Arsitektur Perangkat Lunak

Departemen Informatika, Universitas Pradita, Indonesia

2023

Daftar Isi

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

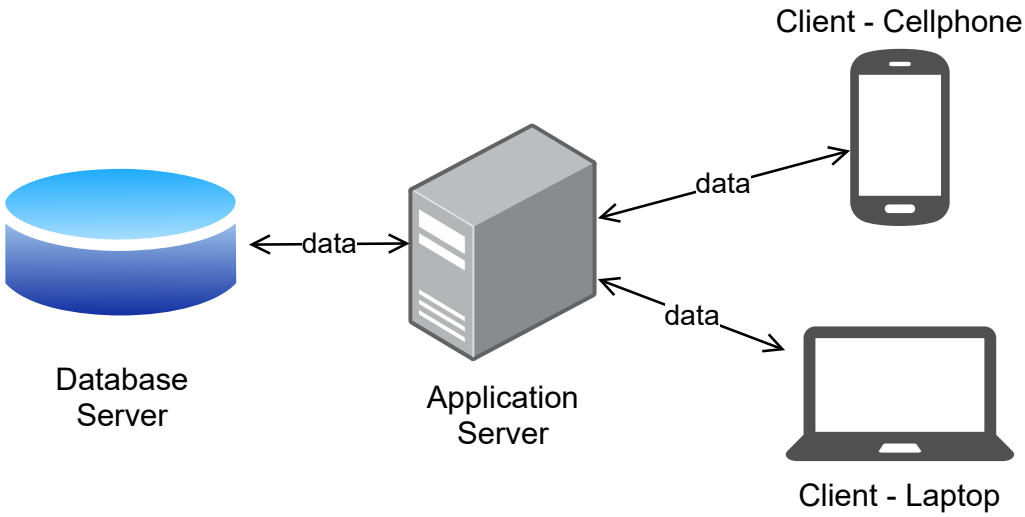
Pada awal komputer bermula sebagai suatu kesatuan, tidak terpisah-pisah. Perangkat lunak hanya berjalan pada satu unit komputer tersebut. Secara perlahan, ada bagian komputer yang dapat terpisah secara fisik dan menjalankan tanggung jawab tertentu. Sebagai contoh, data storage terpisah dari komputer 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 sesautu 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 2.1.



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 spefikasi 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 etwork 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

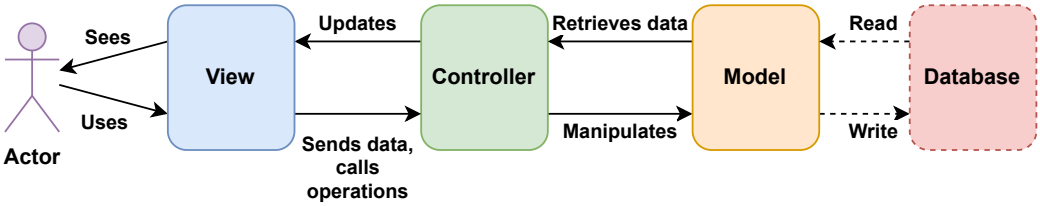
Arsitektur MVC adalah pola arsitektur untuk pengembangan *Graphical User Interface* (GUI). Arsitektur tersebut membagi logika progam menjadi 3 bagian yang saling terhubung: Model, View, dan Controller. Skema dari MVC dapat dilihat pada Gambar 3.1..

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 pegguna 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, sendangkan *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

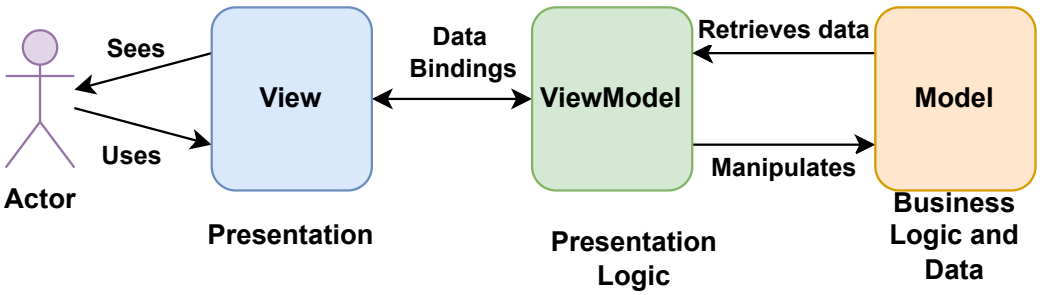
Arsitektur MVVM (Model-View-ViewModel)

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 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 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 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

Pendahuluan

Alfa Yohannis, Charlie Chaplin

5.1 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 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

Pendahuluan

Alfa Yohannis, Charlie Chaplin

12.1 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 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

Pendahuluan

Hendra Lijaya, Oktavianus Hendry Wijaya

14.1 Pengertian

DevOps merupakan metode pengembangan software dengan mengkolaborasikan *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 Kelebihan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur DevOps:

14.3.1 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.

14.3.2 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*.

14.4 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 pengentahuan 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 perilisan 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*.

14.5 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/di*trigger* oleh *events* seperti push kode, request pull, atau pembuatan perilisan 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 fleksibilitasm, 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 di*design* 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.

14.6 Contoh Kasus

14.7 Code

Bibliography