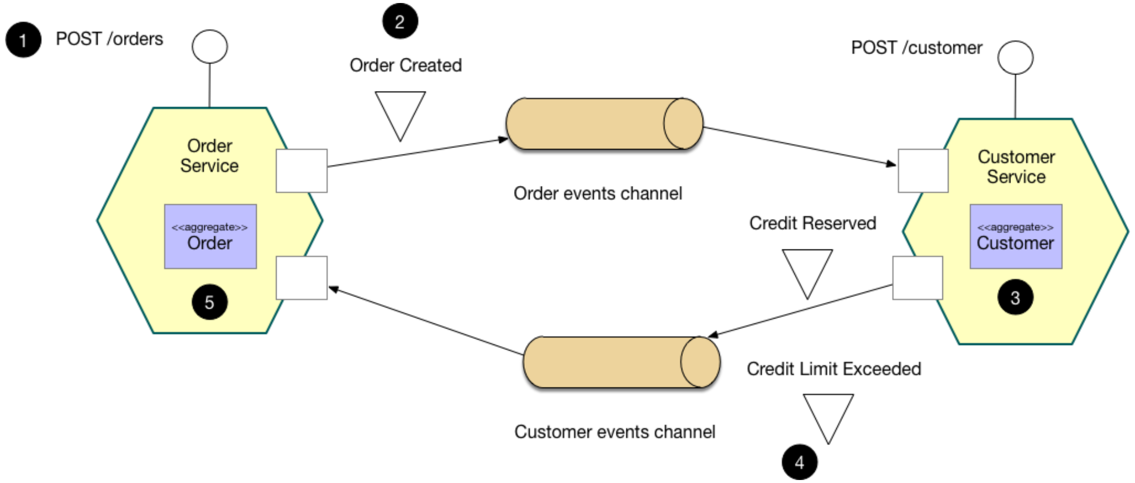
Copyright © 2020. Chris Richardson Consulting, Inc. All rights reserved.

Learn-Build-Assess Microservices http://adopt.microservices.io

Saga Design Pattern

Context: Each service has its own database. Some business transactions, however, span multiple service so you need a mechanism to implement transactions that span services. For example, let's imagine that you are building an e-commerce store where customers have a credit limit. The application must ensure that a new order will not exceed the customer's credit limit. Since Orders and Customers are in different databases owned by different services <u>the application cannot simply use a local ACID* transaction</u>.

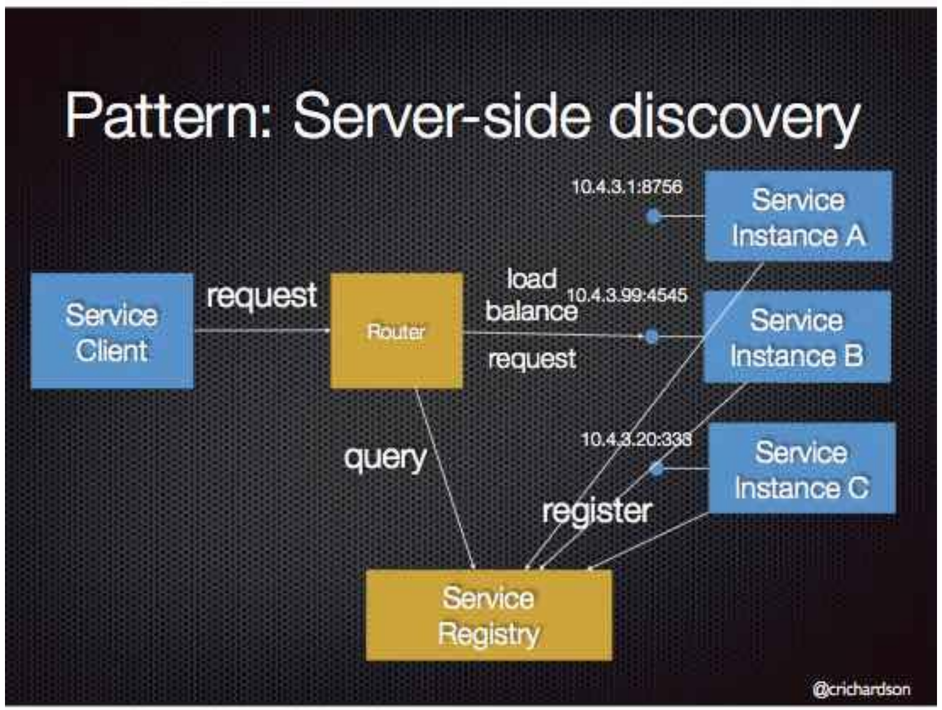
Solution: Implement each business transaction that spans multiple services is a saga. A **saga is a sequence of local transactions**. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. <u>If a local transaction fails because it violates a business rule, then the saga executes a series of compensating transactions that undo the changes</u> that were made by the preceding local transactions.

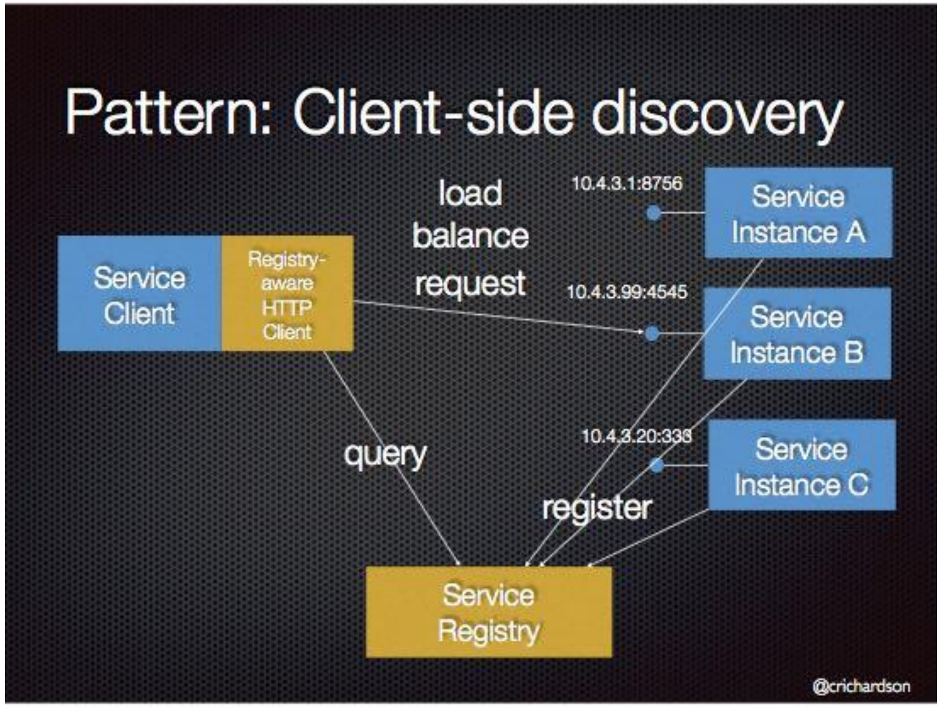


**ACID* (atomicity, consistency, isolation, durability) is a set of properties of database *transactions.*

Service Discovery

Context: Service typically need to call each other. In a traditional distributed system deployment, services run at fixed, well-known locations (hosts and ports) and so can easily call one another using HTTP/REST or some RPC mechanism.
Solutions: Service-side discovery, client-side discovery

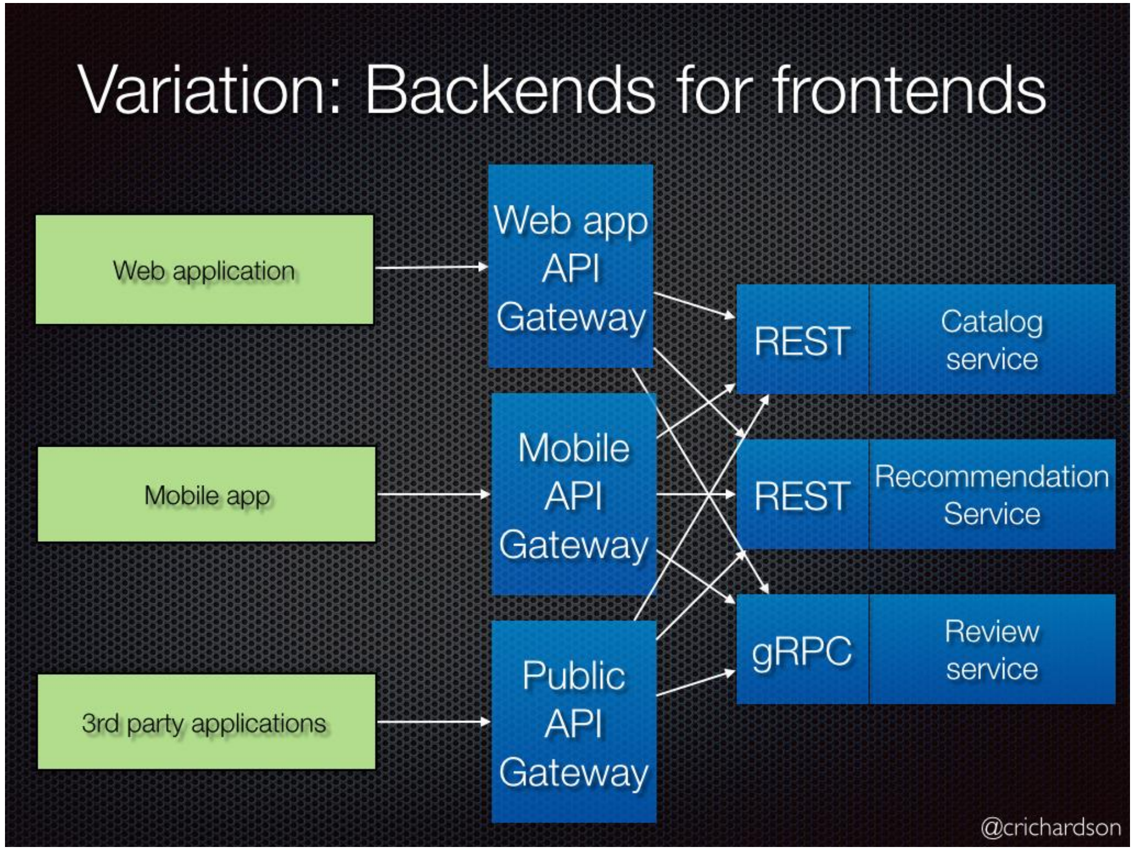




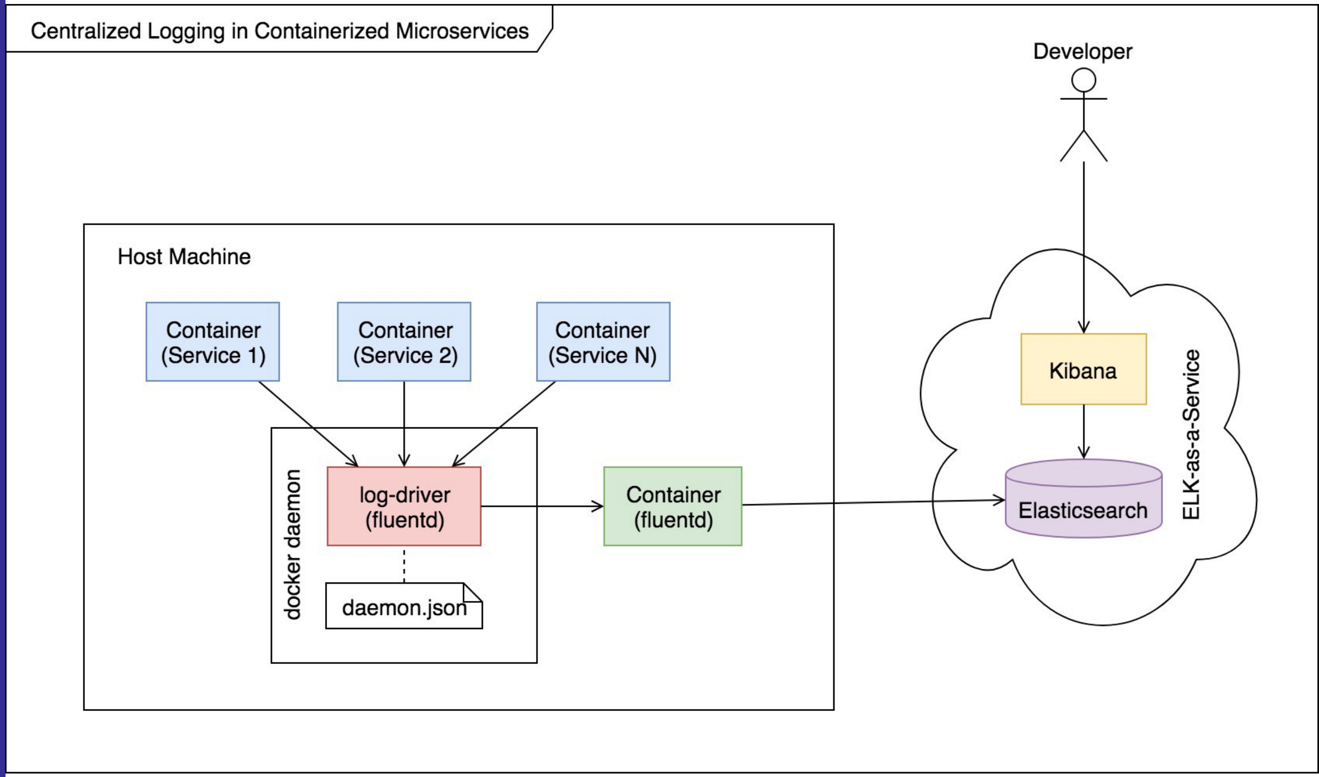
Gateway Design Pattern

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

Solution: Implement a for API gateway that is the **single-entry point all clients**. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.



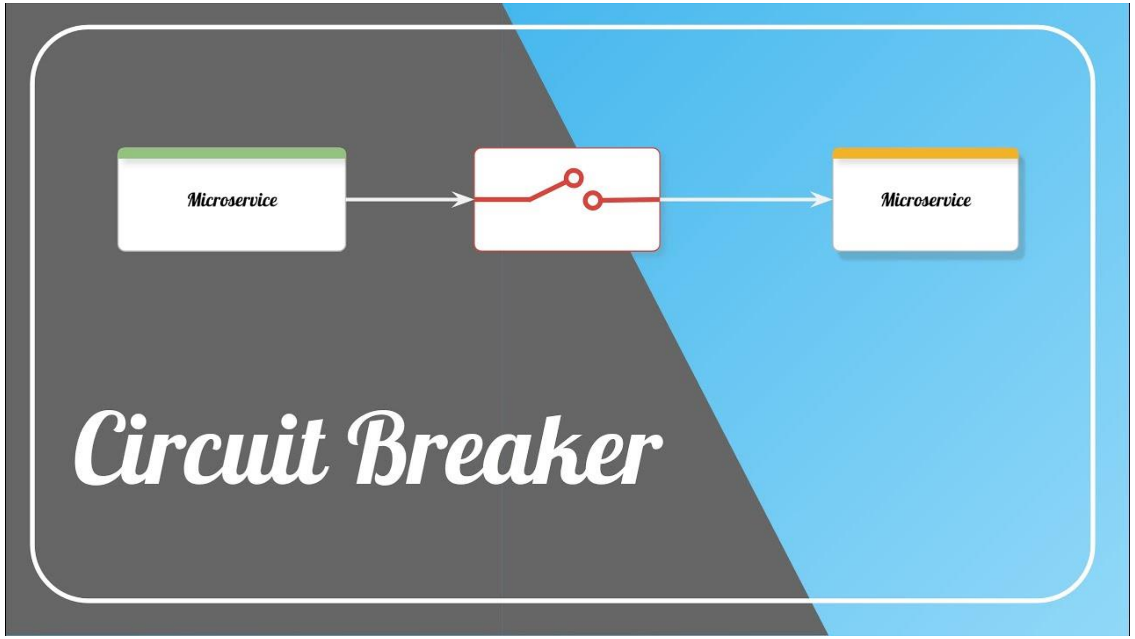
Log Aggregation Pattern



Circuit Breaker Pattern

Context: When one service synchronously invokes another there is always the possibility that the other service is unavailable or is exhibiting such high latency it is essentially unusable. Precious resources such as threads might be consumed in the caller while waiting for the other service to respond. This might lead to resource exhaustion, which would make the calling service unable to handle other requests. The **failure of one service can potentially cascade** to other services throughout the application.

Solution: Create a software circuit breaker



5. Interactive Demo