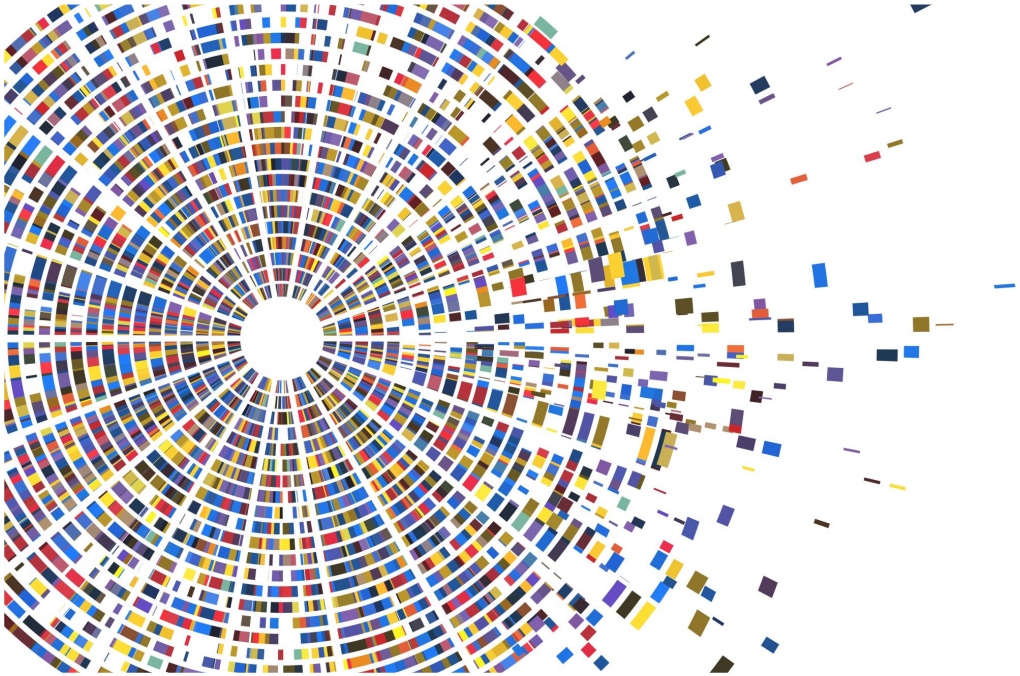


Software Engineering Update : SaaS, SOA, API, Microservices

Royana Afwani

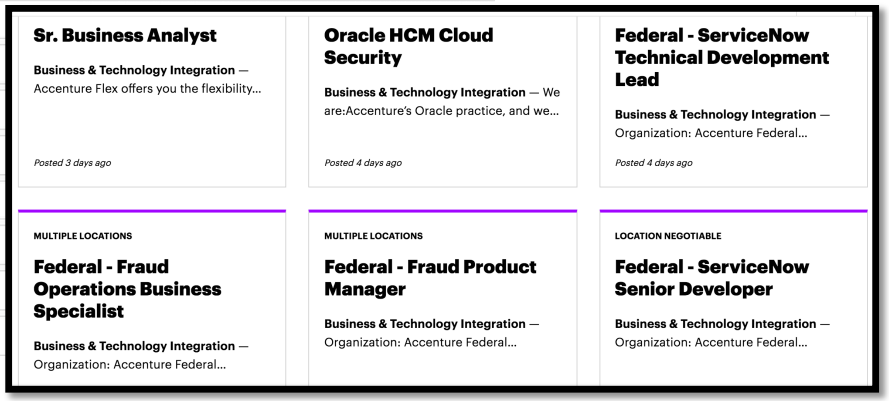
September 2020

Soft. Eng. Update

TelkomGroup mengajak talenta terbaik bangsa untuk bergabung bersama memberikan sentuhan digital bagi Indonesia. melalui inovasi dan kreasi tanpa batas, wujudkan Indonesia 4.0

* Registrasi akan ditutup sampai dengan kebutuhan talent terpenuhi
** ket : khusus untuk posisi Mobile Developer akan dibuka segera

- ARTIFICIAL INTELLIGENCE ENGINEER
- DATA SCIENTIST
- UI DESIGNER
- UX DESIGNER
- UX RESEARCHER
- BACKEND DEVELOPER
- DevSecOps ENGINEER
- FRONTEND DEVELOPER
- SCRUM MASTER
- SOFTWARE ARCHITECT
- SOFTWARE DOCUMENTATION ENGINEER
- SOFTWARE QUALITY ASSURANCE ENGINEER



Sr. Business Analyst
Business & Technology Integration — Accenture Flex offers you the flexibility...
Posted 3 days ago

Oracle HCM Cloud Security
Business & Technology Integration — We are:Accenture's Oracle practice, and we...
Posted 4 days ago

Federal - ServiceNow Technical Development Lead
Business & Technology Integration — Organization: Accenture Federal...
Posted 4 days ago

MULTIPLE LOCATIONS
Federal - Fraud Operations Business Specialist
Business & Technology Integration — Organization: Accenture Federal...

MULTIPLE LOCATIONS
Federal - Fraud Product Manager
Business & Technology Integration — Organization: Accenture Federal...

LOCATION NEGOTIABLE
Federal - ServiceNow Senior Developer
Business & Technology Integration — Organization: Accenture Federal...

- Participate in the entire application lifecycle, focusing on coding and debugging
- Build reusable code and libraries for future use
- Follow emerging technologies
- Develop, improve, and maintain high quality back-end services and APIs
- Doing test driven development
- Within a cross-functional team, collaborate with other developers specializing in backend, frontend,
- quality assurance, product owner, scrum master, and etc
- Apply design patterns and design principles to produce maintainable code
- Learn multiple tech stacks to use the best tools for the job
- Solve technical problems

Requirements :

Pengalaman Kerja : Minimal 2 tahun bekerja sebagai Backend Developer

Digital Technical Skill :
- Knowledge about Microservices Architecture.
- Good Knowledge of RESTFul APIs.
- Experienced in building large-scale web apps/services/APIs.
- Knowledge of Unix/Linux environments and CLI
- Knowledge of SQL / NoSQL database
- Experienced using Cloud Service such as AWS, Azure, etc
- Experienced using Docker & CI/CD deployment
- Familiarity with Test Driven Development
- Good Practice using Gitflow

https://rekrutmen.telkom.co.id/index.php?r=site/detailprohire&id=LoAIrYiPov8QKarZpgtwWjTBoI4RbUstOC64rxdlIRM

https://www.accenture.com/us-en/careers/jobsearch?jk=&sb=1

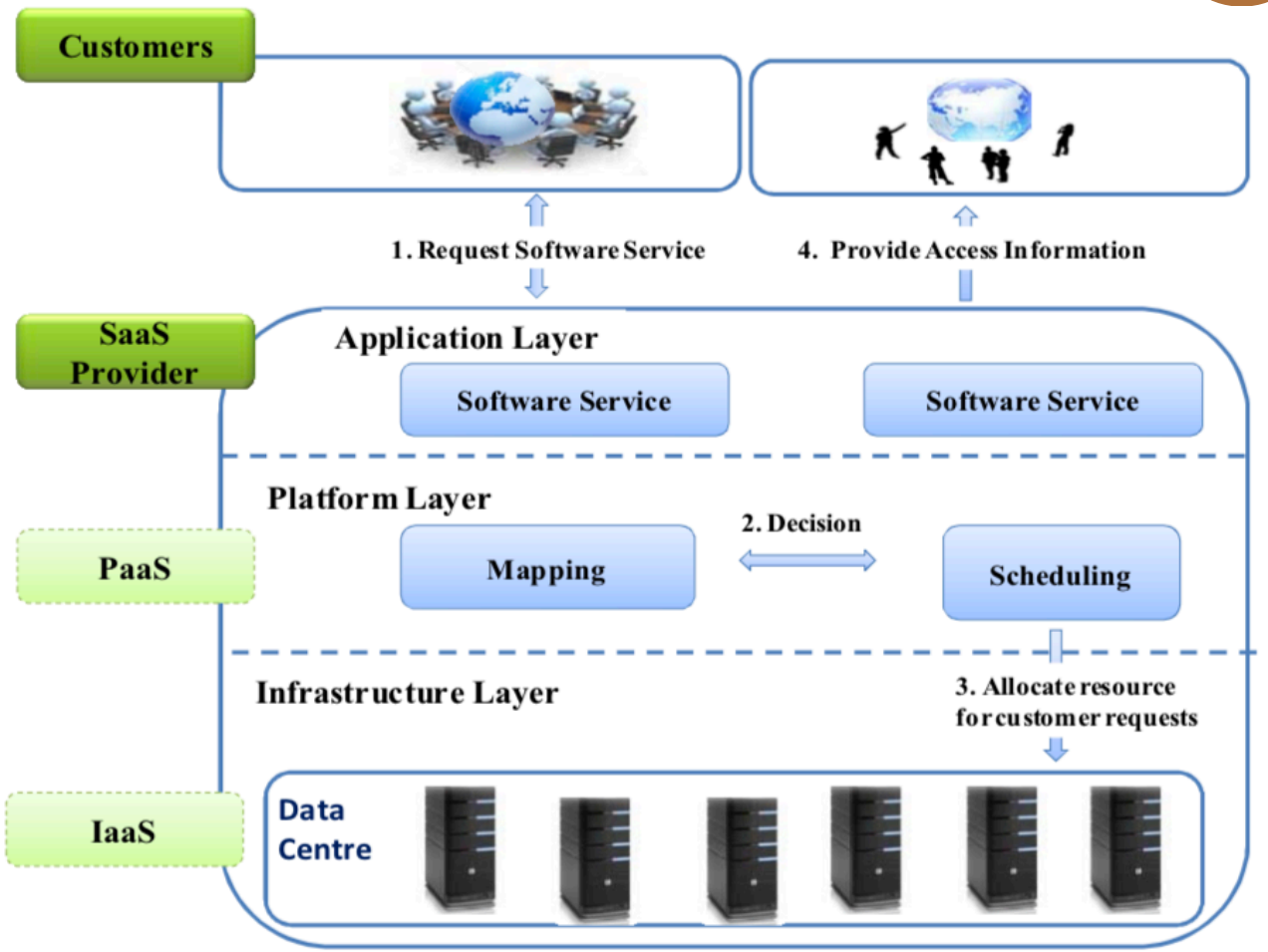
Software as a Services (SaaS)

Cloud Computing : IaaS, PaaS, SaaS

SaaS

- The online delivery of software
- Enterprise customers access business applications over the Internet
    - Rather than buying a software license for an application such as enterprise resource planning (ERP) or customer relationship management (CRM) and installing this software on individual machines, a business signs up to use the application hosted by the company that develops and sells the software, giving the buyer more flexibility to switch vendors and perhaps fewer headaches in maintaining the software

A system model of a SaaS layer structure



Wu, Linlin, Saurabh Kumar Garg, and Rajkumar Buyya. "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments." *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011.

Sample deployment of customer-relationship-management (CRM) software (200-seat license)

	Total cost of ownership, $ thousand		
	Software on premises	**Software as a service**	**Sources of savings with software as a service**
Implementation, deployment			
Customization, integration	108	72	• Reduced deployment time, limited customization, self-service through on-boarding scripts
Basic infrastructure testing, deployment	54	0	• Does not require infrastructure and application testing
Application infrastructure testing, deployment	30	0	
Ongoing operations			
Training	101	34	• Lowers training requirements through –Simpler user interfaces –Self-training, service capabilities
Management, customization of business process change	94	0	• Does not require ongoing business process change management – Vendors monitor customer usage to enhance offering – Customers provide feedback to influence feature functionality
Data center facilities rental, operations; security, compliance; monitoring of incident resolution	750	0	• Includes vendor's costs to serve in subscription price (ongoing operations, back-end hardware and software)
Software			
User licenses, subscriptions; maintenance	480	1,500	
Other			
Unscheduled downtime	308	0	• Provides 99.9% general-server availability vs 99%
Unused licenses	92	0	• Reduces unused licenses by 20%, users added as needed
Total costs (including those not shown here)	2,298	1,640	

Perbedahan Kebutuhan

Aplikasi yang diinstall dan dikelola sendiri

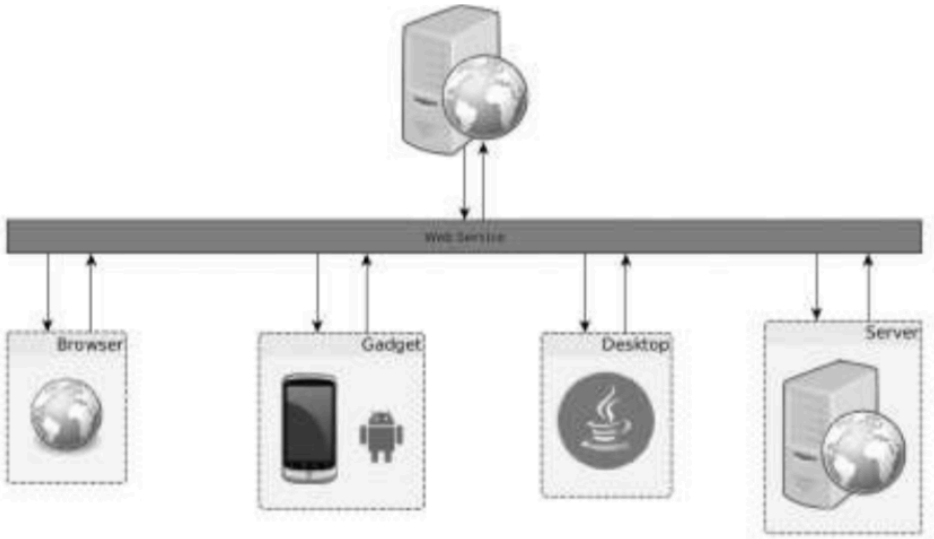
VS

Aplikasi yang disewa melalui internet (SaaS)

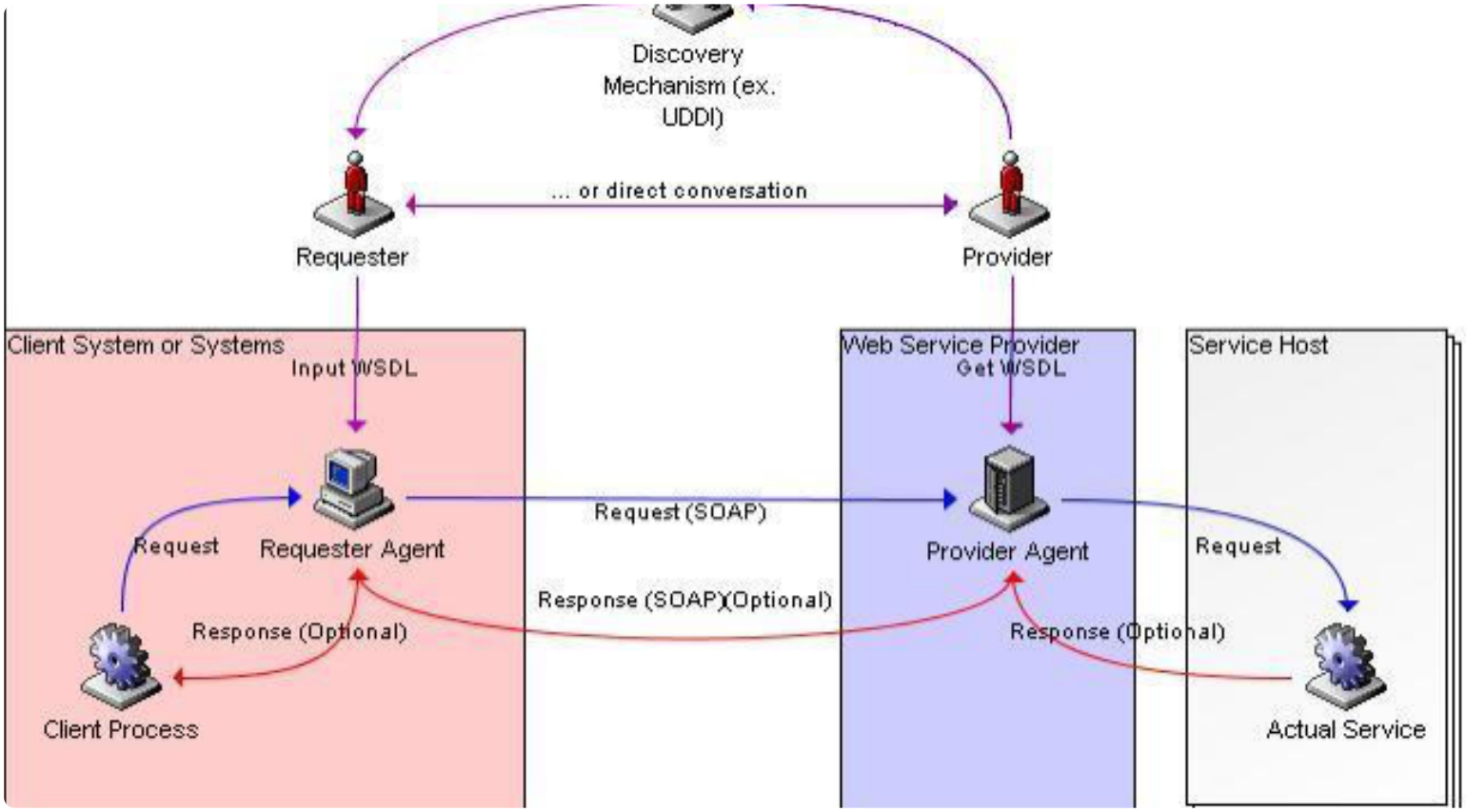
Referensi Tabel :
Dubey, Abhijit, and Dilip Wagle. "Delivering software as a service." *The McKinsey Quarterly* 6.2007 (2007): 2007.

Service Oriented Architecture (SOA)

- SOA : Logika bisnis yang dienkapsulasi sebagai *service*, dan proses komunikasi antar *service* dengan menggunakan *message*. Dalam hal ini, *service layer* akan menjembatani hubungan antara *business logic* dan *application logic* [Thomas Erl]
- *Service* merupakan komponen umum yang digunakan oleh beberapa sistem aplikasi (*reusable*). *Service* direalisasikan dengan menambahkan *interface* (*wrapper*) pada satu atau sekelompok sistem aplikasi. Sistem aplikasi berkomunikasi dengan *service interface* melalui API (*application programmer interface*).
- Implementasi dari *service dapat* menggunakan teknologi *web service* yang berbasis *XML* (*eXtensible Markup Language*), *XSD* (*XML Schema Definition*), *SOAP* (*Simple Object Access Protocol*), *WSDL* (*Web Service Definition Language*), Restful web / Rest API (JSON) , *dll*



Alur *request* dari *service requestor* ke *service provider*



Basic Framework: Anatomy of a Web Service Interaction

https://www.w3.org/TR/ws-i18n-scenarios/

Microservices

- Before : monolith architecture
  - Business- critical enterprise applications are typically large monoliths developed by large teams
  - We couldn’t, change the UI without testing and redeploying the entire application

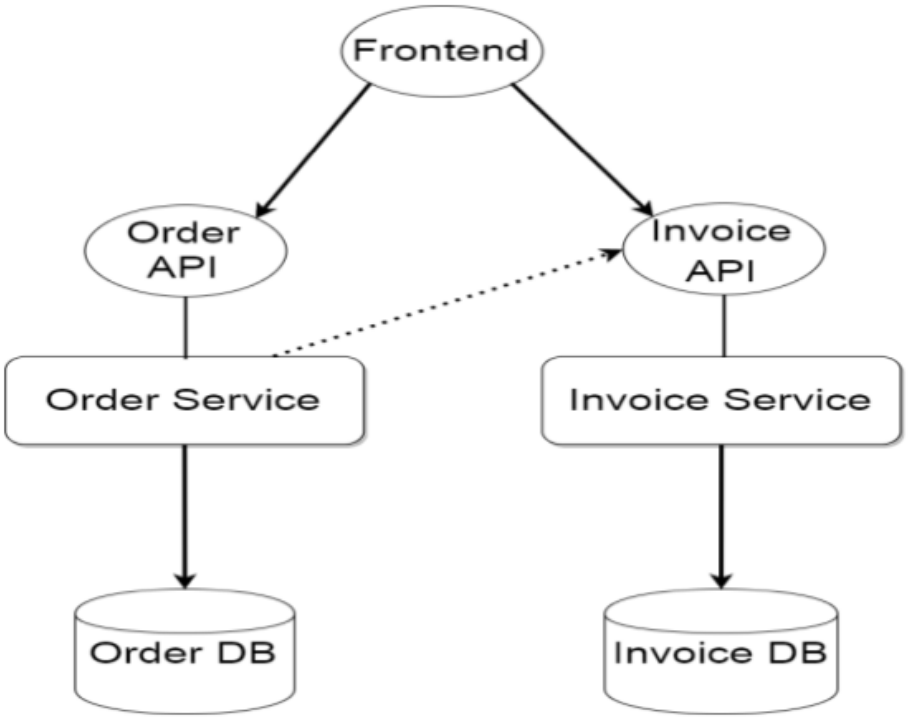
- After : microservices architecture
  - A collection of loosely coupled services
  - The idea of small, loosely coupled teams, rapidly and reliably developing and delivering microservices

Microservices

- *Microservice* adalah kumpulan proses independen dan kecil yang berkomunikasi antara satu dengan lainnya untuk membentuk aplikasi kompleks yang 'bebas' terhadap bahasa API apa pun
- Servis-servis ini terdiri dari blok-blok kecil, terpisah, dan fokus pada tugas-tugas ringan untuk memfasilitasi metode modular dalam pembangunan sistem
- Microservices : Pengembangan lanjutan dari SOA

Buku *Microservices Patterns* oleh Chris Richardson :

> "*Using the microservice architecture with a waterfall development process is like driving a horse-drawn Ferrari—you squander most of the benefit of using microservices. If you want to develop an application with the microservice architecture, it's essential that you adopt agile development and deployment practices such as Scrum or Kanban. Better yet, you should practice continuous delivery/deployment, which is a part of DevOps*"

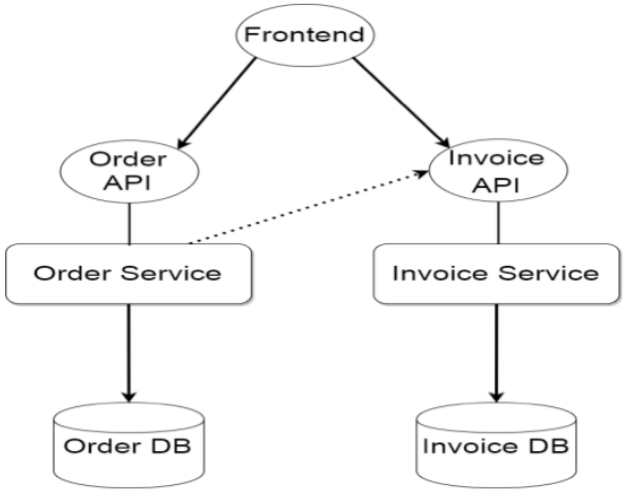


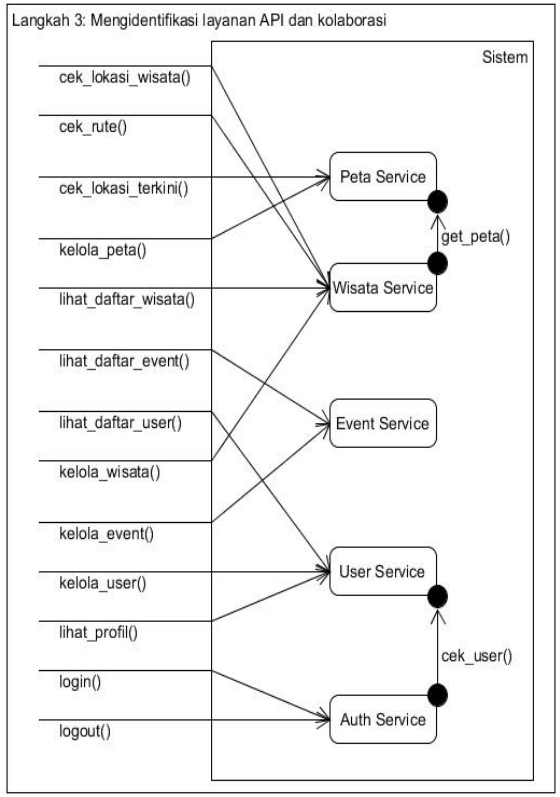
Contoh Microservice

- *frontend* aplikasi memanggil dua layanan yang berbeda, *service* order dan *service invoice*
- Kedua *service* ini memiliki basis kode terpisah dan jika ada kebutuhan untuk berkomunikasi di antara mereka, komunikasi dilakukan melalui API yang disediakan layanan ini

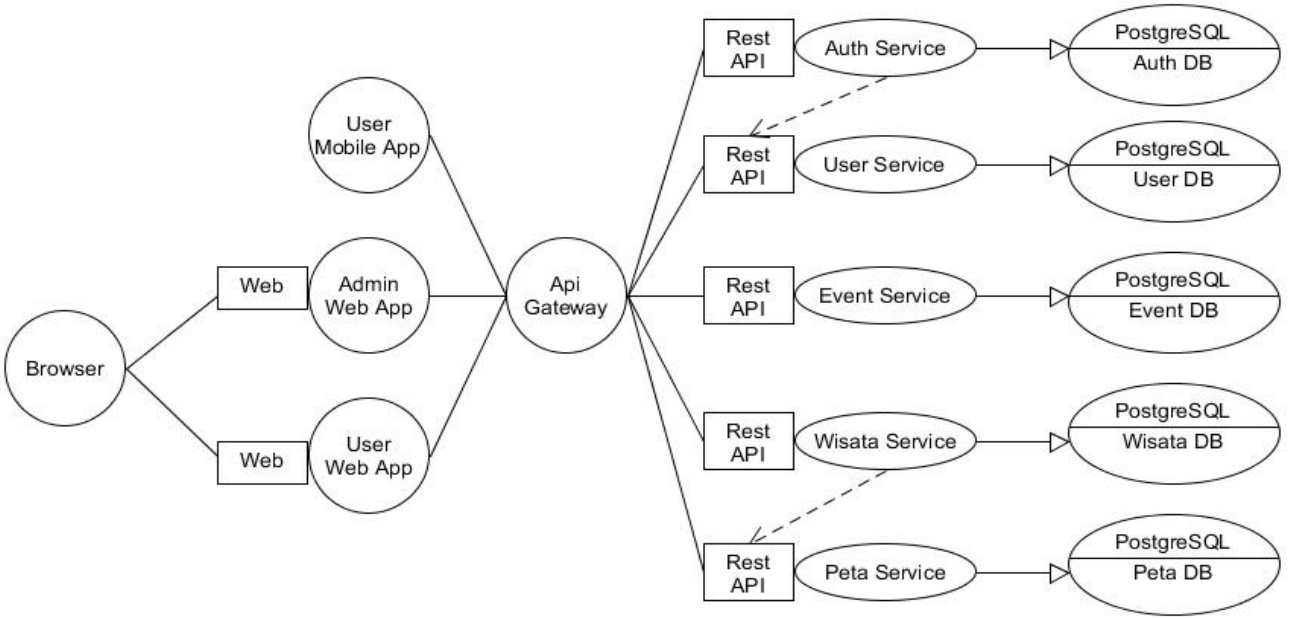
Contoh Microservice

- Karena ukurannya yang kecil dan fokus hanya pada satu konteks bisnis, *microservice* memungkinkan untuk mencapai modularitas yang baik dalam basis kode.
- Modularitas berarti merancang komponen aplikasi secara terpisah dan mandiri.
- Modularitas memudahkan pengembangan karena membuat perubahan dalam satu modul independen dari modul lain.
- Modularitas mudah dipertahankan dengan *microservice* karena ada batasan yang jelas antara masing-masing *service*. Satu *microservice* hanya dapat melihat antarmuka *microservice* lainnya, mencegah panggilan ke metode internal *microservice* lainnya.





Contoh : komunikasi antar *service* pada sistem



Tugas Akhir : Habiburrahman (Angkt. 16)

Kelebihan dan kekurangan microservices

Kategori	Kelebihan & Kekurangan
Time to Market	Lebih lambat pada awalnya, karena tantangan teknis yang dimiliki microservice. Lebih cepat ketika telah berkembang nanti
Refactoring	Lebih mudah dan aman karena perubahan terdapat di dalam microservice
Deployment	Dapat digunakan dalam bagian-bagian kecil, hanya satu service pada satu waktu.
Coding Language	Bahasa dan alat dapat dipilih per service. Servicenya kecil-kecil sehingga mudah berubah
Scaling	Penskalaan dapat dilakukan per service.
DevOps Skills	Berbagai teknologi yang berbeda membutuhkan banyak keterampilan DevOps.
Understandability	Mudah dimengerti karena basis kode bersifat modular dan services menggunakan SRP (Single Responsibility Principle)
Transactions	Sulit diimplementasikan. Konsistensi akhir harus disepakati dalam beberapa kasus
CI, CD (continuous integration, continuous delivery, and continuous deployment)	Dibutuhkan CI dan CD harus digunakan. CD memungkinkan siklus rilis yang lebih cepat.

Contoh penggunaan CRP : GOJEK chat service (which internally called **Icebreaker**). It allowed GOJEK users to communicate with drivers through the app itself, rather than use SMS (which cost both the driver and customer money).
https://blog.gojekengineering.com/applying-the-single-responsibility-principle-to-microservices-7edeb7e4d108

Issues

Sumber → https://microservices.io/patterns/microservices.html

When to use the microservice architecture?

- One challenge with using this approach is deciding when it makes sense to use it. When developing the first version of an application, you often do not have the problems that this approach solves. Moreover, using an elaborate, distributed architecture will slow down development. This can be a major problem for startups whose biggest challenge is often how to rapidly evolve the business model and accompanying application. Using Y-axis splits might make it much more difficult to iterate rapidly. Later on, however, when the challenge is how to scale and you need to use functional decomposition, the tangled dependencies might make it difficult to decompose your monolithic application into a set of services.

How to decompose the application into services?

Another challenge is deciding how to partition the system into microservices. This is very much an art, but there are a number of strategies that can help:

- Decompose by business capability and define services corresponding to business capabilities.
- Decompose by domain-driven design subdomain.
- Decompose by verb or use case and define services that are responsible for particular actions. e.g. a Shipping Service that's responsible for shipping complete orders.
- Decompose by by nouns or resources by defining a service that is responsible for all operations on entities/resources of a given type. e.g. an Account Service that is responsible for managing user accounts.
- Ideally, each service should have only a small set of responsibilities. (Uncle) Bob Martin talks about designing classes using the Single Responsibility Principle (SRP). The SRP defines a responsibility of a class as a reason to change, and states that a class should only have one reason to change. It make sense to apply the SRP to service design as well.

Issues

How to maintain data consistency?

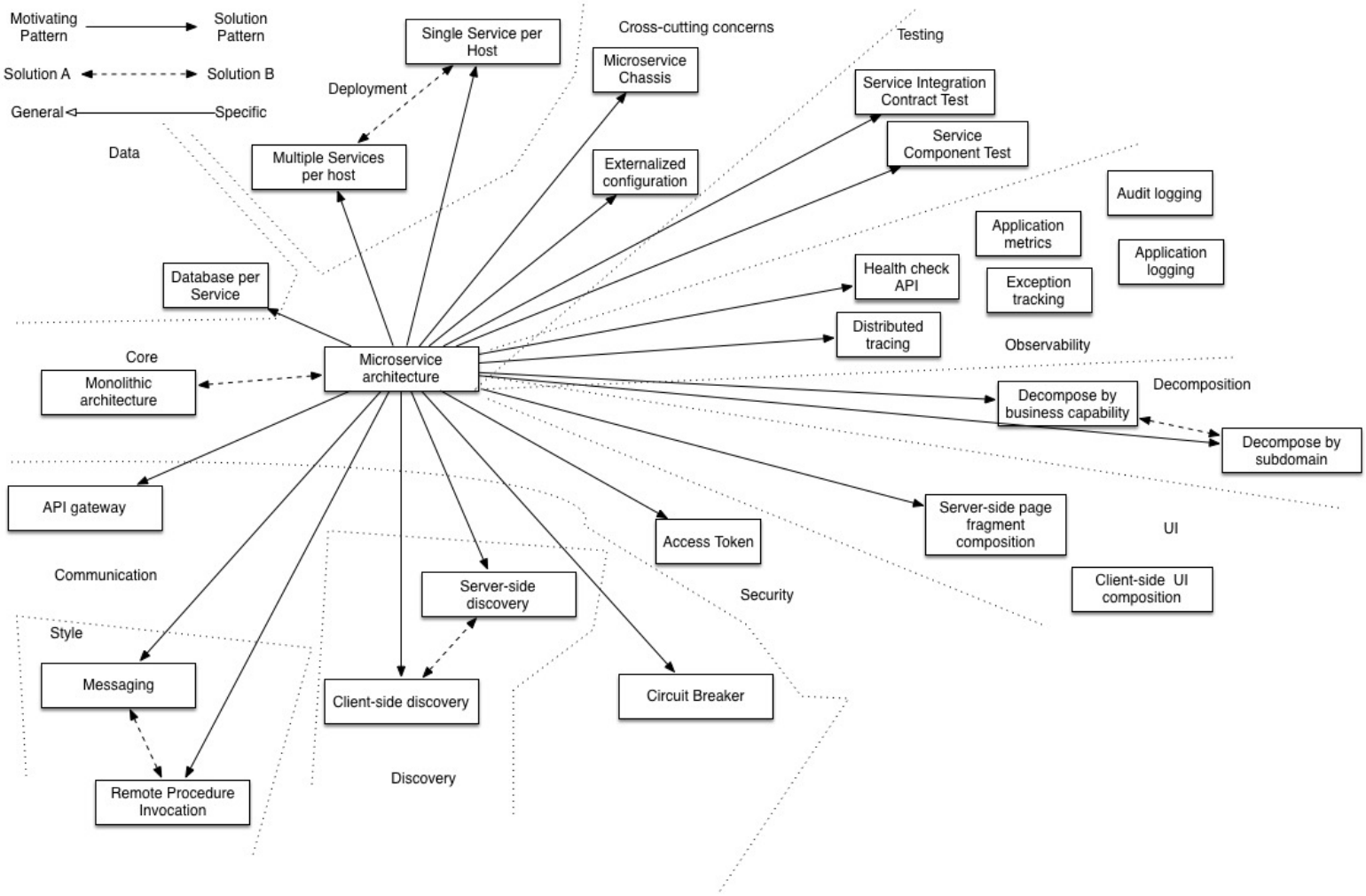
- In order to ensure loose coupling, each service has its own database. Maintaining data consistency between services is a challenge because 2 phase-commit/distributed transactions is not an option for many applications. An application must instead use the Saga pattern. A service publishes an event when its data changes. Other services consume that event and update their data. There are several ways of reliably updating data and publishing events including Event Sourcing and Transaction Log Tailing.

How to implement queries?

- Another challenge is implementing queries that need to retrieve data owned by multiple services. The API Composition and Command Query Responsibility Segregation (CQRS) patterns.

Sumber → https://microservices.io/patterns/microservices.html

Microservices : Pattern



Sumber → https://microservices.io/patterns/microservices.html

Applying the Single Responsibility Principle to Microservices : Case Study (Gojek) .

- In 2018, Gojek released a new chat service (which we internally called **Icebreaker**). It allowed our users to communicate with drivers through the app itself, rather than use SMS (which cost both the driver and customer money).

The Problem(s)

In a nutshell, Icebreaker depended on too many other services to function properly. Let's look at some of the tasks Icebreaker performed in order to create a channel:

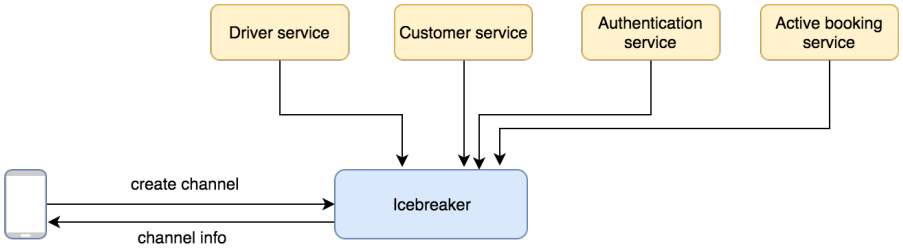
1.Authorise the API call: This made a call to our authentication service.

2.Fetch the customer profile: This required an HTTP call to our customer service.

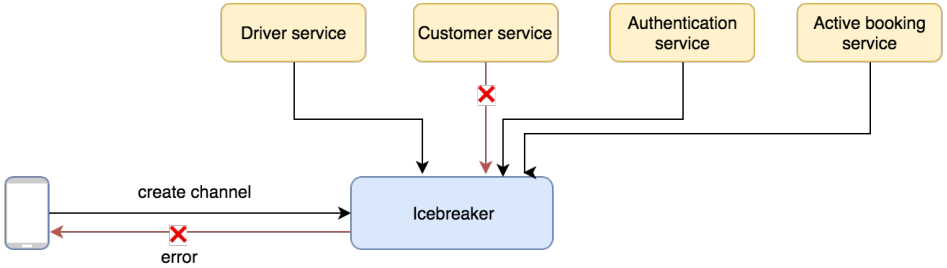
3.Fetch the drivers' profile: This required an HTTP call to our driver service.

4.Verify if the customer-driver pair are in an active order: This made a call to our active booking storage service.

5.Create the channel.



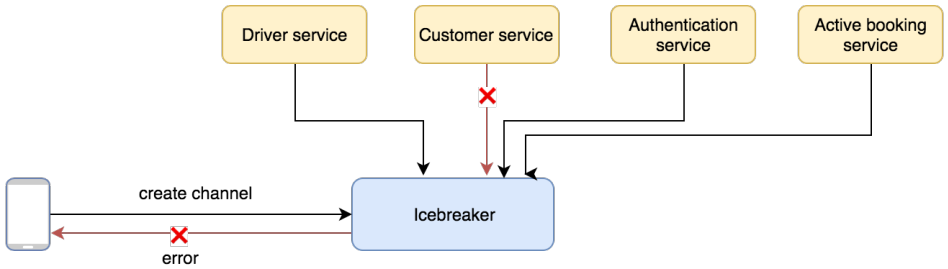
If any of these services failed, Icebreaker would fail as well.



When a service starts to do too many things, it's bound to fail sooner or later. In this case, Icebreaker's job was to create a channel between a customer and a driver. However, it was doing all this extra stuff: like authentication, verification, and profile retrieval.

Applying the Single Responsibility Principle to Microservices : Case Study (Gojek) . Part 2

- In 2018, Gojek released a new chat service (which we internally called **Icebreaker**). It allowed our users to communicate with drivers through the app itself, rather than use SMS (which cost both the driver and customer money).



Simultaneous dependence on multiple services ensures that the dependent service is less stable than any of them

SOLUTION ?

https://blog.gojekengineering.com/applying-the-single-responsibility-principle-to-microservices-7edeb7e4d108

Penerapan Microservices

- Most large scale web sites including Netflix, Amazon and eBay have evolved from a monolithic architecture to a microservice architecture.
- Netflix, which is a very popular video streaming service that's responsible for up to 30% of Internet traffic, has a large scale, service-oriented architecture. They handle over a billion calls per day to their video streaming API from over 800 different kinds of devices. Each API call fans out to an average of six calls to backend services.
- Amazon.com originally had a two-tier architecture. In order to scale they migrated to a service-oriented architecture consisting of hundreds of backend services. Several applications call these services including the applications that implement the Amazon.com website and the web service API. The Amazon.com website application calls 100-150 services to get the data that used to build a web page.
- The auction site ebay.com also evolved from a monolithic architecture to a service-oriented architecture. The application tier consists of multiple independent applications. Each application implements the business logic for a specific function area such as buying or selling. Each application uses X-axis splits and some applications such as search use Z-axis splits. Ebay.com also applies a combination of X-, Y- and Z-style scaling to the database tier.

Here are list of articles published by companies about their experiences using microservices:

- Comcast Cable
- Uber
- Netflix
- Amazon
- Ebay
- Sound Cloud
- Karma
- Groupon
- Hailo
- Gilt
- Zalando
- Capital One Why Capital One is at Re:Invent and Keynote
- Lending Club
- AutoScout24

Sumber → https://microservices.io/patterns/microservices.html

Selanjutnya?

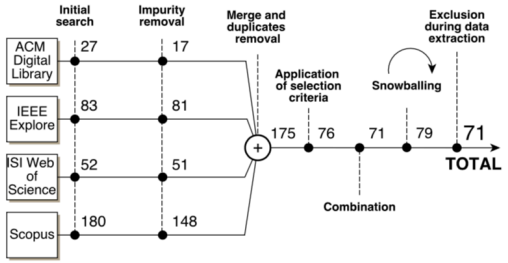


Fig. 1. Overview and numbers of the search and selection process

Problems	#Studies
Complexity	19
Low flexibility	19
Resources management	16
Service composition	15
Data management	13
Modernization	11
Low auditability	7
Runtime uncertainty	7
Low portability	5
Low testability	5
Realtime communication	4
Security	4
Time to market	4

Lifecycle scope	#Studies
Design	65
Implementation	24
Operation	22
Maintenance	12
Testing	10
Requirements	1

Arch. activities	#Studies
	Intr
Architectural Analysis	56
Architectural Implementation	29
Architecture Description	23
Architectural Evaluation	18
Architectural Maintenance and Evolution	15
Architecture Understanding	10
Architecture Reuse	6
Architectural Synthesis	6
Architecture Recovery	5
Architecture Impact Analysis	2

Quality attr.	#Studies
Performance efficiency	40
Maintainability	28
Security	17
Functional suitability	14
Reliability	14
Compatibility	14
Usability	13
Portability	12

Res. strategies	#Studies
Solution proposal	48
Validation research	14
Opinion paper	8
Experience paper	8
Philosophical paper	3
Evaluation research	3

Contribution	#Studies
Application	21
Method	18
Reference architecture	16
Problem framing	14
Middleware	12
Design pattern	2
Architectural language	2

Perspective	#Studies
Cloud	21
System quality	21
Migration	16
Domain-specific	11
IoT	6
Mobile oriented	6
Other	10

Design patterns	#Studies
API Gateway	11
Publish/subscribe	8
Circuit breaker	6
Proxy	4
Load balancer	3
Other	9

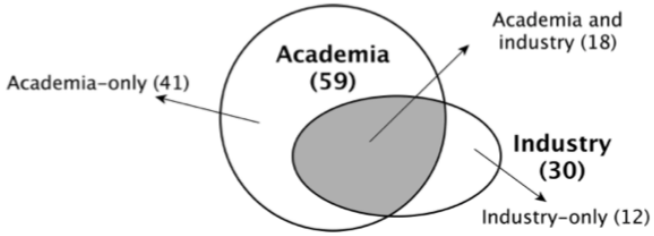


Fig. 4. Distribution of industry involvement

Di Francesco, Paolo, Ivano Malavolta, and Patricia Lago. "Research on architecting microservices: Trends, focus, and potential for industrial adoption." *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017.