

# Arsitektur Perangkat Lunak

Departemen Informatika, Universitas Pradita, Indonesia

2023



# Daftar Isi

<b>1</b>	<b>Pendahuluan</b>	<b>1</b>
	Alfa Yohannis	
1.1	Materi . . . . .	1
<b>2</b>	<b>Arsitektur Client-Server</b>	<b>3</b>
	Alfa Yohannis	
2.1	Latar Belakang . . . . .	3
2.2	Arsitektur Client-Server . . . . .	3
2.3	Kelebihan dan Kekurangan . . . . .	4
2.3.1	Kelebihan . . . . .	4
2.3.2	Kekurangan . . . . .	5
2.4	Contoh Kasus . . . . .	5
2.4.1	Deskripsi . . . . .	5
2.4.2	Penjelasan Implementasi . . . . .	5
2.5	Kesimpulan . . . . .	6
<b>3</b>	<b>Arsitektur MVC (Model-View-Controller)</b>	<b>7</b>
	Alfa Yohannis	
3.1	Latar Belakang . . . . .	7
3.2	Arsitektur Model-View-Controller . . . . .	7
3.3	Kelebihan dan Kekurangan . . . . .	8
3.3.1	Kelebihan . . . . .	8
3.3.2	Kekurangan . . . . .	9
3.4	Contoh Kasus . . . . .	9
3.4.1	Deskripsi . . . . .	9
3.4.2	Penjelasan Implementasi . . . . .	9
3.5	Kesimpulan . . . . .	10
<b>4</b>	<b>Arsitektur MVVM (Model-View-ViewModel)</b>	<b>11</b>
	Alfa Yohannis	
4.1	Latar Belakang . . . . .	11

4.2	Arsitektur Model-View-ViewModel . . . . .	12
4.3	Kelebihan dan Kekurangan . . . . .	12
4.3.1	Kelebihan . . . . .	12
4.3.2	Kekurangan . . . . .	13
4.4	Contoh Kasus . . . . .	13
4.4.1	Deskripsi . . . . .	13
4.4.2	Penjelasan Implementasi . . . . .	14
4.5	Kesimpulan . . . . .	15
<b>5</b>	<b>Layered Architecture</b>	<b>17</b>
	Austin Nicholas Tham, Darren Valentio, Muhammad	
5.1	Definisi <i>Layered Architecture</i> . . . . .	17
5.2	Latar Belakang . . . . .	18
5.3	Pros Cons . . . . .	18
5.3.1	Pros . . . . .	18
5.3.2	Cons . . . . .	19
5.4	Software Architecture Pattern . . . . .	19
5.5	Design Patterns . . . . .	20
5.5.1	Contoh penerapan layered architecture: . . . . .	21
<b>6</b>	<b>Pendahuluan</b>	<b>23</b>
6.1	Materi . . . . .	23
<b>7</b>	<b>Pendahuluan</b>	<b>25</b>
7.1	Materi . . . . .	25
<b>8</b>	<b>Pendahuluan</b>	<b>27</b>
8.1	Materi . . . . .	27
<b>9</b>	<b>Pendahuluan</b>	<b>29</b>
9.1	Materi . . . . .	29
<b>10</b>	<b>Pendahuluan</b>	<b>31</b>
10.1	Materi . . . . .	31
<b>11</b>	<b>Pendahuluan</b>	<b>33</b>
11.1	Materi . . . . .	33
<b>12</b>	<b>Microservices</b>	<b>35</b>
12.1	Definisi <i>Microservices</i> . . . . .	35
12.2	Karakteristik <i>Microservices</i> . . . . .	35
12.3	Kelebihan <i>Microservices</i> . . . . .	36

12.4 Kekurangan <i>Microservices</i> . . . . .	36
12.5 Contoh dari <i>Microservices</i> . . . . .	36
12.6 Kesimpulan . . . . .	36
<b>13 Pendahuluan</b>	<b>37</b>
13.1 Materi . . . . .	37
<b>14 DevOps</b>	<b>39</b>
Hendra Lijaya, Oktavianus Hendry Wijaya	
14.1 Pengertian . . . . .	39
14.2 Fungsi . . . . .	39
14.3 Arsitektur . . . . .	39
14.4 Kelebihan Kekurangan . . . . .	41
14.4.1 Kelebihan . . . . .	41
14.4.2 Kekurangan . . . . .	42
14.5 Perbedaan DevOps dan nonDevOps . . . . .	42
14.6 Tools . . . . .	43
14.7 Contoh Kasus . . . . .	45
14.8 Code . . . . .	46
<b>Daftar Pustaka</b>	<b>47</b>



# Bab 1

## Pendahuluan

ALFA YOHANNIS

### 1.1 Materi

1. Introduction
2. Client-Server Architecture
3. Model-View-Controller Architecture
4. Model-View-ViewModel Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

AAAA [?]





# Bab 2

## Arsitektur Client-Server

ALFA YOHANNIS

### 2.1 Latar Belakang

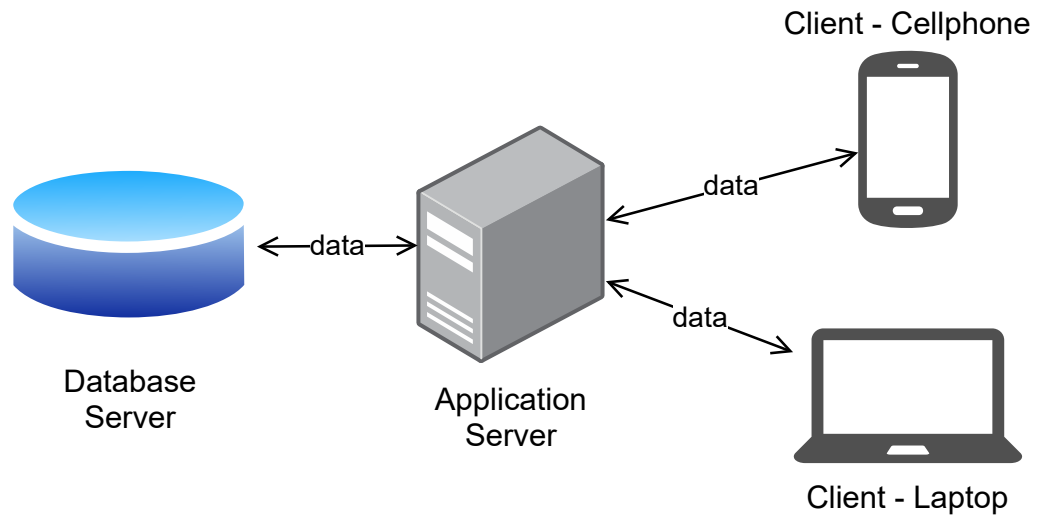
Pada awal komputer bermula sebagai suatu kesatuan, tidak terpisah-pisah. Perangkat lunak hanya berjalan pada satu unit komputer tersebut. Secara perlahan, ada bagian komputer yang dapat terpisah secara fisik dan menjalankan tanggung jawab tertentu. Sebagai contoh, data storage terpisah dari komputer utama. Lalu, beberapa fungsionalitas akhirnya terpisah dan membutuhkan mesin tersendiri. Misalnya, komputer yang didedikasikan untuk menyimpan data atau yang kita sebut sebagai *database server*. Di sisi lain, jaringan komputer juga berkembang dan kemudian menjadi sesuatu yang umum. Komputer-komputer saling berkomunikasi satu sama yang lain, dan setiap komputer dapat memiliki peran-peran tertentu yang memungkinkan lahirnya sistem terdistribusi.

### 2.2 Arsitektur Client-Server

Suatu sistem *client-server* terdiri dari satu *server* dan satu *client* atau lebih. *Server* biasanya memiliki kemampuan komputasi dan penyimpanan data yang lebih cepat dan banyak dibanding *client*. Oleh karena itu, *client* menugaskan *server* untuk melakukan komputasi tertentu dan menerima hasilnya atau sekedar menarik data dari *server*.

Terdapat 2 jenis *client-server architecture*: *two-tier architecture* dan *three-tier architecture*. Two tier-architecture umumnya hanya terdiri dari *desktop application* yang berada di sisi klien dan *database* yang berada di sisi server. Contoh lain adalah *web browser* yang memuat *web application* dan *web server*

untuk melakukan *backend computation*. Arsitektur tersebut dapat diperluas menjadi *three-tier architecture*, dengan menambahkan *database server* seperti yang ditampilkan pada Gambar 14.2.



Gambar 2.1: Skema dari 3-tier client-server arsitektur.

## 2.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur client-server:

### 2.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur client-server adalah:

- Kemampuan komputasi (dan penyimpanan data) dapat diakses dari berbagai lokasi berjauhan dan oleh banyak komputer/pengguna.
- Komputasi-komputasi yang membutuhkan kinerja tinggi dapat didelegasikan ke server.
- Data dapat disentralisasikan sehingga meningkatkan konsistensi data dan mengurangi duplikasi data.
- Sistem dapat menerapkan *horizontal scaling* untuk skalabilitas. Horizontal scaling adalah meningkatkan kinerja komputer dengan penambahan komputer agar beban komputasi dibagi ke komputer-komputer

yang tersedia. Misalnya, awalnya terdapat 10 000 requests perhari yang ditangani oleh suatu *application server*. Jika *application server* ditambah, maka beban tersebut dibagi di antara kedua *server* tersebut. Vertical scaling adalah meningkat kinerja suatu komputer dengan menaikkan spesifikasi komputer tersebut, misalnya dengan menggunakan prosesor yang lebih cepat atau meningkatkan kapasitas memori.

### 2.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur client-server adalah sistem jadi lebih kompleks untuk dikelola:

- Biaya akan meningkat karena terdapat komponen/mesin tambahan yang perlu dikelola.
- Faktor keamanan juga perlu diperhatikan karena server dan client beroperasi dalam suatu jaringan komputer yang mana rawan terhadap *cyber attack*.
- Perlunya koordinasi antar-komputer, misalnya komunikasi sinkron dan asinkron serta komputasi parallel.
- Kompatibilitas antara *server* dan *client* maupun sesama klien.
- Masalah-masalah yang umum terdapat pada jaringan komputer et-work problems, misalnya *network latency*, kesalahan dalam konfigurasi jaringan, dsb.

## 2.4 Contoh Kasus

### 2.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

### 2.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

## **2.5 Kesimpulan**

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.

## Bab 3

# Arsitektur MVC (Model-View-Controller)

ALFA YOHANNIS

### 3.1 Latar Belakang

Pada mulanya pengembangan perangkat lunak menyatukan fungsi-fungsi dari *graphical user interface* (GUI) dan pengelolaan data ke dalam satu kode tanpa memisahkan mereka sesuai dengan perhatian (*concerns*) mereka masing-masing. Konsekuensinya, pola tersebut akan menimbulkan masalah ketika *developer* diminta untuk membangun aplikasi skala besar, misalnya aplikasi yang menolong pengguna berinteraksi dengan dataset yang besar dan kompleks. Kode program akan menjadi lebih tidak terstruktur (*spaghetti code*) dan sulit untuk dipahami. Sebagai solusi, kode program perlu dibagi ke dalam komponen-komponen sesuai dengan perhatian mereka (*separation of concerns*). Arsitektur Model-View-Controller (MVC) kemudian diajukan untuk membagi kode program ke dalam tiga abstraksi utama: *model*, *view*, dan *controller*.

### 3.2 Arsitektur Model-View-Controller

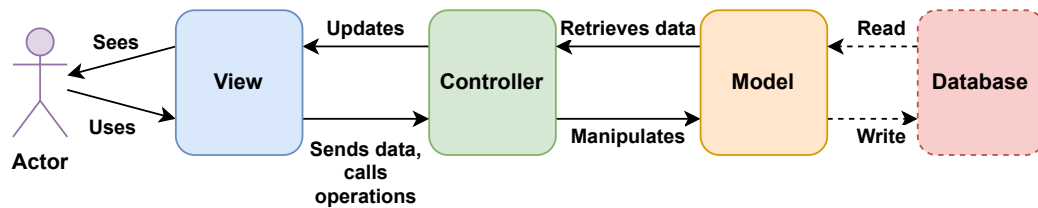
Arsitektur MVC adalah pola arsitektur untuk pengembangan *Graphical User Interface* (GUI). Arsitektur tersebut membagi logika program menjadi 3 bagian yang saling terhubung: Model, View, dan Controller. Skema dari MVC dapat dilihat pada Gambar 3.1..

**Model** ditujukan untuk berinteraksi dengan data: menyimpan, memperharui, menghapus, dan menarik data dari database. Model juga digunakan

untuk menggagregasi data sesuai dengan logika bisnis yang dijalankan.

**View** merupakan presentasi yang ditampilkan ke pengguna yang dengannya pengguna dapat berinteraksi. Misalnya, halaman web, GUI desktop, diagram, *text fields*, *buttons*, dsb.

**Controller** bertugas untuk menerima input dari pengguna melalui *view* dan meneruskan input tersebut ke model untuk disimpan atau diproses lebih lanjut. Controller juga menarik data dari *model* dan memembetuknya demikian rupa sehingga siap untuk dikirimkan ke *view* untuk ditampilkan ke pengguna.



Gambar 3.1: Arsitektur Model-View-Controller (MVC).

### 3.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur MVC:

#### 3.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur MVC adalah:

- Pemisahan presentasi dan data membolehkan model ditampilkan di banyak *view* secara bersamaan.
- View bersifat *composable* artinya view dapat dibangun dari berbagai atau berisi *subviews/fragments*.
- Controller satu dapat diganti (*switchable*) dengan controller lain pada saat *runtime*.
- Developer dapat membuat berbagai macam mekanisme pemrosesan data dari input ke output dengan mengkombinasikan berbagai macam fungsionalitas yang dimiliki oleh views, controllers, dan models.
- *Data engineers*, *backend* dan *frontend developers* masing-masing dapat fokus mengerjakan tugas utama mereka. Misal, *data engineers* hanya mengerjakan tugas yang berkaitan dengan data, sedangkan *frontend developers* fokus ke *user interface*.

### 3.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur MVC adalah sebagai berikut:

- Derajat kompleksitas kode program bertambah karena kode harus dibagi ke dalam tiga abstraksi yang berbeda.
- *Developers* harus mengikuti aturan ketat tertentu dalam mendefinisikan *controllers*, *models*, dan *views*.
- Secara relative, MVC lebih sulit dipahami dikarenakan struktur bawaannya.
- Terlalu berlebihan (*overkill*) untuk aplikasi sederhana.
- Cocok untuk pembangunan Graphical User Interface tetapi belum tentu cocok untuk pengembangan aplikasi atau komponen yang lain.
- Adanya lapisan-lapisan abstraksi dapat mengurangi kinerja (*performance*) aplikasi.

## 3.4 Contoh Kasus

### 3.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

### 3.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

Listing 3.1: Model dari Rate.

```
1 import javax.persistence.Entity;
2 import javax.persistence.Id;
3 import javax.persistence.IdClass;
4
5 @Entity
6 @IdClass(RateId.class)
7 public class Rate {
8     @Id
9     private String fromCurrency;
```

```

10    @Id
11    private String toCurrency;
12    private Double rate;
13    ...
14    Rate(String fromCurrency, String toCurrency, Double
        rate) {
15        ...
16    }
17    ...
18 }

```

Listing 3.2: RateRepository.

```

1 import java.util.Collection;
2 import org.springframework.data.jpa.repository.Query;
3 import org.springframework.data.repository.
    CrudRepository;
4
5 public interface RateRepository extends CrudRepository<
    Rate, Integer> {
6    @Query("SELECT _r FROM _Rate _r WHERE _r.fromCurrency = ?1
    _and _r.toCurrency = ?2")
7    Collection<Rate> findFirstByFromCurrencyAndToCurrency(
    String fromCurrency, String toCurrency);
8
9    @Query("SELECT DISTINCT(r.fromCurrency) FROM _Rate _r")
10    Collection<String> findAllFromCurrency();
11
12    @Query("SELECT DISTINCT(r.toCurrency) FROM _Rate _r")
13    Collection<String> findAllToCurrency();
14 }

```

### 3.5 Kesimpulan

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.



## Bab 4

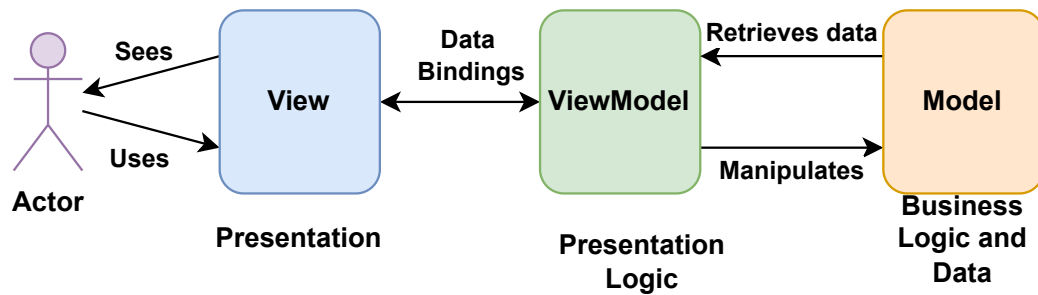
# Arsitektur MVVM (Model-View-ViewModel)

ALFA YOHANNIS

### 4.1 Latar Belakang

Pada mulanya, dalam pengembangan perangkat lunak, kode yang bertanggung jawab terhadap data, logika bisnis, dan tampilan (Graphical User Interface) bercampur jadi satu, tidak ada pemisahan abstraksi. Pola Model-View-Controller kemudian muncul memisahkan kode program ke dalam 3 abstraksi utama berdasarkan perhatian mereka: *model* untuk data, *view* untuk tampilan, dan *controller* untuk logika bisnis. Hanya saja, MVC tidak memiliki abstraksi yang secara eksplisit mengelola *states* dari tampilan (*views*). Pola MVP (Model-View-Presenter) kemudian diajukan di mana komponen *Presenter*-nya bertanggung jawab mengelola logika presentasi dari *views*. Walaupun demikian, kode program yang mengelola sinkronisasi antara views dan state dari logika presentasi mereka masih harus dibuat secara manual.

Keunikan dari Model-View-ViewModel adalah pola tersebut memiliki komponen *binder* yang mengotomasi komunikasi/sinkronisasi antara view dengan properties yang ada pada *view model*. Nilai-nilai pada *view* ditautkan dengan properties pada view model sehingga perubahan nilai pada salah komponen di view (misalnya perubahan pada *textbox*) akan memperbarui juga nilai pada *property*-nya di *view model* yang ditautkan pada komponen tersebut. Adanya binder mengurangi jumlah kode yang harus ditulis oleh developer secara manual untuk melakukan sinkronisasi antara *view* dan *view model*.



Gambar 4.1: Arsitektur Model-View-ViewModel (MVVM).

## 4.2 Arsitektur Model-View-ViewModel

- Separation of the view layer by moving all GUI code to the view model via data binding.
- UI developers don't write the GUI, instead a markup language is used.
- The separation of roles allows UI designers to focus on the UX design rather than programming of the business logic.
- A proper separation of the view from the model is more productive, as the user interface typically changes frequently and late in the development cycle based on end-user feedback.
- Data bindings and properties are used to synchronise the relevant values in the view and the view model, that represents the state of the view, so that they are always the same.
- It eliminates or minimises application logic that directly manipulates the view.

## 4.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur MVVM:

### 4.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur MVVM adalah:

- Separation of the view layer by moving all GUI code to the view model via data binding.

- UI developers don't write the GUI, instead a markup language is used.
- The separation of roles allows UI designers to focus on the UX design rather than programming of the business logic.
- A proper separation of the view from the model is more productive, as the user interface typically changes frequently and late in the development cycle based on end-user feedback.
- Data bindings and properties are used to synchronise the relevant values in the view and the view model, that represents the state of the view, so that they are always the same.
- It eliminates or minimises application logic that directly manipulates the view.

### 4.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur MVVM adalah sebagai berikut:

- It can be overkill for small projects.
- Generalizing the viewmodel upfront can be difficult for large applications.
- Large-scale data binding can lead to lower performance.
- It's best for UI development but might not be the best for other types of developments and applications.

## 4.4 Contoh Kasus

### 4.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

### 4.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

Listing 4.1: Model dari Rate.

```

1  import javax.persistence.Entity;
2  import javax.persistence.Id;
3  import javax.persistence.IdClass;
4
5  @Entity
6  @IdClass(RateId.class)
7  public class Rate {
8      @Id
9      private String fromCurrency;
10     @Id
11     private String toCurrency;
12     private Double rate;
13     ...
14     Rate(String fromCurrency, String toCurrency, Double
        rate) {
15         ...
16     }
17     ...
18 }

```

Listing 4.2: RateRepository.

```

1  import java.util.Collection;
2  import org.springframework.data.jpa.repository.Query;
3  import org.springframework.data.repository.
    CrudRepository;
4
5  public interface RateRepository extends CrudRepository<
    Rate, Integer> {
6      @Query("SELECT r FROM Rate r WHERE r.fromCurrency = ?1
        and r.toCurrency = ?2")
7      Collection<Rate> findFirstByFromCurrencyAndToCurrency(
        String fromCurrency, String toCurrency);
8
9      @Query("SELECT DISTINCT(r.fromCurrency) FROM Rate r")
10     Collection<String> findAllFromCurrency();
11
12     @Query("SELECT DISTINCT(r.toCurrency) FROM Rate r")
13     Collection<String> findAllToCurrency();

```

14 }

## 4.5 Kesimpulan

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.



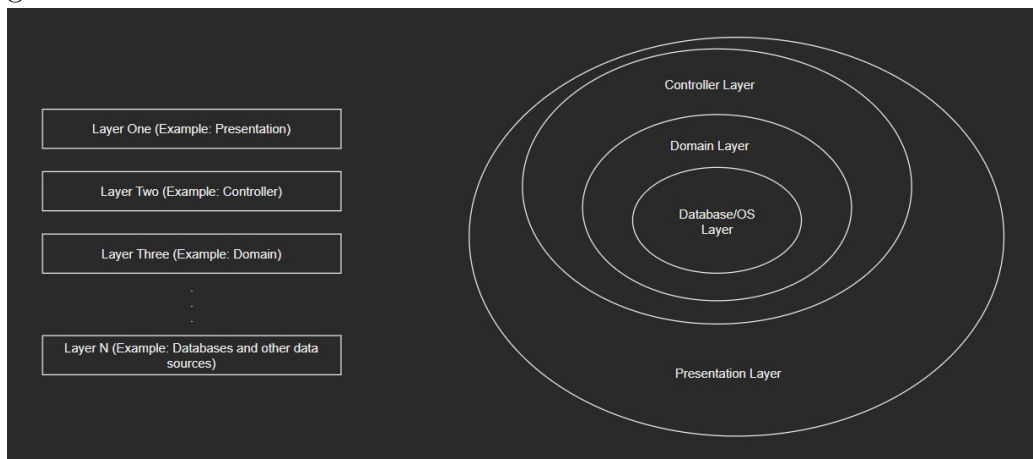
# Bab 5

## Layered Architecture

AUSTIN NICHOLAS THAM, DARREN VALENTIO, MUHAMMAD

### 5.1 Definisi *Layered Architecture*

Pola arsitektur layered adalah pola n-tiered di mana komponen disusun dalam lapisan horizontal. Ini adalah metode tradisional untuk merancang sebagian besar perangkat lunak dan dimaksudkan untuk pengembangan mandiri sehingga semua komponen saling berhubungan tetapi tidak saling bergantung.



Seperti yang ditunjukkan pada gambar, layering biasanya dilakukan dengan mengemas fungsionalitas khusus aplikasi di lapisan atas, penyebaran fungsionalitas spesifik menjadi lapisan bawah dan fungsionalitas yang membentang di seluruh domain aplikasi di lapisan tengah. Jumlah lapisan dan bagaimana lapisan-lapisan ini disusun ditentukan oleh kompleksitas masalah dan solusinya.

Di sebagian besar arsitektur berlapis, ada beberapa lapisan (atas ke bawah):

- **The application layered:** Berisi layanan spesifik aplikasi.
- **The business layer:** Menangkap komponen yang umum di beberapa aplikasi.
- **The middleware layer:** Lapisan ini mengemas beberapa fungsi seperti pembangun GUI, antarmuka ke basis data, laporan, dan dll.
- **The database/System Software Layer:** Berisi OS, database, dan antarmuka ke komponen perangkat keras tertentu.

## 5.2 Latar Belakang

Penilaian untuk setiap karakteristik berdasarkan kecenderungan alami untuk implementasi tipikal pola layered.

- Kemampuan untuk merespon dengan cepat terhadap lingkungan yang terus berubah. (monolitik)
- Bergantung pada implementasi pola, penyebaran bisa menjadi masalah. Satu perubahan kecil ke komponen dapat memerlukan redeployment seluruh aplikasi.
- Pengembang dapat memberikan pengujian singkat untuk menguji aplikasi sebelum klien menggunakannya
- Mudah dikembangkan karena polanya sudah terkenal dan tidak terlalu rumit untuk melakukan implementasinya.

## 5.3 Pros Cons

### 5.3.1 Pros

- Mudah untuk diuji karena komponen-komponennya termasuk lapisan khusus sehingga dapat diuji secara terpisah.
- Sederhana dan mudah diimplementasikan karena secara alami, sebagian besar aplikasi bekerja berlapis-lapis

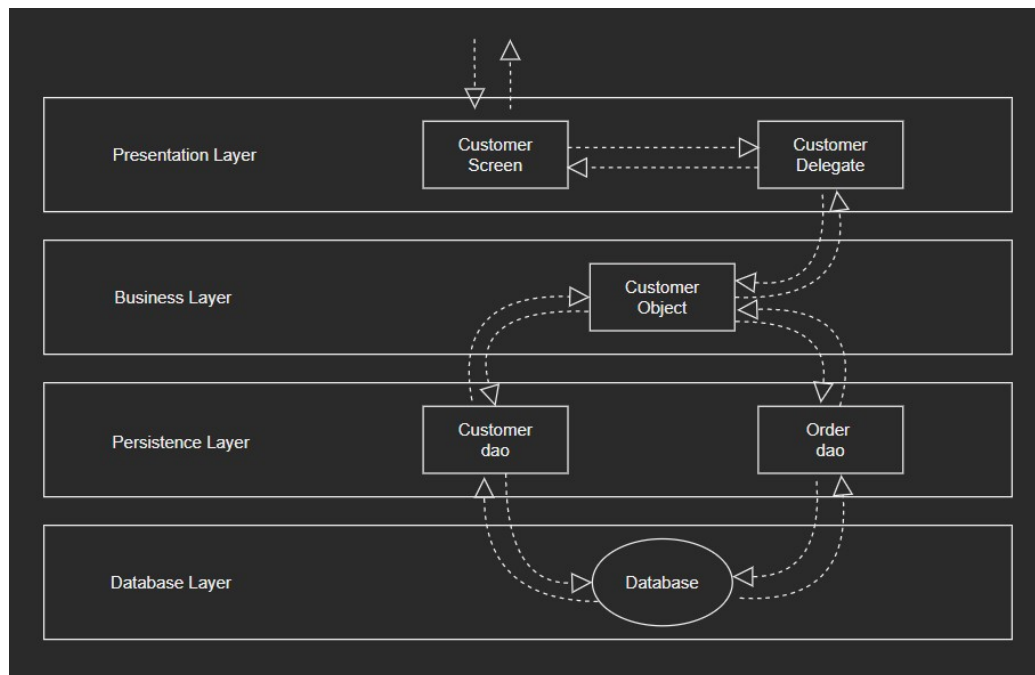


### 5.3.2 Cons

- Tidak mudah untuk melakukan perubahan pada lapisan tertentu karena aplikasi merupakan unit tunggal.
- Kopling antar lapisan cenderung membuatnya lebih sulit. Hal ini membuatnya sulit untuk diukur.
- Harus digunakan sebagai unit tunggal sehingga perubahan ke lapisan tertentu berarti seluruh sistem harus dipekerjakan kembali.
- Semakin besar, semakin banyak sumber daya yang dibutuhkan untuk permintaan untuk melewati beberapa lapisan dan dengan demikian akan menyebabkan masalah kinerja.

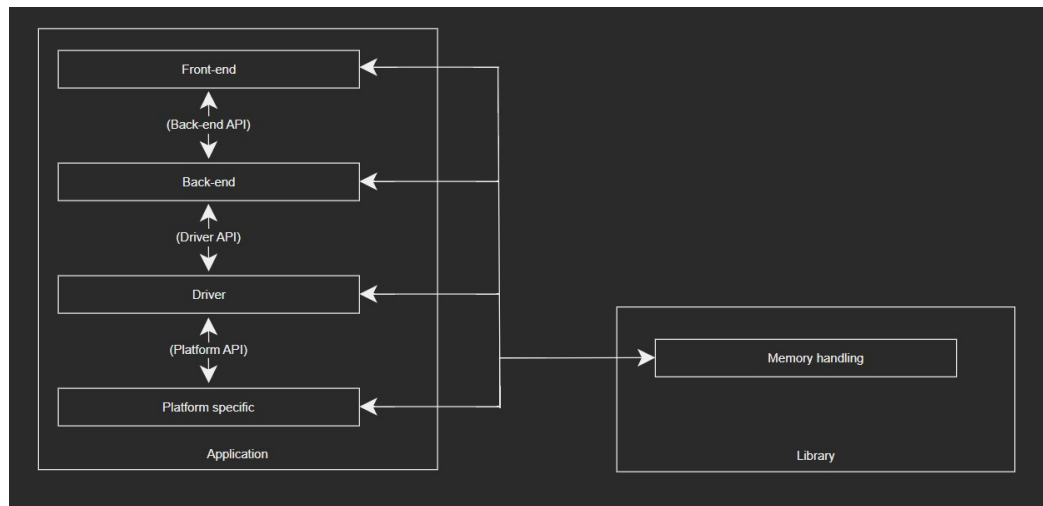
## 5.4 Software Architechture Pattern

Ini adalah pola arsitektur paling umum di sebagian besar aplikasi tingkat perusahaan. Ini juga dikenal sebagai pola n-tier, dengan asumsi n jumlah tingkatan. Contoh Skenario:



## 5.5 Design Patterns

Anggap mock-up software design, susunan “stack” nya seperti layered architecture:

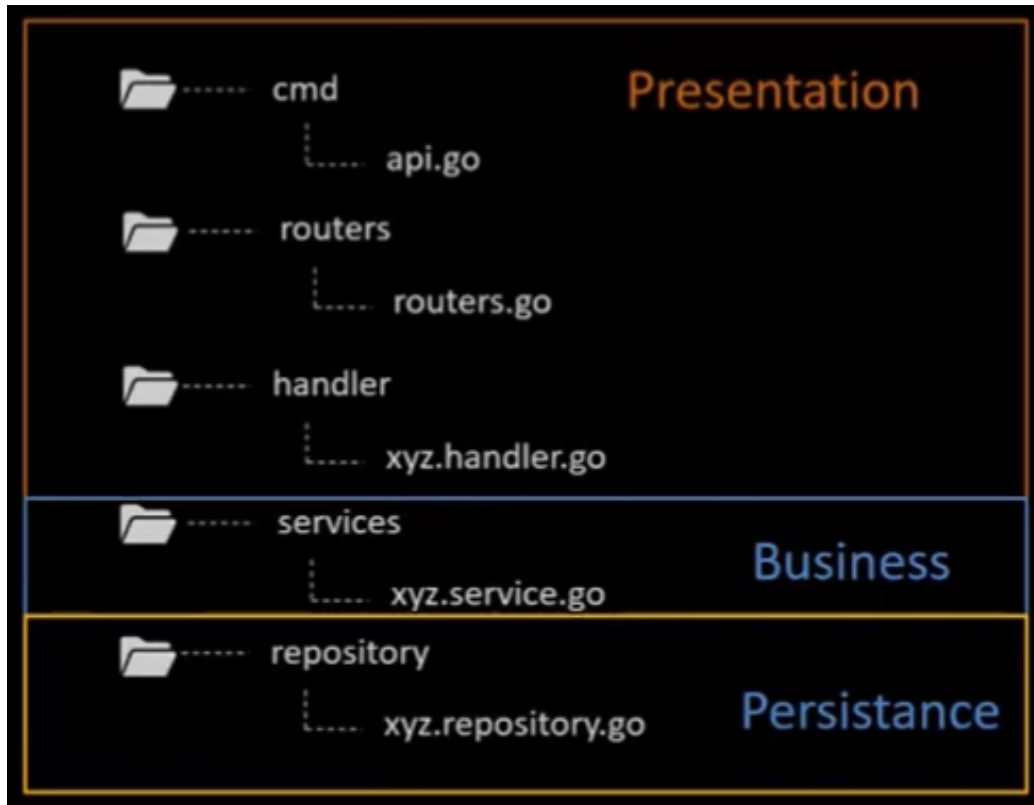


Setiap layer dari aplikasi terpisah dengan cara penggunaan metode API, namun yang masih saling berhubungan adalah memory handling , karena setiap komunikasi layer akan membawa/mengirim data sehingga akan terjadi alokasi memory dan pada akhirnya membutuhkan memory handling.

Ada 4 bagian dari layered architecture yang di mana setiap layer memiliki hubungan antara komponen yang ada di dalamnya dari atas ke bawah yaitu:

- **The presentation layer:** Semua bagian yang berhubungan dengan layer presentasi.
- **The business layer:** Berhubungan dengan logika bisnis.
- **The persistence layer:** Berguna untuk mengurus semua fungsi yang berhubungan dengan objek relasional
- **The database layer:** Tempat penyimpanan semua data layer.

## 5.5.1 Contoh penerapan layered architecture:





# Bab 6

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 6.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps



# Bab 7

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 7.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps





# Bab 8

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 8.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps



# Bab 9

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 9.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps



# Bab 10

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 10.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps



# Bab 11

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 11.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps





# Bab 12

## Microservices

INZAGHI POSUMA AL KAHFI, LUCKY RUSANDANA, ALFRED  
GERALD THENDIWIJAYA

### 12.1 Definisi *Microservices*

Microservices adalah desain arsitektur untuk membuat aplikasi dari banyak unit layanan yang terpisah namun masih saling berhubungan. Setiap unit layanan dalam aplikasi menjalankan fungsi yang berbeda, namun tetap saling mendukung.

Dengan kata lain, microservices sama dengan membangun aplikasi dalam aplikasi. Contohnya, penerapan Microservices architecture pada super-app seperti Shopee. Dalam satu aplikasi, Shopee menggunakan beberapa microservice untuk berbagai jenis service-nya seperti Shopee Food, ShopeePay, Shopee Mall, dsb. Atau untuk contoh lainnya seperti Gojek memiliki Microservices seperti GoRide, GoPay, GoFood, GoSend, dsb.

Gambar 12.1: Microservices pada E-Commerce.

### 12.2 Karakteristik *Microservices*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum

lum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

### 12.3 Kelebihan *Microservices*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

### 12.4 Kekurangan *Microservices*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

### 12.5 Contoh dari *Microservices*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

### 12.6 Kesimpulan

# Bab 13

## Pendahuluan

ALFA YOHANNIS, CHARLIE CHAPLIN

### 13.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps



# Bab 14

## DevOps

HENDRA LIJAYA, OKTAVIANUS HENDRY WIJAYA

### 14.1 Pengertian

DevOps merupakan metode pengembangan software dengan mengkolaborasi *software developer* dengan *IT operation*.

### 14.2 Fungsi

Tujuan akhir atau *goal* dari DevOps adalah untuk menciptakan lingkungan kolaborasi yang berkelanjutan untuk membawa software menjadi lebih berkualitas, lebih cepat, dan dapat diandalkan.

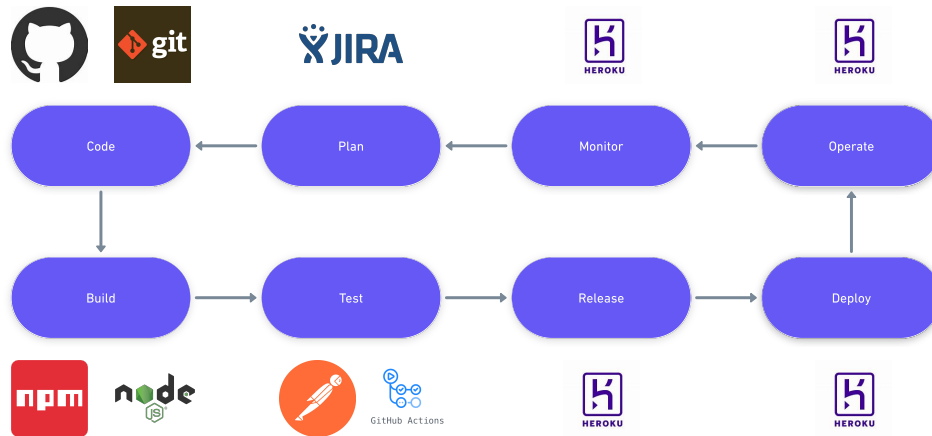
### 14.3 Arsitektur

- *Plan*

Tahap paling awal dalam SDLC (*Software Development Life Cycle*). Mulai dari tahap pengumpulan data, membuat *roadmap*, menetapkan tujuan, *timelines* serta mengidentifikasi *resources* yang diperlukan untuk menyelesaikan sebuah proyek.

- *Code*

Pada tahap ini, developer mulai menulis kode untuk mengembangkan *software* berdasarkan requirement yang telah dikumpulkan pada tahap *Plan*.



Gambar 14.1: Arsitektur DevOps.

- Build**  
 Pada tahap ini, kode di *compile*, dan di *package* kedalam format yang bisa di *deliver*. Tujuan tahap ini membuat kode yang telah di *compile* agar dapat melakukan tahap *Testing* dan *Release*.
- Test**  
 Pada tahap ini, perangkat lunak diuji untuk memastikan bahwa perangkat lunak sudah memenuhi requirement dan fungsinya sudah berjalan tanpa ada *bug*.
- Release**  
 Pada tahap ini, perangkat lunak sebelum di *deploy* ke *staging* atau *production environment* dapat dilakukan data migrasi, konfigurasi dan lainnya. Tujuannya adalah untuk memastikan bahwa perangkat lunak tersedia dan dapat digunakan oleh audiens yang dituju.
- Deploy**  
 Pada tahap ini, perangkat lunak di *deploy* ke *production* ataupun *staging* bisa menggunakan tools atau *automation script*. Proses ini mencakup juga instalasi library, dan konfigurasi server dan lainnya.

- *Operate*  
Pada tahap ini, perangkat lunak sudah beroperasi di *production environment* dan dapat dikelola dan dipantau untuk memastikan kinerjanya seperti yang diharapkan.
- *Monitor*  
Pada tahap ini, pengumpulan dan analisis data dari log sistem. Informasi ini dapat digunakan untuk mengidentifikasi area untuk perbaikan, mengoptimalkan kinerja, dan menginformasikan upaya pengembangan perangkat lunak kedepannya.

## 14.4 Kelebihan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur DevOps:

### 14.4.1 Kelebihan

Keuntungan dari menerapkan arsitektur DevOps adalah:

- DevOps menjadi pilihan yang bagus untuk *development* dan *deployment* aplikasi yang cepat
- Merespon lebih cepat ke perubahan market untuk meningkatkan *business growth* (pertumbuhan bisnis)
- Data dapat disentralisasikan sehingga meningkatkan konsistensi data dan mengurangi duplikasi data.
- DevOps meningkatkan profit bisnis dengan mengurangi waktu *delivery software* dan biaya.
- DevOps menghilangkan proses deskriptif sehingga memberikan kejelasan mengenai *development* dan *delivery product*.
- Meningkatkan *experience* dan kepuasan *customer*.
- DevOps menyederhanakan kolaborasi dan menggunakan semua *tools* di cloud untuk diakses pengguna.
- Meningkatkan keterlibatan dan produktivitas tim.

### 14.4.2 Kekurangan

Kekurangan dari penerapan arsitektur DevOps adalah:

- DevOps *professional* atau *expert* masih belum umum ditemukan.
- *Developing* dengan DevOps mahal.
- Penerapan DevOps baru ke dalam industri sulit untuk dikelola dalam waktu singkat.
- Kurangnya pengetahuan mengenai DevOps dapat menyebabkan masalah pada *Continuous Integration* dari *project automation*.

## 14.5 Perbedaan DevOps dan nonDevOps

Perbedaan besar dari *development* dengan DevOps dan nonDevOps yaitu:

1. Kolaborasi

Pada *Software development* nonDevOps, developer dan tim operation bekerja secara terpisah. Sedangkan pada DevOps, kedua hal tersebut bekerja secara kolaboratif dalam 2 tim yang berbeda dengan berbagi pengetahuan dan skill sehingga memastikan proses software development dapat disederhanakan.

2. *Continuous Integration and Delivery (CI/CD)*

DevOps menerapkan CI/CD yang dimana melibatkan sistem *automation* pada proses *software development* mulai dari *building* dan *testing* hingga ke *deployment* dan *maintenance*. Dengan menerapkan ini, perubahan dapat di tes dan diintegrasikan ke software secara cepat dan efisien mungkin.

3. *Automation*

DevOps bergantung secara penuh pada automation untuk meningkatkan efisiensi dan mengurangi error. Tools seperti *management configuration*, *continuous integration*, dan *continuous delivery* memungkinkan tim untuk mengautomatis proses yang awalnya manual dan memastikan konsistensi.

4. *Monitoring*

Tim yang menerapkan DevOps menggunakan *tools* untuk *monitoring* dan analitik untuk mengumpulkan data mengenai performa pada software saat *production*. Hal ini membantu tim dalam mengidentifikasi



dan menyelesaikan isu dengan cepat sehingga mengurangi downtime dan meningkatkan *user experience* secara keseluruhan.

#### 5. Agile Development

DevOps berdasarkan pada prinsip *agile development* yang dimana fleksibilitas, adaptabilitas, dan kolaborasi sangat ditekankan. Tim DevOps memprioritaskan dalam *delivery* dalam perubahan kecil dan perubahan *incremental* dengan cepat dibandingkan perilisian monolitik yang bersifat besar.

Kesimpulannya adalah DevOps bersifat lebih kolaboratif, *automated*, dan *agile* pada proses *software development* yang menekankan *continuous integration and delivery*, *automation*, dan *monitoring*.

## 14.6 Tools

Tools yang digunakan dalam pembuatan DevOps:

#### 1. Git - GitHub Action

GitHub Action adalah fitur dari *platform* GitHub yang memungkinkan developer untuk mengautomasi *workflows* dan *build*, *test*, dan *deploy* kode langsung dari *platform* GitHub. GitHub Action menyediakan *library* dari *pre-built actions* yang dapat digunakan untuk membangun *workflows* dan juga kemampuan untuk membuat action kustom menggunakan JavaScripts atau Docker containers. *Workflows* dapat dipicu/ditriggery oleh *events* seperti push kode, request pull, atau pembuatan perilisian baru.

Keuntungan menggunakan GitHub:

- Terintegrasi dengan GitHub
- Workflows yang dapat dikustomisasi
- Reusability
- Kolaborasi
- Skalabilitas
- Gratis

Kesimpulan, GitHub Action merupakan *tools* yang sangat berguna untuk *automating software development workflows*, menyediakan developer fleksibilitas, kustomisasi, dan platform yang terintegrasi untuk *building*, *testing*, dan *deploy* kode

## 2. Heroku

Heroku merupakan *platform cloud* yang memungkinkan developer untuk *build*, *deploy*, dan mengelola aplikasi secara cepat dan mudah. Heroku mendukung beberapa Bahasa pemrograman seperti Java, Ruby, Node.js, Python, PHP, dan Go. Heroku menyediakan platform yang dikelola secara penuh sehingga developer tidak perlu mengkhawatirkan mengenai mengelola infrastruktur, sistem operasi, dan server.

Heroku didasarkan pada arsitektur yang berbasis *container* dan menggunakan Dyno untuk menjalankan aplikasi. Dyno merupakan *container* linux yang ringan dan terisolasi yang berjalan diatas platform Heroku. Dyno didesign untuk menjalankan satu proses atau layanan yang membantu meningkatkan performa, skalabilitas, dan ketahanan.

Fitur-fitur Heroku:

- *Command Line Interface (CLI)*
- *Web Based Dashboard*
- Beragam *add-ons* dan *extensions*
- *Support continuous integration and continuous delivery (CI/CD) workflows.*

Adapun kekurangan dari Heroku yaitu:

- Kustomisasi yang terbatas.
- Bergantung pada *add-on third party*.
- Memerlukan biaya dan kartu kredit.

## 3. Postman

Postman merupakan tools software yang sering digunakan oleh developer untuk *test*, dokumentasi, dan berbagi API. API atau *Application Programming Interfaces* memungkinkan software aplikasi yang berbeda untuk berkomunikasi melalui pertukaran data antara satu dengan yang lain.

Dengan menggunakan Postman, developer dapat dengan mudah membuat dan mengeksekusi *HTTP requests*, yang memungkinkan mereka untuk mencoba API dan memastikan API berjalan dengan benar. Postman juga menyediakan berbagai fitur yang memudahkan pendokumentasian API, termasuk kemampuan untuk membuat dokumentasi API dan membuat *code snippets* dalam berbagai variasi bahasa pemrograman.

Fitur utama pada Postman:

- *Collections*: Postman memungkinkan developers untuk mengorganisasikan *request* ke sebuah *collections*, yang dimana memudahkan dalam *grouping request* berdasarkan fungsionalitas.
- *Environments*: Postman mendukung penggunaan *environments* yang memungkinkan developer untuk pendefinisian variabel dan *values* yang berbeda untuk *testing environments* yang berbeda (seperti *development*, *testing*, atau *production*).
- *Test automation*: Postman memungkinkan developer untuk mengotomatisasi *testing* dengan membuat *scripts* yang dapat dijalankan sebagai bagian dari *test*. Hal ini memudahkan dalam memastikan API berjalan secara benar dan memudahkan mencari isu diawal dalam proses *development*.
- *Collaboration*: Postman memudahkan dalam berkolaborasi dengan developer lain dengan memungkinkan berbagai *collection*, *environment*, dan *documentation* dengan yang lain.
- *Integrations*: Postman terintegrasi dengan bermacam-macam tools dan layanan lain, seperti GitHub, Jira, Slack, yang memudahkan dalam memasukkan Postman kedalam *workflows* yang sudah ada.

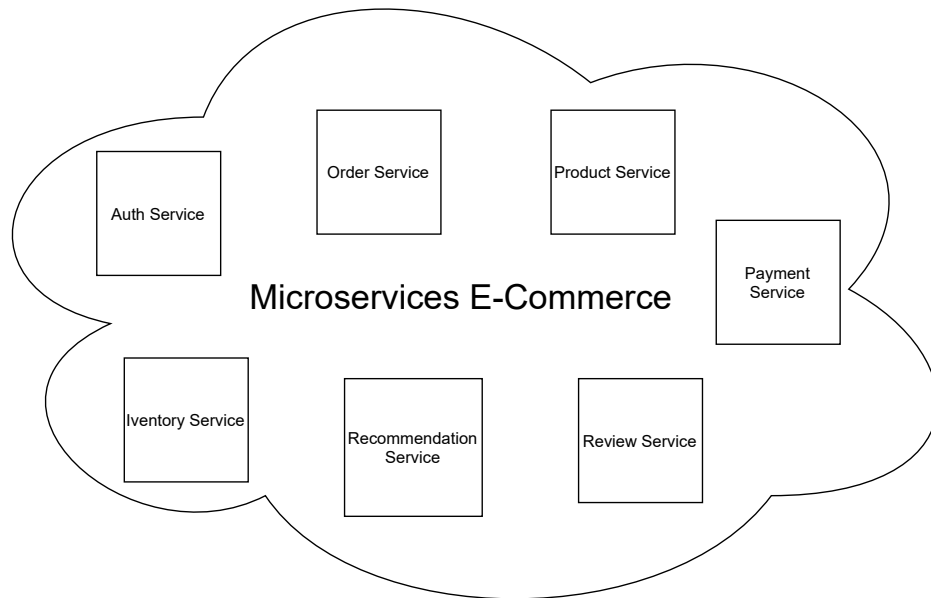
## 14.7 Contoh Kasus

Contoh Kasus Aplikasi *E-Commerce* Biasanya pada aplikasi *e-commerce* yang menggunakan arsitektur *microservice*. Beberapa *service* pada *microservice* yaitu *Authentication Service*, *product catalog service*, *order management service*, *payment service*, and *shipping service*, yang masing-masing bertanggung jawab untuk fungsi tertentu. Dengan menggunakan DevOps dapat memberikan beberapa kemudahan pada developer dengan beberapa fitur.

Fitur utama DevOps:

*CI/CD Pipeline* digunakan untuk melakukan *compile code*, *unit testing*, *integration testing*, *packaging*, *deployment* dan *monitoring*. Setiap kali developer melakukan *push* pada *repository git*, *CI/CD Pipeline* otomatis melakukan *compile code*, menjalankan *unit tests*, *deploy* perubahan pada *staging environment* untuk *integration testing*. Jika *integration test* dilewati, maka akan di *deploy* di *production environment*.

Menggunakan DevOps dapat membuat developer fokus pada fitur aplikasi yang ingin dibangun sehingga tidak terlalu lama memikirkan setup infrastruktur untuk *testing* dan lainnya.



Gambar 14.2: Studi Kasus *E-Commerce Microservice*.

## 14.8 Code

Code dapat dilihat di link berikut

# Bibliography