Desenvolvimento de Aplicações com Arquitetura Baseada em Microservices

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1007] - Tópicos Avançados em SI 4
https://github.com/vinicius3w/if1007-Microservices

AssertLab
Advanced Software and Systems Engineering Research Technologies

Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-CompartilhaIgual 3.0 Não Adaptada

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

AssertLab

Resources

- There is no textbook required. However, the following are some books that may be recommended:
  - Building Microservices: Designing Fine-Grained Systems
  - Spring Microservices
  - Spring Boot: Acelere o desenvolvimento de microsserviços
  - Microservices for Java Developers A Hands-on Introduction to Frameworks and Containers
  - Migrating to Cloud-Native Application Architectures
  - Continuous Integration

DOES YOUR COMPANY NEED MICROSERVICES?

Demystifying Microservices

AssertLab
Advanced Software and Systems Engineering Research Technologies

Micro-what????

- Microservices are an architecture style and an approach for software development to satisfy modern business demands

- We will see...
  - The evolution of microservices
  - The definition of the microservices architecture with examples
  - Concepts and characteristics of the microservices architecture
  - Typical use cases of the microservices architecture
  - The relationship of microservices with SOA and Twelve-Factor Apps

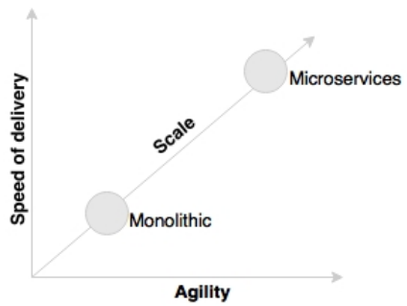
Microservices are not invented; they are more of an evolution from the previous architecture styles.

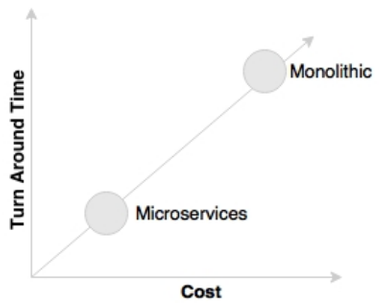
The evolution of microservices

- One of the increasingly popular architecture patterns next to SOA
  - … complemented by DevOps and cloud
- Evolution is greatly influenced by the disruptive digital innovation trends in modern business and the evolution of technologies in the last few years.

Business demand as a catalyst for microservices evolution

- Technologies as one of the key enablers for radically increasing their revenue and customer base





Gone are the days when businesses invested in large application developments with the turnaround time of a few years. Enterprises are no longer interested in developing consolidated applications to manage their end-to-end business functions as they did a few years ago.

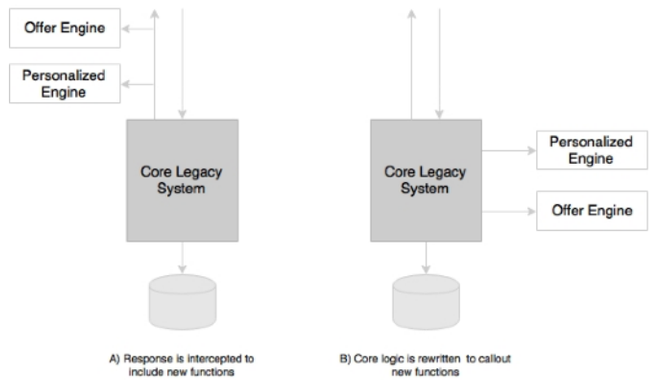
Business demand as a catalyst for microservices evolution

- Institutions do not invest in rebuilding their core mainframe systems as another monolithic monster
- Retailers and other industries do not rebuild heavyweight supply chain management applications, such as their traditional ERPs
- Focus has shifted to building quick-win point solutions that cater to specific needs of the business in the most agile way possible

AssertLab
Advanced Software and Systems Engineering Research Technologies

Shift to building quick-win point solutions

- Modern architectures are expected to maximize the ability to replace their parts and minimize the cost of replacing their parts
- The microservices approach is a means to achieving this



As shown in the diagram, rather than investing in rebuilding the core legacy system, this will be either done by passing the responses through the new functions, as shown in the diagram marked A, or by modifying the core legacy system to call out these functions as part of the processing, as shown in the diagram marked B. These functions are typically written as microservices

Technology as a catalyst for the microservices evolution

- A few decades back, we couldn't even imagine a distributed application without a two-phase commit. Later, NoSQL databases made us think differently

- Platform as a Services (PaaS) providers made us rethink the way we build middleware components
  - Integration Platform as a Service (iPaaS)

AssertLab
Advanced Software and Systems Engineering Research Technologies

The container revolution created by Docker radically influenced the infrastructure space. These days, an infrastructure is treated as a commodity service

Imperative architecture evolution

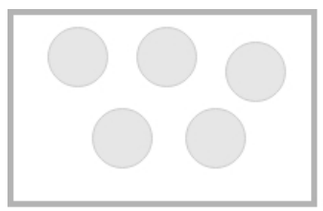
- Architectures have gone through the evolution of age-old mainframe systems to fully abstract cloud services such as AWS Lambda
- Irrespective of the choice of architecture styles, we always used to build one or the other forms of monolithic architectures

Read more about Lambda at: https://aws.amazon.com/documentation/lambda/

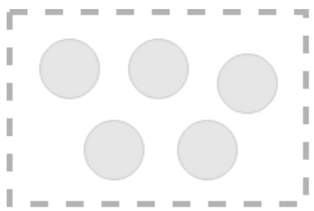
Using AWS Lambda, developers can now drop their "functions" into a fully managed compute service.

Imperative architecture evolution

- The microservices architecture evolved as a result of modern business demands [agility and speed of delivery], emerging technologies, and learning from previous generations of architectures



Monolithic Architecture



Microservices Architecture

AssertLab
Advanced Software and Systems Engineering Research Technologies

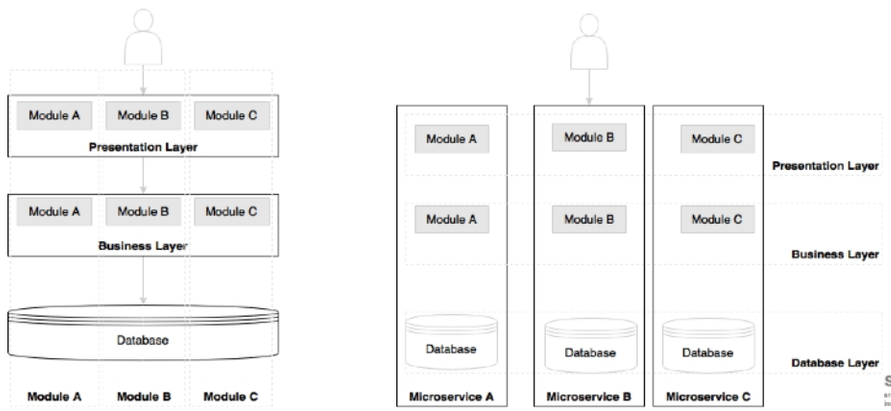
Microservices help us break the boundaries of monolithic applications and build a logically independent smaller system of systems

So, what are microservices?

- Microservices are an architecture style and an approach for software development to satisfy modern business demands
- Originated from the idea of hexagonal architecture coined by Alistair Cockburn
  - Hexagonal architecture is also known as the Ports and Adapters pattern

Architectural style

- Microservices are an architectural style or an approach to building IT systems as a set of business capabilities that are autonomous, self-contained, and loosely coupled



The preceding diagram depicts a traditional N-tier application architecture having a presentation layer, business layer, and database layer. The modules A, B, and C represent three different business capabilities.
As we can note in the preceding diagram, the boundaries are inversed in the microservices architecture. Each vertical slice represents a microservice. Each microservice has its own presentation layer, business layer, and database layer. Microservices are aligned towards business capabilities. By doing so, changes to one microservice do not impact others.

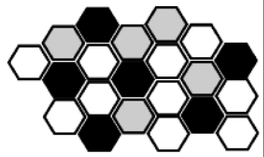
Communication

- There is no standard for communication or transport mechanisms for microservices
- Widely adopted lightweight protocols, such as HTTP and REST, or messaging protocols, such as JMS or AMQP
  - ...such as Thrift, ZeroMQ, Protocol Buffers, or Avro

Independently manageable life cycles

- As microservices are more aligned to business capabilities and have independently manageable life cycles
- The ideal choice for enterprises embarking on DevOps and cloud.
  - DevOps and cloud are two facets of microservices

The honeycomb analogy



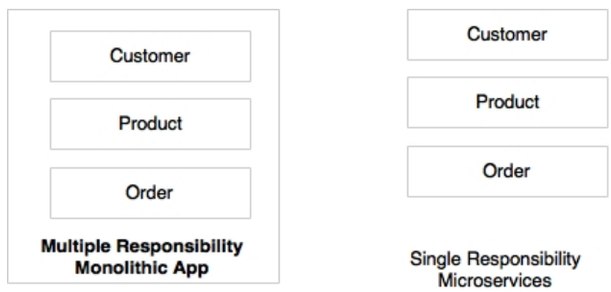
- They start small, using different materials to build the cells
- Construction is based on what is available at the time of building
- Repetitive cells form a pattern and result in a strong fabric structure
- Each cell in the honeycomb is independent but also integrated with other cells
- By adding new cells, the honeycomb grows organically to a big, solid structure
- The content inside each cell is abstracted and not visible outside
- Damage to one cell does not damage other cells, and bees can reconstruct these cells without impacting the overall honeycomb.

AssertLab
Advanced Software and Systems Engineering Research Technologies

In the real world, bees build a honeycomb by aligning hexagonal wax cells. They start small, using different materials to build the cells. Construction is based on what is available at the time of building. Repetitive cells form a pattern and result in a strong fabric structure. Each cell in the honeycomb is independent but also integrated with other cells. By adding new cells, the honeycomb grows organically to a big, solid structure. The content inside each cell is abstracted and not visible outside. Damage to one cell does not damage other cells, and bees can reconstruct these cells without impacting the overall honeycomb.

Principles of microservices

- Single responsibility per service
  - One of the principles defined as part of the SOLID design pattern
  - It states that a unit should only have one responsibility



These principles are a "must have" when designing and developing microservices

Customer, Product, and Order are different functions of an e-commerce application. Rather than building all of them into one application, it is better to have three different services, each responsible for exactly one business function, so that changes to one responsibility will not impair others. In the preceding scenario, Customer, Product, and Order will be treated as three independent microservices.

Principles of microservices

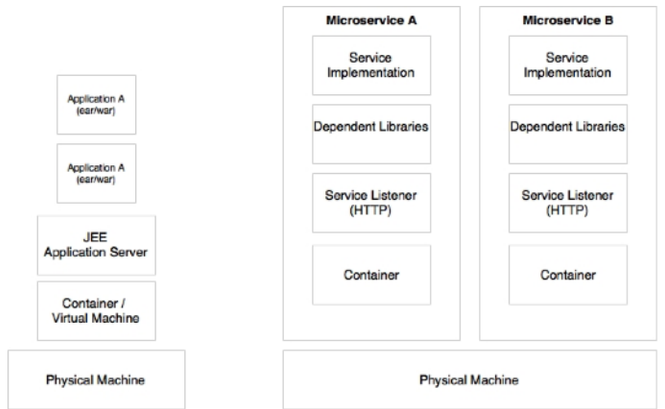
- Microservices are autonomous
  - Microservices are self-contained, independently deployable, and autonomous services that take full responsibility of a business capability and its execution
  - They bundle all dependencies, including library dependencies, and execution environments
    - such as web servers and containers or virtual machines

- SOA vs Microservices
  - SOA implementations provide service-level abstraction
  - Microservices go further and abstract the realization and execution environment

In traditional application developments, we build a WAR or an EAR, then deploy it into a JEE application server, such as with JBoss, WebLogic, WebSphere, and so on. We may deploy multiple applications into the same JEE container. In the microservices approach, each microservice will be built as a fat Jar, embedding all dependencies and run as a standalone Java process

Microservices are autonomous

- Microservices may also get their own containers for execution
- Containers are portable, independently manageable, lightweight runtime environments
- Container technologies, such as Docker, are an ideal choice for microservices deployment



Characteristics of microservices

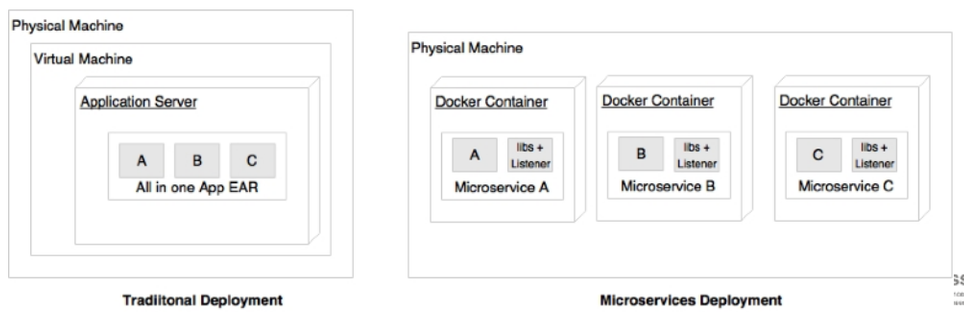
- Services are first-class citizens
  - expose service endpoints as APIs and abstract all their realization details
  - there is no more application development; instead, organizations focus on service development
  - From SOA...
    - Service contract; Loose coupling; Service abstraction; Service reuse; Statelessness; Services are discoverable; Service interoperability; Service composeability
    - More detail on SOA principles can be found here

AssertLab
Advanced Software and Systems Engineering Research Technologies

Evangelists and practitioners have strong but sometimes **different opinions** on microservices. There is no **single**, **concrete**, and **universally accepted** definition for microservices.
However, all successful microservices implementations exhibit a number of **common characteristics**.

Characteristics of microservices

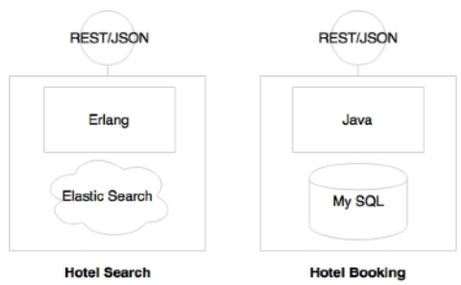
- Microservices are lightweight
  - A single business capability, so they perform only one function ~> smaller footprints
  - When selecting supporting technologies, we will have to ensure that they are also lightweight so that the overall footprint remains manageable



For example, Jetty or Tomcat are better choices as application containers for microservices compared to more complex traditional application servers such as WebLogic or WebSphere

Characteristics of microservices

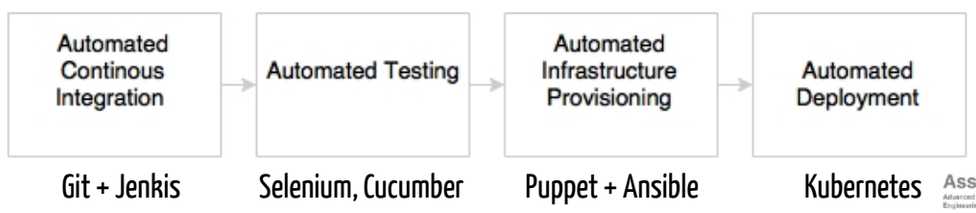
- Microservices with polyglot architecture
  - Different architectures for different microservices
  - Different services use different versions of the same technologies
  - Different languages are used to develop different microservices



Different architectures are used, such as one microservice using the Redis cache to serve data, while another microservice could use MySQL as a persistent data store

Characteristics of microservices

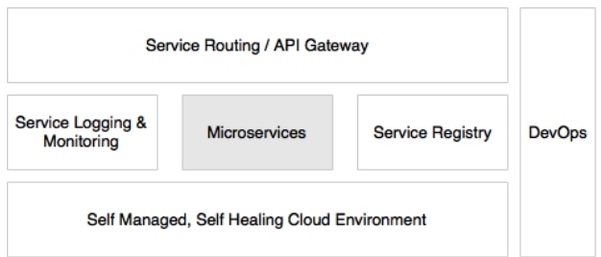
- Automation in a microservices environment
  - A large number of microservices is hard to manage until and unless automation is in place
  - microservices are automated end to end: automated builds, automated testing, automated deployment, and elastic scaling



The development phase is automated using version control tools such as Git together with CI tools such as Jenkins, Travis CI, and so on.
The testing phase will be automated using testing tools such as Selenium, Cucumber, and other AB testing strategies.
Infrastructure provisioning is done through container technologies such as Docker, together with release management tools such as Chef or Puppet, and configuration management tools such as Ansible.
Automated deployments are handled using tools such as Spring Cloud, Kubernetes, Mesos, and Marathon.

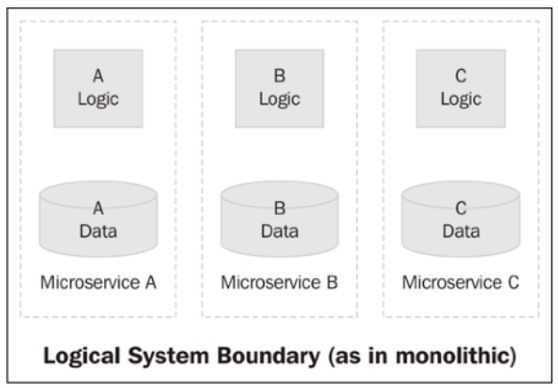
Characteristics of microservices

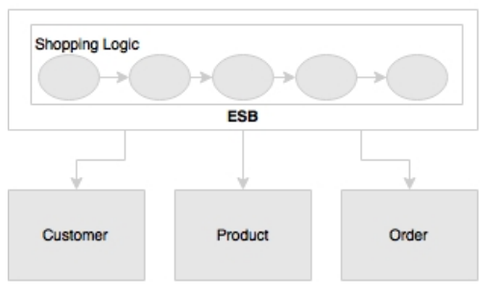
- Microservices with a supporting ecosystem
  - Most of the large-scale microservices implementations have a supporting ecosystem in place
  - The ecosystem capabilities include DevOps processes, centralized log management, service registry, API gateways, extensive monitoring, service routing, and flow control mechanisms



Characteristics of microservices

- Microservices are distributed and dynamic
  - Distributed data and logic and decentralized governance!





SOA system

A typical SOA implementation is shown in the preceding diagram. Shopping logic is fully implemented in ESB by orchestrating different services exposed by Customer, Order, and Product. In the microservices approach, on the other hand, Shopping itself will run as a separate microservice, which interacts with Customer, Product, and Order in a fairly decoupled way

Characteristics of microservices

- Antifragility, fail fast, and self-healing
  - The opposite of fragility is antifragility, or the quality of a system that gets stronger when subjected to stressors
  - How quickly the system can fail and if it fails, how quickly it can recover from this failure
    - Mean Time Between Failures (MTBF) to Mean Time To Recover (MTTR)
  - Self-healing ~> the system automatically learns from failures and adjusts itself

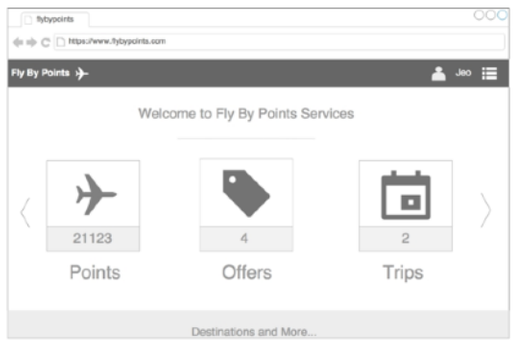
AssertLab
Advanced Software and Systems Engineering Research Technologies

O conceito de antifragilidade foi introduzido no livro Antifragile (Random House) de Nassim Taleb: Se a fragilidade é a qualidade de um sistema que fica mais fraco ou quebra quando submetido a estressores, então, o que é o oposto disso?
Muitos responderiam com a idéia de robustez ou resiliência - coisas que não quebram ou ficam mais fracas quando submetidas a estressores.
No entanto, o Taleb apresenta o oposto da fragilidade como antifragilidade, ou a qualidade de um sistema que se torna mais forte quando submetido a estressores.

Warm up

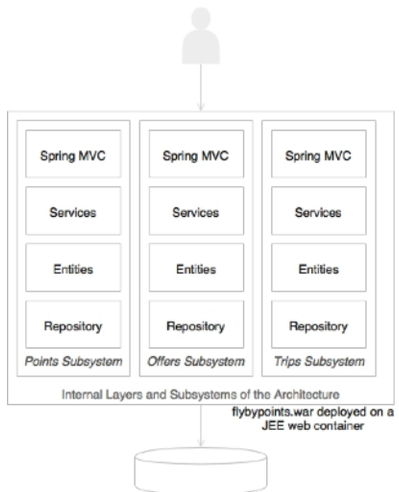
Microservices examples

- A holiday portal: FLY BY POINTS
  - Fly By Points collects points that are accumulated when a customer books a hotel, flight, or car through the online website.
  - When the customer logs in to the Fly By Points website, he/she is able to see the points accumulated, personalized offers that can be availed of by redeeming the points, and upcoming trips if any.



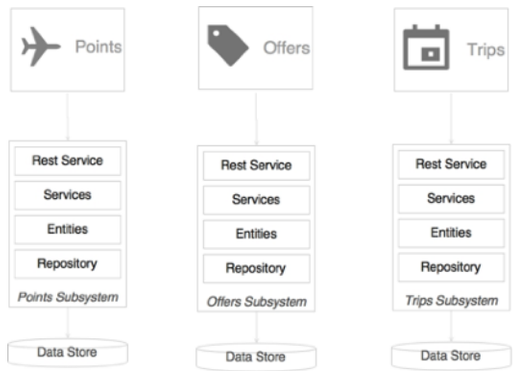
Fly By Points

- The holiday portal has a Java Spring-based traditional monolithic application architecture
- Following the usual practice, the holiday portal is also deployed as a single WAR file on a web server such as Tomcat
- As the business grows, the user base expands, and the complexity also increases



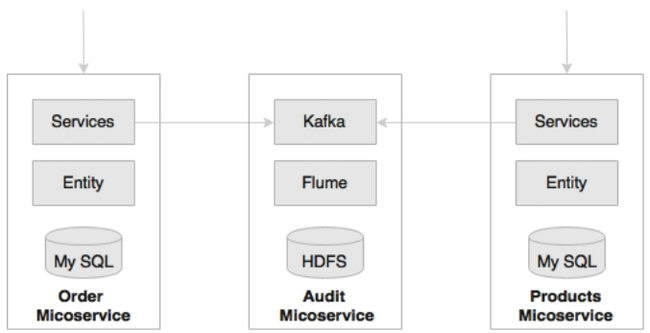
Fly By Points

- Rearchitecting the monolithic application to microservices for better speed of delivery, agility, and manageability
  - Each subsystem has now become an independent system by itself, a microservice
  - Each service encapsulates its own database as well as its own HTTP listener
  - Each microservice exposes a REST service to manipulate the resources/entity that belong to this service



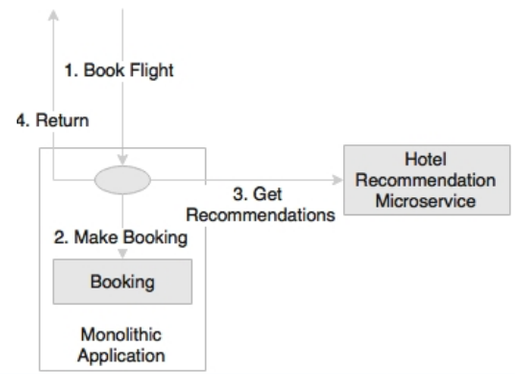
Microservices benefits

- Supports polyglot architecture



Microservices benefits

- Enabling experimentation and innovation

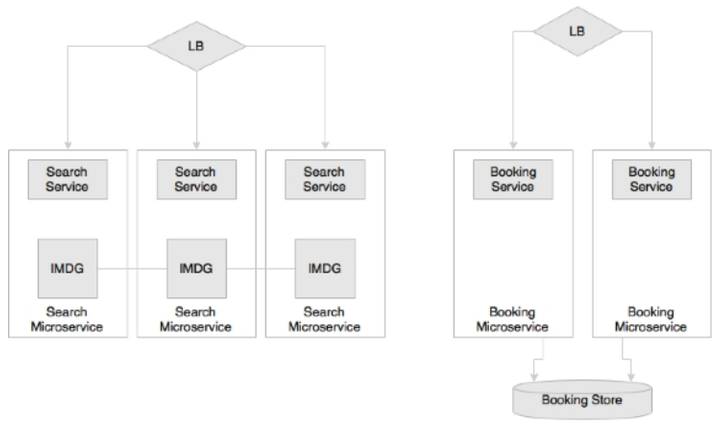


Microservices benefits

- Elastically and selectively scalable
  - As microservices are smaller units of work, they enable us to implement selective scalability
  - Scale Cube defines primarily three approaches to scaling an application:
    - Scaling the x axis by horizontally cloning the application
    - Scaling the y axis by splitting different functionality
    - Scaling the z axis by partitioning or sharding the data

Microservices benefits

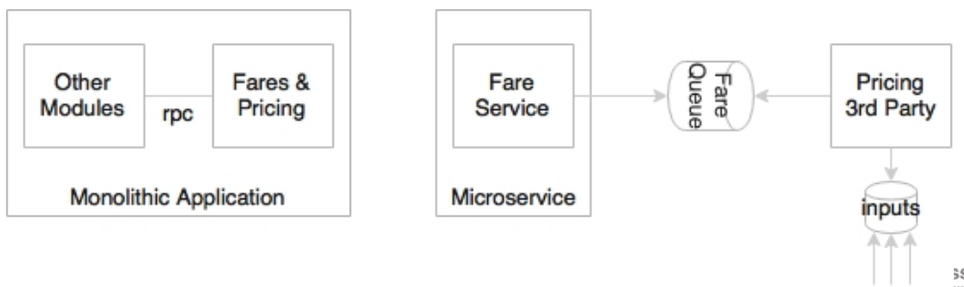
- Elastically and selectively scalable



For instance, in a typical airline website, statistics indicate that the ratio of flight searching to flight booking could be as high as 500:1. This means one booking transaction for every 500 search transactions. In this scenario, the search needs 500 times more scalability than the booking function. This is an ideal use case for selective scaling.

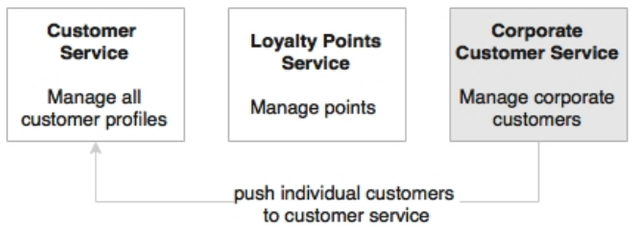
Microservices benefits

- Allowing substitution
  - Architecturally, a microservice can be easily replaced by another microservice developed either in-house or even extended by a microservice from a third party



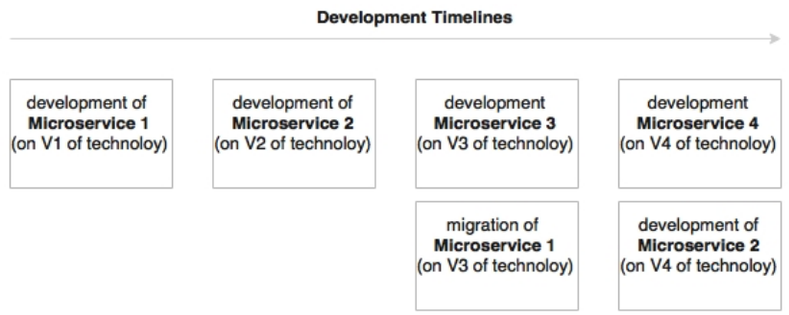
Microservices benefits

- Enabling to build organic systems
  - Organic systems are systems that grow laterally over a period of time by adding more and more functions to it
  - This enable us to keep adding more and more services as the need arises with minimal impact on the existing services



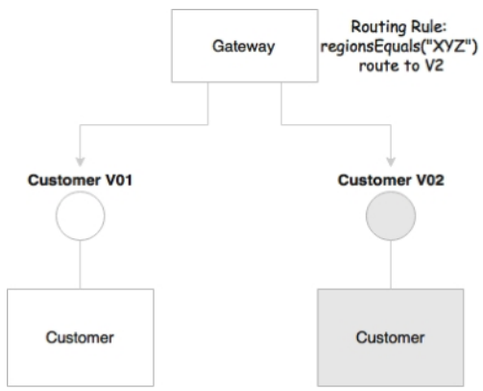
Microservices benefits

- Helping reducing technology debt
  - As microservices are smaller in size and have minimal dependencies, they allow the migration of services that use end-of-life technologies with minimal cost



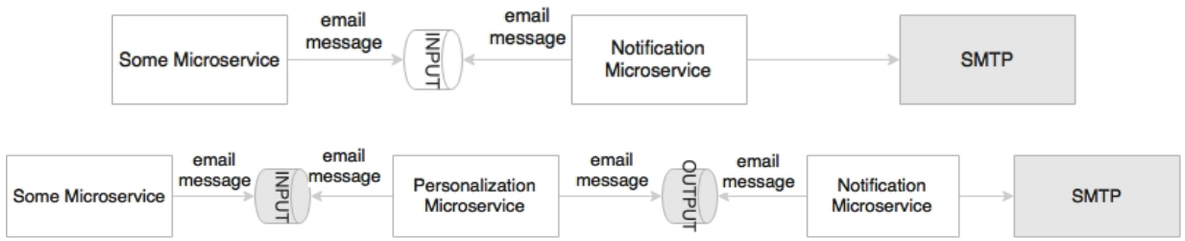
Microservices benefits

- Allowing the coexistence of different versions



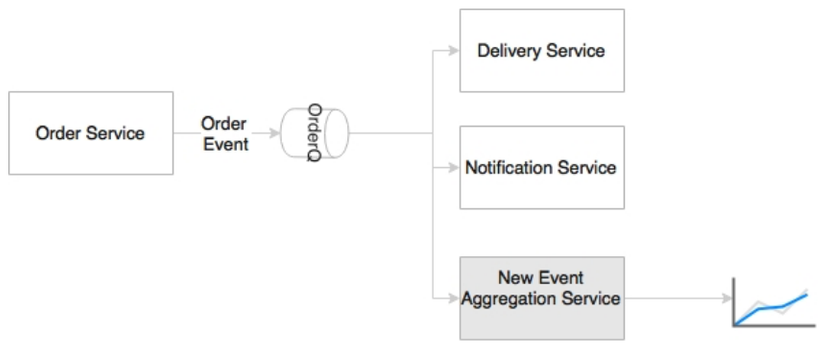
Microservices benefits

- Supporting the building of self-organizing systems
  - A self-organizing system support will automate deployment, be resilient, and exhibit self-healing and self-learning capabilities



Microservices benefits

- Supporting event-driven architecture
  - A well-architected microservice always works with events for both input and output. Once extracted, events can be used for a variety of use cases



Microservices benefits

- Enabling DevOps
  - Microservices are not the ultimate answer, but microservices are at the center stage in many DevOps implementations

Homework 2.1

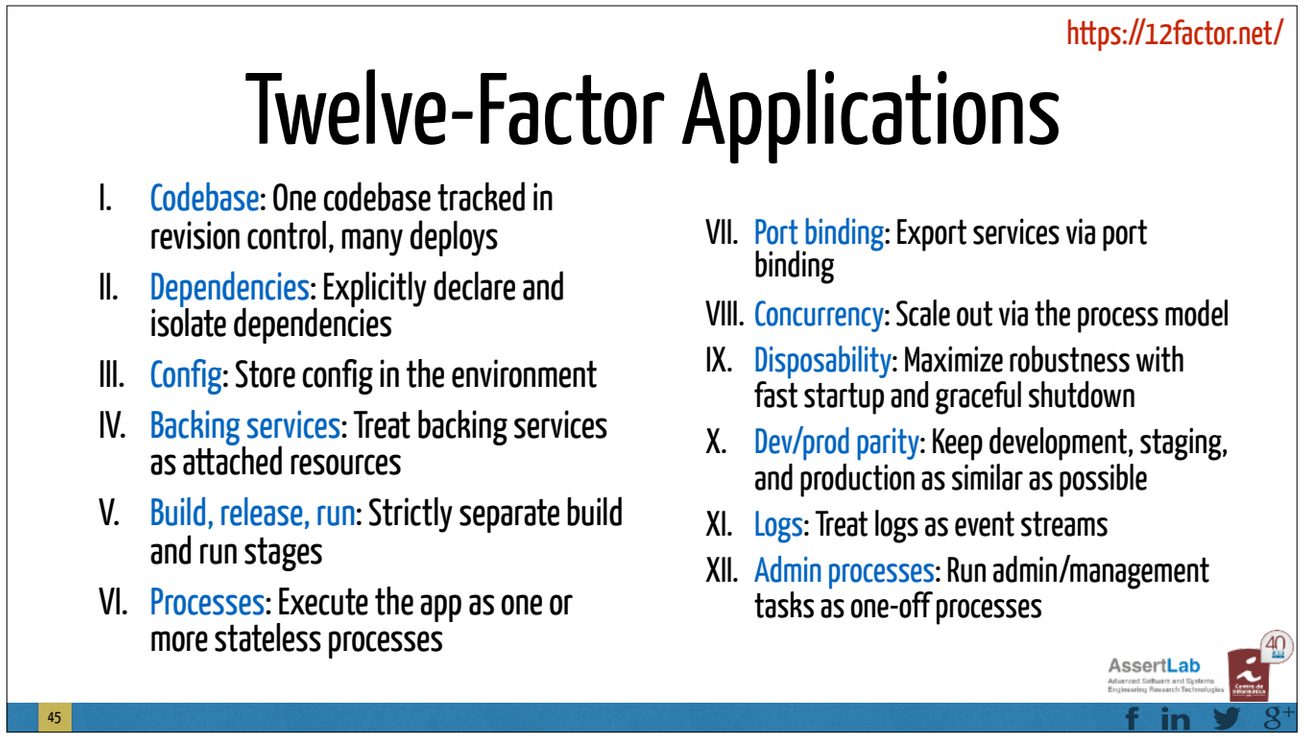
- What are the relationships with other architecture styles?
  - Relations with SOA: Concepts & Principles; Service-oriented integration; Legacy modernization; Service-oriented application; Monolithic migration using SOA

Twelve-Factor Applications

- twelve-factor app é uma coleção de padrões para aplicações nativas pra nuvem, originalmente desenvolvido pelo time de engenheiros da Heroku

- Cloud Foundry, Heroku, e Amazon Elastic Beanstalk são otimizados para implantação de aplicações twelve-factor

- Se refere a uma única unidade de implantação



Essas características se prestam bem à implantação de aplicativos rapidamente, pois **eles fazem poucas ou nenhuma suposição sobre os ambientes nos quais serão implantados**. Essa falta de premissas permite que a plataforma de nuvem subjacente use um mecanismo simples e consistente, facilmente automatizado, para fornecer **rapidamente novos ambientes e implantar esses aplicativos neles**.

Características 12factor

- Fazem poucas ou nenhuma suposição sobre os ambientes nos quais serão implantados
- Mecanismo simples e consistente, facilmente automatizado, para fornecer rapidamente novos ambientes e implantar as apps neles
- Também se prestam bem à idéia de efemeridade, ou aplicações que podemos "jogar fora" com muito pouco custo.
  - Recuperação automática de eventos de falha muito rapidamente

Homework 2.2

- What are the relations between microservices and Twelve-Factor apps?

Microservice use cases

- A microservice is not a silver bullet and will not solve all the architectural challenges
  - Migrating a monolithic application due to improvements required in scalability, manageability, agility, or speed of delivery
  - Utility computing scenarios such as integrating an optimization service, forecasting service, price calculation service... independent stateless computing units that accept certain data, apply algorithms, and return the results
  - Highly agile applications, applications demanding speed of delivery or time to market

Microservice use cases

- There are few scenarios in which we should consider avoiding microservices:
  - If the organization's policies are forced to use centrally managed heavyweight components such as ESB
  - If the organization's culture, processes, and so on are based on the traditional waterfall delivery model, lengthy release cycles, matrix teams, manual deployments...

- Tip: Read more about the Conway's Law

Microservice use cases

- Microservices early adopters
  - Netflix (www.netflix.com)
  - Uber (www.uber.com)
  - Airbnb (www.airbnb.com)
  - Orbitz (www.orbitz.com)
  - eBay (www.ebay.com)
  - Amazon (www.amazon.com)
  - Gilt (www.gilt.com)
  - Twitter (www.twitter.com)
  - Nike (www.nike.com)

Homework 2.3

- Choose 3 early adopters case to tell (briefly) the history

Microservice use cases

- The common theme is monolithic migrations
  - Advantage: they have all the information upfront, readily available for refactoring
  - There is no state called "definite or ultimate microservices".
  - It is a journey and is evolving and maturing day by day

All these enterprises started with monolithic applications and transitioned to a microservices architecture by applying learning and pain points from their previous editions