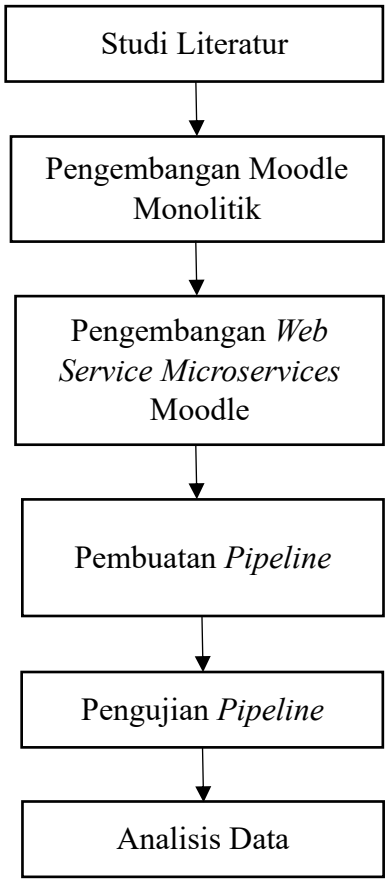
# BAB III
# METODE PENELITIAN

## 3.1 Alur Penelitian

Alur penelitian ini dibuat dengan struktur yang sistematis untuk mempermudah pemahaman tahapan penelitian. Ilustrasi alur penelitian dapat dilihat pada Gambar 3.1.




Gambar 3.1 Alur Penelitian

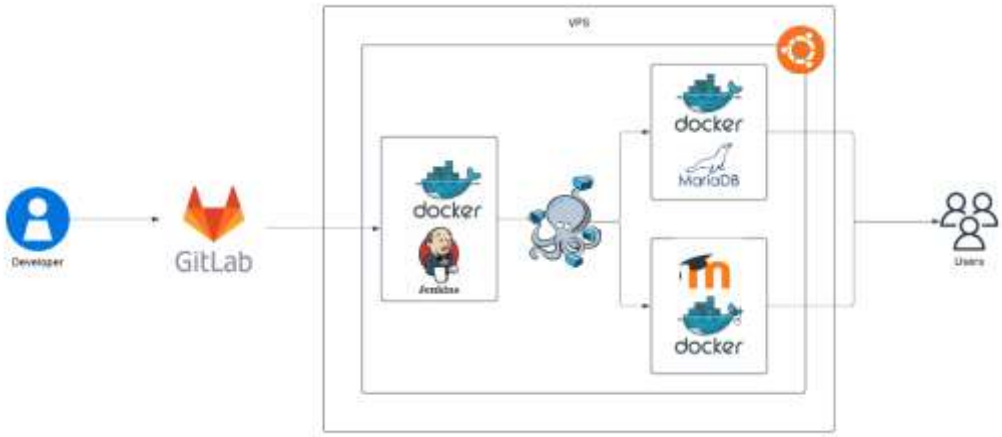
### 3.1.1 Studi Literatur

Studi literatur bertujuan untuk mengkaji berbagai sumber dan referensi terkait dengan proses *deployment* Moodle. Melalui kajian ini, kebutuhan spesifik serta berbagai masalah yang muncul dalam proses implementasi *deployment* Moodle dapat diidentifikasi secara komprehensif. Hasil kajian tersebut menjadi dasar fundamental dalam merumuskan tujuan penelitian secara tepat sehingga

penelitian ini dapat memberikan kontribusi dalam pengembangan *deployment* Moodle.

### 3.1.2 Pengembangan Moodle Monolitik

Platform Moodle dikembangkan dan dijalankan menggunakan teknologi *containerization* Docker serta Docker Compose, dengan mengunduh *image* resmi dari Docker Hub. Secara *default* Moodle berjalan dengan arsitektur monolitik. Dalam arsitektur ini, seluruh komponan sistem Moodle terintegrasi dalam satu kesatuan, dengan proses CI/CD dijalankan secara terpadu dalam satu alur *pipeline*.




Gambar 3.2 Pengembangan Moodle Monolitik

Gambar 3.2 memperlihatkan pengembangan Moodle dengan arsitektur monolitik, yang menggambarkan tahapan utama mulai dari *push* kode oleh *developer* hingga penerapan sistem yang dapat diakses oleh *users*. Berikut adalah penjelasan dari setiap tahapan alur tersebut.

a. ***Developer***: Melakukan *push* kode ke GitLab setelah menyelesaikan perubahan atau penambahan fitur baru pada aplikasi.
b. **GitLab**: Berfungsi sebagai repositori kode sumber untuk menyimpan dan mengelola aplikasi secara terpusat.
c. **Jenkins di Docker:** Jenkins mengambil kode dari GitLab dan menjalankan *pipeline* sesuai kode yang ada di *Jenkinsfile*.

d. **Docker Compose:** Digunakan untuk mengatur dan menjalankan *multi-container deployment* secara bersamaan. *Container* yang dijalankan di antaranya adalah:
   - Aplikasi utama (Moodle)
   - Database (MariaDB)

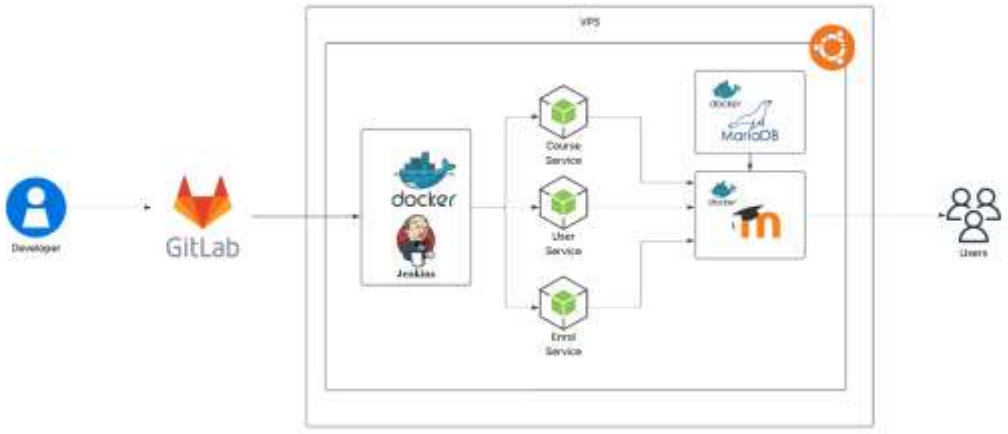
e. **Moodle di Docker:** Moodle adalah platform LMS (*Learning Management System*) yang di *deploy* sebagai *container* dan Aplikasi ini diakses oleh pengguna akhir.

f. **MariaDB di Docker:** Berjalan sebagai *container* dan terhubung ke *container* Moodle via Docker *network*.

g. ***Users*:** Mengakses Moodle yang telah di *deploy* melalui *browser*.

### 3.1.3 Pengembangan *Web Service Microservices* Moodle

Beberapa komponen aplikasi Moodle monolitik dimodifikasi menjadi *web service* yang dapat dijalankan terpisah dalam arsitektur *microservices*. Pada arsitektur ini setiap layanan yang terintegrasi dengan Moodle dipisahkan dan dijalankan secara independen sehingga setiap layanan memiliki alur *pipeline* sendiri.




Gambar 3.3 Pengembangan *Web Service Microservices* Moodle

Berdasarkan pengembangan *Web Service Microservices* Moodle pada Gambar 3.3, berikut penjelasan mengenai setiap tahapan yang terlibat dalam arsitektur tersebut.

a. ***Developer*:** Menulis atau memperbarui kode untuk salah satu *service* (*Course, User, Enrol*) dan Melakukan *push* kode ke repositori GitLab.
b. **GitLab:** Menyimpan *source code* dari berbagai komponen *microservices*.
c. **Jenkins di Docker:** Jenkins mengambil kode dari GitLab dan menjalankan *pipeline* sesuai dengan *service* yang dijalankan.
d. ***Microservices (Course, User, Enrol*):** Tiga *service* terpisah masing-masing menangani bagian spesifik:
   - ***Course Service*:** Membuat kategori dan *course* baru.
   - ***User Service*:** Membuat pengguna baru.
   - ***Enrol Service*:** Manual *enrol* kursus.

   Setiap *service* ini dibuat dengan Node.js dan berkomunikasi dengan moodle melalui REST API.

e. **Moodle di Docker:** Moodle tetap digunakan sebagai *frontend* utama untuk pengguna dan Terhubung dengan *microservices* tersebut untuk memproses data.
f. **MariaDB di Docker:** Digunakan sebagai database utama yang mendukung Moodle dan *service* lainnya dan Diakses dari *container* Moodle (*service* tertentu jika dibutuhkan).
g. ***Users*:** Mengakses platform Moodle melalui *browser* dan mengambil data dari layanan-layanan *microservice* untuk ditampilkan ke pengguna.

### 3.1.4 Pembuatan *Pipeline*

Jenkins *pipeline* digunakan sebagai alat CI/CD dalam sistem ini, *pipeline* dirancang untuk menangani alur *deployment* secara otomatis dari pengambilan kode dari Gitlab dan memastikan aplikasi berjalan dengan lancar di *production*. Berikut penjelasan *stage pipeline* pada Moodle di kedua arsitektur:

#### 3.1.4.1 Monolitik

Pada arsitektur monolitik, *Jenkinsfile* digunakan untuk mengotomatisasikan seluruh proses *deployment* secara keseluruhan dalam satu *pipeline*. *Pipeline* ini mengelola semua komponen Moodle secara bersamaan dengan path dan port tunggal untuk aplikasi monolitik.


Gambar 3.4 Tahapan *pipeline* Moodle monolitik

Pada Gambar 3.4 ditampilkan tahapan *pipeline* Moodle monolitik yang akan digunakan dalam pengujian. Berikut penjelasan mengenai tahapan *pipeline* yang dijalankan sesuai dengan skenario penelitian.

***a. Deploy***

Jenkins akan otomatis menarik (*pull*) kode terbaru dari Gitlab setelah terjadi *push*. Kode sumber tersebut kemudian akan digunakan untuk membangun aplikasi dalam bentuk *docker container* melalui *docker compose*. Proses ini akan melakukan penghentian (*down*) pada *container* yang sedang berjalan di lingkungan *production*, sehingga akan menyebabkan *downtime*, lalu dilanjutkan dengan menjalankan *up container* baru dari hasil *push* terbaru.

***b. Health Check***

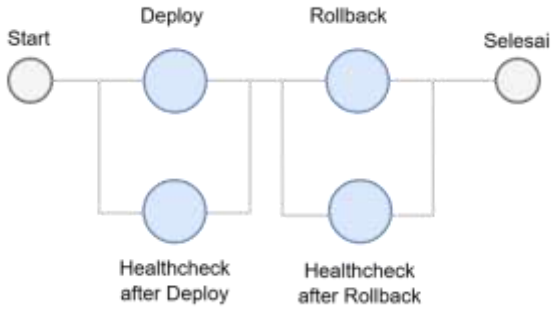
Setelah *container* berhasil di *up*, *pipeline* akan melakukan pengecekan *health check* pada *container* di *production*, apabila aplikasi memberikan respons dengan status kode 200, maka proses *deployment* akan dianggal berhasil dan *pipeline* akan melakukan proses selesai tanpa eksekusi *stage* lain. Namun, jika aplikasi tidak memberikan respons dalam batas waktu yang ditentukan, *pipeline* akan mengakses *container logs* untuk mencari kesalahan. Apabila ditemukan pesan error, proses akan dilanjutkan ke tahap *rollback*.

***c. Rollback***

*Rollback* dilakukan dengan menghentikan (*down*) *container* yang gagal di *deploy*, kemudian menggantikannya dengan *container* versi stabil sebelumnya yang telah tersimpan. Setelah proses penggantian, sistem secara otomatis menjalankan *health check* untuk memastikan bahwa aplikasi dapat diakses kembali. Jika aplikasi merespons dengan status kode 200, maka *rollback* dianggap berhasil. Namun, apabila aplikasi masih gagal diakses atau tidak memberikan respons yang sesuai, maka proses *rollback* dinyatakan gagal dan memerlukan intervensi manual untuk pemulihan lebih lanjut.

### 3.1.4.2 *Microservices*

Pada arsitektur *microservices*, *Jenkinsfile* digunakan untuk otomatisasi secara independen pada masing-masing layanan yaitu *user service, course service* dan *enrol service*. *Pipeline* ini dirancang dengan alur yang seragam, namun masing-masing *service* memiliki path dan port yang berbeda.




Gambar 3.5 Tahapan *pipeline* Moodle *microservices*

Pada Gambar 3.5 ditampilkan tahapan *pipeline* Moodle berbasis *microservices* yang akan digunakan dalam pengujian *pipeline*. Berikut penjelasan mengenai tahapan *pipeline* yang dijalankan sesuai dengan skenario penelitian.

***a. Deploy***

Pada tahap ini, *pipeline* akan menghentikan *service* yang sedang berjalan di *production* dengan cara melakukan *kill* terhadap port yang digunakan oleh *service* tersebut yang akan menyebabkan *downtime*. Kemudian kode terbaru akan di *deploy* ke *production*. Karena *service* di bangun dengan menggunakan Node.js dan Swagger UI maka, *deploy* dilakukan dengan proses `node server.js`. *Pipeline* langsung masuk ke tahap *health check* yang dijalankan secara paralel.

***b. Health Check***

*Pipeline* akan memeriksa apakah *service* yang baru di *deploy* dapat diakses dengan status kode 200, jika status ini terpenuhi maka *deployment* akan dianggap berhasil. Namun, jika tidak ada respons atau status error lainnya, maka

*pipeline* akan melakukan proses ke tahap *rollback* untuk memulihkan kondisi sebelumnya.

***c. Rollback***

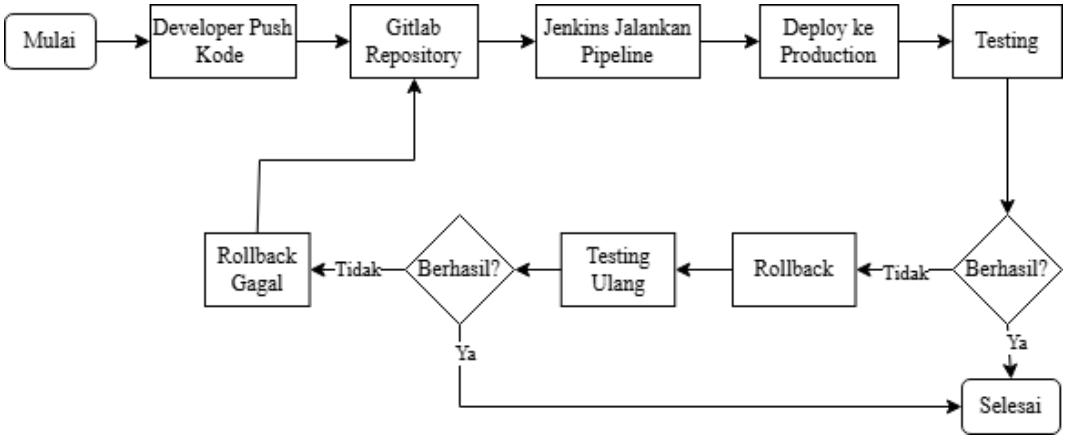
Jika *health check* gagal, maka *pipeline* akan melakukan *rollback* dengan menghentikan *service* yang bermasalah dengan *kill* port, lalu menjalankan ulang kode versi stabil yang sebelumnya dengan `node server.js`. Setelah itu dilakukan *health check* kembali, jika status kode 200 diterima maka *rollback* akan dinyatakan berhasil, namun jika tidak ada respon setelah di *rollback* maka *rollback* dianggap gagal dan membutuhkan intervensi manual.

### 3.1.5 Pengujian *Pipeline*

Pengujian *pipeline* merupakan tahapan penting dalam memastikan bahwa proses integrasi dan *deployment* Moodle sesuai dengan skenario penelitian. Pada tahap ini akan dijelaskan langkah-langkah pengujian *pipeline* yang diterapkan untuk menguji keandalan *pipeline* dalam *deployment* Moodle.

#### 3.1.5.1 Alur Pengujian

Alur pengujian ini akan mencakup serangkaian langkah terstruktur untuk melakukan penelitian dan memperoleh data pengukuran yang relevan, mulai dari proses *push* kode oleh *developer* hingga melakukan *rollback* apabila terjadi kegagalan dalam *deployment*.




Gambar 3.6 Alur Pengujian

Gambar 3.6 menunjukan alur pengujian *pipeline* yang digunakan dalam penelitian ini. Berikut adalah penjelasan mengenai setiap tahapan dalam proses pengujian tersebut.

a. ***Developer Push* Kode:** *Developer* melakukan *push* kode terbaru ke repositori GitLab.
b. **GitLab Repositori:** Kode sumber tersimpan di repositori GitLab.
c. **Jenkins Jalankan *Pipeline*:** Jenkins mengambil perubahan dari GitLab dan menjalankan *pipeline*.
d. ***Deploy* ke *Production*:** *Pipeline* akan otomatis di *deploy* aplikasi ke lingkungan *production*.
e. ***Testing*:** Aplikasi yang sudah di *deploy* akan diuji dengan curl.
f. **Berhasil?:** Jika *testing* berhasil, lanjut ke langkah Selesai. Jika gagal, lanjut ke proses *rollback*.
g. ***Rollback*:** Sistem akan mengembalikan aplikasi ke versi stabil sebelumnya.
h. ***Testing* ulang:** Setelah *rollback*, sistem diuji kembali apakah aplikasi yang di *rollback* bisa berjalan normal.
i. **Berhasil:** Jika *rollback* berhasil, alur berakhir di Selesai. Jika *rollback* gagal, maka proses akan kembali ke Github repositori untuk melakukan perubahan kode.

#### 3.1.5.2 Spesifikasi Perangkat

Penelitian ini memerlukan perangkat keras dan perangkat lunak yang memadai untuk mendukung seluruh tahapan implementasi serta pengujian sistem secara optimal. Oleh karena itu, spesifikasi lengkap perangkat keras dan perangkat lunak yang digunakan akan disajikan secara sistematis dan terperinci berikut ini guna memberikan gambaran yang jelas dan komprehensif.

1) Spesifikasi Perangkat Keras:
   - Laptop: Lenovo V14
   - Processor: AMD Ryzen 3
   - RAM: 8 GB
   - Penyimpanan: SSD 256 GB + HDD 500 GB

2) Spesifikasi Perangkat Lunak dan Tools:
   - Sistem Operasi & Lingkungan Pengembangan: Windows 11 + WSL 2, Ubuntu 24.04
   - *Containerization* & *Orchestration*: Docker 27.5.1, Docker Compose 1.29.2
   - CI/CD Tool: Jenkins BlueOcean (2.492.3-1)
   - Aplikasi Moodle: Bitnami Moodle (4.5)
   - Database: Bitnami MariaDB (11.7.2)
   - Bahasa Pemrograman: Node.js (v18.19.1)
   - API *Documentation*: Swagger UI
   - Version Control & Editor: Git 2.43.0, Visual Studio Code
   - VPS (*Hosting Server*):
     - 4 Core CPU
     - 4 GB RAM
     - 100 GB NVMe Storage
     - Up to 10 Gbps Network Speed

### 3.1.5.3 Skenario Kesalahan Dependensi pada Arsitektur Monolitik Moodle

Moodle dirancang agar berjalan stabil di lingkungan production dengan menggunakan image bitnami/mariadb:11.7.2 (latest) dan bitnami/moodle:4.5, yang dipilih karena telah mendukung kestabilan dan kompatibilitas antar layanan. Berikut tampilan konfigurasi pada file docker-compose.yml yang digunakan.

**docker-compose.yml**

```
services:
  mariadb:
    image: docker.io/bitnami/mariadb:latest
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=*****
      - MARIADB_DATABASE=*****
```

Karena akan dilakukan test kegagalan *deployment* karena kesalahan dependensi, maka sistem akan dibuat salah dengan mengubah versi *image MariaDB* menjadi bitnami/mariadb:10.5, maka dari itu jenis kesalahan dependensinya adalah ketidaksesuaian versi antar layanan. Berikut docker-composes.yml dengan kesalahan dependensi.

**docker-compose.yml**

```
services:
  mariadb:
    image: docker.io/bitnami/mariadb:10.5
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=*****
      - MARIADB_DATABASE=*****
```

#### 3.1.5.4 Skenario Kesalahan Dependensi pada Arsitektur *Microservices* Moodle

*Service-service* yang terhubung dengan Moodle dirancang dalam Node.js pada server.js dengan package.json sebagai tempat pengelolaan dependensinya. File server.js kemudian memanggil dependensi yang tercantum dalam package.json untuk digunakan dalam membangan *service* Moodle. Berikut server.js dengan versi yang stabil.

**server.js**

```
import express from 'express';
import axios from 'axios';
import FormData from 'form-data';
import cors from 'cors';
import swaggerJsdoc from 'swagger-jsdoc';
import swaggerUi from 'swagger-ui-express';
```

Selanjutnya, *service-service* yang terhubung ke Moodle akan diuji dengan skenario kegagalan *deployment* karena kesalahan dependensi. Untuk itu server.js akan dimodifikasi dengan menghilangkan kode dependensi express, sehingga dependensi express yang terdaftar di package.json tidak akan ter-*load* dengan benar dalam server.js. Hal ini akan menyebabkan kegagalan dalam proses *deployment*. Berikut kode server.js dengan kesalahan dependensi tersebut.

**Server.js**

```
//import express from 'express';
import axios from 'axios';
import FormData from 'form-data';
import cors from 'cors';
import swaggerJsdoc from 'swagger-jsdoc';
import swaggerUi from 'swagger-ui-express';
```

### 3.1.6 Analisis Data

Pengukuran data dilakukan dengan metode eksperimen, dimana setiap skenario dijalankan sebanyak 20 kali untuk mendapatkan hasil yang lebih representatif. Parameter yang akan diukur dalam penelitian ini meliputi:

a. ***Downtime***

*Downtime* dalam penelitian ini dihitung berdasarkan nilai *Mean Time to Recovery* (MTTR), yaitu rata-rata waktu yang dibutuhkan sistem untuk pulih dari kegagalan dan kembali berfungsi normal. MTTR mencakup seluruh proses pemulihan, termasuk waktu deteksi kegagalan, *rollback* (jika terjadi), hingga sistem kembali aktif dan memberikan status kode 200.

b. **Waktu *Rollback***

Waktu *rollback* Merupakan bagian dari MTTR yang menunjukkan durasi yang dibutuhkan untuk mengembalikan sistem ke kondisi stabil sebelum kegagalan terjadi. Waktu *rollback* dicatat secara terpisah untuk menunjukkan kontribusinya terhadap total waktu pemulihan, meskipun perhitungan rata-rata *downtime* secara keseluruhan tetap mengacu pada MTTR.

c. **Keberhasilan *Deployment***

Penelitian ini dinilai berdasarkan aplikasi yang telah di *deploy* harus dapat dijalankan dan memberikan status kode 200, yang menunjukan bahwa layanan sudah berjalan dan dapat diakses oleh pengguna. Proses *deployment* ini harus selesai tanpa dilakukan *rollback*, dengan kata lain, jika *pipeline* menjalankan *rollback* maka *deployment* akan dianggap gagal.

Setelah melakukan perancangan sistem, skenario pengujian dilaksanakan dengan mengimplementasikan *pipeline* pada Moodle dengan arsitektur monolitik dan *microservices*. Proses pengujian ini mencakup pengukuran dan evaluasi kinerja CI/CD *pipeline*, dengan fokus pada identifikasi dan analisis kegagalan yang terjadi akibat kesalahan dependensi. Pengujian dilakukan sebanyak 20 kali percobaan untuk memperoleh data yang representatif. Data yang terkumpul dari setiap percobaan mencakup *downtime* dan waktu *rollback* pada proses *deployment*, yang keduanya dihitung menggunakan Persamaan 1. Selain itu, keberhasilan *deployment* dihitung sebagai persentase keberhasilan dari total percobaan *deployment* yang dilakukan, dengan persamaan sebagai berikut:

$$Keberhasilan\ deployment\ (\%) = \left(\frac{N_{deploy\ sukses}}{N_{total\ deploy}}\right) \times 100\% \qquad (2)$$

di mana:

- $N_{deploy\ sukses}$ adalah jumlah *deployment* yang berhasil tanpa mengalami kegagalan.
- $N_{total\ deploy}$ adalah total jumlah *deployment* yang dilakukan dalam skenario pengujian.

Nilai ini digunakan sebagai dasar untuk menghitung persentase keberhasilan *deployment* dari seluruh percobaan yang dilakukan. Dengan demikian, hasil tersebut memberikan gambaran yang representatif mengenai tingkat keandalan sistem dalam menyelesaikan proses *deployment* tanpa mengalami kegagalan atau *rollback*.