

1.1 Pengenalan DevOps

1.1.1 Definisi DevOps

DevOps adalah metodologi yang mengintegrasikan *Development* dan *Operations* untuk menciptakan kultur kolaborasi, otomatisasi, dan *continuous improvement* dalam *software delivery lifecycle*. Konsep ini muncul sebagai respons terhadap kebutuhan industri untuk mempercepat *delivery software* dengan tetap menjaga kualitas dan stabilitas sistem. DevOps memecah tanggung jawab antara tim *development* yang fokus pada pembuatan fitur baru dan tim *operations* yang bertanggung jawab atas stabilitas dan maintenance sistem produksi.

Fundamental dari DevOps bukan hanya tentang *tools* dan teknologi, tetapi merupakan *cultural transformation* yang menekankan *communication, collaboration, and shared responsibility* antara semua anggota yang terlibat dalam pengembangan perangkat lunak. Transformasi ini mengharuskan perubahan mindset dari "*blame culture*" menjadi "*collective ownership*", dimana setiap anggota tim bertanggung jawab atas keberhasilan *product* dari *development* hingga *production*. Filosofi ini didukung oleh penerapan seperti *infrastructure as code, continuous integration, automated testing, and monitoring*.

Implementasi DevOps yang berhasil akan menghasilkan peningkatan yang signifikan dalam metrik kunci seperti *deployment frequency, lead time for changes, mean time to recovery, and change failure rate*. Organisasi yang sudah menerapkan penerapan DevOps dapat melakukan deployment puluhan bahkan ratusan kali per-hari dengan tingkat kepercayaan diri yang tinggi, hal ini cukup berbeda dengan pendekatan tradisional yang hanya melakukan deployment bulanan atau setiap tiga bulan dengan risiko yang besar apabila terjadi kesalahan.

1.1.2 Pendekatan Tradisional vs Pendekatan DevOps (CI/CD)

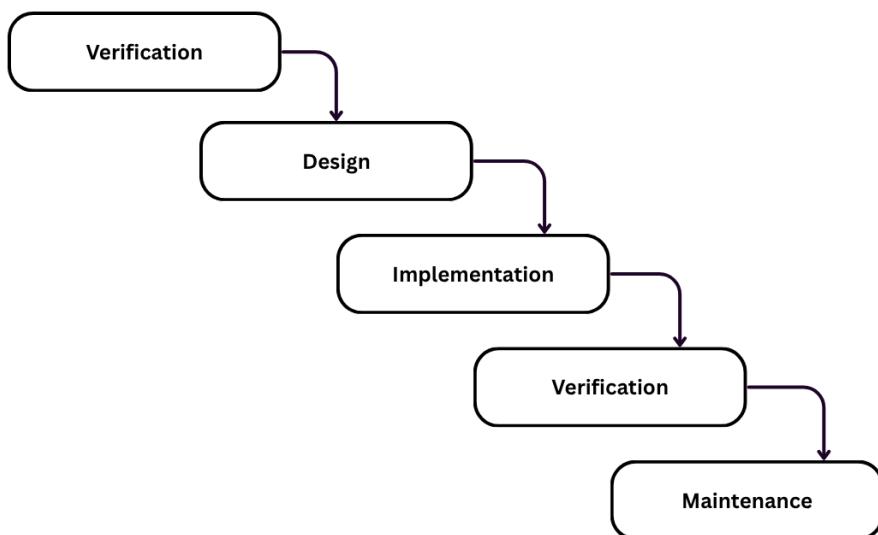
Dalam praktik pengembangan perangkat lunak, terdapat berbagai metodologi yang dapat diterapkan. Salah satu pendekatan yang telah lama dikenal dan digunakan secara tradisional adalah model Waterfall. Pendekatan ini memiliki karakteristik sekuensial, di mana setiap fase pengembangan—mulai dari perencanaan, analisis, desain, implementasi, pengujian, hingga pemeliharaan—dilakukan secara berurutan dan linear. Artinya, fase berikutnya tidak akan dimulai sebelum fase sebelumnya selesai sepenuhnya, dan seringkali sulit untuk kembali ke fase sebelumnya jika ditemukan masalah.

Namun, seiring dengan evolusi kebutuhan bisnis dan teknologi yang semakin cepat, muncul pendekatan baru yang lebih adaptif dan iteratif, salah satunya adalah DevOps. Berbeda dengan model Waterfall yang cenderung kaku, DevOps mengedepankan kolaborasi dan integrasi antara tim pengembangan (Development) dan operasional (Operations). Tujuan utamanya adalah mempercepat siklus pengembangan perangkat lunak dan meningkatkan kualitas produk dengan meminimalkan hambatan antara kedua tim.

Dalam konteks DevOps, tim pengembang perangkat lunak cenderung akan melakukan *deployment* atau perilisan produk secara lebih sering dan dalam siklus yang lebih pendek. Frekuensi *deployment* yang tinggi ini bukan tanpa alasan; tujuannya adalah untuk mendapatkan umpan balik dari pengguna atau pemangku kepentingan secepat mungkin. Dengan umpan balik yang cepat, tim dapat segera mengidentifikasi masalah, memahami kebutuhan pasar yang berubah, dan melakukan perbaikan atau penyesuaian pada produk yang sedang dibangun.

Pendekatan ini memungkinkan tim untuk beradaptasi lebih cepat terhadap perubahan, mengurangi risiko kegagalan besar di akhir siklus pengembangan, dan secara berkelanjutan meningkatkan kualitas serta relevansi produk. *Deployment* yang sering juga didukung oleh otomasi dalam berbagai tahapan, mulai dari *build*, *test*, hingga *release*, yang pada akhirnya mempercepat proses pengiriman nilai kepada pengguna akhir. Dengan demikian, DevOps tidak hanya sekadar metodologi, tetapi juga budaya yang mengubah cara tim bekerja sama demi mencapai tujuan pengembangan perangkat lunak yang lebih efisien dan responsif..

1.1.2.1 Contoh pendekatan tradisional dengan waterfall



Gambar 1.1 Waterfall Methodology

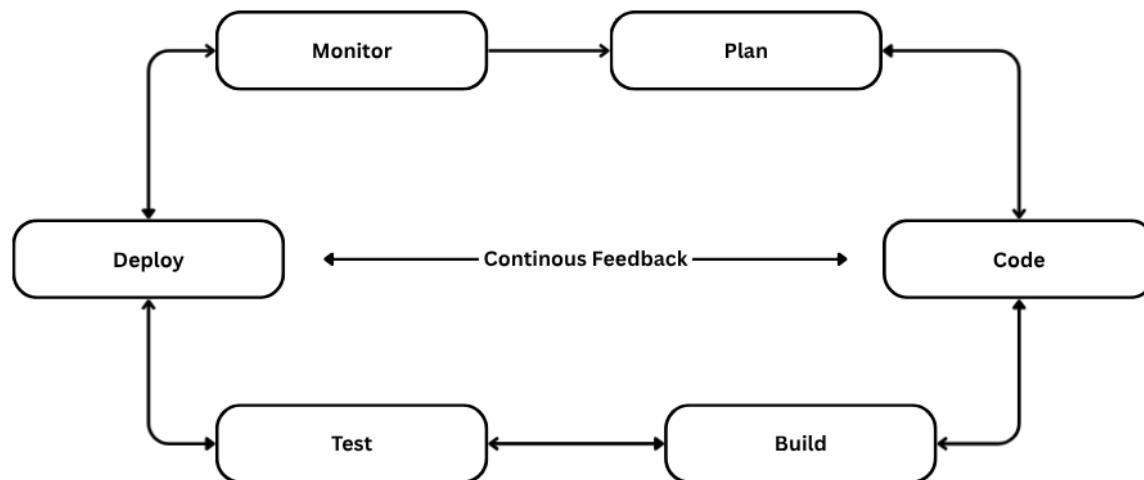
Dapat dilihat berdasarkan diagram di atas, bahwa pendekatan tradisional dengan model waterfall memiliki tahapan yang berjalan secara *sequential* dan *linear*. Proses dimulai dari fase **Verification** (perencanaan dan analisis kebutuhan), dilanjutkan ke fase **Design** (perancangan sistem), kemudian **Implementation** (pengembangan dan coding), diikuti oleh fase **Verification** (testing dan quality assurance), dan terakhir fase **Maintenance** (pemeliharaan sistem).

Setiap fase harus diselesaikan secara lengkap sebelum dapat melanjutkan ke fase berikutnya, dan tidak ada mekanisme untuk kembali ke tahap sebelumnya tanpa mengulang seluruh proses. Hal ini menyebabkan fase planning dan development akan memakan waktu yang sangat lama, bahkan hingga berbulan-bulan. Model waterfall ini cenderung rigid dan kurang fleksibel dalam mengakomodasi perubahan requirements yang mungkin muncul di tengah pelaksanaan project, sehingga diskusi mengenai budget, timeline, dan perubahan sistem hanya dapat dilakukan setelah satu fase sepenuhnya selesai.

1.1.2.2 Contoh pendekatan dengan DevOps Continuous Cycle

Sedangkan menggunakan pendekatan DevOps dengan siklus yang terus berkembang dan iteratif, proses pengembangan hingga tahap deploy akan terus dilakukan seiring waktu dalam cycle yang berkesinambungan. Diagram di atas menunjukkan alur kerja DevOps yang berbentuk circular dan continuous, dimulai dari fase **Plan** (perencanaan), **Code** (development), **Build** (kompilasi dan packaging), **Test** (quality assurance), **Deploy** (release ke environment), hingga **Monitor** (pemantauan performa), yang kemudian kembali lagi ke fase Plan untuk iterasi berikutnya.

Konsep fundamental dari DevOps adalah **Continuous Feedback**, dimana setiap fase saling terhubung dan memberikan *feedback* secara *real-time* untuk peningkatan. Pendekatan DevOps memungkinkan tim untuk mendapatkan umpan balik dengan cepat dari setiap perubahan yang dilakukan, sehingga jika ternyata fitur dari sebuah produk dirasa tidak tepat guna atau tidak sesuai dengan target *audience*, fitur tersebut dapat segera disesuaikan berdasarkan kebutuhan atau bahkan dihentikan tanpa harus menunggu selesainya fase dari seluruh proyek. Fleksibilitas ini mengurangi risiko pemborosan *resources* dan memastikan produk yang dikembangkan selalu sesuai dengan kebutuhan user yang terus berkembang.



Gambar 1.2 Metodologi dengan Kultur DevOps

1.1.3 Prinsip Utama DevOps

1.1.3.1 Collaboration

Pendekatan dengan metode tradisional sering menciptakan tembok dimana tim pengembang fokus kepada kecepatan dan pengiriman fitur, sementara tim operasi memprioritaskan *stability* dan *reliability*. DevOps menghilangkan pembagian ini dengan menciptakan *cross-functional teams* yang dapat bekerja bersama dari *planning phase* hingga *production deployment, sharing knowledge, tools, and responsibilities* untuk mencapai tujuan bersama.

1.1.3.2 Automation

Semua pekerjaan yang bersifat repetitif (mengulang) dan dikerjakan secara manual akan diubah menjadi otomatis guna mendapatkan *consistency, reliability, and speed* dalam *software delivery*. *Automation* mencakup semua aspek dari *software development lifecycle*, mulai dari *code compilation, testing, security scanning, deployment, infrastructure provisioning*, hingga *monitoring alerts*. Menggunakan *automation* akan mengurangi tindakan manual oleh manusia, anggota tim dapat fokus kepada aktivitas lain seperti inovasi, pemecahan masalah, dan perancangan strategis, sekaligus mengurangi *human errors* yang seringkali menjadi sumber dari masalah pada *production*.

1.1.3.3 Continuous Integration/Deployment

Continuous Integration & Continuous Deployment (CI/CD) mengubah paradigma pengembang perangkat lunak dari cara tradisional yang bersifat kompleks dan memiliki resiko tinggi dengan cara modern yang memfokuskan pada publikasi kecil berkala. Artinya kode akan dipublikasikan secara berkala ke *production* untuk mendapatkan umpan balik lebih cepat dengan tetap menjaga prosedur yang telah ditetapkan seperti menjalankan tes otomatis, menuliskan laporan hasil tes, serta melakukan publikasi produk ke *production*. Pendekatan ini tidak hanya mempercepat pengiriman fitur tapi juga memungkinkan mendapatkan umpan balik lebih cepat, mempermudah proses *debugging* ketika ada masalah, serta mengurangi beban yang berat yang harus ditanggung oleh tim ketika terjadi masalah pada *production*.

1.1.3.4 Monitoring & Feedback

Menggunakan pendekatan DevOps, monitoring dan umpan balik akan menjadi salah satu strategi untuk observasi terhadap produk dan fitur yang dibangun. Ada beberapa metrik yang akan terus diperhatikan, misalnya, *performance monitoring, infrastructure metrics, user behavior analytics, and business KPIs*. Konsep DevOps yang menghadirkan monitoring akan membawakan hal yang baik bagi user experience, pengaruh terhadap bisnis, serta membuat tim dapat bergerak berdasarkan data aktual (*data-driven*). Monitoring juga akan berpengaruh terhadap keputusan bisnis yang akan dibuat berdasarkan metrik yang disediakan.

1.1.3.5 Shared Responsibility

Kultur DevOps akan menjadikan sebuah produk menjadi tanggung jawab bersama pada sebuah tim dimana pengembang perangkat lunak tidak hanya bertanggung jawab dalam menulis kode tetapi juga untuk memastikan kode dapat di-deploy, di-monitor, dan dapat dikelola di kemudian hari. Tim operasi juga tidak hanya melakukan manajemen infrastruktur tetapi juga kolaborasi dalam pengambilan keputusan terhadap infrastruktur dan kontribusi pada tahap pengembangan demi mencapai tujuan bersama.

1.2 Fundamental Git & GitHub

1.2.1 Git

Git adalah *distributed version control system* yang dikembangkan oleh Linus Torvalds pada tahun 2005 untuk mengelola *source code Linux kernel*. Git dirancang untuk menangani proyek dengan berbagai ukuran dan kompleksitas, mulai dari *small personal projects* hingga *massive enterprise applications* dengan pengembang yang berjumlah ratusan hingga ribuan. Sebagai *distributed system*, Git memberikan setiap pengembang salinan lengkap dari *project history*, artinya setiap pengembang memiliki keuntungan untuk mengerjakan baris kodennya secara *offline* dan dapat menemukan perubahan yang dari baris kode yang dikerjakan oleh pengembang lainnya.

Menggunakan git, para pengembang dapat bekerja secara independen, melakukan eksperimen terhadap baris kode tanpa mengganggu pekerjaan pengembang lainnya, serta melakukan kolaborasi dari perubahan pada *file* yang berbeda. Git memiliki banyak fitur diantaranya adalah *branching* dan *merging operations*, ini memungkinkan tim pengembang yang membutuhkan perubahan kode dalam waktu proyek yang bersamaan meskipun perubahannya melibatkan banyak perubahan pada file maupun baris kode.

Git bekerja berdasarkan *content-addressed storage system* yang menggunakan hashing dengan algoritma SHA-1 untuk mengidentifikasi perubahan, memastikan integritas data dan perubahan pada baris kode. Repository yang berisi Git terdiri dari objek (*blobs*, *trees*, *commits*, dan *tags*) yang terhubung secara bersamaan dalam sebuah *graph* yang menyediakan informasi dalam melakukan pengecekan perubahan setiap waktu dan memahami perbedaan perubahan *file* ataupun baris kode dari versi yang berbeda-beda yang ditulis oleh pengembang lainnya.

1.2.2 GitHub

GitHub adalah *cloud-based hosting service* untuk menyimpan kode yang menggunakan Git sebagai *tools version control* yang diluncurkan pada tahun 2008 dan sekarang dimiliki oleh Microsoft. GitHub memanfaatkan fungsi dari **git** dengan menyediakan tampilan berbasis web, *platform* untuk berkolaborasi antar pengembang perangkat lunak, fitur proyek management, dan ekosistem yang baik untuk pengembangan perangkat lunak. *Platform* ini menjadi salah satu standar untuk pembuatan proyek

berbasis *open-source* dan diadopsi oleh korporasi untuk menyimpan repositori privat serta kolaborasi antar tim internal.

GitHub menawarkan salah satu fitur yang memudahkan pemeriksaan kode yang disebut *pull requests*, *tracking issues* untuk submisi bug (kesalahan program), *GitHub Actions* untuk CI/CD *automation*, serta beberapa fitur lainnya seperti project boards, dan permission system untuk akses kontrol ke repositori. GitHub juga membangun platformnya sebagai media sosial untuk para pengembang perangkat lunak dengan menghadirkan fitur favorit untuk repositori, fitur *follows*, dan halaman diskusi untuk proyek yang dibuka ke publik dan bersifat *open-source*.

GitHub menyediakan integrasi dengan tools developer seperti Bash/Command Line Interface, IDE, dan layanan pihak ketiga untuk membuat automasi ke repositori github. Fitur AI seperti GitHub Copilot (Pelengkap kode berbasis AI), *security vulnerability scanning* pada repositori, *dependency management*, serta package registry yang membuat GitHub tidak hanya sebagai platform untuk menyimpan repositori kode melainkan juga mendukung seluruh ekosistem *software development cycle* dari *planning* hingga *deployment* bahkan *maintenance*.

1.2.3 Perbandingan Git dengan Github

Penting untuk memahami bahwa **Git** dan **GitHub** adalah dua hal berbeda. **Git** adalah *version control tool* yang berjalan secara lokal pada perangkat yang dimiliki pengembang perangkat lunak, sementara **GitHub** adalah layanan *cloud service* yang menyediakan penyimpanan repository dengan tools **Git** dan menyediakan fitur kolaborasi tambahan. Analogi yang bisa membantu adalah **Git** seperti mesin dari sebuah mobil, sedangkan GitHub seperti pabrik mobil yang memungkinkan siapa saja untuk datang berkunjung, kolaborasi, atau bahkan mencoba mobil dengan versi yang berbeda.

Pengembang perangkat lunak dapat menggunakan Git tanpa GitHub dengan membuat repositori lokal pada perangkat masing-masing atau menyimpan kode tersebut pada layanan lain seperti **GitLab**, **Bitbucket**, atau **self-hosted solutions**. Meskipun dapat disimpan pada layanan lain, **GitHub** adalah salah satu platform populer yang digunakan oleh kebanyakan pengembang perangkat lunak dan perusahaan.

1.2.4 Instalasi Git

Git perlu diunduh dan dipasang sebelum digunakan, kunjungi git-scm.com dan pilih perangkat yang digunakan. Sistem operasi windows dapat menggunakan file .exe dan pilih *standalone installer* untuk instalasi, sedangkan mac OS dan linux dapat menggunakan terminal.

Mac OS

```
$ brew install git
```

Linux (Debian/Ubuntu)

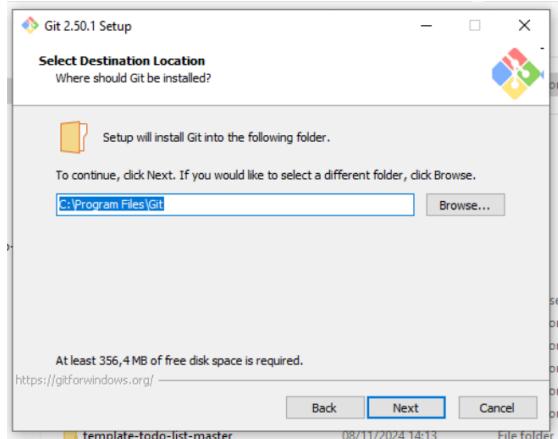
```
$ apt-get install git
```

Sesaat setelah mengunduh, klik file dengan format .exe lalu lakukan instalasi secara bertahap.



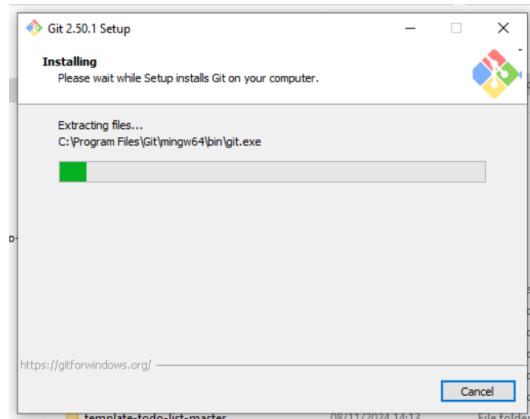
Gambar 1.3 Popup setup Git versi 2.50.1

Catatan: Tidak harus versi yang sama seperti yang tertera di atas.



Gambar 1.4 Popup lokasi instalasi Git versi 2.50.1

Setelahnya cukup klik *next* secara terus menerus, biarkan semua konfigurasi mengikuti orisinal bawaan dari *installer* Git.

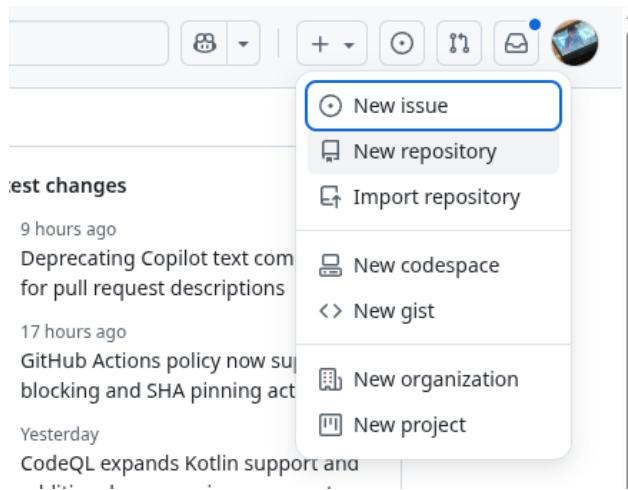


Gambar 1.5 Proses ekstraksi dan instalasi Git versi 2.50.1

Ketika *installer* Git melakukan ekstraksi *file* dan proses instalasi, tunggu sejenak hingga muncul tulisan *finish*.

1.2.5 Membuat proyek Pertama menggunakan Github

Untuk membuat proyek pertama pada Github, daftar akun terlebih dahulu melalui website github.com. Setelahnya kamu akan otomatis masuk ke dalam dashboard.



Gambar 1.6 Menu profil Github

Pada bagian pojok kanan atas klik new repository untuk membuat repositori baru penyimpanan di Github.

The screenshot shows the 'Create a new repository' form. At the top, there are links for 'Preview' and 'Switch back to classic experience'. Below that, a note says 'Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#). Required fields are marked with an asterisk (*).'

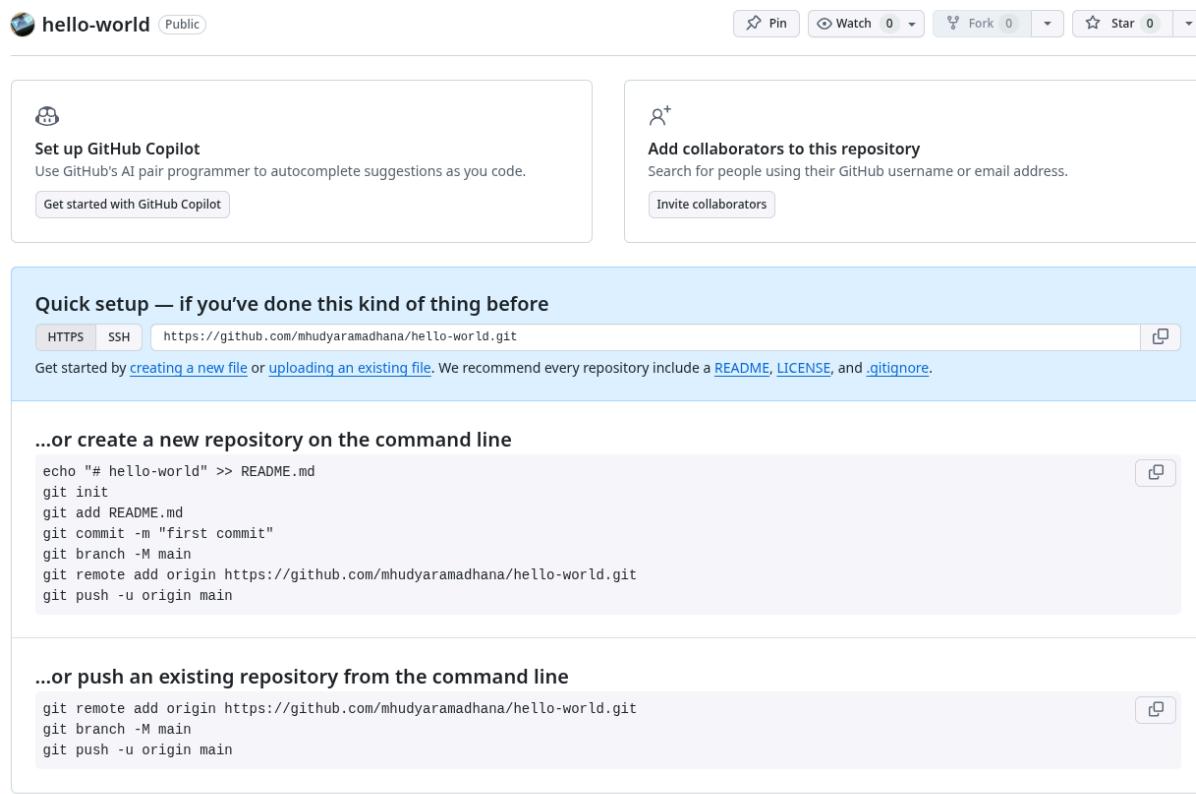
The form is divided into two sections:

- 1 General**
 - Owner *: A dropdown menu showing 'mhudyaramadhana'.
 - Repository name *: An input field with a placeholder '/'.
 - A note below says: 'Great repository names are short and memorable. How about [bug-free-adventure](#)?'.
 - Description: A text area with a character limit of '0 / 350 characters'.
- 2 Configuration**
 - Choose visibility *: A dropdown menu set to 'Public'.
 - A note below says: 'Choose who can see and commit to this repository'.

Gambar 1.7 Proses pembuatan repositori baru

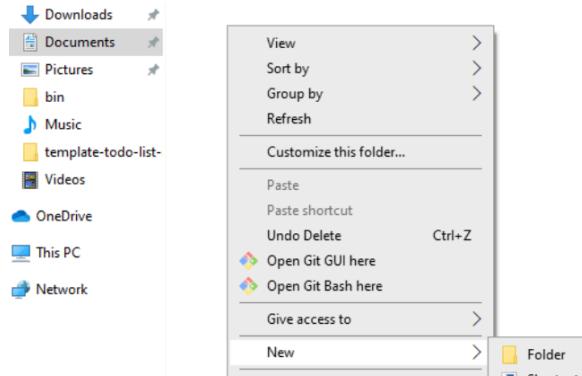
Selanjutnya, tuliskan nama repositori. Secara otomatis semua spasi akan diubah menjadi strip (-) sebagai pemisah. Untuk deskripsi juga bisa dimasukkan dengan teks apapun yang menjelaskan proyek tersebut. Berikut penjelasan untuk masing-masing konfigurasi:

- Visibility akan mempengaruhi apakah proyek tersebut bisa diakses secara publik atau hanya pemilik akun dan kontributor saja nantinya.
- README berfungsi sebagai file tentang deskripsi proyek secara teks yang ditulis dalam format markdown, format yang memungkinkan kita mengelola teks dengan format seperti bold, italic, bahkan memasukkan gambar pada deskripsi tersebut. **Catatan: tidak perlu dicentrang (diganti ke ON) pada bagian Add README.**
- .gitignore adalah file yang bisa digunakan untuk melakukan konfigurasi dalam mencegah file-file yang akan diunggah ke repositori misalnya seperti file yang berisi konfigurasi dari proyek atau yang biasa disebut .env (dot env)
- License adalah lisensi yang bisa digunakan untuk menjelaskan apakah proyek tersebut boleh disalin dan dikembang oleh pengembang lainnya atau tidak.



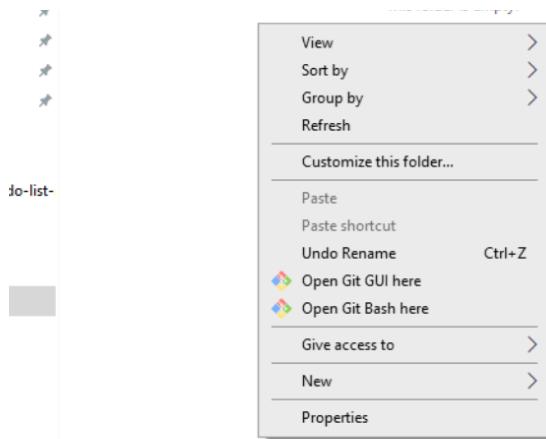
Gambar 1.8 Tampilan setelah pembuatan repositori

Gambar di atas adalah tampilan dari repositori yang baru saja dibuat, pada bagian bawah terdapat panduan untuk melakukan push (upload kode) ke Github.



Gambar 1.9 Menu explore windows

Selanjutnya, buat folder baru untuk mencoba mengirim kode ke repositori Github.



Gambar 1.10 Menu explore windows

Masuk ke dalam folder, sebagai contoh pada gambar di atas folder yang dibuat bernama **latihan**. Klik kanan di dalam folder tersebut lalu pilih opsi **Open Git Bash here**. Git Bash adalah terminal bawaan Git yang sudah terintegrasi dengan Windows, terminal ini adalah gambaran dari terminal pada Linux / Mac.

A screenshot of a Git Bash terminal window titled 'MINGW64:/c/Users/Hudya/Documents/latihan'. The window shows the command '\$ git init' being run, followed by the message 'Initialized empty Git repository in C:/Users/Hudya/Documents/latihan/.git/'. The terminal prompt ends with '\$'. At the top of the window, there's a navigation bar with 'This PC > Documents > latihan' and a search bar labeled 'Search latihan'. Below the title bar, there's a header with columns for 'Name', 'Date modified', 'Type', and 'Size'. The terminal window is set against a background of a file explorer view.

Gambar 1.11 Git bash

Perintah pertama yang perlu dijalankan adalah ***git init***. Perintah tersebut melakukan inisiasi terhadap folder untuk menyiapkan kebutuhan yang berhubungan dengan ***Git***. Nantinya akan ada folder bernama ***.git*** yang secara bawaan disembunyikan, folder ini bisa dimunculkan apabila opsi tampilkan hidden folder dinyalakan pada konfigurasi Explorer Windows.

Meskipun pada repositori Github yang dibuat terdapat panduan untuk melakukan push pertama kali, namun terdapat cara yang lebih sederhana untuk melakukannya yang akan ditulis pada bagian bawah. Secara teori untuk melakukan push terhadap kode menuju Github **saat pertama kali proyek dibuat**, perintah yang diperlukan adalah:

```
-- untuk inisiasi folder saat pertama kali folder dibuat  
git init  
  
-- untuk membuat file percobaan berisi teks Hello World pada file hello-world.txt  
echo "Hello, World" >> hello-world.txt  
  
-- untuk menambahkan semua file ke dalam sebuah penampung sebelum dikirim  
git add .  
  
-- untuk memberikan pesan pada file yang telah ditambahkan  
git commit -m "masukkan pesan"  
  
-- untuk menambahkan remote url dari repositori github  
git remote add <namaremote> <url>  
  
contoh:  
git remote add origin https://github.com/mhudayaramadhana/hello-world.git  
  
-- untuk mengirimkan kode ke repositori online  
git push <namaremote> <branch>  
  
contoh:  
git push origin master
```

Ketika melakukan commit dan perangkat yang digunakan baru pertama kali melakukannya, maka akan ada kemungkinan error identity.

Untuk mengatasi error ini, perlu dilakukan konfigurasi Git dengan menjalankan dua command yang ditunjukkan pada output error: mengatur email dan nama user secara global. Setelah konfigurasi ini dilakukan, Git akan dapat melakukan commit dengan menyertakan *author information* yang valid.

```
Author identity unknown  
  
*** Please tell me who you are.
```

Run

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

Gambar 1.12 Error commit identity

Gambar di atas menunjukkan bahwa konfigurasi terhadap git belum dijelaskan atas nama dan email siapa yang melakukan commit caranya adalah:

```
-- config email  
git config --global user.email <email>  
  
-- config username  
git config --global user.name <name>  
  
-- Contoh:  
git config --global user.email mhudyaramadhana@gmail.com  
git config --global user.name perogeremmer
```

Setelahnya silahkan coba kembali lakukan **git commit**.

Kesalahan "*Author identity unknown*" dalam Git terjadi karena Git belum mengkonfigurasi identitas pengguna, yaitu nama dan alamat email. Informasi ini krusial untuk kontrol versi, karena setiap *commit* mencatat siapa yang melakukan perubahan, kapan perubahan dilakukan, dan apa yang diubah. Pencatatan identitas ini penting untuk beberapa alasan, termasuk keterlacakkan, kolaborasi, dan jejak audit.

Tanpa identitas yang benar, Git tidak dapat menjalankan fungsi intinya, menyebabkan pengelolaan proyek menjadi kacau dan menghambat kolaborasi serta pemeliharaan kode. Oleh karena itu, setelah instalasi Git, langkah pertama yang harus dilakukan adalah mengkonfigurasi nama pengguna dan email.

```
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan
$ git init
Initialized empty Git repository in C:/Users/Hudya/Documents/latihan/.git/
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ echo "Hello, World" >> hello-world.txt

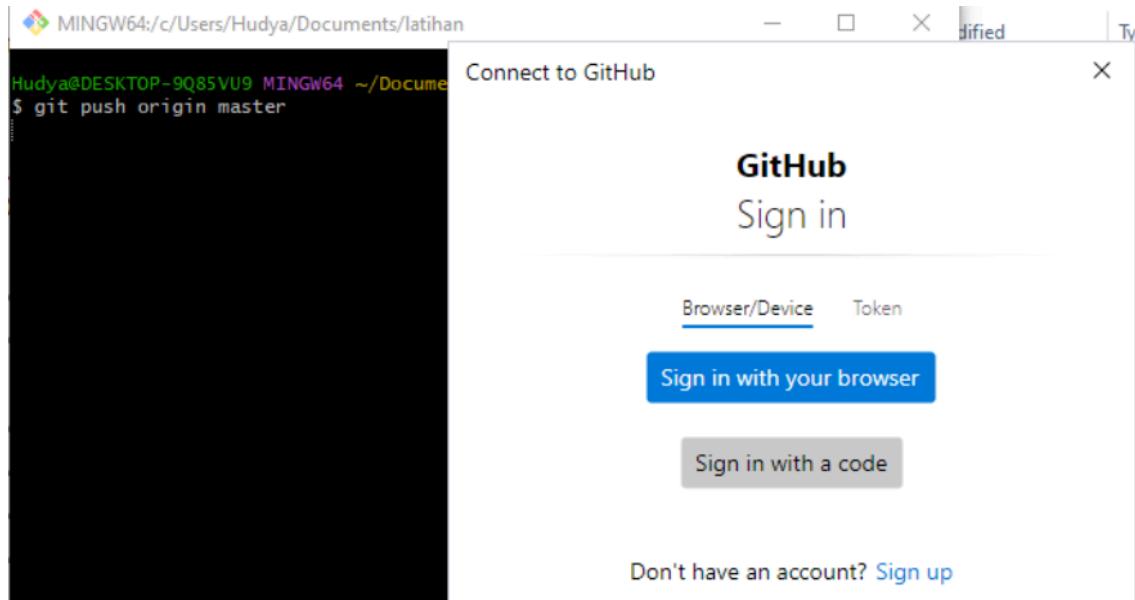
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ git add .
warning: in the working copy of 'hello-world.txt', LF will be replaced by CRLF t
he next time Git touches it

Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ git commit -m "First Commit"
[master (root-commit) d6977d] First Commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello-world.txt

Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ git remote add origin https://github.com/mhudyaramadhana/hello-world.git

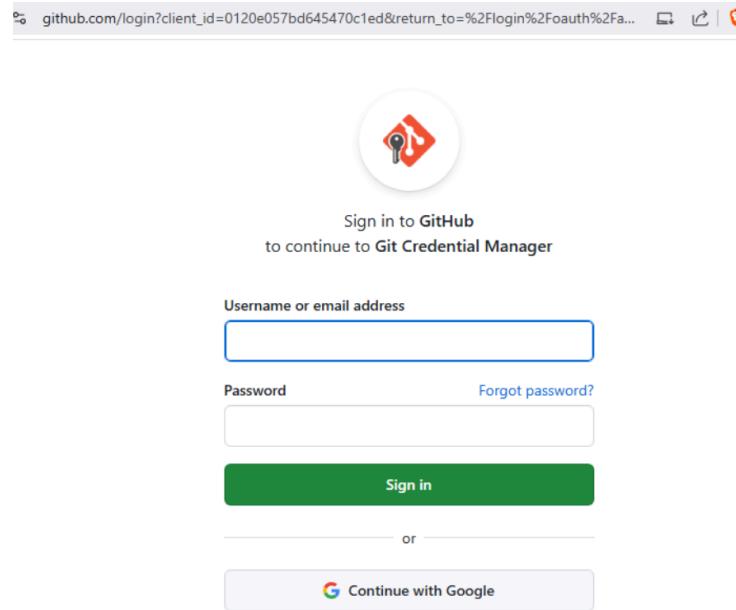
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ git push origin master
```

Gambar 1.13 Proses git dari init



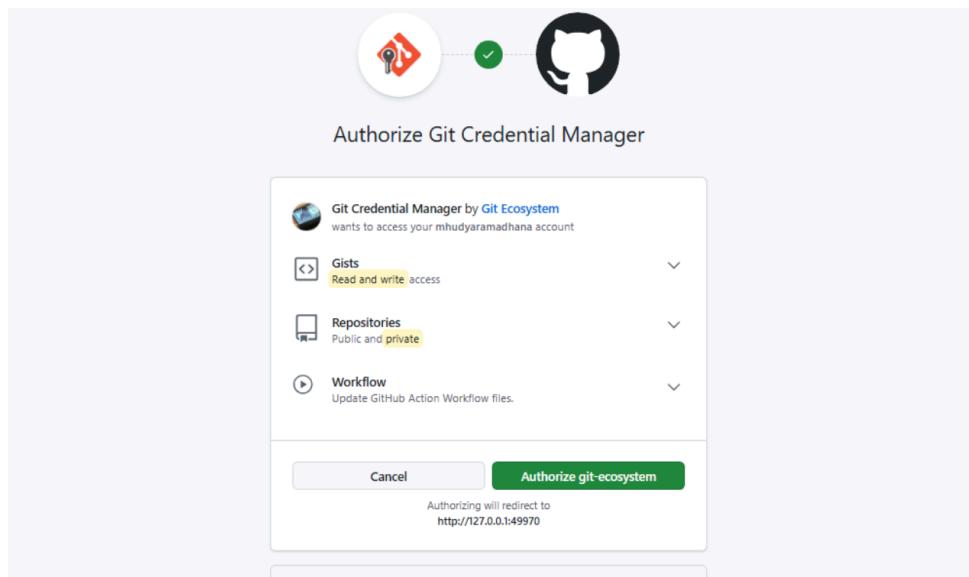
Gambar 1.14 Proses *sign in* github

Apabila perangkat yang digunakan belum pernah melakukan autentikasi, akan muncul sebuah jendela yang meminta pengguna untuk masuk menggunakan akun Github. Cukup gunakan opsi ***Sign in with your browser***.



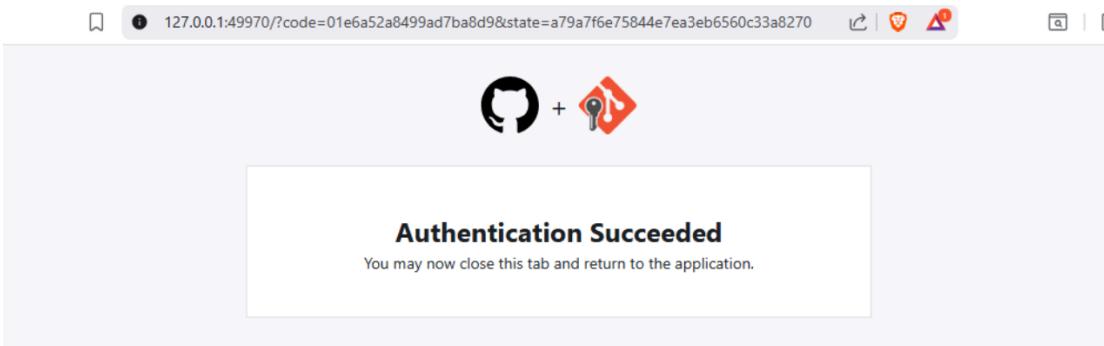
Gambar 1.15 Sign in ke Github

Pengguna akan diarahkan untuk melakukan autentikasi menggunakan username dan password, atau dapat menggunakan akun Google.



Gambar 1.16 Proses otorisasi git credential manager

Pengguna akan diminta untuk memberikan perizinan terhadap tools yang digunakan yaitu **Git Credential Manager**, cukup tekan authorize pada tombol berwarna hijau pada bagian bawah lalu tunggu.



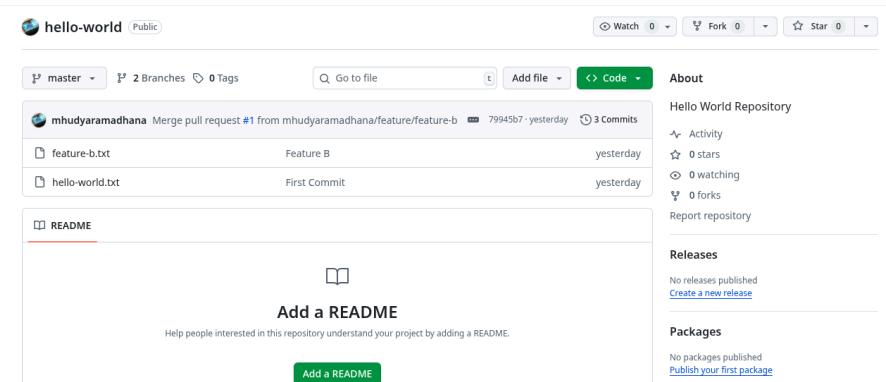
Gambar 1.17 Proses otorisasi berhasil

Apabila autentikasi berhasil dilakukan, silahkan kembali buka terminal Git yang sebelumnya sedang melakukan proses *push (upload)* kode.

A screenshot of a terminal window titled 'MINGW64:c/Users/Hudya/Documents/latihan'. The window shows the command \$ git push origin master being run. The output indicates that authentication is required in the browser, objects are being enumerated, counted, and written, and finally pushed to the 'hello-world' repository on GitHub. The message ends with '* [new branch] master -> master'.

Gambar 1.18 Proses git push

Apabila tampilan pada terminal sudah menampilkan tampilan seperti pada gambar di atas, bisa dipastikan bahwa kode sudah di-push ke Github.



Gambar 1.19 Tampilan repositori setelah di-push

Untuk memastikannya, silahkan *refresh* halaman repositori dan lihat *file* pertama sudah berhasil diunggah. Berikut merupakan elemen dan fungsinya:



Gambar 1.20 Header Section

Header Section:

- **Repository Name** (`hello-world`): Nama repository yang ditampilkan dengan status Public
- **Pin** (📌): Untuk menyematkan repository di profile Anda
- **Watch** (👁): Subscribe ke repositori untuk mendapatkan notifikasi pembaruan
- **Fork** (🍴): Membuat salinan dari repositori ke akun sendiri
- **Star** (⭐): Bookmark repositori dan menunjukkan appreciation (seperti "like")



Gambar 1.21 Navigation Tabs

Navigation Tabs:

- **Code**: Halaman utama untuk browsing source code dan files
- **Issues**: Bug tracking, feature requests, dan diskusi project
- **Pull Requests**: Code review dan merge requests dari contributors
- **Actions**: CI/CD workflows dan automation yang running di repositori
- **Projects**: Project management boards untuk organizing tasks
- **Security**: Security advisories, vulnerability scanning, dan security policies
- **Insights**: Analytics tentang commits, contributors, traffic, dan repository metrics



Gambar 1.22 Main Content Area

Main Content Area:

- **Branch Selector** (main): Dropdown untuk mengganti antar *branch*
- **Commit Count** (1 commit): Jumlah *total commits* dalam pada branch saat ini
- **Latest Commit Info**: Menampilkan commit message dan timestamp terakhir
- **Add File** (dropdown): Options untuk create new file atau upload existing files
- **Code** (tombol hijau): Clone options dengan HTTPS, SSH, dan GitHub CLI commands
- **Go to file** (ikon pencarian): Pencarian cepat untuk mencari file di dalam repositori
- **Branches (ikon cabang)**: Daftar cabang yang ada pada repositori
- **Tags (ikon tag)**: Daftar tags yang dimiliki pada repositori

About

Hello World Repository

Activity

0 stars

0 watching

0 forks

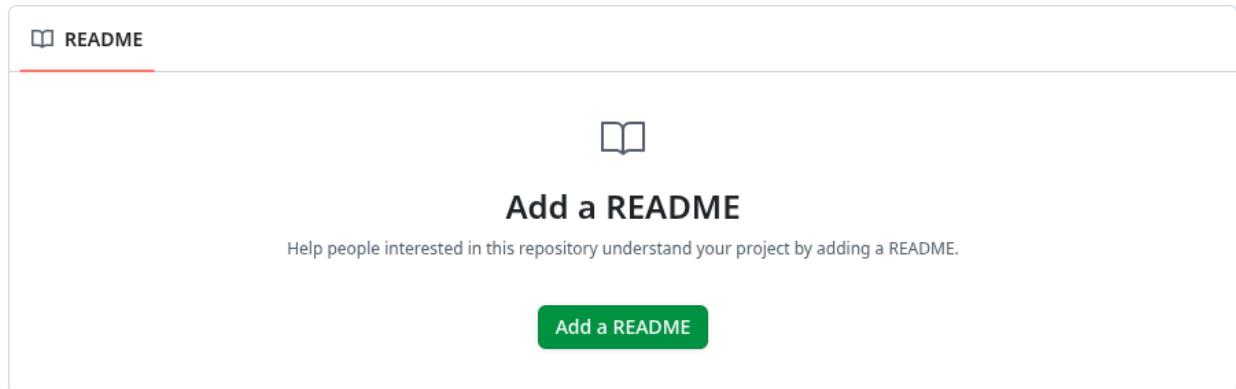
Report repository

Releases

Gambar 1.23 Sidebar Information

Sidebar Information:

- **About:** Deskripsi repositori, topics/tags, dan URL website
- **Releases:** Versi software yang telah dibungkus dalam *versioning*
- **Packages:** Published packages (NPM, Docker, etc.) yang terhubung dengan repositori
- **Activity:** Aktivitas terbaru dari repositori
- **Contributors:** Kontributor dari proyek repositori



Add a README

Help people interested in this repository understand your project by adding a README.

Add a README

Gambar 1.24 File-Specific Actions

File-Specific Actions:

- **README.md:** File yang ditampilkan sebagai informasi ketika repositori dibuka.
- **Add a README button:** Tombol cepat untuk membuat README file bila belum ada.



Gambar 1.25 Additional Interface Elements

Additional Interface Elements:

- **Latest commit Author:** Kontributor terakhir yang mengirimkan commit
- **Latest commit hash:** Short SHA yang bisa di-klik untuk melihat rincian *commit*
- **Commit message preview:** Pesan terakhir commit yang bisa di-klik untuk melihat rincian *commit*

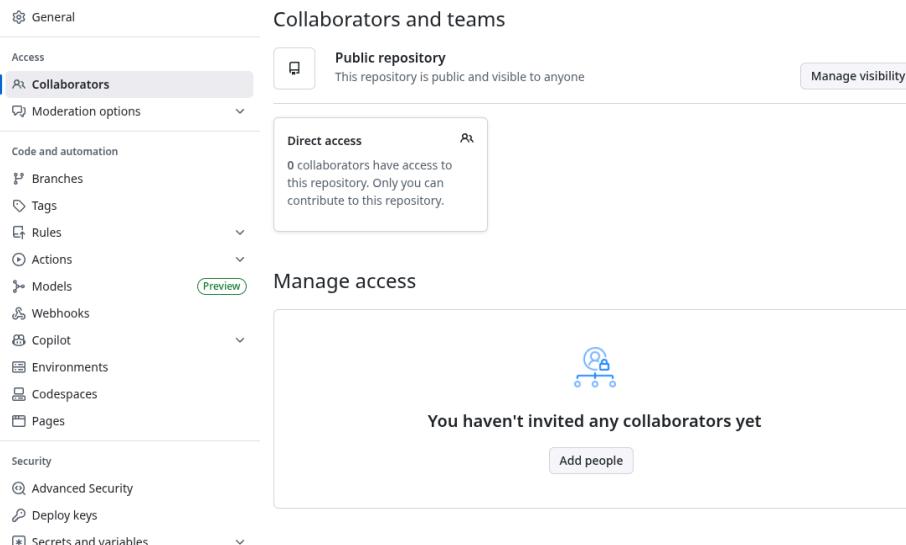
1.2.6 Merubah isi file

Apabila isi file **hello-world.txt** diubah, untuk mengirimkan perubahannya ke Github cukup lakukan tiga langkah berikut:

```
git add .
git commit -m "masukkan pesan"
git push origin master
```

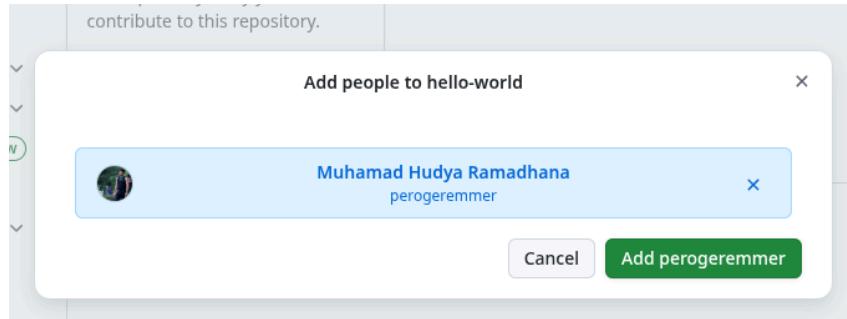
Jalankan satu persatu perintah di atas pada setiap barisnya, maka perubahan bisa dilihat pada halaman Github.

1.2.7 Mengundang Kontributor



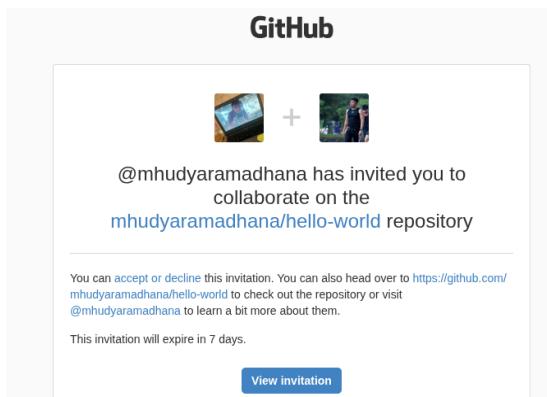
Gambar 1.26 Halaman undangan kontributor

Dalam mengerjakan sebuah proyek, biasanya sebuah repositori dikerjakan tidak selalu sendirian. Untuk mengundang kontributor baru, klik tombol settings lalu pilih *collaborators*. Klik tombol add people dan kontributor baru bisa diundang dengan memasukkan usernamenya.



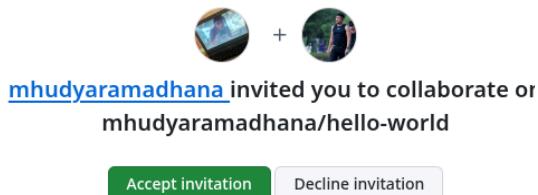
Gambar 1.27 *Popup* konfirmasi undangan

Setelah diundang, kontributor harus membuka email untuk mendapatkan URL undangan untuk melakukan konfirmasi.



Gambar 1.28 *Popup* konfirmasi

Klik tombol *view invitation* pada gambar di atas. Selanjutnya akan ditampilkan sebuah halaman untuk menerima undangan sebagai kontributor (*collaborator*).



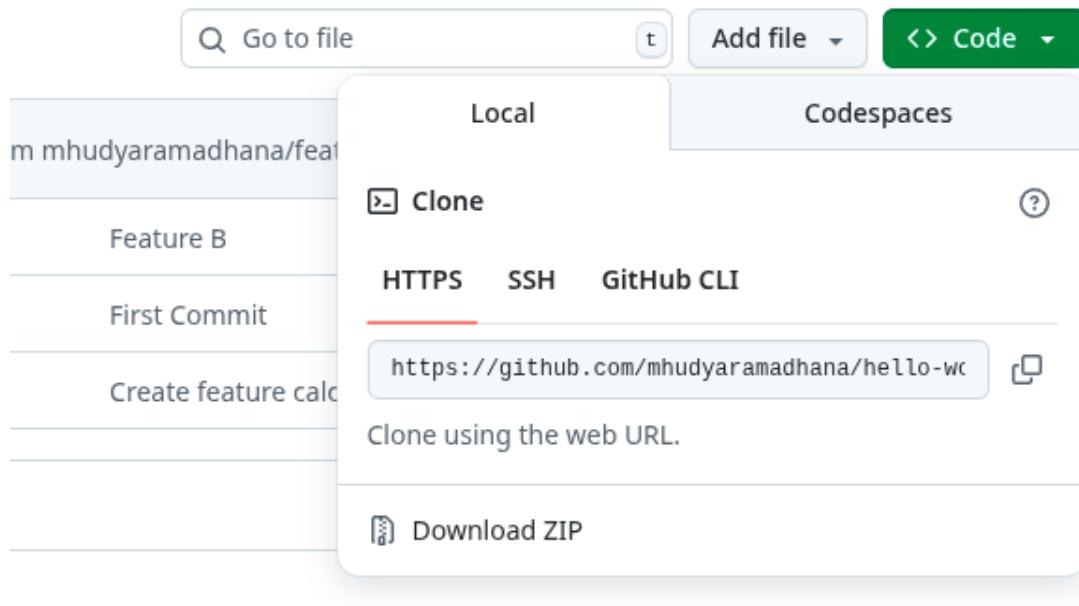
Gambar 1.29 *Popup* terima undangan

Klik tombol *accept invitation*, setelahnya kontributor baru bisa melakukan push terhadap repositori yang telah diundang.

1.2.8 Kloning Repository

Setelah berhasil menerima undangan untuk menjadi kontributor, perangkat penerima undangan bisa melakukan kloning terhadap repositori. Kloning adalah proses untuk mendownload isi kode repositori termasuk dengan keseluruhan sejarah perubahan git yang telah terjadi pada proyek tersebut.

Pertama, buka URL repositori yang ingin di-clone, lalu klik bagian code.



Gambar 1.30 Menu code

Salin URL HTTPS dengan klik icon copy pada bagian kanan, lalu masukkan perintah untuk kloning menggunakan terminal:

```
-- untuk melakukan cloning
git clone <url>

-- Contoh:
hudya@perogeremmer-pc:~/code$ git clone
https://github.com/mhudyaramadhana/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 19 (delta 6), reused 10 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (19/19), 4.04 KiB | 4.04 MiB/s, done.
Resolving deltas: 100% (6/6), done.
```

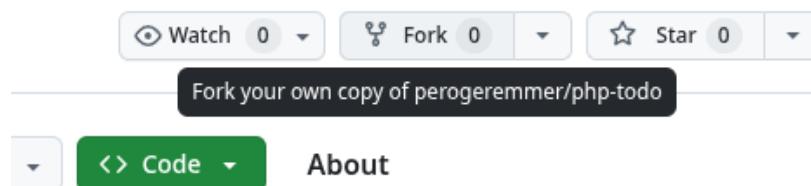
1.2.9 Forking

Forking pada GitHub adalah proses membuat salinan repositori (proyek) milik pengguna lain ke akun GitHub Anda sendiri. Salinan ini sepenuhnya independen dari repositori asli, artinya Anda dapat membuat perubahan, menambahkan fitur, atau memperbaiki bug pada salinan tersebut tanpa memengaruhi repositori aslinya.

Tujuan dan Manfaat Forking:

- **Kontribusi Open Source:** Forking adalah langkah pertama yang paling umum dalam berkontribusi pada proyek open source. Setelah melakukan fork terhadap repositori, pengembang dapat mengerjakan perubahan pada salinan pengembang dan kemudian mengajukan ***pull request*** ke pemilik repositori asli. Jika perubahan pengembang diterima, ***commit*** pengembang akan diintegrasikan ke dalam proyek utama.
- **Eksperimen dan Pengembangan Independen:** Pengembang dapat menggunakan fork untuk bereksperimen dengan ide-ide baru, mencoba fitur yang berbeda, atau mengembangkan fungsionalitas tambahan tanpa khawatir merusak proyek asli. Ini sangat berguna ketika pengembang ingin menguji sesuatu secara menyeluruh sebelum mengusulkannya untuk digabungkan.
- **Dasar untuk Proyek Baru:** Terkadang, pengembang mungkin menemukan sebuah proyek yang sangat bagus sebagai titik awal untuk ide pengembang. Dengan melakukan fork, pengembang mendapatkan salinan lengkap yang dapat diubah dan kembangkan menjadi proyek yang sepenuhnya baru dan terpisah.
- **Belajar dan Memahami Kode:** Mem-fork sebuah proyek memungkinkan pengembang untuk menjelajahi kode sumber secara mendalam, memahami bagaimana proyek tersebut dibangun, dan melihat bagaimana fitur-fitur tertentu diimplementasikan. Ini adalah cara yang bagus untuk belajar dari sudut pandang pengembang lain.

Singkatnya, forking adalah mekanisme inti di GitHub yang memfasilitasi kolaborasi, pengembangan independen, dan kontribusi dalam ekosistem open source. Pertama, buka URL repositori yang ingin di-fork, lalu klik bagian fork.



Gambar 1.31 Tombol Fork

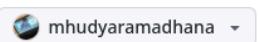
Setelahnya, akan muncul halaman layaknya membuat repositori baru. Karena memang fork akan membuat sebuah repositori baru yang akan tersimpan pada profil Github Pengembang

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Required fields are marked with an asterisk (*).

Owner *



Repository name *

/ php-todo

• php-todo is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description

0 / 350 characters

Copy the master branch only

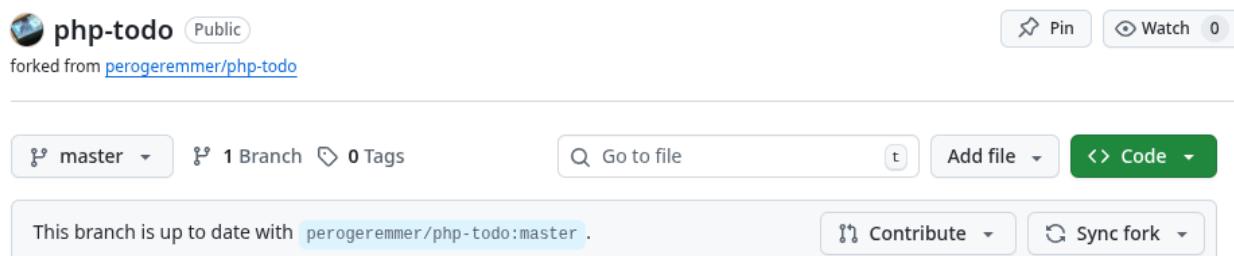
Contribute back to perogeremmer/php-todo by adding your own branch. [Learn more](#).

ⓘ You are creating a fork in your personal account.

Create fork

Gambar 1.32 Menu code

Setelah klik create fork, maka repositori akan ditampilkan dimana ada keterangan **forked from ...**, yang menjelaskan bahwa repositori tersebut merupakan repositori fork dari repositori asli.



Gambar 1.33 Hasil Repositori Fork

Fitur *sync fork* memungkinkan pengembang untuk mendapatkan kode terbaru dari repositori asli.

1.3 Strategi *Branching*

Pada bagian sebelumnya proyek Github di-push pada branch bernama master. Branch ini merupakan branch bawaan (*default*) yang digunakan sebagai branch utama. Meskipun pada pembaruan terakhir

standarisasi dari branch utama diganti menjadi ***main***, namun masih banyak pengembang perangkat lunak yang menggunakan istilah ***master*** sebagai *branch* utama.

Pada Github ada sebuah fitur yang bernama branch, artinya pengembang perangkat lunak bisa membuat sebuah cabang pada repositori yang tujuannya memisahkan perubahan kode yang dikerjakan antar pengembang. Contohnya sebagai berikut:

Pengembang A mengerjakan fitur A dan akan membuat sebuah file baru bernama feature-a.txt sedangkan pengembang B akan mengerjakan fitur B dengan membuat sebuah file baru bernama feature-b.txt, nantinya apa yang dikerjakan pengembang B tidak akan mengganggu pekerjaan pengembang A karena dikerjakan pada cabang yang berbeda.

Fitur branching nantinya akan digabungkan pada sebuah fitur yang bernama pull requests dimana pengembang akan membuka *branch* dari fitur yang dikerjakan agar kodennya dapat digabung dengan *branch* utama yaitu *master*.

1.3.1 Standar penamaan branch

Dalam manajemen proyek menggunakan konsep DevOps, konsistensi penamaan branch sangat crucial untuk:

Clarity & Communication: Nama branch yang jelas membantu anggota tim memahami tujuan dan scope dari setiap branch tanpa perlu membuka code atau membaca documentation. Ketika pengembang perangkat lunak melihat *branch* bernama ***feature/user-authentication***, mereka langsung tahu bahwa *branch* tersebut berisi kode untuk fitur autentikasi user.

Organization & Management: Dengan penamaan branch yang konsisten, *project manager* dan *team leader* dapat melakukan monitor terhadap progress lebih mudah, menemukan *bottleneck*, dan menyusun beban pekerjaan. *Tools* seperti GitHub dan GitLab juga dapat secara otomatis menyusun dan filter branch berdasarkan *naming patterns*, yang membuat manajemen proyek menjadi lebih efisien.

Automation & CI/CD Integration: Banyak tools automation dan pipeline CI/CD bergantung pada pattern penamaan branch untuk mengaktifkan tindakan spesifik. Misalnya, *branch* dengan prefix ***feature/*** akan di-deploy ke *development environment*, sedangkan *branch* ***hotfix/*** akan mendeploy kode ke *production deployment* pipeline dengan tambahan pengecekan keamanan.

1.3.2 Struktur Dasar Penamaan Branch

Pada dasarnya struktur dasar penamaan branch dibagi menjadi dua, yaitu ***prefix*** dan ***suffix***. Contohnya adalah **<prefix>/<suffix>**. Berikut merupakan beberapa contoh penamaan branch menggunakan prefix dan deskriptif:

- feature/user-login
- hotfix/security-patch
- hotfix/user-login-wrong-capital
- feature/user-hash-password-authentication
- release/v2.1.0

Berikut merupakan contoh **prefix (awalan)** untuk menunjukkan tipe atau purpose dari branch:

- **feature/** - Pengembangan fitur baru
- **bugfix/** - Perbaikan bug yang tidak urgent
- **hotfix/** - Perbaikan critical bug di production
- **release/** - Persiapan release version baru
- **chore/** - Maintenance tasks, refactoring, atau housekeeping

Berikut merupakan panduan pembuatan **suffix** (penjelasan) untuk menjelaskan secara singkat apa yang dikerjakan:

- Gunakan lowercase letters
- Pisahkan kata dengan hyphen (-)
- Hindari spaces dan special characters
- Maksimal 3-4 kata untuk readability

Contoh:

- Ordinary feature: feature/**user-login**
- Developer initials: feature/**security-patch**
- Priority level: hotfix/**security-patch-critical**

1.3.3 Jenis-Jenis Prefix dan Penggunaannya

1.3.3.1 Feature Branches

Prefix **feature/** digunakan untuk pengembangan fitur baru atau *enhancement* dari fitur yang telah dibangun. Secara *lifecycle*, branch ini dibuat dari branch **master** lalu di-merge ke branch **development** untuk ujicoba internal, dan terakhir di-merge ke branch **master** apabila sudah layak ujicoba. Berikut contoh *branch feature*:

- feature/shopping-cart
- feature/payment-integration
- feature/user-profile-page
- feature/email-notifications
- feature/advanced-search

Penggunaan *branch feature* secara garis besar ditujukan ketika:

- Menambahkan fitur baru yang tidak relate dengan *user* secara langsung
- Implementasi fitur dari kebutuhan *user*
- Membuat komponen atau modul kode baru
- Perubahan tampilan yang signifikan

1.3.3.2 Bugfix Branches

Prefix **bugfix/** digunakan untuk memperbaiki *bug* yang ditemukan pada saat fase **development** atau **testing**. Secara *lifecycle*, branch ini dibuat ketika sebuah bug ditemukan dan akan langsung di-deploy ke **development environment** untuk ujicoba dan akan di-merge ke **production environment** setelah kode dinyatakan lulus ujicoba. Berikut merupakan contoh *branch bugfix*:

- **bugfix/login-validation-error**
- **bugfix/mobile-responsive-layout**
- **bugfix/database-connection-timeout**

Penggunaan *branch bugfix* secara garis besar ditujukan ketika:

- Bug yang bersifat tidak kritis dan tidak mempengaruhi production
- Isu yang ditemukan ketika ujicoba oleh tim Quality Assurance
- Kesalahan tampilan secara minor
- Isu performa yang bersifat tidak kritis

1.3.3.3 Hotfix Branches

Prefix **hotfix/** digunakan untuk memperbaiki *bug* yang ditemukan fixes yang harus segera di-deploy ke production. Secara *lifecycle*, branch ini dibuat dari *branch utama (master)* dan akan langsung di-merge ke *branch master* karena bersifat kritikal. Berikut merupakan contoh *branch hotfix*:

- hotfix/security-vulnerability
- hotfix/payment-gateway-down
- hotfix/data-corruption-fix
- hotfix/server-crash-patch

Penggunaan *branch hotfix* secara garis besar ditujukan ketika:

- Terjadi bug kritis pada production
- Ada masalah keamanan pada production
- Pencegahan kehilangan data
- Terjadi *performance issues* pada production yang bersifat kritis

1.3.3.4 Release Branches

Prefix **release/** digunakan untuk menyiapkan sebuah *branch* sebelum digabung ke *branch* utama (*master*). Biasanya pekerjaan para pengembang yang menggunakan branch berbeda akan digabung pada sebuah branch yang sama yang diberi nama **release/** yang pada akhirnya akan digabung ke branch **master** setelah semua kode tidak memiliki konflik (sebuah file yang memiliki perubahan dari dua pengembang dan bersinggungan baris kodennya). Secara *lifecycle*, branch ini dibuat dari branch utama (*master*) dan akan digabung dari branch *feature* lainnya yang telah dikerjakan oleh para pengembang. Berikut merupakan contoh *branch release*:

- release/v1.2.0
- release/2024-spring-update
- release/mobile-app-v3.1
- release/hotfix-1.1.1

Penggunaan *branch release* secara garis besar ditujukan ketika:

- Menyiapkan rilis yang dijadwalkan
- Branch final yang akan diuji coba oleh tim quality assurance

1.3.4 Membuat Branch Baru Pada Terminal

1.3.4.1 Menggunakan Terminal Bawaan

Menggunakan terminal bawaan Git (Git Bash) ataupun terminal windows, pengembang dapat membuat branch baru dengan cara sebagai berikut:

```
-- untuk membuat branch baru  
git checkout -b <nama branch>  
  
Contoh:  
git checkout -b feature/feature-a
```

Perintah git checkout membuat pengembang langsung pindah ke *branch* (cabang) yang baru, opsi -b adalah opsi yang menunjukkan bahwa apabila branch tersebut tidak ada maka buatkan branch tersebut.

```
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (master)
$ git checkout -b feature/feature-a
Switched to a new branch 'feature/feature-a'
```

Gambar 1.34 Proses *checkout branch*

Setelah membuat *branch* baru masukkan perintah berikut:

```
-- untuk membuat file baru
echo "New Feature" >> "new-feature.txt"

-- untuk menyiapkan file yang akan di-push
git add .

-- untuk menuliskan pesan commit
git commit -m "Feature A"

-- untuk melakukan push (upload) masukkan nama branch yang telah dibuat
git push origin feature/feature-a
```

```
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)
$ echo "New Feature" >> "new-feature.txt"

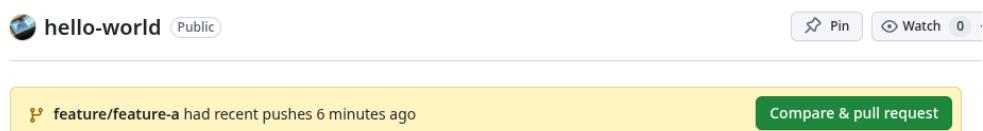
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)
$ git add .
warning: in the working copy of 'new-feature.txt', LF will be replaced by CRLF t
he next time Git touches it

Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)
$ git commit -m "Feature A"
[feature/feature-a d1621f5] Feature A
 1 file changed, 1 insertion(+)
 create mode 100644 new-feature.txt

Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)
$ git push origin feature/feature-a
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
```

Gambar 1.35 Proses *commit* dari *branch* baru

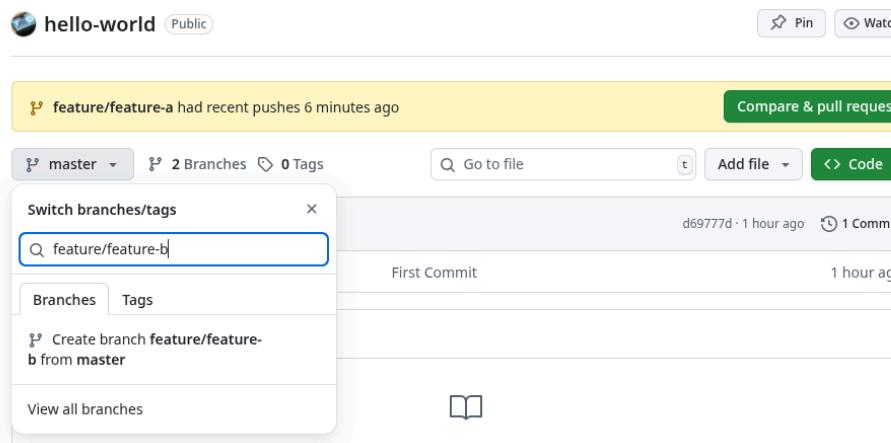
Setelah berhasil melakukan *push*, cobalah untuk membuka kembali halaman Git. Nantinya akan ada tulisan *branch* **feature/feature-a** memiliki data terbaru yang baru saja di-push dan ada opsi tombol hijau bernama *Compare & pull request*.



Gambar 1.36 Notifikasi setelah push

1.3.4.2 Menggunakan Halaman Repositori

Menggunakan halaman repositori, membuat *branch* baru juga bisa dilakukan. Caranya adalah dengan klik *dropdown branch master*, lalu tulisankan *feature/feature-b* di dalam kotak pencarian. Nantinya, tulisan *create branch feature/feature-b from master* akan muncul dan klik tulisan tersebut.



Gambar 1.37 Pembuatan *branch* dari repositori

Untuk mendapatkan *branch* tersebut pada *local repository* caranya adalah dengan menuliskan `git fetch` pada terminal lalu akan muncul keterangan *new branch*.

```
-- untuk mendapatkan list branch terbaru  
git fetch
```

```
MINGW64:/c/Users/Hudya/Documents/latihan  
hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)  
$ git fetch  
From https://github.com/mhudayaramadhana/hello-world  
 * [new branch]      feature/feature-b -> origin/feature/feature-b
```

Gambar 1.38 Proses `git fetch`

Setelah mendapatkan *branch* baru, cara selanjutnya adalah dengan pindah ke *branch* tersebut dengan perintah sebagai berikut:

```
-- untuk pindah branch  
git checkout <nama branch>  
  
contoh:
```

```
git checkout feature/feature-b
```

Jika diperhatikan, untuk pindah ke *branch feature/feature-b* tidak diperlukan lagi opsi -b. Hal ini disebabkan karena ketika menjalankan perintah *git fetch*, *local repository* sudah mendeteksi keberadaan *branch* tersebut sehingga opsi -b untuk membuat *branch* baru tidak diperlukan.

```
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-a)
$ git checkout feature/feature-b
branch 'feature/feature-b' set up to track 'origin/feature/feature-b'.
Switched to a new branch 'feature/feature-b'
```

Gambar 1.39 Proses checkout

Setelah pindah ke *branch feature/feature-b* baru masukkan perintah berikut untuk melakukan perubahan pada *branch* tersebut:

```
-- untuk membuat file baru
echo "Feature B" >> "feature-b.txt"

-- untuk menyiapkan file yang akan di-push
git add .

-- untuk menuliskan pesan commit
git commit -m "Feature B"

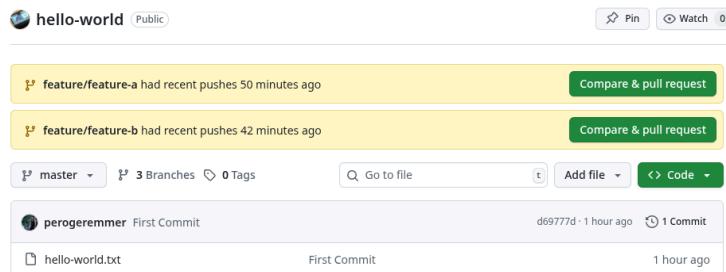
-- untuk melakukan push (upload) masukkan nama branch yang telah dibuat
git push origin feature/feature-b
```

```
Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-b)
$ git commit -m "Feature B"
[feature/feature-b cfd6418] Feature B
 1 file changed, 1 insertion(+)
 create mode 100644 feature-b.txt

Hudya@DESKTOP-9Q85VU9 MINGW64 ~/Documents/latihan (feature/feature-b)
$ git push origin feature/feature-b
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 297.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/mhudyaramadhana/hello-world.git
 d69777d..cf6418  feature/feature-b -> feature/feature-b
```

Gambar 1.40 Proses push

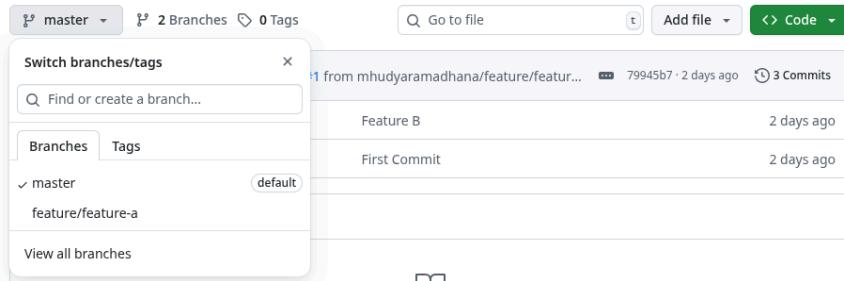
Sesaat setelah berhasil melakukan push terhadap branch feature/feature-b coba kembali buka halaman repositori di Github. Muncul dua notifikasi pada halaman tersebut yaitu branch feature/feature-a mendapatkan push baru, dan branch feature/feature-b mendapatkan push baru juga. Hal ini menunjukkan bahwa kedua cabang memiliki pembaruan dan tidak melibatkan branch utama (master).



Gambar 1.41 Notifikasi setelah push

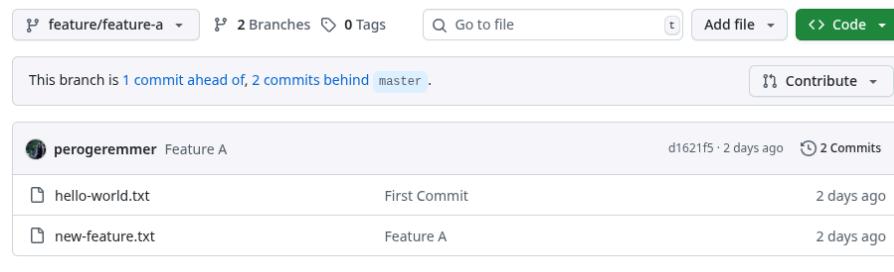
1.3.5 Berpindah Cabang

Untuk berpindah antar cabang cukup klik dropdown bertuliskan *master* dan pilih cabang yang ingin dituju.



Gambar 1.42 Proses pindah cabang

Setelah berpindah cabang, dapat dilihat bahwa pada cabang *feature/feature-a* sudah memiliki dua *file* yang sebelumnya telah dibuat dari *branch master* yaitu *hello-world.txt* dan *new-feature.txt* yang telah dibuat pada *branch feature/feature-a*.



Gambar 1.43 Berhasil membuat cabang

1.4 Pull Requests

Pada Git, istilah **Pull Request** (PR) merupakan mekanisme yang memungkinkan pengembang untuk memberitahu anggota tim bahwa mereka telah menyelesaikan sebuah fitur atau perbaikan dalam branch terpisah dan siap untuk di-review serta di-merge ke branch utama (master). Pull Request bukan bagian dari Git core functionality, melainkan feature yang disediakan oleh GitHub.

Konsep Pull Request menciptakan proses untuk pemeriksaan kode yang melibatkan *review*, diskusi, dan *quality assurance* sebelum perubahan kode masuk ke branch *master*. Proses ini adalah implementation dari "*four-eyes principle*" dalam *software development*, dimana setiap perubahan harus dilihat dan disetujui oleh minimal satu developer lain sebelum di-merge ke *production environment*.

Pull Request juga berfungsi sebagai mekanisme dokumentasi yang menyimpan konteks mengenai mengapa perubahan dibuat, bagaimana implementasinya, dan apa yang telah di-uji. Catatan ini menjadi acuan untuk maintain di masa mendatang, *debugging*, dan *knowledge transfer* dalam tim pengembang.

1.4.1 Lifecycle Pull Request

1.4.1.1. Preparation Phase

Sebelum membuat PR, pengembang diharuskan untuk:

- Menyelesaikan pekerjaan dalam branch *feature*
- Memastikan code telah di-uji secara lokal (*unit test*)
- Meninjau perubahan kode secara mandiri
- Memastikan branch yang dikerjakan *up-to-date* dengan target branch

1.4.1.2. Creation Phase

Pengembang membuat PR melalui *platform interface* dengan:

- Judul yang deskriptif dan menjelaskan perubahan
- Deskripsi yang rinci dan menjelaskan konteks dan implementasi
- Memilih pengembang senior untuk meninjau kode
- Merujuk isu yang berelasi atau dokumentasi

1.4.1.3. Review Phase

Pada masa *review*, pengembang yang ditunjuk untuk melakukan *review* harus melakukan:

- Peninjauan kode untuk kualitas, logika, dan *best practices*

- Ujicoba terhadap perubahan
- *Security assessment* untuk menemukan kerentanan sistem
- Diskusi dan umpan balik pada kolom komentar

1.4.1.4. Iteration Phase

Apabila mendapatkan umpan balik dari peninjau, pengembang harus melakukan:

- Menyelesaikan permintaan perbaikan dengan konteks tambahan
- Merespon terhadap pertanyaan untuk penjelasan
- Memperbarui dokumentasi jika diperlukan
- Meminta peninjauan kembali setelah perubahan dilakukan

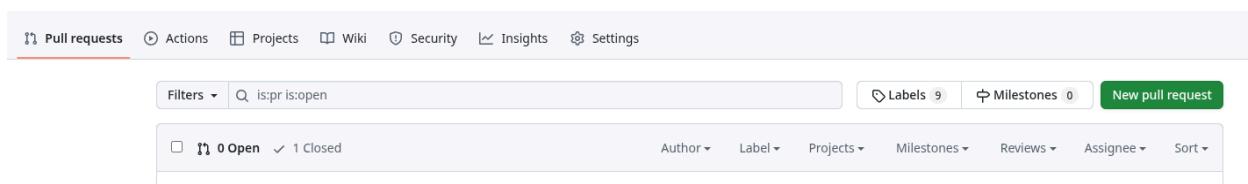
1.4.1.5. Approval & Merge Phase

Setelah fase *approval* dilakukan, tahapan yang harus dilaksanakan oleh pengembang adalah:

- Test automasi otomatis harus dijalankan
- Pemilihan strategi merge (merge commit, squash, atau rebase)
- Kode diintegrasikan ke *target branch*
- *Feature branch* di-delete untuk *cleanup*

1.4.2 Membuat Pull Request yang Efektif

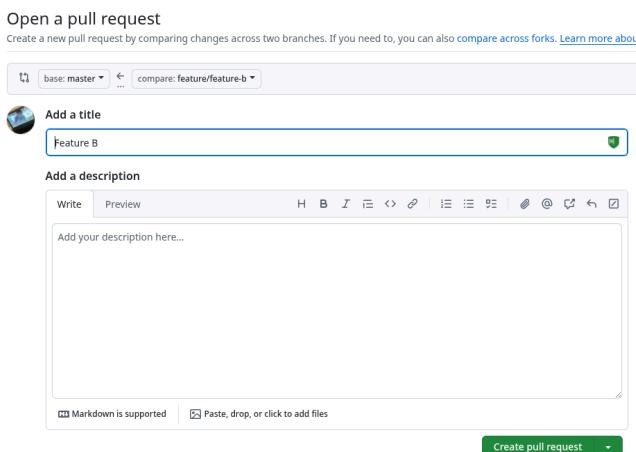
Untuk membuat *pull requests*, klik tab navigasi *pull request* lalu klik tombol hijau **New pull request**.



Gambar 1.44 Halaman *pull request*

Setelahnya akan ditampilkan halaman *Open a pull request*, pilih branch yang ingin di-merge (digabung). Lalu masukkan judul *pull requests*, deskripsi *pull requests*, *reviewers*, *assignees*, *labels*, *projects*, dan *milestone*. Untuk saat ini, cukup isi judul, dan deskripsi saja.

Dikarenakan repositori saat ini hanya berisi satu pengembang saja, untuk memilih *reviewers* rasanya tidak mungkin dilakukan. Apabila ingin melakukan *reviews* dengan pengembang lain, salah satu caranya adalah harus mengundang pengembang lain terlebih dahulu untuk bergabung di dalam repositori.



Gambar 1.45 Pembuatan *pull requests*

Dalam membuat judul Pull Requests ada karakteristik yang disarankan, yaitu:

[Tipe] Penjelasan singkat dari perubahan

Contoh:

[Feature] Add user authentication with OAuth
[Bugfix] Fix mobile responsive layout issues
[Hotfix] Resolve SQL injection vulnerability
[Refactor] Optimize database query performance
[Docs] Update API documentation with new endpoints

- **Concise:** Maksimal 50-60 karakter
- **Descriptive:** Menjelaskan apa yang diubah
- **Consistent:** Mengikuti aturan penamaan tim
- **Actionable:** Menggunakan kalimat aksi ("Add", "Fix", "Update")

1.4.2.1. Contoh Deskripsi Pull Requests

Dalam membuat deskripsi pull request, format yang digunakan adalah markdown. Berikut merupakan contoh dari deskripsi pull requests yang baik:

```
## 📝 Description
Brief summary of the changes and the problem this PR addresses.

## 🎯 Motivation and Context
Why is this change required? What problem does it solve?
Fixes #(issue number) or Relates to #(issue number)
```

```
## 🔖 Type of Change
- [ ] Bug fix (non-breaking change which fixes an issue)
- [x] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)
- [ ] Documentation update
- [ ] Performance improvement
- [ ] Code refactoring
```

Apabila di-render dalam bentuk tampilan maka bentuknya akan menjadi seperti ini:

The screenshot shows a GitHub pull request template with the following structure:

- Description**: Brief summary of the changes and the problem this PR addresses.
- Motivation and Context**: Why is this change required? What problem does it solve? Fixes #(issue number) or Relates to #(issue number)
- Type of Change**: A list of options with checkboxes:
 - Bug fix (non-breaking change which fixes an issue)
 - New feature (non-breaking change which adds functionality)
 - Breaking change (fix or feature that would cause existing functionality to not work as expected)
 - Documentation update
 - Performance improvement
 - Code refactoring

Gambar 1.46 Hasil markdown pada halaman github

1.4.3 Proses Peninjauan Kode Pull Request

Proses peninjauan kode pull request umumnya dilakukan oleh pengembang yang lebih senior atau berpengalaman, hal ini bertujuan karena pengembang tersebut memiliki tanggung jawab dalam menentukan keputusan apakah kode layak untuk di-merge (gabung) untuk proses uji coba atau perlu diperbaiki terlebih dahulu.

Secara teknis, ada standarisasi yang baik dimiliki oleh seorang pengembang:

- Kualitas kode yang mudah dipahami, dipelihara, dan mengikuti standar kode yang dibangun oleh tim.
- Algoritma yang efisien, serta alur data yang tepat dari logika bisnis.
- Pengecekan keamanan, validasi dari nilai yang dimasukkan, autentikasi dan otorisasi.
- Dampak performa seperti query dari database, penggunaan memori, serta kompleksitas kode.
- Pembuatan unit tests secara otomasi.

Dalam melakukan proses review, pengembang senior yang ditunjuk harus memenuhi aspek sebagai berikut:

- Memperhatikan perubahan kode berdasarkan kebutuhan dari logika bisnis yang dibutuhkan.
- Memberikan masukan melalui komentar terhadap kode yang perlu diperbaiki.
- Memperhatikan perubahan yang bersifat *breaking*, perubahan yang bisa menyebabkan proses saat ini berhenti.
- Pertimbangan untuk *deployment* seperti kode perubahan kolom *database*, perubahan konfigurasi, atau rencana *rollback*.

1.4.3.1 Panduan Pemberian Komentar

Pada saat mengulas *pull requests*, penting untuk mengikuti panduan sebagai berikut:

- Spesifik dan mudah dipahami: "Coba pisahkan kode ini menjadi fungsi sebuah yang bisa digunakan kembali pada kode lainnya"
- Jelaskan dengan baik: "Apabila query ini join terhadap tiga table, maka akan membuat proses pengambilan data menjadi lambat, coba pertimbangkan untuk mengurangi join atau gunakan opsi query where in"
- Memberikan solusi: "Daripada menggunakan looping dan setiap looping membuka koneksi ke database, cobalah untuk mengumpulkan semua ID ke dalam sebuah array lalu gunakan query where in"
- Puji progresi yang baik: "Kode ini sudah sangat baik, kamu melakukannya dengan benar"

1.4.4 Strategi penggabungan kode

Dalam standarisasi penggabungan kode setelah proses mengulas dilakukan, ada tiga strategi yaitu:

1.4.4.1. Merge Commit

Strategi *merge commit* adalah strategi untuk menyatukan gabungan *commit* pada sebuah branch yang sama dimana catatan setiap *commit* akan disimpan secara terpisah dan dapat diakses setiap *commit*-nya untuk melakukan penelusuran kebelakang. Kelebihan dari metode ini adalah setiap catatan *commit* akan disimpan dengan baik sehingga apabila diperlukan pengecekan akan memudahkan pengembang. Kekurangannya adalah membuat catatan menjadi kompleks dan semakin panjang.

Cara Kerja Merge Commit

Ketika menggunakan merge commit, Git akan:

1. Mempertahankan semua *commit history* dari *branch* yang dipilih.
2. Membuat *commit* baru khusus yang disebut "*merge commit*".
3. Merge commit ini memiliki dua *parent*: *commit* terakhir dari *target branch* dan *commit* terakhir dari *feature branch*
4. Semua *individual commits* dari *feature branch* tetap dapat diakses dan ditelusuri.
5. Struktur *branch history* tetap terlihat dengan jelas.

Contoh Implementasi

Sebelum Merge Commit:

```
main branch:  
* x9y8z7w Latest commit in main  
  
feature/shopping-cart branch:  
* d4e5f6g Add shopping cart model  
* g7h8i9j Implement add to cart functionality  
* j1k2l3m Add cart item validation  
* m3n4o5p Implement cart total calculation
```

Setelah Merge Commit:

```
main branch:  
* a1b2c3d Merge pull request #123 from feature/shopping-cart  
|\  
| * m3n4o5p Implement cart total calculation  
| * j1k2l3m Add cart item validation  
| * g7h8i9j Implement add to cart functionality  
| * d4e5f6g Add shopping cart model  
|/  
* x9y8z7w Latest commit in main
```

Kapan Menggunakan Merge Commit

- *Feature branches* dengan pekerjaan yang signifikan.
- Pengerjaan fitur jangka panjang yang membutuhkan detailed history.
- *Collaborative work* dimana pengembang lainnya ikut berkontribusi pada fitur yang sama.
- Proyek yang memerlukan rincian yang dilakukan sebagai bukti.

Keuntungan Merge Commit

1. **Complete History Preservation:** Semua *commit* tetap terlihat dan dapat ditelusuri.
2. **Clear Feature Boundaries:** *Merge commit* memberikan penanda bahwa ada fitur baru.
3. **Individual Attribution:** Setiap kontribusi dari pengembang tetap dapat terlihat dengan jejak waktu.
4. **Detailed Context:** Bisa melihat keseluruhan perubahan kode dari fitur yang dibangun.
5. **Easy Feature Rollback:** Dapat *rollback* keseluruhan fitur pada sejarah *commit* tertentu.
6. **Debugging Friendly:** Bisa menemukan isu lebih cepat termasuk siapa yang melakukan kesalahan.

7. **Collaboration Visibility:** Kontributor ganda dalam sebuah *branch* fitur yang sama tetap dapat dilihat.

Kekurangan Merge Commit

1. **Complex History Graph:** Representasi visual dari Git *history* menjadi lebih complicated.
2. **Noisy Commit Log:** Banyak *commits* bisa membuat garis besar fitur menjadi sulit dipahami.
3. **Merge Commit Overhead:** Tambahan commits yang belum tentu menjelaskan perubahan signifikan.
4. **Harder to Follow:** Garis dari *branch* utama menjadi tidak linear dan sulit dibaca.
5. **Tool Limitations:** Sebagian alat visualisasi Git kesulitan menampilkan grafik yang kompleks.
6. **Release Notes Complexity:** Membuat catatan rilis menjadi lebih sulit karena banyaknya *commit*.

1.4.4.2. Squash and Merge

Strategi *squash and merge* adalah strategi yang menggabungkan beberapa commit menjadi sebuah gabungan commit. Apabila dalam sebuah pull request memiliki sepuluh commit, maka ketika menggunakan strategi ini akan digabungkan menjadi sebuah commit. Kelebihannya adalah membuat jumlah *commit* tidak menjadi terlalu banyak sehingga proses pengecekan data dengan git jauh lebih cepat, sedangkan kekurangannya adalah jadi kehilangan catatan *commit* yang dilakukan setiap individu karena strategi ini menggabungkan banyak *commit* menjadi sebuah *commit* baru.

Cara Kerja Squash and Merge

Ketika menggunakan squash and merge, Git akan:

1. Mengambil semua perubahan dari *feature branch*.
2. Menggabungkan semua *commit* menjadi satu *commit* baru
3. Membuat *commit* baru tersebut di target branch
4. Menghapus *history individual commits* dari feature branch.

Contoh Implementasi

Sebelum Squash and Merge:

```
feature/user-login branch:  
* d4e5f6g Add login form HTML structure  
* g7h8i9j Implement form validation  
* j1k2l3m Add CSS styling for login form  
* m3n4o5p Fix validation error messages  
* p6q7r8s Update form accessibility  
* s9t1u2v Add unit tests for login
```

Setelah Squash and Merge:

```
main branch:  
* a1b2c3d Implement user login functionality (#123)  
  - Add login form with validation  
  - Include CSS styling and accessibility  
  - Add comprehensive unit tests  
* x9y8z7w Previous commit in main
```

Kapan Menggunakan Squash and Merge

- Pengembangan fitur dengan banyak commit di dalamnya.
- *Commit Work in progress* yang tidak menjelaskan perubahan secara individu.
- Percobaan eksperimental dengan *trial* dan *error commit*.
- Pengembang yang melakukan commit lebih banyak setiap beberapa jam sekali.

Keuntungan Squash and Merge

1. **Clean History:** *History branch* utama menjadi lebih bersih dan mudah dibaca.
2. **Faster Git Operations:** Sedikit *commit* artinya lebih cepat menjalankan operasi git log, git blame, dan operations lainnya
3. **Focused Changes:** Setiap commit pada branch utama merepresentasikan sebuah fitur yang selesai
4. **Easier Rollback:** Jika perlu rollback *feature*, cukup revert satu commit.
5. **Better Release Notes:** Lebih mudah untuk membuat catatan rilis karena sejarah *commit* yang bersih.

Kekurangan Squash and Merge

1. **Lost Development Context:** *Commit* yang dibuat oleh individu atau selama pengembangan bisa hilang.
2. **Difficult Debugging:** Tidak bisa memeriksa perubahan spesifik pada saat proses pengembangan.
3. **Attribution Issues:** Commit dari kontributor tidak terlihat pada sejarah *commit* pada *branch* utama.
4. **Large Commits:** Gabungan *commit* yang banyak bisa jadi sangat besar dan sulit untuk di-ulas oleh pengembang senior nantinya.

1.4.4.3. Rebase and Merge

Strategi *rebase and merge* adalah strategi yang memindahkan atau "memutar ulang" commit dari feature branch ke atas target branch tanpa membuat merge commit. Proses ini mengambil setiap commit dari feature branch dan menerapkannya satu per satu di atas commit terakhir dari target branch,

sehingga menghasilkan history yang linear dan bersih. Kelebihannya adalah menghasilkan history yang linear tanpa merge commit yang membuat timeline development lebih mudah diikuti, sedangkan kekurangannya adalah mengubah commit history asli dan bisa membingungkan untuk developer yang baru karena commit hash akan berubah setelah rebase.

Cara Kerja Rebase and Merge

Ketika menggunakan rebase and merge, Git akan:

1. Mengambil setiap *commit* dari *feature branch* secara individual.
2. "Memutarnya ulang" atau replay di atas *commit* terakhir target branch.
3. Membuat sebuah *commit* baru dengan hash yang berbeda tetapi content yang sama.
4. Menerapkan *commit* tersebut secara linear ke target branch.
5. Tidak membuat merge commit tambahan.

Contoh Implementasi

Sebelum Rebase and Merge:

```
main branch:  
* x9y8z7w Latest commit in main  
  
feature/user-profile branch:  
* d4e5f6g Add user profile page  
* g7h8i9j Implement profile editing  
* j1k2l3m Add profile image upload
```

Setelah Rebase and Merge:

```
main branch:  
* n1o2p3q Add profile image upload (new hash)  
* k4l5m6n Implement profile editing (new hash)  
* h7i8j9k Add user profile page (new hash)  
* x9y8z7w Latest commit in main
```

Kapan Menggunakan Rebase and Merge

- *Feature branches* yang dikelola dengan baik dan memiliki commit yang jelas.
- Sebuah tim yang memilih *track history Git* secara linear.
- *Commit* yang dilakukan secara individual akan menambahkan value.

- Proyek yang ingin melihat *history* dari *detail commit* tanpa perlu melakukan merge commit.

Keuntungan Rebase and Merge

1. **Linear History:** *Timeline* yang bersih dan kronologis tanpa metode merge commits.
2. **Preserved Individual Commits:** Setiap *commit* tetap terlihat dengan *context*-nya
3. **Better Git Bisect:** Lebih mudah untuk menemukan bug dengan git bisect karena grafik perubahan Gitnya linear.
4. **Cleaner Logs:** perintah git log --oneline menunjukkan garis progresi yang lurus.
5. **Individual Attribution:** Pengembang melakukan kontribusi dan tetap terlihat setiap *commit*-nya.

Kekurangan Rebase and Merge

1. **Changed Commit Hashes:** *Hash* dari SHAs pada *commit* bisa berubah dan menyebabkan kebingungan.
2. **Rewrite History:** Sejarah dari Git tidak akan merefleksikan apa yang sebenarnya terjadi.
3. **Complexity for Beginners:** Konsep dari rebase lebih sulit untuk dimengerti.
4. **Potential Conflicts:** Bisa saja ada konflik pada setiap penggabungan kode.

1.4.5 Memilih Strategi yang Tepat

Dalam memilih strategi yang tepat untuk *Pull Requests*, berikut merupakan panduan dalam bentuk tabel untuk ketiga strategi *pull requests*:

Skenario	Merge Commit	Squash and Merge	Rebase and Merge
Banyak commit kecil/WIP	⚠️ Menyimpan noise	✓ Ideal	✗ Menciptakan noise
Commit atomic yang terorganisir	✓ Sempurna	⚠️ Kehilangan detail	✓ Sempurna
Butuh history yang sederhana	✗ Graf kompleks	✓ Bersih	⚠️ Lebih kompleks
Atribusi individual penting	✓ Rincian lengkap	✗ Kehilangan rincian	✓ Mempertahankan
Tim baru dengan Git	⚠️ Kompleksitas sedang	✓ Lebih sederhana	✗ Lebih kompleks
Pengembangan fitur besar	✓ Menunjukkan perkembangan	✓ Satu entri	⚠️ Banyak <i>commit</i>

Fitur multi-developer	<input checked="" type="checkbox"/> Sangat baik	<input checked="" type="checkbox"/> Kehilangan kolaborasi	<input checked="" type="checkbox"/> Baik
Butuh audit trail detail	<input checked="" type="checkbox"/> Sejarah lengkap	<input checked="" type="checkbox"/> Sejarah terbatas	<input checked="" type="checkbox"/> Sejarah linear
Manajemen rilis	<input checked="" type="checkbox"/> Batas yang jelas	<input checked="" type="checkbox"/> Rilis bersih	<input checked="" type="checkbox"/> Tanpa batas
Debug masalah kompleks	<input checked="" type="checkbox"/> Konteks penuh	<input checked="" type="checkbox"/> Konteks terbatas	<input checked="" type="checkbox"/> Konteks baik
Kebutuhan compliance	<input checked="" type="checkbox"/> Bisa dilacak secara penuh	<input checked="" type="checkbox"/> Jejak terbatas	<input checked="" type="checkbox"/> Jejak termodifikasi
Kompleksitas CI/CD pipeline	<input checked="" type="checkbox"/> Lebih banyak <i>trigger</i>	<input checked="" type="checkbox"/> Lebih sedikit <i>trigger</i>	<input checked="" type="checkbox"/> Lebih banyak <i>trigger</i>

Tabel 1.1 Skenario Pull Requests

Berdasarkan matriks keputusan di atas, setiap strategi *merge* memiliki karakteristik dan keunggulan yang berbeda:

Merge Commit

Merupakan pilihan terbaik untuk **project enterprise** atau **compliance** yang membutuhkan pelacakan catatan secara lengkap dan bisa ditelusuri secara penuh. Strategi ini ideal untuk pengembangan fitur kompleks dengan pengembang ganda, dimana setiap step penggerjaan perlu didokumentasikan dengan baik. Meskipun menghasilkan sejarah yang lebih kompleks, **merge commit** memberikan konteks lengkap yang sangat baik untuk proses *debugging* dan *maintenance* jangka panjang.

Squash and Merge

Paling cocok untuk **tim yang mengutamakan kesederhanaan dan kebersihan history**. Strategi ini sangat baik untuk pengembang tunggal yang sering melakukan *commit* kecil atau percobaan, serta untuk proyek yang membutuhkan catatan rilis yang bersih. Kelemahan utamanya adalah hilangnya penggerjaan secara rinci dan kehilangan atribusi individu. Cara ini kurang cocok untuk tim yang kolaboratif atau kebutuhan audit.

Rebase and Merge

Memberikan **keseimbangan antara rincian dan kesederhanaan** dengan menghasilkan sejarah yang linear dan bersih namun tetap mempertahankan *commit* dari individu. Cara ini ideal untuk **teams** yang experienced dengan Git dan menghasilkan susunan yang tersusun rapih, atau *commit* kecil. Strategi ini sempurna untuk proyek open source atau tim yang ingin sejarah secara rinci namun tidak dipusingkan

dengan kompleksitas dari merge commit. Meskipun membutuhkan kemampuan Git lebih jauh dari para pengembang.

Rekomendasi

Dalam praktik nyata, **tidak ada satu strategi yang universal untuk semua situasi**. Sebuah tim yang sudah dewasa sering menggunakan **hybrid approach** dimana strategi dipilih berdasarkan keputusan pemimpin atau keputusan bersama. Penting untuk memastikan arahan tim pengembang dan memastikan konsistensi dalam proses pengambilan keputusan untuk menjaga alur kerja yang bisa diprediksi dan tidak membingungkan antara pengembang lainnya.

1.5 Manajemen Konflik pada Kode

Konflik pada kode adalah kondisi dimana apabila sebuah *file* pada kode programming beririsan secara penggeraan oleh seorang pengembang atau lebih. Sebagai contoh sebuah file bernama **main.php** dikerjakan oleh dua orang pengembang. Pengembang pertama mengerjakan baris 20-100, sedangkan pengembang kedua mengerjakan baris 50-120. Ketika kedua perubahan tersebut digabungkan, Git tidak dapat secara otomatis menentukan versi mana yang harus digunakan untuk baris 50-100 yang dikerjakan oleh kedua pengembang, sehingga terjadi konflik yang memerlukan intervensi secara manual.

Konflik terjadi karena Git menggunakan *line-based merging system*, dimana Git membandingkan *file line by line* untuk menentukan perbedaan antara versi yang berbeda. Ketika dua branch mengubah *lines* yang sama atau *adjacent lines* dalam file yang sama, Git tidak memiliki informasi yang cukup untuk menentukan perubahan mana yang harus dipertahankan, sehingga memerlukan pengembang untuk memperbaiki konflik tersebut secara manual.

Pemahaman yang baik tentang *conflict resolution* adalah kemampuan yang penting untuk setiap pengembang yang bekerja dalam sebuah tim, karena konflik adalah bagian normal dari kerja tim dalam pengembangan dan akan sering terjadi dalam proyek dengan kontributor yang berjumlah lebih dari satu.

Poin utamanya adalah, selama baris kode yang dikerjakan beririsan antara dua buah file, akan terjadi konflik pada baris kode yang beririsan.

1.5.1 Konflik berbasis Teks

Text-based conflicts adalah jenis konflik yang paling umum, terjadi ketika multiple developers mengubah same lines dalam file yang sama.

Contoh di bawah adalah dua *file* yang dikerjakan oleh dua orang pengembang:

```
// File: main.php - Perubahan pengembang A
```

```

<?php
function calculatePrice($basePrice, $taxRate) {
    // Calculate final price with tax
    return $basePrice + ($basePrice * $taxRate);
}

// File: main.php - Perubahan pengembang B
<?php
function calculatePrice($basePrice, $taxRate, $discount = 0) {
    // Calculate final price with tax and discount
    $discountedPrice = $basePrice * (1 - $discount);
    return $discountedPrice + ($discountedPrice * $taxRate);
}

```

Apabila *file* tersebut digabungkan pada sebuah *branch* maka akan menghasilkan konflik sebagai berikut:

```

<<<<< HEAD
<?php
function calculatePrice($basePrice, $taxRate) {
    // Calculate final price with tax
    return $basePrice + ($basePrice * $taxRate);
}
=====
<?php
function calculatePrice($basePrice, $taxRate, $discount = 0) {
    // Calculate final price with tax and discount
    $discountedPrice = $basePrice * (1 - $discount);
    return $discountedPrice + ($discountedPrice * $taxRate);
}
>>>>> feature/add-discount

```

1.5.2 Memahami Tanda Konflik

Git menggunakan penanda khusus untuk menandai bagian yang konflik pada sebuah *file*:

```

<<<<< HEAD
<?php
function calculatePrice($basePrice, $taxRate) {

```

```

    // Calculate final price with tax
    return $basePrice + ($basePrice * $taxRate);
}

=====

<?php
function calculatePrice($basePrice, $taxRate, $discount = 0) {
    // Calculate final price with tax and discount
    $discountedPrice = $basePrice * (1 - $discount);
    return $discountedPrice + ($discountedPrice * $taxRate);
}
>>>>> feature/add-discount

```

Explanation dari setiap marker:

- <<<<< HEAD: Start dari current branch changes
- =====: Separator antara conflicting versions
- >>>>> branch-name: End dari incoming branch changes

1.5.3 Skenario Konflik dengan Git Editor

Berikut adalah skenario konflik yang terjadi oleh satu orang pengembang. Sebagai contoh akan dibuat file main.php menggunakan dua baris kode di atas pada dua branch berbeda. Branch pertama diberi nama **feature/conflict**, dan branch kedua diberi nama **feature/conflict-2**.

File **main.php** pada branch feature/conflict akan diberi kode:

```

<?php
function calculatePrice($basePrice, $taxRate) {
    // Calculate final price with tax
    return $basePrice + ($basePrice * $taxRate);
}

```

Sedangkan file **main.php** pada branch feature/conflict-2 akan diberi kode:

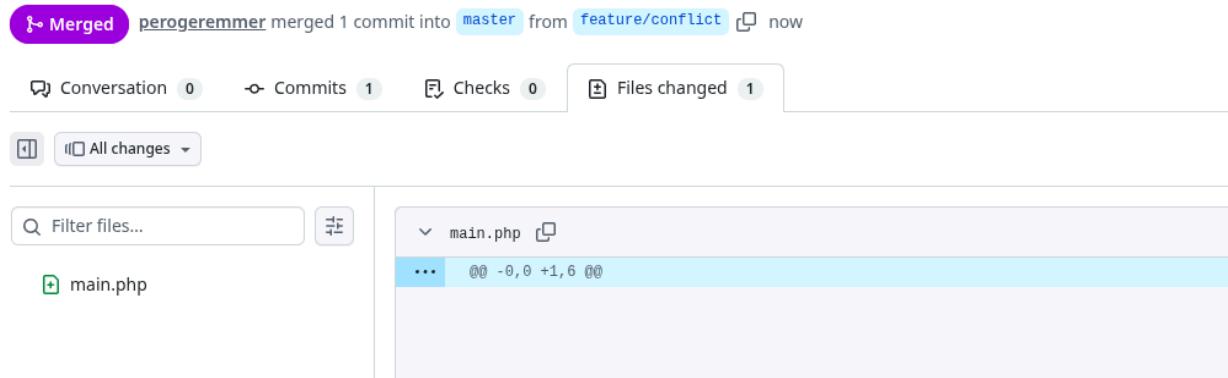
```

<?php
function calculatePrice($basePrice, $taxRate, $discount = 0) {
    // Calculate final price with tax and discount
    $discountedPrice = $basePrice * (1 - $discount);
    return $discountedPrice + ($discountedPrice * $taxRate);
}

```

Kemudian branch **feature/conflict** akan dilakukan *pull requests* dan di-merge ke branch **master**.

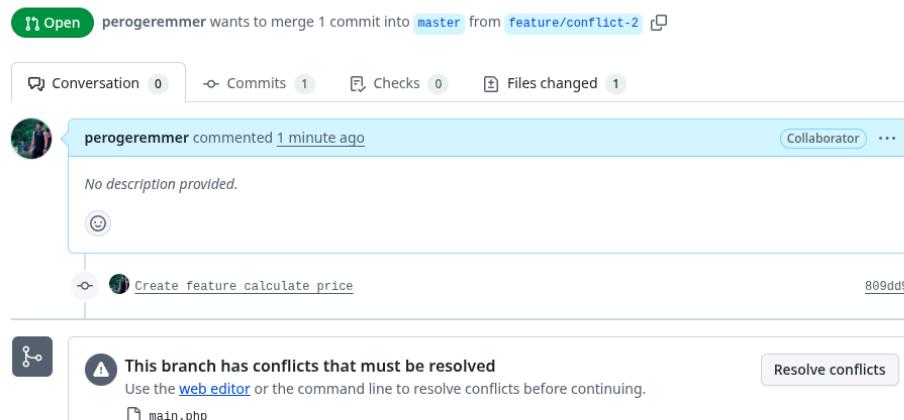
Create feature calculate price #2



Gambar 1.42 Hasil merge feature

Kode pada branch **feature/conflict** telah masuk dimana pull requests dari branch tersebut yang digabung kepada branch **master** memiliki kode PHP berisi fungsi **calculatePrice** dengan memiliki dua parameter saja, namun pada branch **feature/conflict-2** berisi tiga parameter yang tentunya beririsan baris kodennya sehingga akan menimbulkan konflik apabila **feature/conflict-2** dibuka untuk pull requests. Pesan yang akan ditampilkan pada saat branch **feature/conflict-2** dibuka adalah sebagai berikut:

Create feature calculate price #3



Gambar 1.47 Conflict file

Pada gambar di atas, apabila tombol resolve conflict ditekan, maka akan muncul ke halaman baru untuk melakukan perbaikan dari konflik.

Ina committing changes → feature/conflict...

The screenshot shows a commit message: "Ina committing changes → feature/conflict...". Below it is a code editor window titled "main.php". The code contains two conflicting blocks of PHP code. The first block, starting at line 1, is marked with a red bracket and labeled "feature/conflict-2". It includes a multi-line comment and a function definition. The second block, starting at line 9, is marked with a yellow bracket and labeled "master". It also includes a multi-line comment and a function definition. The code editor highlights the overlapping area in yellow.

```
1 <<<< feature/conflict-2
2 // File: main.php - Perubahan pengembang B
3 <?php
4 function calculatePrice($basePrice, $taxRate, $discount = 0) {
5     // Calculate final price with tax and discount
6     $discountedPrice = $basePrice * (1 - $discount);
7     return $discountedPrice + ($discountedPrice * $taxRate);
8 =====
9 // File: main.php - Perubahan pengembang A
10 <?php
11 function calculatePrice($basePrice, $taxRate) {
12     // Calculate final price with tax
13     return $basePrice + ($basePrice * $taxRate);
14 >>>> master
15 }
```

Gambar 1.48 Editor Git

Gambar di atas menunjukkan branch **feature/conflict-2** mencoba untuk digabungkan ke branch master, namun karena terdapat konflik pada baris yang bersinggungan.

Create feature calculate price #3

Resolving conflicts between feature/conflict... and master and committing changes → feature/conflict...

The screenshot shows a conflict resolution interface in VSCode. The title bar says "Create feature calculate price #3". Below it is a status bar: "Resolving conflicts between feature/conflict... and master and committing changes → feature/conflict...". The main area shows a file named "main.php" with a conflict. On the left, there's a sidebar with "1 conflicting file" and a list item "main.php". The right side shows the code with a conflict. The code is identical to the one in the previous screenshot, with the same red and yellow brackets indicating the conflict regions.

```
1 // File: main.php - Perubahan pengembang B
2 <?php
3 function calculatePrice($basePrice, $taxRate, $discount = 0) {
4     // Calculate final price with tax and discount
5     $discountedPrice = $basePrice * (1 - $discount);
6     return $discountedPrice + ($discountedPrice * $taxRate);
7 }
```

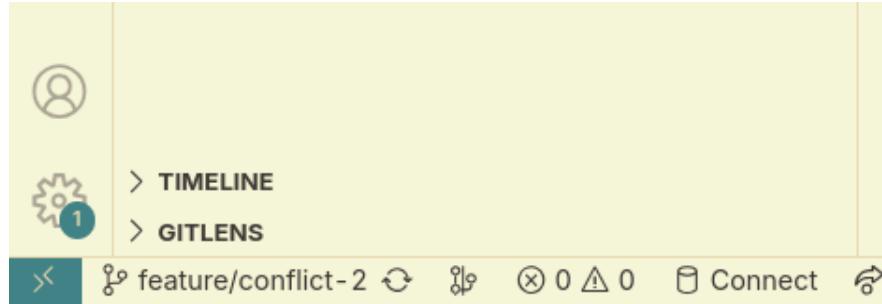
Gambar 1.49 Proses penyelesaian conflict

Untuk memperbaiki konflik, pengembang perlu memilih kode mana yang akan digabung pada dua penanda, apakah dari branch asal (head) dengan marka <<< atau mengambil baris pada branch target dengan marka >>>. Perlu diperhatikan bahwa pengambilan kode harus teliti agar tidak membuat kode berantakan dan rusak. Setelah memperbaikinya, tekan tombol pada bagian kanan yaitu **click to resolve**, tombol akan berubah menjadi warna hijau yaitu **commit merge**.

1.5.4 Skenario Konflik Manajemen dengan VSCode

Dengan VSCode manajemen konflik juga dapat dilakukan dengan tampilan yang lebih maksimal, hal ini dikarenakan VSCode sebagai IDE (Integrated Development Environment) memang didesain untuk menyelesaikan masalah konflik dengan menghadirkan tampilan kode antara satu *branch* dengan *branch* lainnya yang memiliki konflik.

Langkah pertama yang dilakukan adalah, perhatikan *branch* yang sedang aktif, karena perubahan pada *file* maupun isi filenya akan sangat signifikan.



Gambar 1.50 Menu branch pada VSCode

Secara umum *merge* artinya menggabungkan, artinya kode *branch* yang aktif akan digabungkan dari *branch* yang diinginkan. Sehingga analoginya adalah:

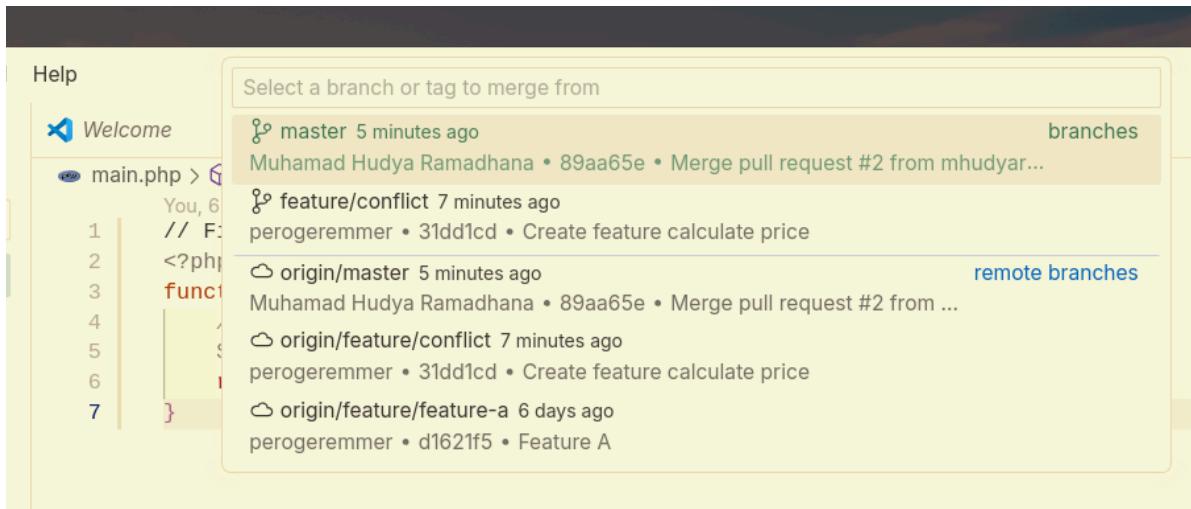
- Apabila *branch* yang aktif adalah **feature/conflict-2**, maka kode yang akan digabung adalah kode dari *branch* **master** ke *branch* **feature/conflict-2**.
- Apabila *branch* yang aktif adalah **master**, maka kode yang akan digabung adalah kode dari *branch* **feature/conflict-2** ke *branch* **master**.

Terkait mana yang lebih baik sebenarnya tidak ada, karena poin utama dari yang ingin dilakukan adalah ingin memperbaiki konflik dengan *branch* tujuan. Kemudian pilih bagian extension Git → titik tiga (opsi) → *branch* → *merge* , perhatikan gambar di bawah ini:



Gambar 1.51 Tab branch pada git

Apabila sudah berhasil nantinya akan ada teks “Select a branch or tag to merge from” artinya akan diminta kode dari *branch* lain yang akan digabungkan pada *branch* aktif. Pada studi kasus saat ini adalah, *branch* yang aktif merupakan **feature/conflict-2** dan akan digabung dari *branch* **master**.



Gambar 1.52 Daftar branch

Pada saat memilih branch target, pastikan pilih yang ada tulisan remote repository-nya, pada gambar di atas pilihlah yang **origin/master** (atau apapun nama remotenya) karena kode dari repositori online tentunya akan lebih *up to date* daripada kode yang berada di lokal. Perbedaan yang ada tulisan origin dan tidak adalah, origin merupakan remote branches atau *online*, sedangkan branches adalah local branches atau *offline*.

Setelah memilih branch **origin/master**, nantinya tampilan yang akan dilihat adalah sebagai berikut:

```

You, 1 second ago | 1 author (You) | Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<< HEAD (Current Change)
2 // File: main.php - Perubahan pengembang B
3 <?php
4 function calculatePrice($basePrice, $taxRate, $discount = 0) {
5     // Calculate final price with tax and discount
6     $discountedPrice = $basePrice * (1 - $discount);
7     return $discountedPrice + ($discountedPrice * $taxRate);
8 =====
9 // File: main.php - Perubahan pengembang A
10 <?php
11 function calculatePrice($basePrice, $taxRate) {
12     // Calculate final price with tax
13     return $basePrice + ($basePrice * $taxRate);
14 >>>>> origin/master (Incoming Change)
15 + } You, 7 minutes ago via PR #3 • Create feature calculate price ...

```

Gambar 1.53 Conflict base

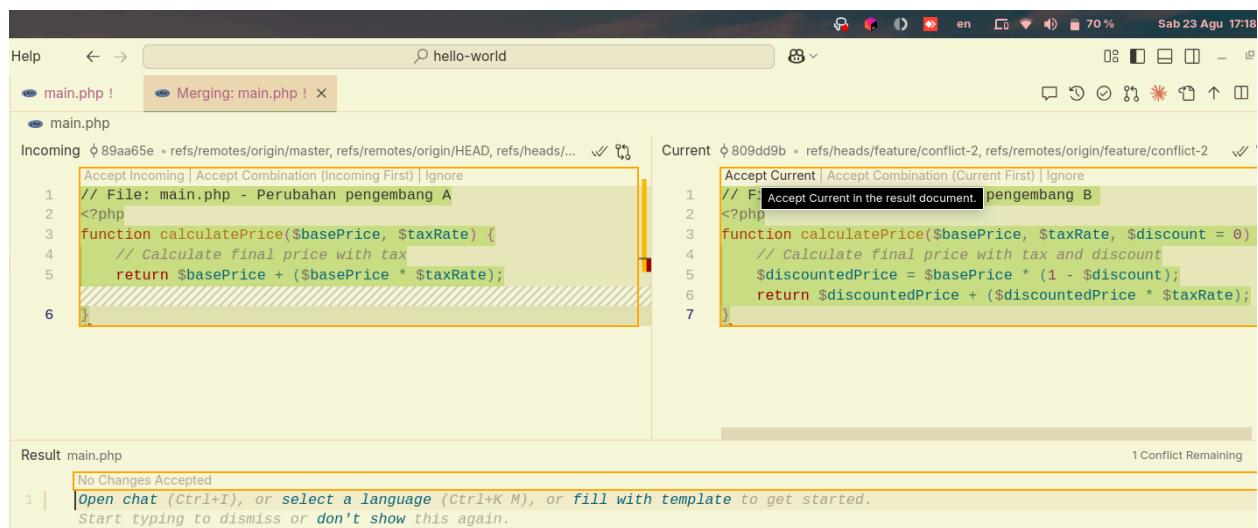
Hal ini karena extension Git dari VSCode mendeteksi adanya konflik yang harus diselesaikan. Untuk melakukan perubahan dengan cepat, pengembang dapat langsung menggantinya disana baik secara

manual seperti editor Git, atau dengan meneka teks **Accept Current Change**, **Accept Incoming Change**, **Accept Both Changes**, atau **Compare Changes** yang terlihat pada bagian atas yang agak samar.



Gambar 1.54 Notifikasi *merge conflict*

Ada sebuah tombol bernama **Resolve in Merge Editor** yang akan memberikan pilihan fleksibel untuk pengembang dalam menentukan basis kode mana yang perlu kamu ambil terlebih dahulu.



Gambar 1.55 Proses komparasi kode

Kemudian tampilan akan berubah seperti gambar di atas, pengembang bisa langsung memilih **accept current**, **accept incoming**, atau **accept all from left** melalui simbol ceklis dua.

```

Result main.php
1 Current | Remove Current
2 // File: main.php - Perubahan pengembang B
3 <?php
4 function calculatePrice($basePrice, $taxRate, $discount = 0) {
5     // Calculate final price with tax and discount
6     $discountedPrice = $basePrice * (1 - $discount);
7     return $discountedPrice + ($discountedPrice * $taxRate);
}

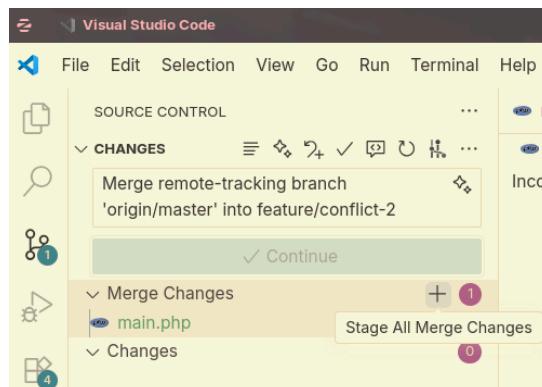
```

You, 8 minutes ago via PR #3 • Create feature calculate price ...

Gambar 1.56 Hasil merge conflict

Hasil perubahannya akan ditampilkan pada bagian bawah, apakah memilih salah satu atau keduanya, kode yang berada pada bagian bawah juga dapat diubah apabila diperlukan, hal ini disebut *combination changes*. Pastikan tulisan *conflicts remaining* pada sebelah kanan menjadi 0, agar proses *merge* bisa dilakukan.

Pada gambar di atas, kode yang dipilih adalah kode dari *branch feature/conflict-2* sehingga kodennya merupakan perubahan pengembang B.



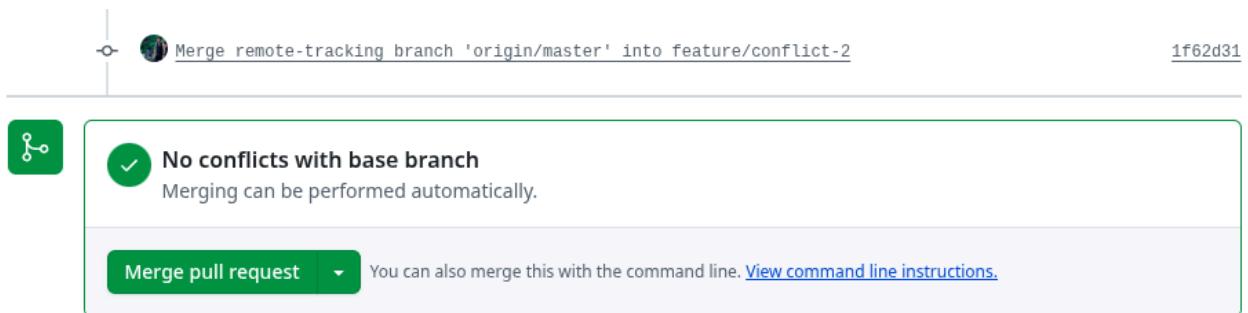
Gambar 1.57 Proses commit

Langkah selanjutnya adalah memastikan perubahan tersebut dilakukan *commit*, proses ini akan jauh lebih mudah dengan VSCode karena hanya perlu menuliskan pesan pada kolom *changes* lalu klik *stage all merge changes*.



Gambar 1.58 Proses push dengan VSCode

Terakhir klik pesan sync changes untuk melakukan push. Perubahan akan dideteksi pada halaman github.



Gambar 1.59 Notifikasi setelah conflict management

Perubahan telah dideteksi dengan commit baru yaitu **merge remote tracking branch 'origin/master' into feature/conflict-2** yang menandakan bahwa kode dari *branch origin/master* telah digabung dengan kode pada branch **feature/conflict-2**.

This screenshot shows a GitHub code diff interface. It displays a single file 'main.php' with the following content:

```

1 // File: main.php - Perubahan pengembang B
2 <?php
3 + function calculatePrice($basePrice, $taxRate, $discount = 0) {
4 +     // Calculate final price with tax and discount
5 +     $discountedPrice = $basePrice * (1 - $discount);
6 +     return $discountedPrice + ($discountedPrice * $taxRate);
7 }

```

The code is color-coded with red for deleted code and green for added code. The status bar at the top indicates '+5 -4' changes, and the bottom right shows a 'Viewed' button.

Gambar 1.60 Hasil pemilihan kode branch

Jika diperhatikan pada tab files changed ketika Pull Requests, maka bisa dilihat bahwa kode dari pengembang B berhasil menimpa kode A. Sekarang proses *merge pull request* bisa dilakukan dan kemudian kodennya akan berubah dimana branch master akan memiliki kode perubahan pengembang B.

This screenshot shows a GitHub repository interface for 'mhudyaramadhana / hello-world'. The 'Code' tab is selected. On the left, the 'Files' sidebar shows branches 'master', 'feature-b.txt', 'hello-world.txt', and 'main.php' (which is currently selected). On the right, the main area shows the contents of 'main.php' under the heading 'hello-world / main.php'. The code is as follows:

```

1 // File: main.php - Perubahan pengembang B
2 <?php
3 + function calculatePrice($basePrice, $taxRate, $discount = 0) {
4 +     // Calculate final price with tax and discount
5 +     $discountedPrice = $basePrice * (1 - $discount);
6 +     return $discountedPrice + ($discountedPrice * $taxRate);
7 }

```

The code is color-coded with red for deleted code and green for added code. The GitHub UI includes standard navigation tabs like Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

Gambar 1.61 Hasil akhir conflict

2.1 Isolasi Lingkungan Perangkat dengan Docker

Membuat isolasi lingkungan sebagai pengembang perangkat lunak merupakan salah satu prinsip manajemen proyek dalam DevOps. Hal ini bertujuan untuk memastikan konsistensi dan stabilitas dalam siklus pengembangan perangkat lunak. Dengan mengisolasi setiap lingkungan (pengembangan, pengujian, *staging*, dan *production*), pengembang dapat mencegah konflik dependensi dan masalah kompatibilitas yang mungkin timbul akibat perbedaan konfigurasi atau versi pustaka.

Isolasi lingkungan memungkinkan tim untuk menguji perubahan kode secara menyeluruh di lingkungan yang mereplikasi kondisi produksi, sehingga meminimalkan risiko kegagalan saat penerapan. Selain itu, praktik ini juga mendukung otomatisasi proses deployment dan integrasi berkelanjutan (CI/CD), yang merupakan inti dari metodologi DevOps. Dengan lingkungan yang terisolasi, proses pengujian dan deployment dapat dilakukan secara otomatis dan berulang tanpa mengganggu lingkungan lain, mempercepat waktu rilis, dan meningkatkan keandalan perangkat lunak.

2.1.1 Docker

Docker adalah platform open-source yang revolusioner dalam dunia pengembangan perangkat lunak modern. Platform ini menggunakan teknologi *containerization* untuk memungkinkan pengembang mengemas aplikasi beserta semua dependensinya ke dalam *container* yang ringan, *portable*, dan dapat dijalankan secara konsisten pada berbagai *environment*.

Konsep Dasar Docker

Docker bekerja dengan prinsip *containerization* - sebuah metode virtualisasi tingkat sistem operasi yang memungkinkan aplikasi berjalan dalam lingkungan yang terisolasi namun tetap berbagi kernel sistem operasi host. Berbeda dengan virtual machine tradisional yang membutuhkan sistem operasi lengkap, Docker container hanya memuat aplikasi dan dependensi yang diperlukan, sehingga jauh lebih efisien dalam penggunaan *resource*.

Komponen Utama Docker

Docker terdiri dari beberapa komponen kunci:

- **Docker Engine:** *Runtime core* yang menjalankan dan mengelola container
- **Docker Images:** Template *read-only* yang berisi instruksi untuk membuat container
- **Docker Containers:** *Instance* yang dapat dijalankan dari Docker image
- **Dockerfile:** File teks yang berisi perintah-perintah untuk membangun image
- **Docker Hub:** *Registry* publik untuk menyimpan dan mendistribusikan images
- **Docker Compose:** *Tool* untuk mendefinisikan dan menjalankan aplikasi multi-container

Keunggulan Docker

Portabilitas menjadi salah satu keunggulan utama Docker. Aplikasi yang dikemas dalam container dapat berjalan identik di laptop developer, server testing, hingga production server tanpa masalah "it works on my machine", sebuah kalimat yang sering diucapkan pengembang karena tidak dapat memastikan aplikasi berjalan pada lingkungan selain perangkatnya. Docker juga memungkinkan isolasi aplikasi yang lebih baik, sehingga konflik antar aplikasi dapat diminimalisir.

Skalabilitas dan efisiensi *resource* adalah nilai tambah lainnya. *Container* dapat di-scale up atau down dengan cepat sesuai kebutuhan, dan konsumsi resource yang lebih ringan dibanding *virtual machine* memungkinkan penggunaan *resource* yang lebih tinggi pada server yang sama.

Implementasi dalam Development Workflow

Docker mengubah cara tim development bekerja dengan memungkinkan standardisasi environment development. Developer dapat dengan mudah setup environment yang kompleks hanya dengan menjalankan beberapa perintah Docker. Continuous Integration/Continuous Deployment (CI/CD) pipeline juga menjadi lebih reliable karena konsistensi environment dari development hingga production.

Microservice architecture sangat terbantu dengan Docker, di mana setiap service dapat dikemas dalam container terpisah dan di-deploy secara independen. Hal ini memungkinkan tim untuk menggunakan teknologi stack yang berbeda untuk setiap service sesuai kebutuhan.

Use Case Populer

Docker banyak digunakan untuk application modernization, memindahkan legacy applications ke cloud, development environment setup, dan implementasi DevOps practices. Perusahaan-perusahaan besar menggunakan Docker untuk meningkatkan deployment speed, mengurangi infrastructure cost, dan meningkatkan developer productivity.

Platform ini telah menjadi standar industri dalam containerization dan menjadi fondasi bagi teknologi cloud-native modern seperti Kubernetes, yang mengorkestrasi container dalam skala enterprise.

2.1.1 Prasyarat Instalasi Docker

Minimum spesifikasi Docker sangat bergantung pada sistem operasi dan tujuan penggunaan, tetapi secara umum memerlukan arsitektur CPU 64-bit (x86_64 atau arm64), RAM minimal 4 GB, dan penyimpanan minimal 20 GB. Beberapa platform spesifik seperti Linux (Ubuntu, CentOS) dapat berjalan dengan Docker Engine dasar pada spesifikasi lebih rendah, sementara Docker Desktop memerlukan sistem operasi 64-bit yang lebih baru dengan dukungan virtualisasi yang diaktifkan.

Spesifikasi Minimum Umum

- **Arsitektur CPU:** 64-bit (x86_64 atau arm64).
- **RAM:** Minimal 4 GB, untuk kenyamanan, minimal 8 GB.

- **Penyimpanan:** Minimal 20 GB

Persyaratan Spesifik Berdasarkan Platform

- **Windows:**
 - **Sistem operasi:** Windows 10/11 64-bit (versi Home, Pro, Enterprise, atau Education) dengan build yang mendukung WSL 2 atau Hyper-V.
 - **Virtualisasi:** Harus diaktifkan di pengaturan BIOS/UEFI.
- **Linux:**
 - **Sistem operasi:** Diterima oleh Docker adalah Ubuntu, Debian, Fedora, CentOS, RHEL, SLES, dan lainnya.
 - **Arsitektur:** 64-bit.
- **MacOS:**
 - **Versi macOS:** 10.14 (Mojave) atau lebih baru.
 - **Perangkat Keras:** Model 2010 atau lebih baru dengan dukungan virtualisasi dari Intel.

2.1.1 Instalasi Linux pada Windows dengan WSL

Docker dapat dipasang pada sistem operasi Windows dengan beberapa cara. Metode yang paling umum adalah menggunakan **Docker Desktop**, yang menyediakan interface grafis yang user-friendly dan mudah digunakan untuk pemula. Docker Desktop memang praktis karena menyediakan GUI yang intuitif dan proses instalasi yang mudah.

Namun, penggunaan Docker Desktop pada Windows memiliki beberapa kelemahan, terutama dalam konteks pembelajaran DevOps:

Kelemahan Docker Desktop:

- **Performa yang kurang optimal** - Docker Desktop menggunakan virtualisasi tambahan yang dapat memperlambat operasi container
- **Resource overhead** - Membutuhkan resource sistem yang lebih besar karena lapisan abstraksi tambahan
- **Keterbatasan networking** - Konfigurasi network yang lebih kompleks dan terbatas dibanding environment native Linux
- **Perbedaan dengan environment production** - Kebanyakan server *production* menggunakan Linux, sehingga ada gap antara development dan production environment
- **File system performance** - Akses file antara Windows host dan container Linux relatif lambat

Meskipun penggunaan docker desktop memiliki kelemahan, ada alternatif lain untuk memasang Docker menggunakan WSL.

Keuntungan menggunakan WSL:

- **Performa native Linux** - Container berjalan langsung di kernel Linux tanpa overhead virtualisasi tambahan
- **Environment yang authentic** - Memberikan pengalaman yang sama dengan server Linux production
- **Resource efficiency** - Penggunaan memory dan CPU yang lebih efisien
- **Command line proficiency** - Terbiasa dengan Linux command line yang essential untuk DevOps
- **Networking yang lebih fleksibel** - Konfigurasi network yang lebih mudah dan powerful

Langkah pertama, adalah membuka **windows power shell** dengan akses administrator lalu jalankan perintah berikut:

```
wsl --install
wsl --set-default-version 2
```

Apabila keluar popup untuk melakukan perubahan maka silahkan tekan saja “**yes**”.

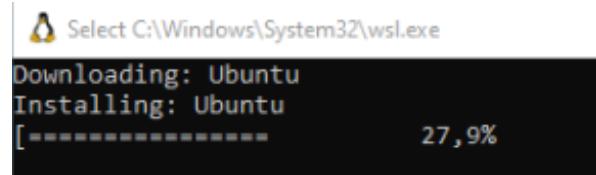


Gambar 2.1 Permission popup

```
Windows PowerShell
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Ubuntu
Ubuntu has been installed.
The requested operation is successful. Changes will not be effective until
PS C:\Users\Hudya>
```

Gambar 2.2 Proses Instalasi

Kemudian proses instalasi akan berjalan dan otomatis akan dipasang ubuntu sebagai OS untuk WSL. Ubuntu merupakan salah satu distro linux yang populer dan banyak digunakan oleh banyak pengembang. Setelah selesai cari WSL pada menu windows lalu buka aplikasinya.



Gambar 2.3 Proses Instalasi Ubuntu

Ketika aplikasi WSL dibuka, proses pengunduhan OS Ubuntu dan instalasi akan dijalankan. Nantinya akan ada permintaan pembuatan password, masukkan saja password yang mudah diingat dan ulangi prosesnya lalu. Tunggu hingga proses selesai dan otomatis akan masuk ke OS Ubuntu secara otomatis.

```
Provisioning the new WSL instance Ubuntu
This might take a while...
Create a default Unix user account: udya
New password:
Retype new password:
No password has been supplied.
New password:
Retype new password:
passwd: password updated successfully
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

udya@DESKTOP-9Q85VU9:/mnt/c/Windows/system32$
```

Gambar 2.4 Instalasi Selesai

Apabila instalasi sudah selesai, jendela WSL bisa ditutup terlebih dahulu untuk kemudian dibuka kembali untuk masuk ke *home directory*.

```
udya@DESKTOP-9Q85VU9: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

udya@DESKTOP-9Q85VU9:~$
```

Gambar 2.5 Home directory WSL

Apabila aplikasi WSL kembali dijalankan, tampilan yang akan dilihat adalah home directory dari distribusi Linux Ubuntu yang baru saja terinstal. Command prompt akan menunjukkan username dan hostname sistem, seperti **username@computername:~\$**, di mana tanda ~ menandakan bahwa posisi saat ini berada pada home directory user.

Sebelum memulai instalasi Docker, langkah pertama yang **sangat penting** adalah memperbarui semua *package repository* dan sistem Ubuntu. Hal ini diperlukan untuk memastikan bahwa semua *package* yang

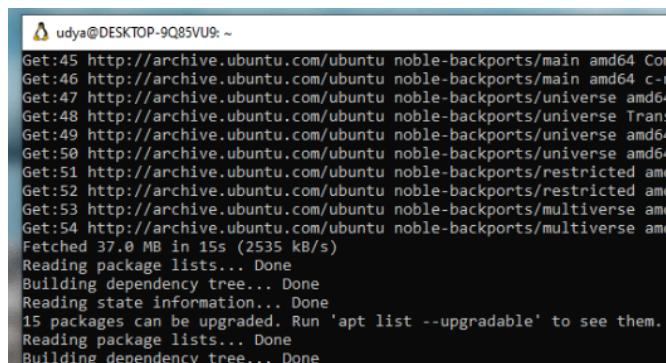
akan di-unduh adalah versi terbaru dan terhindar dari *vulnerability* keamanan. Masukkan perintah berikut:

```
sudo apt update && sudo apt upgrade -y
```

Perintah ini terdiri dari dua bagian:

- sudo apt update - memperbarui daftar package yang tersedia dari repository
- sudo apt upgrade -y - mengupgrade semua package yang sudah terinstal ke versi terbaru (flag -y otomatis menjawab "yes" untuk konfirmasi)

Proses ini mungkin memerlukan beberapa menit tergantung kecepatan internet dan jumlah package yang perlu diperbarui. Setelah proses selesai, sistem Ubuntu dalam WSL siap untuk instalasi Docker.



```
udo@DESKTOP-9Q85VU9: ~
Get:45 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 Com
Get:46 http://archive.ubuntu.com/ubuntu noble-backports/main amd64 c-n
Get:47 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64
Get:48 http://archive.ubuntu.com/ubuntu noble-backports/universe Trans
Get:49 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64
Get:50 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64
Get:51 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd
Get:52 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd
Get:53 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd
Get:54 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd
Fetched 37.0 MB in 15s (2535 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
15 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
```

Gambar 2.6 Proses memperbarui package repository

2.1.1 Instalasi Docker

WSL (Windows) & Linux

Untuk instalasi docker menggunakan WSL (Windows) atau bahkan linux, silahkan buka WSL atau terminal pada linux dan masukkan perintah di bawah ini baris perbaris:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Selanjutnya proses instalasi akan berjalan secara otomatis.

```
udy@DESKTOP-9Q85VU9:~$ curl -fsSL https://get.docker.com -o get-docker.sh
udy@DESKTOP-9Q85VU9:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 5c8855edd778525564500337f5ac4ad65a0c168e

WSL DETECTED: We recommend using Docker Desktop for Windows.
Please get Docker Desktop from https://www.docker.com/products/docker-desktop/

You may press Ctrl+C now to abort this script.
+ sleep 20
```

Gambar 2.7 Proses instalasi docker pada WSL

Langkah selanjutnya adalah menambahkan docker ke dalam user group khusus untuk mengizinkan docker mengeksekusi perintah yang membutuhkan hak akses admin, masukkan perintah berikut:

```
sudo usermod -aG docker $USER
```

Mac OS

Pengguna Mac OS dapat dengan mudah menggunakan Docker Mac Desktop yang tersedia pada website docker, secara teori memang lebih ramah untuk pemula tapi resikonya adalah *resource* yang digunakan akan lebih berat. Cara lain adalah menggunakan brew, pertama pastikan brew sudah terpasang pada terminal kemudian masukkan perintah berikut:

```
brew cask install docker
```

2.1.2 Instalasi Docker Compose

Terakhir, unduh dan pasang **Docker Compose**. **Docker Compose** adalah *tools* untuk mendefinisikan dan menjalankan aplikasi *multi-container* Docker. Menggunakan Docker Compose, pengembang dapat menggunakan file YAML untuk melakukan konfigurasi layanan pada aplikasi yang berjalan, lalu dengan satu perintah, container dapat dibuat dan menjalankan semua service tersebut.

Docker compose biasanya digunakan untuk menjalankan beberapa layanan sekaligus, misalnya aplikasi PHP yang berjalan dan membutuhkan MySQL dan Redis sebagai database maka Docker Compose dapat digunakan untuk menyatukan ketiga layanan tersebut yang dihubungkan pada sebuah jaringan *container*.

Masukkan perintah berikut untuk WSL dan Linux:

```
sudo apt-get update && sudo apt-get install docker-compose-plugin
```

Masukkan perintah berikut untuk memasang docker compose pada Mac OS:

```
brew install docker-compose
```

2.1.3 Konfigurasi Jaringan Docker

Setelah Docker berhasil dipasang, langkah selanjutnya adalah melakukan konfigurasi jaringan agar Docker dapat berjalan dengan optimal. Docker menggunakan sistem *networking* yang kompleks untuk menghubungkan container dengan host system dan container lainnya. Pada umumnya, Docker akan secara otomatis mengkonfigurasi jaringan yang diperlukan saat pertama kali dijalankan.

Namun, ketika menggunakan Docker di lingkungan WSL, terdapat beberapa konfigurasi tambahan yang perlu dilakukan karena WSL bekerja sebagai layer virtualisasi antara Windows dan Linux kernel. Hal ini berbeda dengan instalasi Docker pada sistem Linux native atau Mac OS yang umumnya tidak memerlukan konfigurasi tambahan.

Khusus untuk WSL pastikan langkah berikut dilakukan untuk menghubungkan kernel Linux WSL ke host Windows:

```
sudo update-alternatives --config iptables
```

Kemudian pilih nomor 1, tekan keyboard arah ke bawah satu kali lalu tekan tombol **enter**. Untuk linux perintah di atas tidak perlu dilakukan. Tutup terminal WSL dan buka kembali, lalu jalankan perintah berikut:

```
docker ps
```

Apabila berhasil maka akan muncul teks seperti ini:

```
hudya@perogeremmer-pc:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

Artinya docker sudah siap dijalankan pada perangkat. Kemudian masukkan kembali perintah ini:

```
docker --version  
docker-compose --version
```

Apabila semua langkah sudah benar maka teks yang muncul adalah sebagai berikut:

```
hudya@perogeremmer-pc:~$ docker-compose --version  
docker-compose version 1.29.2, build 5becea4c  
hudya@perogeremmer-pc:~$ docker --version  
Docker version 28.1.1, build 4eba377
```

2.1.4 Perintah Dasar Docker

Docker menggunakan Command Line Interface (CLI) sebagai cara utama untuk berinteraksi dengan sistem. Pemahaman yang baik terhadap perintah dasar Docker akan digunakan untuk pembuatan Dockerfile, Docker Compose, dan *orchestration*.

Perintah-perintah Docker umumnya mengikuti format **docker [COMMAND] [OPTIONS]**, dimana setiap perintah memiliki fungsi spesifik untuk mengelola *images*, *containers*, *networks*, dan *volumes*.

Informasi dan Status

Perintah	Fungsi	Contoh
docker --version	Cek versi Docker	docker --version
docker info	Informasi sistem Docker	docker info
docker ps	Lihat container yang berjalan	docker ps
docker ps -a	Lihat semua container	docker ps -a

Tabel 2.1 Perintah untuk menampilkan informasi dan status container docker

Mengelola Images

Perintah	Fungsi	Contoh
----------	--------	--------

<code>docker images</code>	Lihat daftar images	<code>docker images</code>
<code>docker pull <image></code>	Download image	<code>docker pull nginx</code>
<code>docker search <image></code>	Cari image di Docker Hub	<code>docker search ubuntu</code>
<code>docker rmi <image></code>	Hapus image	<code>docker rmi nginx</code>

Tabel 2.2 Perintah untuk mengelola images docker

Menjalankan Container

Perintah	Fungsi	Contoh
<code>docker run <image></code>	Jalankan container	<code>docker run hello-world</code>
<code>docker run -d <image></code>	Jalankan di background	<code>docker run -d nginx</code>
<code>docker run -it <image></code>	Jalankan interaktif	<code>docker run -it ubuntu</code>
<code>docker run -p <host>:<container> <image></code>	Mapping port	<code>docker run -p 8080:80 nginx</code>
<code>docker run --name <nama> <image></code>	Beri nama container	<code>docker run --name web nginx</code>

Tabel 2.3 Perintah untuk menjalankan perintah container docker

Mengelola Container

Perintah	Fungsi	Contoh
<code>docker start <container></code>	Mulai container	<code>docker start web</code>
<code>docker stop <container></code>	Hentikan container	<code>docker stop web</code>
<code>docker restart <container></code>	Restart container	<code>docker restart web</code>
<code>docker rm <container></code>	Hapus container	<code>docker rm web</code>
<code>docker logs <container></code>	Lihat log container	<code>docker logs web</code>

<code>docker exec -it <container> <command></code>	Masuk ke container	<code>docker exec -it web bash</code>
--	--------------------	---------------------------------------

Tabel 2.4 Perintah untuk mengelola container docker

Pembersihan (Cleanup)

Perintah	Fungsi	Contoh
<code>docker container prune</code>	Hapus container yang tidak aktif	<code>docker container prune</code>
<code>docker image prune</code>	Hapus images yang tidak terpakai	<code>docker image prune</code>
<code>docker system prune</code>	Hapus semua yang tidak terpakai	<code>docker system prune</code>

Tabel 2.5 Perintah untuk menghapus dan membersihkan container docker

Perintah Latihan Docker

Skenario	Perintah
Test instalasi	<code>docker run hello-world</code>
Web server sederhana	<code>docker run -d -p 8080:80 --name webserver nginx</code>
Ubuntu interaktif	<code>docker run -it --name ubuntu-test ubuntu</code>
Lihat web di browser	Buka http://localhost:8080
Stop dan hapus container	<code>docker stop webserver && docker rm webserver</code>

Tabel 2.6 Perintah untuk latihan sederhana docker

Untuk memulai percobaan cobalah untuk menjalankan menjalankan perintah **test instalasi** berdasarkan tabel 2.6, dimana perintahnya adalah:

```
docker run hello-world
```

Hasilnya adalah sebagai berikut:

```
hudya@perogeremmer-pc:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:a0dfb02aac212703bfcb339d77d47ec32c8706ff250850ecc0e19c8737b18567
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Berdasarkan contoh di atas, perintah yang dijalankan adalah menjalankan image hello-world, namun karena image hello-world belum ada maka docker otomatis mencarikannya dan mengunduhnya. Kemudian dijalankan secara otomatis dimana pesannya adalah "*Hello from Docker!*".

2.2 Container Docker PHP

Menggunakan Docker, aplikasi PHP dapat dijalankan di dalam container yang terisolasi dan konsisten. Container Docker untuk PHP menyediakan environment yang standardized, portable, dan mudah untuk di-deploy across different systems.

Keuntungan Menggunakan Container pada PHP

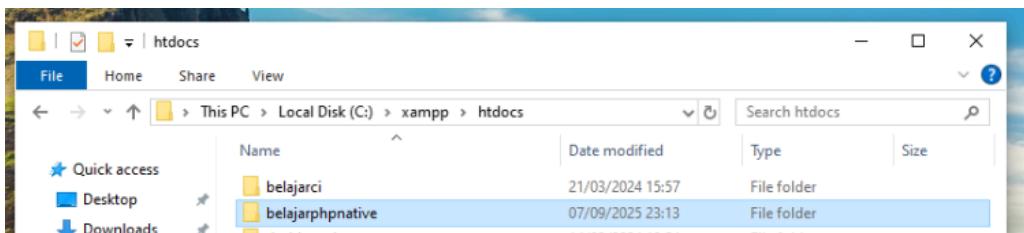
Docker container memungkinkan pengembang untuk mengemas aplikasi PHP beserta semua *dependencies*-nya (PHP runtime, extensions, libraries) ke dalam satu unit yang dapat dijalankan di mana saja. Hal ini mengatasi masalah "it works on my machine" yang sering terjadi dalam development.

Cara kerja Container untuk aplikasi PHP

Container PHP menggunakan base *image* yang sudah include PHP runtime dan web server (seperti Apache atau Nginx). Pengembang dapat menjalankan aplikasi PHP tanpa perlu menginstall PHP secara manual pada perangkat yang digunakan. Container ini bersifat *ephemeral* dan *stateless*, sehingga mudah untuk di-scale dan di-maintain.

Membuat Container di Windows

Pertama, pastikan sudah ada folder berisi aplikasi PHP. Pada panduan ini, aplikasi akan berada pada folder C:/xampp/htdocs dengan nama folder **belajarphpnative**. Apabila belum ada foldernya, silahkan dibuat terlebih dahulu.



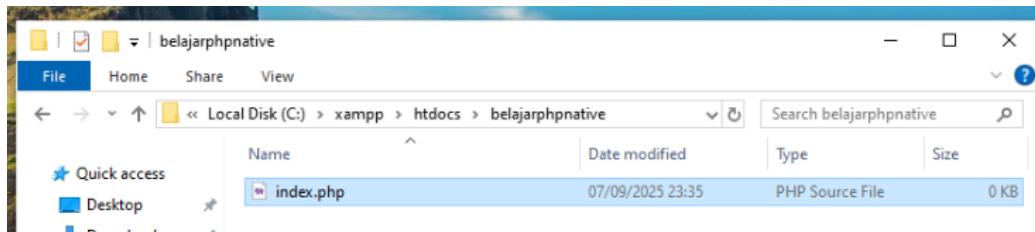
Gambar 2.8 Pembuatan folder container

Folder **belajarphpnative** akan berisi sebuah file **index.php** dengan kode:

```
<?php  
echo "Hello World!";  
?>
```

Catatan: Untuk MacOS dan Linux, silahkan buat folder belajarphpnative dimana saja karena keduanya tidak membutuhkan WSL atau container ubuntu.

Kode di atas hanya ditujukan sebagai contoh untuk menjalankan aplikasi php menggunakan container docker. Jika belum ada, coba buat terlebih dahulu baik folder dan filenya.



Gambar 2.9 Pembuatan file index.php

Sekarang buka WSL, lalu ketik perintah berikut secara baris-perbaris pada terminal WSL:

```
cd /mnt/c  
cd xampp/htdocs/  
cp -r belajarphpnative ~/  
cd ~
```

Berdasarkan kode di atas berikut penjelasannya:

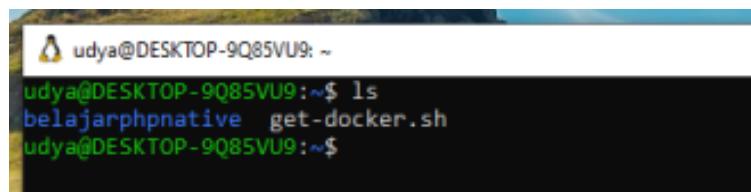
- **cd /mnt/c**: pindah ke direktori C: di windows
- **cd xampp/htdocs**: pindah ke direktori htdocs dari xampp
- **cp -r belajarphpnative ~/**: melakukan copy terhadap folder belajarphpnative ke home directory ubuntu dari WSL (~/ artinya home directory)
- **cd ~**: kembali ke home directory

Catatan: Untuk MacOS dan Linux, langkah di atas tidak perlu dilakukan, cukup buat folder belajarphpnative, buat file baru bernama index.php dan isi dengan kode php pada halaman sebelumnya.

Sekarang masukkan perintah berikut:

```
ls
```

Hasilnya adalah sebagai berikut:



Gambar 2.10 Hasil perintah ls

Hasil yang didapatkan adalah seperti gambar 2.12, ls adalah perintah untuk menampilkan semua folder pada home directory. Sekarang masuk ke directory **belajarphpnative** dengan cara:

```
cd belajarphpnative
```

Untuk menjalankan file **index.php** dengan docker, terdapat dua cara yaitu:

```
# Menjalankan PHP dengan Apache
docker run -p 0.0.0.0:8080:80 -v $(pwd):/var/www/html php:8.2-apache
```

Cara kerja: Menggunakan image `php:8.2-apache` yang sudah include PHP runtime dan Apache web server. Apache berfungsi sebagai full-featured web server yang dapat menangani multiple requests secara concurrent, serving static files, dan memiliki konfigurasi yang robust untuk production environment.

Karakteristik:

- Port mapping: 8080 (host) → 80 (container Apache default port)
- Document root: `/var/www/html` (standar Apache)
- Cocok untuk: Development yang mirip production, aplikasi dengan multiple files
- Performance: Lebih baik untuk handling concurrent requests

```
# Menjalankan PHP built-in server
docker run -p 8080:8080 -v $(pwd):/app -w /app php:8.2-cli php -S 0.0.0.0:8080
```

Cara kerja: Menggunakan image `php:8.2-cli` dan menjalankan PHP built-in server menggunakan command `php -S`. Server ini adalah lightweight development server yang built-in dalam PHP untuk rapid prototyping dan testing.

Karakteristik:

- Port mapping: 8080 (host) → 8080 (container)
- Working directory: `-w /app` (set container working directory)
- Volume mount: Files di-mount ke `/app` directory
- Cocok untuk: Quick testing, simple applications, pembelajaran

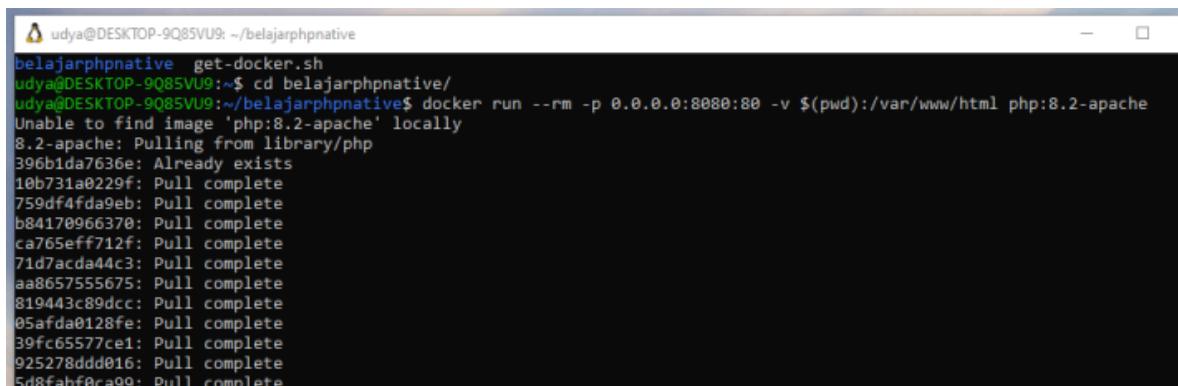
Proses Download Image

Ketika command Docker dijalankan pertama kali, Docker akan secara otomatis mengunduh *image* yang diperlukan dari Docker Hub registry. Proses ini membutuhkan koneksi internet yang stabil karena ukuran image PHP bisa mencapai ratusan MB. Setelah image berhasil diunduh, Docker akan menyimpannya secara lokal sehingga eksekusi berikutnya akan lebih cepat.

Perbedaan Utama:

- **Apache:** Full web server, production-ready, multiple file handling
- **Built-in:** Lightweight, development-only, single-threaded
- **Use case:** Apache untuk pembelajaran web server concepts, Built-in untuk rapid PHP testing

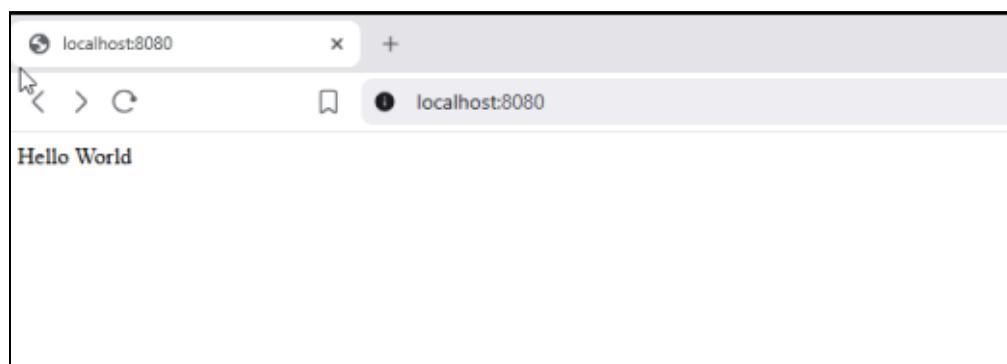
Hasilnya adalah sebagai berikut:



```
udya@DESKTOP-9Q85VU9: ~/belajarphpnative
belajarphpnative get-docker.sh
udya@DESKTOP-9Q85VU9:~/belajarphpnative/
udya@DESKTOP-9Q85VU9:~/belajarphpnative$ docker run --rm -p 0.0.0.0:8080:80 -v $(pwd):/var/www/html php:8.2-apache
Unable to find image 'php:8.2-apache' locally
8.2-apache: Pulling from library/php
396b1da7636e: Already exists
10b731a0229f: Pull complete
759df4fd9eb: Pull complete
b84170966370: Pull complete
ca765eff712f: Pull complete
71d7aca44c3: Pull complete
aa8657555675: Pull complete
819443c89dcc: Pull complete
05afda0128fe: Pull complete
39fc65577ce1: Pull complete
925278ddd016: Pull complete
Ed8fafbf0ra99: Pull complete
```

Gambar 2.11 Proses download image

Apabila sudah berhasil dijalankan, coba akses menggunakan browser ke URL localhost:8080 maka yang akan ditampilkan adalah teks “Hello World”.



Gambar 2.12 Hasil Hello World

Menjalankan Docker dengan Detachment Mode

Menggunakan *detachment mode* pada Docker memungkinkan *container* untuk berjalan di latar belakang, membebaskan terminal agar dapat digunakan untuk tugas lain. Ketika sebuah *container* dijalankan tanpa *detachment mode* (yaitu, dalam *foreground mode*), *log* dan *output* dari *container* akan langsung

ditampilkan di terminal tempat perintah docker run dieksekusi. Ini berarti terminal tersebut akan "terkunci" oleh proses *container* tersebut sampai *container* dihentikan.

Sebaliknya, dengan menggunakan *detachment mode* (ditambahkan dengan opsi -d atau --detach pada perintah docker run), Docker akan menjalankan *container* secara asinkron. Setelah *container* berhasil diluncurkan, Docker akan mengembalikan ID *container* ke terminal dan melepaskan kontrol, memungkinkan pengguna untuk terus bekerja di terminal yang sama. *Container* tersebut akan terus berjalan di latar belakang sampai secara eksplisit dihentikan, dihentikan, atau mati karena kesalahan internal.

Manfaat utama dari *detachment mode* adalah:

1. **Fleksibilitas Terminal:** Pengguna dapat menjalankan beberapa perintah Docker lainnya atau tugas-tugas lain di terminal yang sama tanpa harus membuka jendela terminal baru.
2. **Operasi Latar Belakang:** Ideal untuk menjalankan aplikasi server, basis data, atau layanan lain yang perlu berjalan terus-menerus tanpa intervensi pengguna langsung.
3. **Manajemen Log Terpisah:** Meskipun *container* berjalan di latar belakang, log dari *container* masih dapat diakses menggunakan perintah `docker logs <container_id_atau_nama>`, memungkinkan pemantauan dan *debugging* yang efisien.
4. **Otomatisasi:** Sangat penting dalam skrip otomatisasi atau *pipeline* CI/CD, di mana *container* perlu diluncurkan sebagai bagian dari alur kerja tanpa memerlukan interaksi manual.

Contoh perintahnya adalah:

```
# Menjalankan PHP built-in server
docker run -d -p 8080:8080 -v $(pwd):/app -w /app php:8.2-cli php -S 0.0.0.0:8080

# atau
docker run --detach -p 8080:8080 -v $(pwd):/app -w /app php:8.2-cli php -S
0.0.0.0:8080
```

Setelah docker dijalankan dengan mode detach, maka id dari container akan ditampilkan dan untuk melihat keseluruhan proses docker yang berjalan pada foreground (background) perintah yang diperlukan adalah:

```
docker ps
```

Maka akan ditampilkan keseluruhan container yang sedang berjalan.

```
udy@DESKTOP-9Q85VU9:~/belajarphpnative$ docker run -d --rm -p 0.0.0.0:8080:80 -v $(pwd):/var/www/html php:8.2-apache
482c1736020501e672d318ad2985546a025362141d53081ae9ababcd814f97265
udy@DESKTOP-9Q85VU9:~/belajarphpnative$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
482c17360205 php:8.2-apache "docker-php-entrypoi..." 8 seconds ago Up 6 seconds 0.0.0.0:8080->80/tcp competent_benz
Ludy@DESKTOP-9Q85VU9:~/belajarphpnative$
```

Gambar 2.13 Hasil container *detachment*

Untuk menghentikan container yang sedang berjalan, tidak perlu mengetik atau copy paste keseluruhan idnya, cukup tuliskan beberapa digit huruf depan dari container tersebut, contoh:

```
docker stop 482c
```

```
udy@DESKTOP-9Q85VU9:~/belajarphpnative$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
482c17360205 php:8.2-apache "docker-php-entrypoi..." 8 seconds ago Up
udy@DESKTOP-9Q85VU9:~/belajarphpnative$ docker stop 482c
482c
udy@DESKTOP-9Q85VU9:~/belajarphpnative$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
udy@DESKTOP-9Q85VU9:~/belajarphpnative$
```

Gambar 2.14 Hasil setelah container dihentikan

2.2.1 Membuat Images Docker dengan DockerFile

Dalam ekosistem Docker, "image" dapat diibaratkan sebagai sebuah cetak biru atau *template* yang berisi semua komponen yang diperlukan untuk menjalankan sebuah aplikasi atau layanan secara mandiri. Hal ini mencakup kode aplikasi, pustaka (libraries), dependensi sistem, alat-alat, dan konfigurasi yang spesifik.

Fungsi dan Konsep Utama Image Docker:

- **Templat Standar:** Image Docker memastikan bahwa lingkungan tempat aplikasi berjalan akan selalu konsisten, tidak peduli di mana pun aplikasi tersebut dijalankan (laptop pengembang, server produksi, atau cloud). Ini memecahkan masalah "berjalan di mesin saya tapi tidak di mesin Anda."
- **Lapisan (Layers):** Image Docker dibangun dalam bentuk lapisan-lapisan (layers) yang bersifat *read-only*. Setiap instruksi dalam Dockerfile (file yang digunakan untuk membangun image) akan membuat lapisan baru. Pendekatan berlapis ini membuat image lebih ringan dan efisien karena

lapisan yang sama dapat digunakan kembali oleh banyak image.

- **Portabilitas:** Image Docker dapat dengan mudah didistribusikan dan dibagikan melalui Docker Hub atau registri container lainnya. Ini memungkinkan pengembang untuk membangun aplikasi mereka, membuat image-nya, dan kemudian membagikannya agar dapat dijalankan di lingkungan lain dengan mudah.
- **Immutability:** Setelah sebuah image dibuat, ia tidak dapat diubah. Jika ada perubahan yang diperlukan, sebuah image baru harus dibuat. Konsep immutability ini meningkatkan keandalan dan konsistensi.
- **Dasar untuk Container:** Image adalah fondasi dari sebuah "container." Ketika sebuah image dijalankan, ia menjadi sebuah instance yang dapat dieksekusi, yang disebut container. Container adalah unit yang terisolasi dan mandiri dari aplikasi yang berjalan.

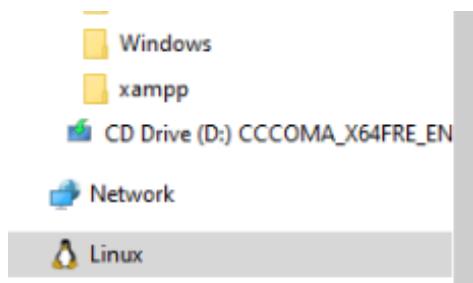
Manfaat Penggunaan Image Docker dalam Pengembangan dan Operasi (DevOps):

- **Konsistensi Lingkungan:** Mengurangi masalah perbedaan lingkungan antara pengembang, pengujian, dan produksi.
- **Isolasi Aplikasi:** Setiap aplikasi berjalan dalam container-nya sendiri, terisolasi dari aplikasi lain dan sistem host, menghindari konflik dependensi.
- **Penyebaran Cepat:** Mempercepat proses deployment dan skala aplikasi.
- **Efisiensi Sumber Daya:** Dengan berbagi kernel OS host dan menggunakan sistem berlapis, image dan container Docker relatif ringan dibandingkan mesin virtual (VM).
- **CI/CD (Continuous Integration/Continuous Delivery):** Memfasilitasi otomatisasi alur kerja CI/CD, karena image dapat dengan mudah dibangun, diuji, dan disebarluaskan secara otomatis.

Secara ringkas, image pada Docker adalah paket yang lengkap dan terstandardisasi dari sebuah layanan, yang memungkinkan aplikasi untuk dikemas bersama semua dependensinya dan dijalankan secara konsisten di berbagai lingkungan.

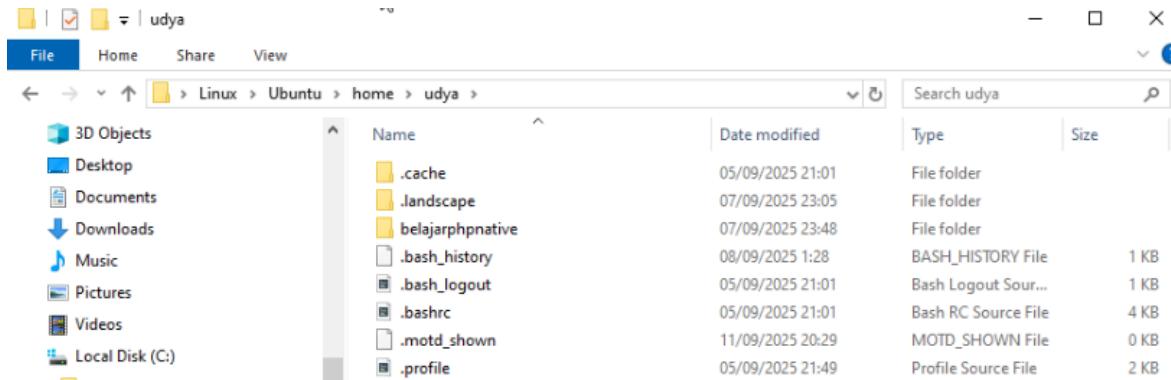
Membuat Dockerfile pada Windows

Untuk membuat Dockerfile pada Windows, caranya cukup mudah. Pertama, pergi dan buka explorer lalu cari bagian Linux pada Network di sebelah kiri bawah.



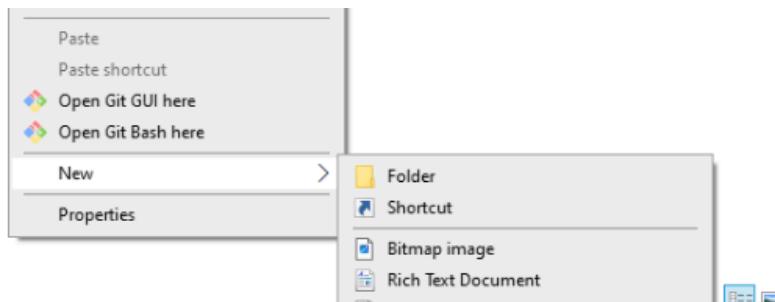
Gambar 2.15 Linux network pada explorer

Klik nama folder Ubuntu lalu klik home→user (nama user bisa berbeda), nanti akan terlihat folder utama dari user ubuntu.



Gambar 2.16 Folder linux pada explorer Windows

Buka folder belajarphnactive lalu klik kanan > new > text document, buat nama file menjadi **Dockerfile** (**tanpa extension**).



Gambar 2.17 Pembuatan folder baru

Catatan: Untuk MacOS, caranya masih sama seperti pembuatan index.php, cukup untuk membuat file Dockerfile pada folder yang sama dengan explorer MacOS.

Selanjutnya masukkan isi file berikut untuk **Dockerfile**:

```
# Menggunakan PHP 8.2 dengan Apache
FROM php:8.2-apache

# Set working directory (equivalent dengan -w flag)
WORKDIR /var/www/html

# Expose port 80 (equivalent dengan -p flag)
EXPOSE 80
```

Untuk menjalankannya, Dockerfile perlu dibuild terlebih dahulu sebagai image.

```
docker build -t my-php-app .
```

Kode di atas akan mengeksekusi file dari Dockerfile untuk dijadikan sebuah image bernama my-php-app, apabila menjalankan perintah “`docker image ls`” maka akan muncul daftar image yang sudah diunduh atau dibangun.

```
docker run -d -p 0.0.0.0:8080:80 -v $(pwd):/var/www/html my-php-app
```

Perbedaan terletak pada baris terakhir konfigurasi: alih-alih menggunakan `php:8.2-apache` seperti sebelumnya, kini diganti dengan *image* **my-php-app**. Perubahan ini memiliki tujuan strategis untuk meningkatkan konsistensi dan portabilitas layanan aplikasi PHP. Setelah membuat *image* (`my-php-app`) melalui Dockerfile, dapat dipastikan bahwa lingkungan eksekusi aplikasi PHP akan selalu sama, terlepas dari di mana aplikasi di-deploy.

Dockerfile memungkinkan pengembang untuk secara eksplisit mendefinisikan semua dependensi, konfigurasi, dan *runtime* yang dibutuhkan aplikasi, memastikan bahwa *build artifact* yang dihasilkan adalah paket mandiri yang dapat dijalankan di mana saja dengan jaminan perilaku yang sama.

Catatan: Untuk menghentikan container di atas yang sudah berjalan, gunakan `docker ps` untuk melihat id container lalu gunakan `docker stop <id>` untuk menghentikan container yang sedang berjalan.

2.3 Docker Compose Untuk Multi-Service

Docker Compose adalah tool orchestration yang memungkinkan developer untuk mendefinisikan dan menjalankan aplikasi multi-container menggunakan file konfigurasi YAML. Dengan Docker Compose, aplikasi PHP dapat digabungkan dengan service lain seperti MySQL, Redis, atau Nginx dalam sebuah jaringan yang terintegrasi dan terkoordinasi.

Konsep Multi-Service Architecture

Dalam development modern, aplikasi web tidak berdiri sendiri melainkan terdiri dari beberapa komponen terpisah yang saling berinteraksi. Aplikasi PHP membutuhkan database untuk menyimpan data, mungkin memerlukan cache server untuk performance, dan web server untuk serving content. Docker Compose memungkinkan semua service ini berjalan dalam container terpisah namun dapat berkomunikasi satu sama lain.

Keuntungan Docker Compose:

- **Service Isolation:** Setiap service (PHP, MySQL, Redis) berjalan dalam container terpisah dengan environment yang terisolasi
- **Network Integration:** Semua container dapat berkomunikasi melalui jaringan internal Docker.
- **Easy Configuration:** Satu file docker-compose.yml untuk mengatur semua layanan.
- **Environment Consistency:** Semua pengembang menggunakan stack yang sama persis
- **One-Command Deployment:** Semua services dapat dijalankan atau dihentikan dengan satu command

Contoh Implementasi PHP + MySQL:

```

version: '3.8'
services:
  php:
    image: php:8.2-apache
    ports:
      - "8080:80"
    working_dir: /var/www/html
    volumes:
      - ./:/var/www/html
    depends_on:
      - mysql
    environment:
      - DB_HOST=mysql
      - DB_NAME=myapp
      - DB_USER=root
      - DB_PASS=password

  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: myapp
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

volumes:
  mysql_data:

```

File **docker-compose.yml** di atas mendefinisikan arsitektur multi-container yang mengimplementasikan stack LAMP (Linux, Apache, MySQL, PHP) menggunakan Docker Compose versi 3.8. Konfigurasi ini memungkinkan pengembangan aplikasi web PHP dengan database MySQL dalam lingkungan yang terisolasi dan mudah direproduksi. Struktur ini terdiri dari dua service utama yang saling terhubung: service PHP dengan Apache web server dan service MySQL database server.

Service **php** menggunakan image **php:8.2-apache** yang merupakan kombinasi PHP versi 8.2 dengan Apache web server yang sudah terintegrasi. Port mapping "8080:80" mengalihkan traffic dari **port 8080** di host machine ke **port 80** di dalam container, sehingga aplikasi dapat diakses melalui **http://localhost:8080**. Working directory diset ke **/var/www/html** yang merupakan document root default Apache. Volume binding **./src:/var/www/html** memungkinkan sinkronisasi real-time antara folder root pada host dengan folder aplikasi di container, sehingga perubahan kode langsung terefleksi tanpa perlu rebuild container. Environment variables seperti **DB_HOST=mysql**, **DB_NAME=myapp**, **DB_USER=root**, dan **DB_PASS=password** menyediakan konfigurasi koneksi database yang dapat digunakan oleh aplikasi PHP.

Service **mysql** menggunakan image **mysql:8.0** dengan konfigurasi environment variables **MYSQL_ROOT_PASSWORD** dan **MYSQL_DATABASE** untuk membuat database myapp dengan password root yang telah ditentukan. Volume **mysql_data:/var/lib/mysql** memastikan persistensi data database bahkan ketika container dihapus atau di-restart, karena data disimpan di volume Docker yang terpisah dari lifecycle container. Port 3306 di-expose untuk memungkinkan akses database dari luar container jika diperlukan untuk debugging atau administrasi. Directive **depends_on** pada service **php** memastikan bahwa container MySQL akan dijalankan terlebih dahulu sebelum container PHP, menjamin urutan startup yang benar dan ketersediaan database ketika aplikasi PHP mulai berjalan.

Network Communication

Docker Compose secara otomatis membuat jaringan internal dimana semua layanan dapat berkomunikasi menggunakan layanan dengan nama yang sama sebagai hostname. Pada contoh di atas, aplikasi PHP dapat mengakses MySQL menggunakan hostname **mysql** tanpa perlu mengetahui IP address yang spesifik, hal ini disebabkan karena PHP dan MySQL berada pada sebuah jaringan yang sama.

Membuat File Docker Compose

Menggunakan Cara yang sama dengan pembuatan file **Dockerfile**, buatlah sebuah file baru bernama **docker-compose.yml** dengan kode di atas.

Command Management

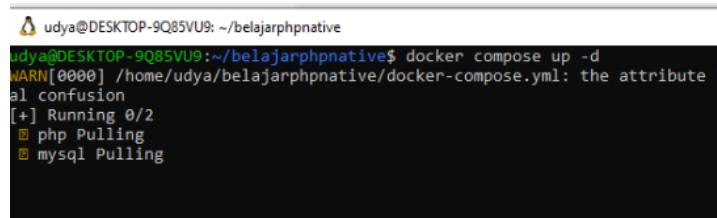
```
# Menjalankan semua services secara detachment
docker compose up -d

# Melihat status services
docker compose ps

# Melihat logs
docker compose logs

# Menghentikan semua services
docker compose down
```

Apabila perintah **docker compose** tidak bisa dijalankan pada MacOS atau linux, coba ganti dengan **docker-compose**. Saat dijalankan maka tunggu proses pengunduhan:



```
udya@DESKTOP-9Q85VU9: ~/belajarphpnative
udya@DESKTOP-9Q85VU9:~/belajarphpnative$ docker compose up -d
WARN[0000] /home/udya/belajarphpnative/docker-compose.yml: the attribute `al confusion
[+] Running 0/2
  □ php Pulling
  □ mysql Pulling
```

Gambar 2.18 Hasil docker compose up

Use Cases dalam Development:

- **LAMP Stack:** Linux + Apache + MySQL + PHP
- **Development Environment:** Consistent development setup untuk tim
- **Microservices:** Aplikasi yang terdiri dari multiple small services
- **Testing Environment:** Isolated environment untuk testing dengan database

2.3.1 Keunggulan Docker Compose

Keunggulan utama Docker Compose terletak pada kemampuannya untuk mengorkestrasikan seluruh lingkungan aplikasi. Daripada harus menjalankan setiap kontainer satu per satu dengan perintah Docker yang panjang, Docker Compose memungkinkan pengembang untuk mendefinisikan semua layanan, jaringan, dan volume dalam satu file YAML (docker-compose.yml). File ini berfungsi sebagai cetak biru yang komprehensif untuk seluruh aplikasi. Fitur-fitur kunci Docker Compose meliputi:

- **Definisi Layanan (Service Definition):** Setiap layanan (misalnya, server web, database, message queue) dapat didefinisikan dengan konfigurasi spesifiknya, termasuk citra Docker yang digunakan, port yang dipetakan, variabel lingkungan, dan dependensi antar layanan.
- **Jaringan Terisolasi (Isolated Networks):** Docker Compose secara otomatis membuat jaringan virtual untuk layanan-layanan dalam aplikasi, memastikan bahwa mereka dapat berkomunikasi satu sama lain secara aman dan terisolasi dari proses lain di host.
- **Manajemen Volume (Volume Management):** Data persisten dapat dikelola menggunakan volume, memastikan bahwa data tidak hilang saat kontainer dihentikan atau dihapus.
- **Orkestrasi Mudah (Easy Orchestration):** Dengan satu perintah (docker-compose up), semua layanan yang ditentukan dalam file YAML akan dibangun, dimulai, dan dihubungkan. Ini sangat mempercepat proses pengembangan dan deployment.
- **Lingkungan Pengembangan yang Konsisten (Consistent Development Environments):** Docker Compose memastikan bahwa setiap pengembang di tim bekerja dengan lingkungan yang identik, mengurangi masalah "berfungsi di mesin saya". Ini juga mempermudah transisi dari pengembangan ke staging dan produksi.

Dalam konteks arsitektur mikroservis, Docker Compose menjadi semakin relevan karena memungkinkan

pengembang untuk dengan mudah mengelola berbagai layanan mikro yang membentuk aplikasi secara keseluruhan. Ini memfasilitasi pengembangan, pengujian, dan debugging aplikasi yang terdistribusi. Dalam arsitektur mikroservis, Docker Compose penting untuk mengelola berbagai layanan mikro aplikasi, memfasilitasi pengembangan, pengujian, dan *debugging* terdistribusi.

2.3.2 Studi Kasus untuk PHP + MySQL + PHPMyAdmin

Untuk memahami materi Docker Compose lebih lanjut, studi kasus yang akan dibuat adalah sebuah konfigurasi untuk menghubungkan PHP, MySQL dan PHPMyAdmin. Menggunakan Docker Compose, ubah isi file **Dockerfile** yang sebelumnya ada pada folder **belajarphpnative** dengan kode berikut:

```
# Menggunakan PHP 8.2 dengan Apache
FROM php:8.2-apache

# Install MySQL extensions yang dibutuhkan
RUN docker-php-ext-install mysqli pdo pdo_mysql

# Enable Apache mod_rewrite (untuk URL rewriting)
RUN a2enmod rewrite

# Set working directory
WORKDIR /var/www/html

# Set proper permissions
RUN chown -R www-data:www-data /var/www/html \
    && chmod -R 755 /var/www/html

# Expose port 80
EXPOSE 80
```

Setelahnya ubah kembali isi file **docker-compose.yml** dengan kode berikut:

```
version: '3.8'

services:
  # PHP dengan Apache Web Server
  php:
    build:
      context: .
      dockerfile: Dockerfile
    working_dir: /var/www/html
    ports:
      - "8080:80"
```

```

volumes:
  - ./var/www/html
depends_on:
  - mysql
environment:
  - DB_HOST=mysql
  - DB_NAME=myapp
  - DB_USER=root
  - DB_PASSWORD=password
networks:
  - app-network

# MySQL Database Server
mysql:
  image: mysql:8.0
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: myapp
    MYSQL_USER: appuser
    MYSQL_PASSWORD: password
  volumes:
    - mysql_data:/var/lib/mysql
  ports:
    - "3306:3306"
  networks:
    - app-network

# phpMyAdmin untuk Database Management
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  ports:
    - "8081:80"
  environment:
    - PMA_HOST=mysql
    - MYSQL_ROOT_PASSWORD=password
  depends_on:
    - mysql
  networks:
    - app-network

# Persistent Volume untuk MySQL Data
volumes:
  mysql_data:

# Custom Network untuk Container Communication

```

```
networks:  
  app-network:  
    driver: bridge
```

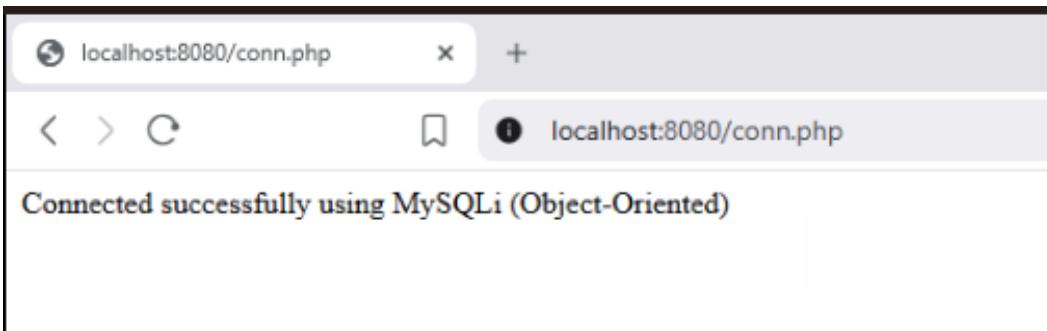
Sekarang buat file baru pada folder **belajarphpnative** dengan nama conn.php, lalu masukkan kode berikut:

```
<?php  
$servername = "mysql";  
$username = "root";  
$password = "password";  
$dbname = "myapp";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
  
// Check connection  
if ($conn->connect_error) {  
    die("Connection failed: " . $conn->connect_error);  
}  
  
echo "Connected successfully using MySQLi (Object-oriented)";  
  
$conn->close();  
?>
```

Jika container sebelumnya sedang berjalan, matikan dengan perintah **docker compose** (atau: **docker-compose**) **down**. Sesudahnya, masukkan perintah berikut untuk membangun container ulang:

```
docker compose up -d --build
```

Sekarang akses **localhost:8080** dan akan melihat pesan Hello World sama seperti sebelumnya, namun ketika mengakses **localhost:8080/conn.php** maka pesan yang akan ditampilkan adalah sebagai berikut.



Gambar 2.19 Tampilan koneksi berhasil

Artinya koneksi PHP ke MySQL sudah bisa disambungkan karena pesan yang ditampilkan *connected successfully using MySQLi* dan pada saat **Dockerfile** dibangun memang sudah memasang extension PHP.

3.1 Konfigurasi Web Server

Pada bab sebelumnya file PHP dijalankan menggunakan PHP Apache atau PHP Cli. Apache adalah salah satu web server, dan PHP CLI adalah metode untuk menjalankan server dengan server bawaan PHP. Sebelumnya, eksekusi file PHP umumnya dilakukan melalui dua metode utama: menggunakan PHP Apache atau PHP CLI (Command Line Interface).

Apache adalah web server populer yang menjembatani permintaan klien dan file PHP. Ia meneruskan permintaan ke modul PHP, yang memproses kode, menghasilkan *output* (HTML), dan mengirimkannya kembali melalui Apache ke klien. Proses ini memungkinkan aplikasi web dinamis berjalan stabil. Apache memiliki fitur manajemen permintaan HTTP, *load balancing*, dan keamanan, ideal untuk *hosting* situs web PHP kompleks.

PHP CLI memungkinkan eksekusi skrip PHP dari baris perintah, ideal untuk otomatisasi atau tugas latar belakang. PHP CLI juga menyediakan *server* bawaan untuk pengembangan lokal, yang praktis untuk pengujian cepat tanpa konfigurasi *web server* eksternal. Namun, *server* bawaan ini tidak disarankan untuk lingkungan produksi karena keterbatasan kinerja dan fitur.

Selain Apache dan PHP CLI, terdapat salah satu *web server* lain yang banyak digunakan dan populer, yaitu **Nginx**.

3.1.1 Nginx

Nginx (dibaca *engine-x*) adalah *web server open-source* yang ringan, berkinerja tinggi, dan dapat berfungsi sebagai *reverse proxy*, *load balancer*, serta *HTTP cache*. Nginx dirancang untuk menangani banyak koneksi bersamaan dengan efisien, menjadikannya pilihan ideal untuk situs web dengan lalu lintas tinggi dan aplikasi web modern.

Dibandingkan dengan Apache, Nginx memiliki arsitektur berbasis *event-driven* dan asinkron, yang memungkinkannya menangani ribuan koneksi secara bersamaan dengan penggunaan memori yang lebih rendah. Hal ini berbeda dengan Apache yang biasanya menggunakan pendekatan berbasis proses atau *thread* per koneksi, yang bisa menjadi kurang efisien pada skala besar.

Fitur-fitur utama Nginx meliputi:

- **Kinerja Tinggi:** Mampu menyajikan konten statis dengan sangat cepat dan efisien.
- **Skalabilitas:** Dirancang untuk menangani beban kerja yang tinggi dan dapat diskalakan dengan mudah.
- **Reverse Proxy:** Berfungsi sebagai perantara antara klien dan *server backend*, meningkatkan keamanan dan kinerja.
- **Load Balancing:** Mendistribusikan permintaan masuk ke beberapa *server backend* untuk

memastikan ketersediaan dan mengurangi beban pada satu *server*.

- **HTTP Cache:** Menyimpan salinan respons *server* untuk mempercepat pengiriman konten kepada pengguna yang sama di kemudian hari.
- **Dukungan Protokol:** Mendukung HTTP, HTTPS, TCP, dan UDP.

Nginx sering digunakan bersama PHP-FPM (FastCGI Process Manager) untuk menjalankan aplikasi PHP, karena Nginx tidak memiliki modul bawaan untuk mengeksekusi skrip PHP seperti Apache. Kombinasi Nginx dan PHP-FPM dianggap sebagai tumpukan yang sangat efisien untuk pengembangan web modern, terutama dalam lingkungan *microservices* dan *container* seperti Docker. Banyak perusahaan besar dan situs web populer seperti Netflix, Dropbox, dan WordPress menggunakan Nginx sebagai bagian dari infrastruktur mereka.

3.1.2 Nginx pada lingkungan *production*

Nginx merupakan salah satu web server paling populer untuk *production environment* karena *performance*, *stability*, dan *scalability* yang baik. Dalam *production*, Nginx tidak hanya berfungsi sebagai web server tetapi juga sebagai reverse proxy, load balancer, dan SSL termination point.

Architecture & Performance

Nginx menggunakan event-driven, asynchronous, dan non-blocking I/O model yang memungkinkannya menangani ribuan concurrent connections dengan resource yang minimal. Berbeda dengan Apache yang menggunakan process-based atau thread-based model, Nginx menggunakan single master process dengan multiple worker processes yang sangat efisien dalam memory usage.

Keuntungan menggunakan Nginx pada production:

- **Low Memory Footprint:** Nginx menggunakan sekitar 2.5MB RAM per worker process
- **High Concurrency:** Mampu handle 10,000+ concurrent connections (C10K problem solver)
- **Fast Static Content:** Zero-copy untuk static files dengan sendfile() system call
- **Efficient Caching:** Built-in proxy caching untuk dynamic content

3.1.2.1 Nginx sebagai static server

Static content adalah file-file yang tidak berubah secara dinamis dan dapat disajikan langsung oleh web server tanpa memerlukan processing tambahan. Content ini bersifat tetap dan identik untuk semua user yang mengaksesnya.

Jenis-jenis Static Content:

- **HTML Files:** Halaman web statis yang sudah jadi

- **CSS Stylesheets:** File styling untuk tampilan web
- **JavaScript Files:** Script client-side untuk interaktivitas
- **Images:** JPG, PNG, GIF, SVG, WebP
- **Media Files:** Video, audio, PDF documents
- **Fonts:** WOFF, WOFF2, TTF untuk typography
- **Icons:** Favicon, app icons

Karakteristik Static Content Serving

Web server seperti Nginx sangat optimal untuk menangani static content karena prosesnya straightforward - server hanya perlu membaca file dari disk dan mengirimkannya ke client tanpa processing. Ini berbeda dengan dynamic content yang memerlukan execution, database queries, atau computation.

Keunggulan Static Content:

- **Performance Tinggi:** Tidak ada processing overhead
- **Caching Friendly:** Content dapat di-cache dengan mudah
- **Bandwidth Efficient:** Dapat dikompresi dengan gzip/brotli
- **CDN Compatible:** Mudah didistribusi via Content Delivery Network
- **Resource Light:** Minimal CPU dan memory usage

Content Delivery Optimization

Nginx menggunakan beberapa teknik optimasi untuk static content serving. **Sendfile** system call memungkinkan transfer data langsung dari disk ke network socket tanpa melalui user space, mengurangi memory copy operations. **Zero-copy** mechanism ini sangat efisien untuk file serving.

Browser caching menjadi crucial untuk static content. Server mengirim HTTP headers seperti **Cache-Control**, **Expires**, dan **ETag** untuk memberitahu browser berapa lama content bisa di-cache. Ini mengurangi bandwidth usage dan meningkatkan user experience karena content loading lebih cepat.

3.1.2.1 Nginx sebagai reverse proxy + load balancer

Ketika nginx digunakan sebagai reverse proxy dikombinasikan dengan load balancing, ia menjadi *intelligent traffic distributor* yang tidak hanya meneruskan *requests*, tetapi juga memutuskan **ke backend mana request tersebut akan dikirim**.

Traditional Single Backend:

Client → Reverse Proxy → Backend Server

Load Balanced Reverse Proxy:

Client → Reverse Proxy → Backend Server 1
→ Backend Server 2
→ Backend Server 3

Kombinasi ini sangat powerful dikarenakan sebuah *backend server* memiliki keterbatasan kapasitas trafik. Ketika trafik meningkat, sebuah server bisa kepuasan *request* yang menyebabkan respon time melambat atau server *down*. Beberapa server *backend* dengan distribusi trafik menggunakan Nginx menjadi *decision maker* yang menentukan server mana yang paling appropriate untuk menangani setiap *request* yang masuk.

3.1.2.2 Algoritma Load Balancing

1. Round Robin

Konsep: *Requests* didistribusikan secara berurutan ke setiap backend server.

Cara Kerja:

- Request 1 → Server A
- Request 2 → Server B
- Request 3 → Server C
- Request 4 → Server A (kembali ke awal)

Keunggulan: Mudah dipahami dan pengoperasian yang sederhana

Kelemahan: Tidak mempertimbangkan beban server.

Use Case: Ketika semua server memiliki spesifikasi yang sama dan beban yang bisa diprediksi.

2. Least Connections

Konsep: *Request* dikirim ke server yang sedang menangani **paling sedikit active connections**.

Cara Kerja:

- Server A: 5 active connections
- Server B: 3 active connections
- Server C: 7 active connections
- New request → dikirim ke Server B

Keunggulan: Lebih baik untuk requests yang membutuhkan *request time* lebih panjang.

Kelemahan: Butuh monitoring jumlah aktif koneksi, cukup sulit.

Use Case: Aplikasi dengan variable *request processing times*, seperti *database queries* atau *file uploads*.

3. Weighted Distribution

Konsep: Server diberi "beban" berdasarkan kapasitasnya.

Cara Kerja:

- Server A (weight: 3) → dapat 3x lebih banyak requests
- Server B (weight: 2) → dapat 2x lebih banyak requests
- Server C (weight: 1) → dapat 1x requests

Keunggulan: Optimal untuk infrastruktur yang berbeda spesifikasi antar server

Kelemahan: Perlu konfigurasi manual untuk hasil optimal

Use Case: Mixed hardware environment dimana beberapa servers lebih powerful dari yang lain.

4. IP Hash

Konsep: Client IP address di-hash untuk menentukan backend server.

Cara Kerja:

- User dengan IP 192.168.1.10 selalu ke Server A
- User dengan IP 192.168.1.15 selalu ke Server B
- Same IP = same server (session affinity)

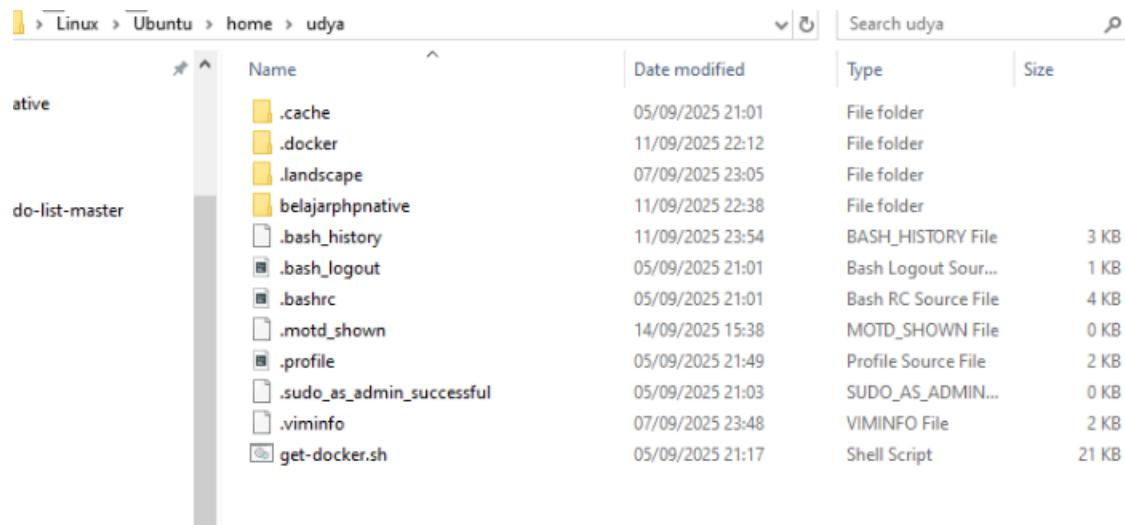
Keunggulan: Pembagian sesi berdasarkan user

Kelemahan: Bisa menyebabkan masalah pada user yang tidak diset ipnya.

Use Case: Aplikasi yang membutuhkan session persistence atau caching per-user.

3.2 Studi Kasus Konfigurasi Nginx

Untuk memulai latihan studi kasus Nginx, buatlah folder baru pada directory user WSL Ubuntu / Mac OS dengan nama **belajarnginx**.



Name	Date modified	Type	Size
.cache	05/09/2025 21:01	File folder	
.docker	11/09/2025 22:12	File folder	
.landscape	07/09/2025 23:05	File folder	
belajarnginx	11/09/2025 22:38	File folder	
.bash_history	11/09/2025 23:54	BASH_HISTORY File	3 KB
.bash_logout	05/09/2025 21:01	Bash Logout Sour...	1 KB
.bashrc	05/09/2025 21:01	Bash RC Source File	4 KB
.motd_shown	14/09/2025 15:38	MOTD_SHOWN File	0 KB
.profile	05/09/2025 21:49	Profile Source File	2 KB
.sudo_as_admin_successful	05/09/2025 21:03	SUDO_AS_ADMIN...	0 KB
.viminfo	07/09/2025 23:48	VIMINFO File	2 KB
get-docker.sh	05/09/2025 21:17	Shell Script	21 KB

Gambar 3.1 Pembuatan folder belajarnginx

Setelah folder dibuat, buatlah sebuah file **default.conf**, **Dockerfile**, **index.html**, dan **docker-compose.yml** di dalam folder **belajarnginx**. Berikut isi file **default.conf**:

```

server {
    listen 80 default_server;
    server_name localhost;

    root /var/www/belajarnginx;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}

```

Kode di atas merupakan contoh konfigurasi server block (virtual host) dasar pada Nginx yang berfungsi sebagai web server sederhana. Server block ini dikonfigurasi untuk mendengarkan pada port 80 dengan directive `listen 80 default_server`, dimana `default_server` menandakan bahwa konfigurasi ini akan menjadi server default yang menangani semua request yang masuk ke port 80 jika tidak ada server block lain yang cocok. Parameter `server_name` localhost mendefinisikan nama domain yang akan ditangani oleh server block ini, dalam hal ini adalah localhost yang biasanya digunakan untuk pengembangan lokal.

Directory root untuk website didefinisikan melalui directive `root /var/www/belajarnginx`, yang berarti semua file website akan disimpan dan diakses dari direktori `/var/www/belajarnginx` di dalam sistem file. Ketika user mengakses website tanpa menyebutkan file tertentu, Nginx akan mencari file `index.html` sebagai halaman default berdasarkan konfigurasi `index index.html`.

Bagian location block `location /` mengatur bagaimana Nginx menangani semua request yang masuk. Directive `try_files $uri $uri/ =404` memberikan instruksi kepada Nginx untuk mencoba mencari file sesuai dengan URI yang diminta (`$uri`), jika tidak ditemukan maka akan mencoba mencari sebagai direktori dengan menambahkan slash (`$uri/`), dan jika kedua opsi tersebut gagal maka akan mengembalikan error 404 (file tidak ditemukan). Konfigurasi ini sangat cocok untuk website statis sederhana dan merupakan dasar yang penting untuk dipahami sebelum mempelajari konfigurasi Nginx yang lebih kompleks.

Berikut isi file **Dockerfile**:

```

# Dockerfile
FROM nginx

# Copy file konfigurasi default nginx dengan yang baru
COPY ./default.conf /etc/nginx/conf.d/default.conf

# Buat direktori dan copy file HTML

```

```

RUN mkdir -p /var/www/belajarnginx

# Salin file index.html ke belajarnginx
COPY index.html /var/www/belajarnginx/

# Ubah permission untuk folder
RUN chown -R www-data:www-data /var/www/belajarnginx
RUN chmod -R 775 /var/www/belajarnginx

# Expose port 80
EXPOSE 80

# Command untuk menjalankan nginx
CMD [ "nginx", "-g", "daemon off;" ]

```

Dockerfile di atas menjelaskan cara membuat container image kustom berbasis Nginx untuk melayani website statis. Proses dimulai dengan menggunakan **FROM nginx** sebagai base image, yang berarti menggunakan official Nginx image dari Docker Hub sebagai fondasi container. Image ini sudah memiliki Nginx yang siap digunakan dengan konfigurasi default, sehingga hanya perlu melakukan kustomisasi sesuai kebutuhan.

Langkah selanjutnya adalah menyalin konfigurasi server block kustom dengan perintah **COPY ./default.conf /etc/nginx/conf.d/default.conf**, yang akan menggantikan konfigurasi default Nginx dengan konfigurasi yang telah dibuat sebelumnya. Setelah itu, direktori **/var/www/belajarnginx** dibuat menggunakan **RUN mkdir -p** untuk menyimpan file-file website, dan file **index.html** disalin ke dalam direktori tersebut menggunakan instruksi **COPY index.html /var/www/belajarnginx/**.

Aspek keamanan dan permission menjadi penting dalam container, sehingga ownership direktori website diubah ke user **www-data** dengan perintah **chown -R www-data:www-data**, di mana **www-data** adalah user default yang digunakan oleh Nginx untuk menjalankan worker processes. Hak akses direktori juga diatur menjadi 775 menggunakan **chmod -R 775**, yang memberikan akses read, write, dan execute untuk owner dan group, serta read dan execute untuk others. Port 80 di-expose menggunakan **EXPOSE 80** untuk memberi tahu Docker bahwa container akan menerima koneksi pada port tersebut. Terakhir, **CMD ["nginx", "-g", "daemon off;"]** menjalankan Nginx dalam mode foreground (non-daemon) yang diperlukan agar container tetap berjalan, karena jika Nginx berjalan sebagai daemon, container akan langsung terminate setelah proses utama selesai.

Berikut isi file **index.html**:

```

<!DOCTYPE html>
<html>
<head>

```

```
<title>Docker Nginx Demo</title>
<style>
    body { font-family: Arial; text-align: center; margin-top: 100px; }
    h1 { color: #4CAF50; }
</style>
</head>
<body>
    <h1>Hello World!</h1>
    <p>Nginx berhasil berjalan di Docker Container</p>
    <p>Port: 80</p>
</body>
</html>
```

Terakhir, isi file **docker-compose.yml**:

```
version: '3.8'

services:
  nginx:
    build: .
    ports:
      - "80:80"
```

Sekarang jalankan perintah berikut untuk melihat hasilnya.

```
docker compose up -d --build
```

Akses **localhost** pada browser, pesan **Nginx berhasil berjalan di Docker Container** akan ditampilkan.

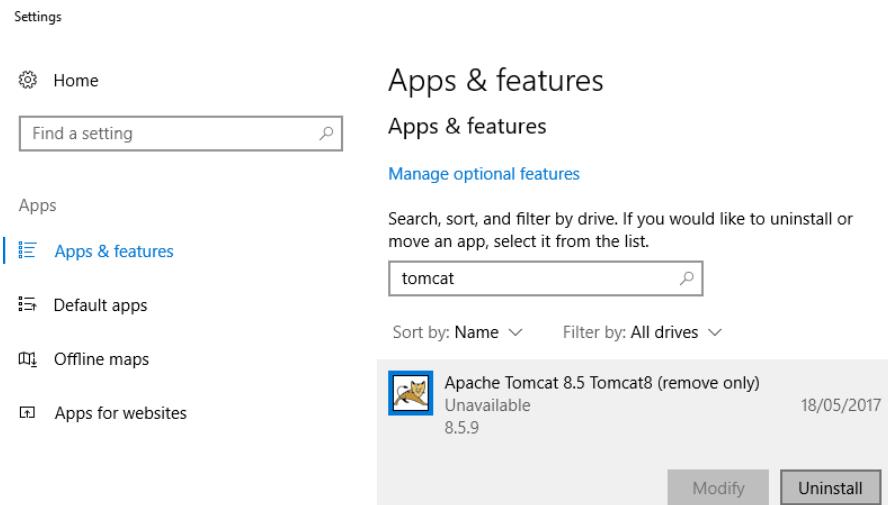


Hello World!

Nginx berhasil berjalan di Docker Container
Port: 80

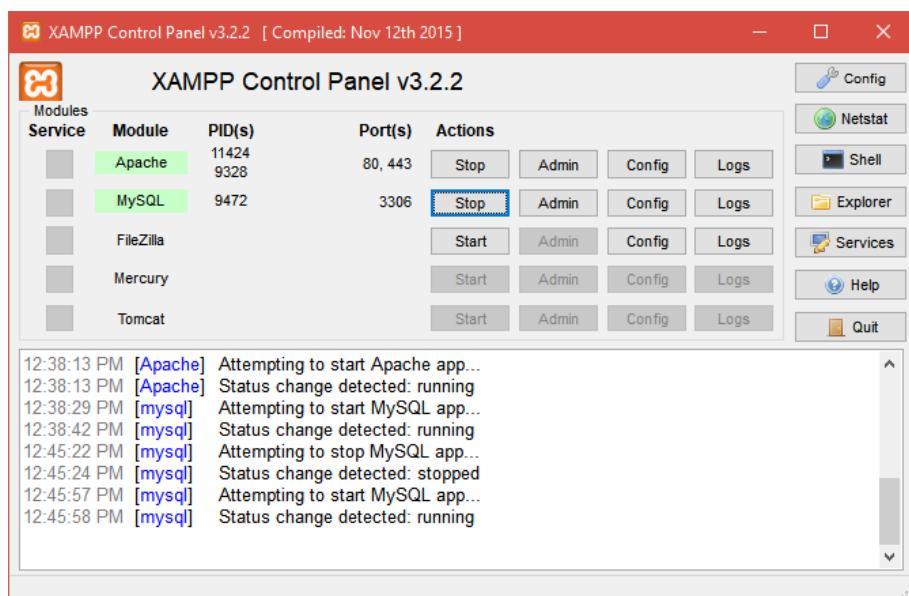
Gambar 3.2 Hasil Nginx Container

Ada kemungkinan terjadi *cache* pada browser, oleh karena itu untuk menguji coba nginx maka gunakan *incognito mode*. Pada sistem operasi Windows, ada kemungkinan juga port 80 tidak bisa dijalankan, hal ini kemungkinan Apache sudah terinstall pada WIndows atau XAMPP yang masih berjalan. Untuk mengatasi hal ini, silahkan *uninstall* aplikasi terkait yang berjalan pada port-port tertentu seperti Apache, atau XAMPP.



Gambar 3.3 Aplikasi Windows Apache yang sudah terpasang

Sedangkan untuk XAMPP, pastikan bahwa program tidak berjalan. Pastikan prosesnya sudah stop karena biasanya XAMPP memblokir port 80 dan 443, serta 3306.



Gambar 3.4 XAMPP Control Panel

3.3 Studi Kasus Nginx + PHP + MySQL

Dalam implementasi Docker Compose, Nginx dapat berperan sebagai *reverse proxy* yang bekerja dengan PHP-FPM (FastCGI Process Manager) dan database MySQL. Konfigurasi ini memungkinkan Nginx untuk mengelola semua permintaan HTTP yang masuk, lalu meneruskannya ke kontainer PHP-FPM untuk diproses.

Ketika permintaan web tiba di Nginx, ia akan memeriksa *request header* dan URL yang diminta. Jika permintaan tersebut memerlukan pemrosesan PHP, Nginx tidak akan memprosesnya secara langsung. Sebaliknya, Nginx akan bertindak sebagai "**gerbang**" yang meneruskan permintaan tersebut ke PHP-FPM melalui protokol FastCGI.

PHP-FPM (FastCGI Process Manager) adalah implementasi FastCGI untuk PHP yang dirancang untuk beban tinggi, berjalan terpisah dari *web server*. PHP-FPM, yang berjalan dalam kontainernya sendiri, bertanggung jawab untuk mengeksekusi *script* PHP dan menghasilkan *output*. Setelah PHP-FPM selesai memproses, ia akan mengirimkan *output* kembali ke Nginx, yang kemudian akan meneruskan *output* tersebut ke *browser* pengguna.

Manfaat utama dari arsitektur ini dengan Docker Compose adalah:

- **Isolasi Lingkungan:** Setiap komponen (Nginx, PHP-FPM, MySQL) berjalan dalam kontainernya sendiri, terisolasi satu sama lain. Ini mencegah konflik dependensi dan memudahkan pengelolaan versi.
- **Skalabilitas:** Masing-masing layanan dapat diskalakan secara independen. Jika beban pada PHP-FPM tinggi, Anda dapat dengan mudah menambahkan lebih banyak *instance* kontainer PHP-FPM tanpa memengaruhi Nginx atau MySQL.
- **Portabilitas:** Lingkungan pengembangan dan produksi dapat dicocokkan dengan sempurna karena semua konfigurasi didefinisikan dalam file docker-compose.yml, memastikan aplikasi berperilaku konsisten di mana pun ia dijalankan.
- **Manajemen Sumber Daya yang Efisien:** Sumber daya CPU dan memori dapat dialokasikan secara granular ke setiap layanan sesuai kebutuhannya.
- **Kemudahan Pengembangan:** Pengembang dapat dengan cepat menyiapkan lingkungan kerja lokal yang mencerminkan produksi hanya dengan menjalankan perintah docker-compose up.

Dengan demikian, kombinasi Docker Compose, Nginx, PHP-FPM, dan MySQL menyediakan tumpukan aplikasi web yang kokoh, fleksibel, dan mudah untuk dikelola.

Pertama, salin folder **belajarphpnative** menjadi **belajarphpnginx** pada sub bab 2.1.9. Kemudian masukkan satu persatu kode konfigurasi di bawah ini untuk menggabungkan antara Nginx, PHP - FPM, MySQL, dan PHPMyAdmin.

	Name	Date modified	Type	Size
ix	.cache	05/09/2025 21:01	File folder	
native	.docker	11/09/2025 22:12	File folder	
odo-list-master	.landscape	07/09/2025 23:05	File folder	
5	belajarnginx	14/09/2025 17:09	File folder	
i	belajarphpnative	11/09/2025 22:38	File folder	
	belajarphnginx	16/09/2025 20:44	File folder	
	.bash_history	14/09/2025 18:12	BASH_HISTORY File	3
	.bash_logout	05/09/2025 21:01	Bash Logout Sour...	1
	.bashrc	05/09/2025 21:01	Bash RC Source File	4
	.motd_shown	14/09/2025 15:38	MOTD_SHOWN File	0
	.profile	05/09/2025 21:49	Profile Source File	2
	.sudo_as_admin_successful	05/09/2025 21:03	SUDO_AS_ADMIN...	0
	~	07/09/2025 22:48	UNKNOWN File	

Gambar 3.5 Pembuatan folder **belajarphnginx**

Buat file bernama **Dockerfile.nginx** di dalam folder **belajarphnginx** dengan kode berikut:

```
# Custom Nginx Dockerfile
FROM nginx

COPY ./default.conf /etc/nginx/conf.d/default.conf

# Expose port 80
EXPOSE 80

# Command untuk menjalankan nginx
CMD [ "nginx", "-g", "daemon off;" ]
```

Dockerfile.nginx adalah custom build configuration untuk membuat Nginx container yang mengikuti struktur professional server management. File ini menggunakan base image nginx kemudian menyalin konfigurasi default bawaan nginx dengan konfigurasi baru yang disiapkan.

Selanjutnya, ubah isi file **Dockerfile** di dalam folder **belajarphnginx** dengan kode berikut:

```
# PHP-FPM untuk Nginx
FROM php:8.2-fpm

# Install MySQL extensions
RUN docker-php-ext-install mysqli pdo pdo_mysql

# Set working directory
```

```

WORKDIR /var/www/myphpapp

# Set proper permissions
RUN chown -R www-data:www-data /var/www/myphpapp

# Expose port 9000 untuk PHP-FPM
EXPOSE 9000

```

Dockerfile di atas adalah PHP-FPM container yang dirancang untuk bekerja dengan Nginx sebagai reverse proxy. Base image php:8.2-fpm dipilih karena menggunakan FastCGI Process Manager (FPM) yang merupakan implementation PHP yang optimal untuk production environments, berbeda dengan mod_php yang embedded dalam Apache. PHP-FPM menjalankan PHP sebagai separate processes yang berkomunikasi dengan web server melalui FastCGI protocol, memberikan isolasi yang lebih baik, stabilitas, dan performa yang lebih baik karena web server dan PHP processor dapat di-scale secara independen.

Instalasi MySQL extensions (mysqli, pdo, pdo_mysql) dilakukan menggunakan docker-php-ext-install yang merupakan helper script khusus dari official PHP images untuk mengcompile dan menginstall extensions dengan konfigurasi yang baik. *Working directory* di-set ke /var/www/myphpapp untuk consistency dengan Nginx configuration, dan ownership serta permissions diatur ke www-data:www-data yang merupakan default user untuk web services di Linux systems. Port 9000 yang di-expose adalah standard FastCGI port dimana PHP-FPM process akan *listen* untuk menerima requests dari Nginx, komunikas via port ini membuat komunikasi internal yang aman tanpa perlu *expose* ke external network.

Kemudian, berikut isi file **default.conf**:

```

server {
    listen 80;
    root /var/www/myphpapp;
    index index.php index.html;

    access_log /var/log/nginx/myphpapp_access.log;
    error_log /var/log/nginx/myphpapp_error.log;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass php:9000;
        fastcgi_index index.php;
    }
}

```

```

    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}
}

```

File **default.conf** adalah site-specific configuration yang mendefinisikan bagaimana Nginx menangani requests untuk aplikasi kita. Configuration ini mengatur document root ke /var/www/myphpapp, mendefinisikan index files yang akan dicari, dan yang paling penting adalah konfigurasi FastCGI untuk komunikasi dengan PHP-FPM container. Location block ~ \.php\$ menggunakan regular expression untuk menangkap semua PHP files dan meneruskannya ke php:9000 (service name dan port PHP-FPM).

Terakhir, ubah isi file dari **docker-compose.yml**:

```

version: '3.8'

services:
  # Nginx Web Server (Custom Build)
  nginx:
    build:
      context: .
      dockerfile: Dockerfile.nginx
    ports:
      - "80:80"
    volumes:
      - .:/var/www/myphpapp
    depends_on:
      - php
    networks:
      - app-network

  # PHP-FPM (untuk Nginx)
  php:
    build:
      context: .
      dockerfile: Dockerfile
    user: "1000:1000"
    working_dir: /var/www/myphpapp
    volumes:
      - .:/var/www/myphpapp
    depends_on:
      - mysql
    environment:
      - DB_HOST=mysql

```

```

        - DB_NAME=myapp
        - DB_USER=root
        - DB_PASSWORD=password
    networks:
        - app-network

# MySQL Database Server
mysql:
    image: mysql:8.0
    environment:
        MYSQL_ROOT_PASSWORD: password
        MYSQL_DATABASE: myapp
        MYSQL_USER: appuser
        MYSQL_PASSWORD: password
    volumes:
        - mysql_data:/var/lib/mysql
    ports:
        - "3306:3306"
    networks:
        - app-network

# phpMyAdmin untuk Database Management
phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
        - "8081:80"
    environment:
        - PMA_HOST=mysql
        - MYSQL_ROOT_PASSWORD=password
    depends_on:
        - mysql
    networks:
        - app-network

# Persistent Volume untuk MySQL Data
volumes:
    mysql_data:

# Custom Network untuk Container Communication
networks:
    app-network:
        driver: bridge

```

Hasilnya akan sama saja dengan folder **belajarphpnative** namun perbedaannya adalah terletak pada web server. Menggunakan konfigurasi di atas, PHP akan berjalan sendiri menggunakan PHP FPM lalu akan ada

web server dengan Nginx yang menerima *requests*. Artinya setiap permintaan yang masuk akan diteruskan oleh Nginx ke PHP melalui port khusus, yaitu 9000.

4.1 Logging

Pencatatan (logging) dalam DevOps adalah bagian penting untuk memantau, menganalisis, dan memecahkan masalah pada seluruh siklus hidup pengembangan perangkat lunak, mulai dari pengembangan, pengujian, hingga produksi. Proses ini melibatkan pengumpulan, penyimpanan, dan analisis log yang dihasilkan oleh aplikasi, sistem, dan infrastruktur.

Tujuan Utama Logging dalam DevOps:

- **Pemantauan Kinerja Aplikasi:** Log memberikan wawasan tentang bagaimana aplikasi berjalan, mengidentifikasi hambatan kinerja, dan memantau penggunaan sumber daya.
- **Deteksi dan Pemecahan Masalah:** Ketika terjadi kesalahan atau anomali, log adalah sumber informasi utama untuk mengidentifikasi akar masalah, memahami konteks kejadian, dan mempercepat proses perbaikan.
- **Audit dan Keamanan:** Log menyediakan jejak aktivitas yang dapat digunakan untuk tujuan audit, mendeteksi upaya akses tidak sah, dan memastikan kepatuhan terhadap standar keamanan.
- **Analisis Tren dan Prediksi:** Dengan menganalisis log dari waktu ke waktu, tim dapat mengidentifikasi tren, memprediksi potensi masalah, dan membuat keputusan yang lebih baik tentang peningkatan sistem.
- **Pemahaman Pengguna:** Log yang berkaitan dengan interaksi pengguna dapat membantu tim memahami perilaku pengguna dan mengoptimalkan pengalaman pengguna.

Jenis-jenis Log yang Umum Digunakan dalam DevOps:

- **Log Aplikasi:** Dihasilkan oleh kode aplikasi itu sendiri, berisi informasi tentang eksekusi program, kesalahan, peringatan, dan informasi debug.
- **Log Sistem Operasi:** Dihasilkan oleh sistem operasi, seperti log peristiwa Windows atau syslog Linux, mencatat aktivitas sistem, kesalahan kernel, dan informasi boot.
- **Log Server Web:** Dihasilkan oleh server web (misalnya, Apache, Nginx), mencatat permintaan HTTP, respons, kode status, dan informasi klien.
- **Log Database:** Dihasilkan oleh sistem manajemen basis data, mencatat kueri, transaksi, kesalahan basis data, dan aktivitas pengguna.
- **Log Jaringan:** Dihasilkan oleh perangkat jaringan (router, firewall), mencatat lalu lintas jaringan, koneksi, dan potensi ancaman keamanan.
- **Log Keamanan:** Berfokus pada peristiwa yang berkaitan dengan keamanan, seperti upaya login yang gagal, perubahan hak akses, atau deteksi intrusi.

Praktik Terbaik untuk Logging dalam DevOps:

- **Standardisasi Format Log:** Menggunakan format log yang konsisten (misalnya, JSON, ELK stack-compatible) memudahkan agregasi dan analisis.

- **Kecukupan dan Relevansi:** Pastikan log mengandung informasi yang cukup untuk diagnostik tetapi tidak berlebihan agar tidak membebani sistem.
- **Agregasi Log Terpusat:** Mengumpulkan log dari berbagai sumber ke dalam satu lokasi terpusat (misalnya, menggunakan Elasticsearch, Splunk, Graylog) untuk analisis yang lebih mudah.
- **Visualisasi dan Dashboard:** Menggunakan alat visualisasi (misalnya, Kibana, Grafana) untuk membuat dashboard yang memungkinkan pemantauan real-time dan identifikasi tren.
- **Peringatan (Alerting):** Mengatur peringatan berdasarkan pola atau ambang batas tertentu dalam log untuk memberitahu tim tentang masalah kritis secara proaktif.
- **Manajemen Siklus Hidup Log:** Menentukan kebijakan retensi log untuk mengelola ruang penyimpanan dan memastikan kepatuhan terhadap peraturan.
- **Otomatisasi:** Mengotomatiskan proses pengumpulan, pengiriman, dan analisis log.
- **Konteks yang Luas:** Memasukkan konteks yang relevan dalam log, seperti ID transaksi, ID pengguna, atau nama mikroservis, untuk membantu pelacakan masalah.

Dengan menerapkan praktik logging yang efektif, tim DevOps dapat meningkatkan visibilitas operasional, mengurangi waktu henti (downtime), mempercepat penyelesaian masalah, dan pada akhirnya, memberikan perangkat lunak yang lebih stabil dan andal.

4.2 Mengakses Log pada Docker

Ketika Docker dijalankan tanpa mode *detachment*, apa yang terlihat pada *Command Line Interface* (CLI) sebenarnya adalah *log*. *Log* ini merupakan catatan semua aktivitas yang terjadi di dalam kontainer Docker sejak dimulai. Informasi yang ditampilkan meliputi keluaran dari aplikasi yang berjalan di dalam kontainer, pesan *debug*, pesan kesalahan, dan informasi status lainnya.

Tampilan *log* secara langsung pada CLI sangat berguna untuk memantau perilaku aplikasi secara *real-time* selama pengembangan atau *debugging*. Pengembang dapat melihat apakah aplikasi berfungsi seperti yang diharapkan, mengidentifikasi masalah, dan memahami alur eksekusi. Namun, mode ini juga berarti bahwa sesi CLI akan terblokir selama kontainer berjalan. Jika Anda menutup terminal, kontainer akan berhenti berjalan karena tidak ada lagi proses yang terhubung.

4.2.1 Mengakses Log pada Container Docker

Ini berbeda dengan menjalankan Docker dalam mode *detached* (menggunakan flag `-d` atau `--detach`), di mana kontainer akan berjalan di latar belakang dan melepaskan kontrol dari terminal. Dalam mode *detached*, *log* masih dihasilkan tetapi tidak ditampilkan langsung di CLI. Untuk melihat *log* dari kontainer yang berjalan di latar belakang, Anda perlu menggunakan perintah berikut:

```
docker logs <container_id_or_name>
```

Masukkan perintah `docker ps` untuk melihat container idnya terlebih dahulu.

```
udya@DESKTOP-9Q85VU9:~$ docker ps
CONTAINER ID   IMAGE
93f7de37dabb  belajarphpnginx-nginx
```

Misalnya di atas untuk container nginx bertuliskan id container **93f7de37dabb**, maka masukkan perintah:

```
docker logs 93f7de37dabb
```

Hasilnya adalah sebagai berikut:

```
udya@DESKTOP-9Q85VU9:~$ docker logs -f 93f7
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh:                                                               Launching
/docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh:      info:      Getting      the      checksum      of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from
the packaged version
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/09/20 09:09:56 [notice] 1#1: using the "epoll" event method
```

4.2.2 Mengakses Container Docker Compose

Misalnya menggunakan docker compose pada submodul 3.3, pertama nyalakan terlebih dahulu containernya dengan perintah:

```
docker compose up -d
```

Kemudian akses localhost atau localhost/conn.php untuk melihat isi dari logs nginx maka perintahnya adalah:

```
docker compose logs nginx
```

Mengapa Nginx? Karena proses yang biasanya diamati adalah request yang masuk. Sekarang yang akan dilihat adalah sebagai berikut:

```
udya@DESKTOP-9Q85VU9:~/belajarphpnginx$ docker compose logs nginx
WARN[0000] /home/udya/belajarphpnginx/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
nginx-1 | 2025/09/20 08:35:24 [notice] 1#1: start worker process 29
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:41:22 +0000] "GET / HTTP/1.1" 200 21 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
```

Bisa dilihat dari baris log di atas bahwa URL / (localhost) telah diakses. Masalah dari penggunaan logs di atas adalah proses berhenti setelah perintah dimasukkan pada terminal. Untuk membuatnya menjadi active (following) maka diperlukan flag -f pada perintahnya, yaitu:

```
docker compose logs -f nginx
```

Sekarang baris log akan terus muncul pada saat web diakses, cobalah akses localhost atau localhost/conn.php maka yang akan tampil adalah request yang baru saja diakses:

```
udya@DESKTOP-9Q85VU9:~/belajarphpnginx$ docker compose logs -f nginx
WARN[0000] /home/udya/belajarphpnginx/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
nginx-1 | 2025/09/20 08:35:24 [notice] 1#1: start worker process 29
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:41:22 +0000] "GET / HTTP/1.1" 200 21 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:50:16 +0000] "GET / conn.php HTTP/1.1" 200 64 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:50:18 +0000] "GET /conn.php HTTP/1.1" 200 64 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:50:19 +0000] "GET /conn.php HTTP/1.1" 200 64 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
nginx-1 | 172.19.0.1 - - [20/Sep/2025:08:50:21 +0000] "GET / HTTP/1.1" 200 21 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36" "-"
```

Sebagai catatan penting, log yang disebutkan di atas sebenarnya adalah log akses dan error yang dihasilkan serta disimpan secara langsung oleh *container* Nginx. Nginx, dalam konfigurasi standarnya, menulis log ini ke dalam sistem *file* di dalam *container* itu sendiri, tepatnya di lokasi /var/log/nginx. Folder ini merupakan direktori standar tempat Nginx menyimpan semua informasi mengenai permintaan yang masuk (log akses) dan masalah yang mungkin terjadi selama operasi (log error). Memahami lokasi ini sangat krusial untuk proses *debugging*, pemantauan kinerja, dan analisis keamanan aplikasi yang dilayani oleh Nginx. Untuk menghentikan proses penampilan logs cukup tekan **CTRL + C**.

Sekarang masuklah ke dalam container nginx untuk melihat lognya dengan perintah berikut:

```
docker compose exec -it nginx /bin/bash
```

Masukkan perintah berikut baris perbaris untuk masuk ke foldernya:

```
cd /var/log/nginx/  
cat myphpapp_access.log
```

Perintah di atas adalah perintah untuk membuka atau menuju direktori **/var/log/nginx** dan disana terdapat file bernama **myphpapp_access.log** yang telah dikonfigurasi pada file **default.conf** yang merupakan konfigurasi nginx dan sudah ditulis modul 3.3 dimana file **myphpapp_access.log** adalah log *requests* yang digunakan untuk mencatat *request* yang masuk. Apabila ingin memastikan apa saja yang ada di dalam folder **/var/log/nginx** maka perintahnya adalah **ls**. Setelah dua perintah di atas dijalankan, maka yang tampil pada terminal adalah sebagai berikut:

```
root@93f7de37dabb:/var/log/nginx# cat myphpapp_access.log  
172.19.0.1 - - [20/Sep/2025:09:10:19 +0000] "GET / HTTP/1.1" 200 21 "-"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/140.0.0.0 Safari/537.36"  
172.19.0.1 - - [20/Sep/2025:09:10:26 +0000] "GET / HTTP/1.1" 200 21 "-"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/140.0.0.0 Safari/537.36"  
172.19.0.1 - - [20/Sep/2025:09:10:29 +0000] "GET /conn.php HTTP/1.1" 200 64 "-"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/140.0.0.0 Safari/537.36"
```

Perintah cat adalah perintah yang dapat menampilkan semua isi file di dalamnya, untuk melihatnya secara following (*live*) maka perintahnya adalah:

```
tail -f myphpapp_access.log
```

Sekarang akses kembali localhost atau localhost/conn.php dan lihat baris akan bertambah secara otomatis. Untuk keluar dari container cukup masukkan perintah **exit**.

4.3 Membuat Log Aplikasi PHP

Membuat log pada aplikasi PHP merupakan tindakan yang sangat penting karena beberapa alasan mendasar yang berkaitan dengan keberlanjutan, pemeliharaan, dan keamanan aplikasi. Log adalah catatan kronologis dari peristiwa-peristiwa yang terjadi di dalam aplikasi, mulai dari operasi normal, peringatan, hingga kesalahan.

Secara keseluruhan, logging bukan hanya tentang mencatat kejadian, tetapi merupakan bagian integral dari strategi observabilitas aplikasi. Dengan implementasi logging yang baik, pengembang dan tim operasional dapat memastikan aplikasi PHP berjalan dengan lancar, aman, dan dapat diandalkan.

Menggunakan proyek pada **modul 3.3**, buatlah sebuah file baru bernama **access_log.php** pada folder **belajarphpnginx** dan masukkan kode berikut:

```
<?php
// Simple logging function
function writeLog($level, $message, $context = []) {
    $timestamp = date('Y-m-d H:i:s');
    $contextStr = !empty($context) ? json_encode($context) : '';

    $logEntry = "[$timestamp] [$level] $message $contextStr" . PHP_EOL;

    // Write ke /tmp yang pasti writable
    file_put_contents('/var/www/myphpapp/app.log', $logEntry, FILE_APPEND | LOCK_EX);
}

// Log halaman diakses
writeLog('INFO', 'Halaman diakses!', [
    'ip' => $_SERVER['REMOTE_ADDR'] ?? 'unknown',
    'user_agent' => $_SERVER['HTTP_USER_AGENT'] ?? 'unknown'
]);

// Echo hasil
echo "Log berhasil ditulis!<br>";
echo "Timestamp: " . date('Y-m-d H:i:s') . "<br>";
```

```
echo "IP Address: " . ($_SERVER['REMOTE_ADDR'] ?? 'unknown') . "<br>";
echo "User Agent: " . ($_SERVER['HTTP_USER_AGENT'] ?? 'unknown') . "<br>";
echo "<br>";
echo "Cek log dengan: docker compose exec -it php cat /var/www/myphpapp/app.log";
echo "<br>";
echo "atau dengan: docker compose exec -it php tail -f /var/www/myphpapp/app.log";
?>
```

Kemudian akses halaman `localhost/access_log.php`, pesan yang akan ditampilkan adalah:

```
Log berhasil ditulis!
Timestamp: 2025-09-20 10:17:53
IP Address: 172.19.0.1
User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/140.0.0.0 Safari/537.36

Cek log dengan: docker compose exec -it php cat /var/www/myphpapp/app.log
atau dengan: docker compose exec -it php tail -f /var/www/myphpapp/app.log
```

Kode di atas mendemonstrasikan implementasi sistem logging sederhana dalam PHP yang terdiri dari dua bagian utama: pembuatan function logging dan penggunaannya untuk mencatat aktivitas user.

Function `writeLog()` dirancang untuk menerima tiga parameter yaitu level log, pesan, dan context data tambahan, kemudian memformat informasi tersebut menjadi entry log yang terstruktur dengan timestamp, level, dan data context dalam format JSON. Function ini menggunakan `file_put_contents()` dengan flag `FILE_APPEND` dan `LOCK_EX` untuk menambahkan entry baru ke file log tanpa menimpa data sebelumnya sambil mencegah corruption saat multiple processes menulis secara bersamaan.

Bagian implementasi menunjukkan penggunaan praktis dari function logging dengan memanggil `writeLog()` setiap kali halaman diakses, mencatat informasi penting seperti IP address pengunjung dan User Agent browser yang diambil dari superglobal `$_SERVER`. Kode juga menggunakan null coalescing operator (`??`) untuk handle kasus dimana data server tidak tersedia dengan memberikan nilai default 'unknown'. Setelah menulis log, script memberikan feedback kepada user melalui browser dengan menampilkan konfirmasi bahwa log telah berhasil ditulis beserta informasi yang dicatat, dan menyediakan instruksi command untuk melihat isi file log baik melalui container maupun host system. Pendekatan logging ini sangat berguna untuk tracking user activity, debugging aplikasi, monitoring system behavior, dan audit trail dalam development maupun production environment.

4.4 Memahami Pentingnya Logging

Logs adalah "jendela" untuk melihat apa yang terjadi di dalam container. Ketika aplikasi dalam container mengalami masalah, logs menjadi sumber informasi pertama yang harus diperiksa untuk memahami root cause masalah tersebut. Dalam environment production, kemampuan membaca dan memahami logs merupakan skill fundamental yang harus dimiliki setiap DevOps engineer.

Docker menyimpan semua output dari aplikasi yang berjalan dalam container, termasuk stdout (standard output) dan stderr (standard error). Pemahaman dasar tentang cara membaca dan menginterpretasi logs akan membantu mahasiswa dalam troubleshooting dan monitoring aplikasi containerized.

4.4.1 Jenis-Jenis Log Messages

Setiap aplikasi umumnya menghasilkan beberapa jenis *log messages*, yaitu:

1. INFO/INFO Level

- Informasi normal operasi aplikasi
- Tidak menunjukkan masalah
- Contoh: "Server started on port 80", "User logged in"

2. WARNING/WARN Level

- Situasi yang perlu perhatian tapi tidak critical
- Aplikasi masih berjalan normal
- Contoh: "Disk space low", "Deprecated API used"

3. ERROR Level

- Masalah serius yang mengganggu operasi
- Bagian dari aplikasi mungkin tidak berfungsi
- Contoh: "Database connection failed", "File not found"

4. FATAL/CRITICAL Level

- Error yang menyebabkan aplikasi berhenti
- Memerlukan immediate action
- Contoh: "Out of memory", "Configuration file missing"

4.4.2 Contoh Log Berdasarkan Tipe

Berikut merupakan beberapa contoh log yang sering ditemui:

Contoh 1: Log Normal Web Server (Nginx)

Berikut adalah analisis lebih lanjut dari log yang diberikan:

```
2024-09-26 10:30:15 [INFO] nginx/1.21.0
2024-09-26 10:30:15 [INFO] server started on port 80
2024-09-26 10:30:20 [INFO] 192.168.1.100 - GET /index.html - 200
2024-09-26 10:30:25 [INFO] 192.168.1.101 - GET /about.html - 200
2024-09-26 10:30:30 [INFO] 192.168.1.102 - GET /contact.html - 200
```

Analisis Log:

Ini adalah contoh ideal dari log web server yang beroperasi dalam kondisi normal dan sehat. Pada pukul 10:30:15, Nginx versi 1.21.0 berhasil startup dan mengikat dirinya ke port 80, yang merupakan port standar untuk HTTP traffic. Kedua pesan INFO ini menunjukkan bahwa proses initialization berjalan sempurna tanpa ada error atau warning yang mengganggu.

Setelah server siap, mulai terlihat aktivitas dari client yang mengakses website. Pada pukul 10:30:20, user pertama dengan IP address 192.168.1.100 mengakses halaman utama (index.html) dan berhasil mendapat response dengan status code 200, yang menandakan request berhasil diproses dan content berhasil dikirimkan ke client. Lima detik kemudian, user kedua dari IP 192.168.1.101 mengakses halaman "about" dengan hasil yang sama - sukses dengan code 200.

Pattern yang sehat ini berlanjut ketika user ketiga (192.168.1.102) mengakses halaman "contact" pada pukul 10:30:30, juga dengan response code 200. Interval waktu yang konsisten antara request (setiap 5 detik) dan uniformitas response code menunjukkan bahwa server bekerja dengan stabil dan responsive. Tidak ada indikasi bottleneck, error, atau masalah performa yang mengganggu user experience.

Kesimpulan:

Log ini merepresentasikan operasi web server yang optimal, di mana semua komponen berfungsi sebagaimana mestinya. Status code 200 pada semua request mengkonfirmasi bahwa semua file yang diminta tersedia dan dapat diakses dengan benar. Pattern traffic yang teratur juga menunjukkan bahwa server mampu menangani multiple concurrent users tanpa mengalami performance degradation.

Dalam konteks monitoring dan maintenance, log seperti ini adalah yang diharapkan administrator untuk production environment. Tidak ada action yang diperlukan ketika melihat log pattern seperti ini, selain tetap memantau untuk memastikan konsistensi performa. Untuk pembelajaran, contoh ini menunjukkan bagaimana seharusnya aplikasi web containerized beroperasi ketika semua konfigurasi sudah tepat dan tidak ada external dependencies yang bermasalah.

Contoh 2: Log dengan Warning

```
2024-09-26 10:35:10 [INFO] nginx/1.21.0
2024-09-26 10:35:10 [INFO] server started on port 80
```

```
2024-09-26 10:35:15 [WARN] worker_connections exceed open file limit: 1024
2024-09-26 10:35:20 [INFO] 192.168.1.100 - GET /index.html - 200
2024-09-26 10:35:25 [WARN] client request body is buffered to a temporary file
```

Analisis Log:

Nginx server memulai operasinya dengan normal pada pukul 10:35:10, menunjukkan versi 1.21.0 dan berhasil binding ke port 80. Proses startup berjalan lancar tanpa ada error yang menghalangi server untuk mulai menerima koneksi dari client.

Lima detik kemudian, muncul warning pertama yang mengindikasikan masalah konfigurasi resource. Sistem mencatat bahwa "worker_connections exceed open file limit: 1024", yang berarti Nginx dikonfigurasi untuk menangani lebih banyak concurrent connections daripada yang diizinkan oleh sistem operasi. Operating system membatasi jumlah file descriptor yang dapat dibuka secara bersamaan (dalam hal ini 1024), sementara konfigurasi Nginx mungkin diset untuk worker_connections yang lebih tinggi. Meskipun ini adalah warning dan bukan error, hal ini dapat membatasi kemampuan server untuk menangani traffic tinggi.

Server tetap dapat melayani request dengan baik, seperti terlihat pada pukul 10:35:20 ketika user dari IP 192.168.1.100 berhasil mengakses index.html dan mendapat response code 200 (success). Namun lima detik kemudian, muncul warning kedua yang menunjukkan masalah performa lainnya. Pesan "client request body is buffered to a temporary file" mengindikasikan bahwa ada request dengan body yang cukup besar sehingga Nginx harus menyimpannya ke file sementara di disk daripada di memory. Hal ini bisa memperlambat response time karena disk I/O umumnya lebih lambat dari memory access.

Kesimpulan:

Log ini menunjukkan bahwa meskipun server berfungsi dengan normal dan dapat melayani request, ada beberapa aspek konfigurasi yang perlu dioptimasi untuk performa yang lebih baik. Warning level messages tidak menghentikan operasi server namun memberikan indikasi tentang bottleneck atau limitasi yang mungkin mempengaruhi performa dalam situasi high-traffic.

Untuk mengatasi warning pertama, administrator dapat meningkatkan system file descriptor limit atau menyesuaikan konfigurasi worker_connections di Nginx. Sedangkan untuk warning kedua, solusinya bisa berupa peningkatan client_body_buffer_size di konfigurasi Nginx atau mengoptimasi aplikasi client untuk mengirim request body yang lebih kecil. Dalam konteks Docker, masalah ini sering terjadi karena default resource limits yang terlalu konservatif, sehingga perlu penyesuaian melalui Docker run parameters atau Docker Compose configuration.

Contoh 3: Log dengan Error

```
2024-09-26 10:40:10 [INFO] nginx/1.21.0
2024-09-26 10:40:10 [INFO] server started on port 80
```

```
2024-09-26 10:40:15 [ERROR] cannot access /var/www/html/missing.html: No such file or
directory
2024-09-26 10:40:15 [INFO] 192.168.1.100 - GET /missing.html - 404
2024-09-26 10:40:20 [ERROR] upstream connect error: Connection refused
2024-09-26 10:40:20 [INFO] 192.168.1.101 - GET /api/data - 502
```

Analisis Log:

Nginx web server berhasil startup dengan normal pada pukul 10:40:10, menampilkan versi 1.21.0 dan konfirmasi bahwa service telah aktif di port 80. Hingga titik ini, semua berjalan sesuai ekspektasi dan server siap menerima request dari client.

Masalah pertama muncul lima detik kemudian ketika ada user dengan IP 192.168.1.100 yang mencoba mengakses file "missing.html". Server mencatat error bahwa file tersebut tidak dapat diakses karena tidak ditemukan di direktori /var/www/html/. Sebagai response yang tepat, Nginx mengembalikan HTTP status code 404 (Not Found) kepada client, menunjukkan bahwa server berfungsi dengan benar dalam menangani file yang tidak ada.

Lima detik berikutnya, situasi menjadi lebih serius ketika user lain (192.168.1.101) mencoba mengakses endpoint "/api/data". Kali ini masalahnya bukan pada file yang missing, melainkan pada upstream service yang tidak dapat diakses. Log menunjukkan "upstream connect error: Connection refused", yang mengindikasikan bahwa ada backend service (mungkin API server atau database) yang seharusnya melayani request ini namun tidak responding. Akibatnya, Nginx mengembalikan HTTP status code 502 (Bad Gateway), menunjukkan bahwa proxy server tidak dapat mendapatkan response yang valid dari upstream server.

Kesimpulan:

Kasus ini mengilustrasikan dua jenis error yang berbeda dalam web server environment. Error pertama (404) adalah client-side error yang disebabkan oleh request terhadap resource yang tidak ada, sedangkan error kedua (502) adalah server-side error yang mengindikasikan masalah pada backend infrastructure. Nginx dalam hal ini berfungsi sebagai reverse proxy yang berusaha meneruskan request ke upstream service namun gagal karena service tersebut tidak available.

Untuk troubleshooting, error 404 relatif mudah diatasi dengan memastikan file yang diminta benar-benar ada di direktori yang tepat. Sedangkan error 502 memerlukan investigasi lebih mendalam terhadap backend services, seperti memeriksa apakah API server running, network connectivity antar services, dan konfigurasi upstream di Nginx. Dalam konteks Docker, masalah ini sering terjadi ketika container dependencies tidak startup dengan urutan yang benar atau ada network connectivity issue antar container.

Contoh 4: Log Database Connection Error

```
2024-09-26 10:45:10 [INFO] Application starting...
```

```
2024-09-26 10:45:12 [ERROR] Database connection failed: Connection refused  
(localhost:5432)  
2024-09-26 10:45:12 [ERROR] Could not connect to PostgreSQL server  
2024-09-26 10:45:13 [FATAL] Application cannot start without database connection  
2024-09-26 10:45:13 [INFO] Application terminated with exit code 1
```

Analisis Log:

Cerita dimulai pada pukul 10:45:10 ketika aplikasi mencoba untuk startup dengan pesan "Application starting..." yang menunjukkan bahwa proses initialization berjalan normal. Namun hanya dalam waktu 2 detik kemudian, masalah mulai terlihat.

Pada pukul 10:45:12, aplikasi mencoba melakukan koneksi ke database PostgreSQL yang seharusnya berjalan di localhost port 5432, namun mendapat response "Connection refused". Ini adalah tanda pertama bahwa ada masalah dengan database service. Dalam detik yang sama, aplikasi mencatat error kedua yang lebih spesifik: "Could not connect to PostgreSQL server", yang memperjelas bahwa masalahnya memang pada database PostgreSQL.

Situasi menjadi critical pada detik berikutnya (10:45:13) ketika aplikasi mengeluarkan log FATAL level, menunjukkan bahwa database connection bukanlah optional feature melainkan critical dependency. Aplikasi menyadari bahwa tidak ada gunanya melanjutkan operasi tanpa akses database, sehingga memutuskan untuk terminate completely. Pesan terakhir menunjukkan bahwa aplikasi berhenti dengan exit code 1, yang dalam convention Unix/Linux menandakan abnormal termination.

Kesimpulan:

Kasus ini menunjukkan pentingnya dependency management dalam aplikasi containerized. Timeline yang singkat (3 detik dari startup hingga termination) mengindikasikan bahwa aplikasi memiliki mekanisme fail-fast yang efektif, di mana sistem langsung menghentikan operasi ketika critical dependency tidak tersedia daripada mencoba terus atau hanging. Hal ini merupakan praktik yang baik dalam development karena memberikan feedback yang cepat kepada developer tentang masalah konfigurasi.

Untuk mengatasi masalah serupa, developer perlu memastikan bahwa semua service dependencies (dalam hal ini database PostgreSQL) sudah running sebelum memulai aplikasi utama. Dalam konteks Docker, ini bisa diatasi dengan menggunakan Docker Compose untuk mengelola startup order antar container, atau mengimplementasikan *retry mechanism* dengan proper delay untuk menunggu database *service fully initialized*.

5.1 Implementasi CI/CD

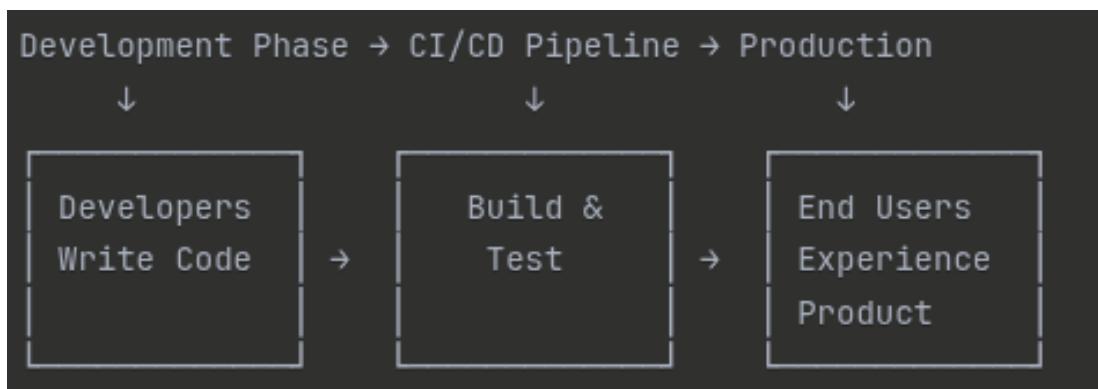
Continuous Integration (CI) dan **Continuous Deployment (CD)** merupakan praktik fundamental dalam modern software development yang bertujuan untuk mengotomatisasi proses development, testing, dan deployment aplikasi. Konsep ini menjadi tulang punggung DevOps culture yang menekankan pada collaboration, automation, dan rapid delivery.

Continuous Integration adalah praktik di mana developer secara rutin merge code changes mereka ke dalam shared repository, idealnya beberapa kali sehari. Setiap integration kemudian diverifikasi dengan automated build dan automated tests untuk mendeteksi integration errors secepat mungkin. Tujuan utama CI adalah mengurangi integration problems yang sering muncul ketika multiple developers bekerja pada project yang sama.

Continuous Deployment melanjutkan proses CI dengan otomatis men-deploy setiap code change yang telah melewati automated testing ke production environment. Dalam beberapa kasus, praktik ini disebut juga **Continuous Delivery**, dimana perubahan kode siap untuk di-deploy namun memerlukan manual approval untuk release ke production.

5.1.1 CI/CD Pipeline

Dalam context project management dan *software development lifecycle*, CI/CD pipeline berperan sebagai **automation backbone** yang menghubungkan berbagai tahapan development. Berikut adalah posisi dan peran CI/CD dalam project workflow:



Gambar 5.1 Proses CI/CD

Posisi dalam Software Development Lifecycle:

- **Planning Phase:** CI/CD requirements didefinisikan sebagai part of project architecture
- **Development Phase:** Developers commit code yang akan trigger pipeline automatically
- **Testing Phase:** Automated testing menjadi integral part dari pipeline

- **Deployment Phase:** Automated deployment mengurangi manual intervention dan human error
- **Monitoring Phase:** Pipeline provides feedback dan metrics untuk continuous improvement

Integrasi dengan Project Management Tools:

CI/CD pipeline terintegrasi dengan project management workflow melalui:

- **Issue Tracking:** Pipeline status linked ke project issues dan tasks
- **Code Review Process:** Pull requests trigger pipeline untuk validation
- **Release Management:** Pipeline mengotomatisasi release notes dan deployment scheduling
- **Quality Gates:** Pipeline enforcement untuk quality standards sebelum production

5.1.3 Manfaat Implementasi CI/CD Pipeline

Implementasi CI/CD pipeline memberikan benefits yang significant untuk development team dan organization secara keseluruhan:

Business Benefits:

- **Faster Time to Market:** Automated processes mengurangi delivery cycle dari weeks ke hours atau days
- **Reduced Operational Costs:** Less manual intervention berarti lower operational overhead
- **Improved Customer Satisfaction:** Frequent updates dan bug fixes meningkatkan user experience
- **Competitive Advantage:** Ability untuk rapid deployment memberi edge di market

Technical Benefits:

- **Quality Assurance:** Automated testing pada setiap code change mendeteksi issues early
- **Reduced Risk:** Small, frequent deployments lebih mudah rollback jika ada problems
- **Consistency:** Standardized deployment process mengurangi environment-specific issues
- **Traceability:** Complete audit trail dari code change hingga production deployment

Team Benefits:

- **Developer Productivity:** Focus pada coding rather than manual deployment tasks
- **Faster Feedback Loop:** Quick detection dan resolution of integration problems
- **Reduced Stress:** Automated processes mengurangi pressure pada release days
- **Knowledge Sharing:** Pipeline configuration sebagai shared knowledge dalam team

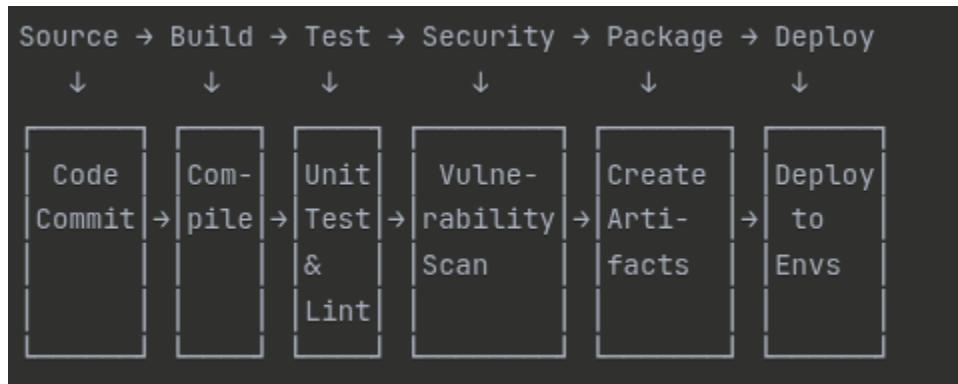
Operational Benefits:

- **24/7 Deployment Capability:** Automated pipelines dapat berjalan kapan saja
- **Scalability:** Pipeline dapat handle multiple concurrent deployments
- **Monitoring Integration:** Automated alerts dan notifications untuk pipeline status

- **Documentation:** Pipeline serves sebagai living documentation untuk deployment process

5.1.4 Komponen Utama CI/CD Pipeline

Pipeline CI/CD terdiri dari beberapa stages yang saling terhubung dalam sequential atau parallel flow:



Gambar 5.2 Proses Build Deployment

1. Source Control Integration:

- Monitor repository untuk code changes
- Webhook triggers untuk automated pipeline execution
- Branch-based pipeline execution strategies

2. Build Stage:

- Source code compilation dan dependency resolution
- Environment setup dan configuration management
- Asset optimization dan bundling

3. Test Stage:

- Unit tests untuk individual components
- Integration tests untuk system interactions
- End-to-end tests untuk complete user workflows
- Performance tests untuk load dan stress testing

4. Security Scanning:

- Vulnerability scanning untuk dependencies
- Code quality analysis dan security best practices
- License compliance checking

5. Artifact Generation:

- Creation of deployable packages (Docker images, JARs, etc.)
- Versioning dan tagging untuk traceability
- Artifact storage dalam registries

6. Deployment Stage:

- Environment-specific deployments (development, staging, production)
- Blue-green atau rolling deployment strategies
- Database migrations dan configuration updates
- Post-deployment verification tests

Setiap *stage* dapat dikonfigurasi dengan specific conditions, approval processes, dan rollback mechanisms untuk memastikan reliability dan control dalam deployment process.

5.2 GitHub Actions

GitHub Actions adalah platform automation yang disediakan langsung oleh GitHub untuk membangun, test, dan deploy code langsung dari repository. Sebagai **cloud-native CI/CD solution**, GitHub Actions terintegrasi seamlessly dengan GitHub ecosystem dan menyediakan powerful automation capabilities untuk various software development workflows.

Arsitektur GitHub Actions didasarkan pada prinsip **event-driven automation**, yang berarti alur kerja (workflows) secara otomatis dipicu oleh kejadian atau peristiwa tertentu. Kejadian ini bisa bervariasi, seperti ketika ada kode baru yang didorong (code push) ke repositori, ketika sebuah permintaan penggabungan (pull request) dibuat, atau bahkan pada interval waktu yang terjadwal (scheduled intervals) seperti setiap malam atau setiap minggu. Pendekatan ini memastikan bahwa tugas-tugas otomatis berjalan tepat pada saat dibutuhkan, menjaga konsistensi dan efisiensi dalam proses pengembangan.

Platform ini menyediakan **hosted runners**, yang merupakan mesin virtual yang disediakan dan dikelola oleh GitHub. Runner ini memiliki berbagai sistem operasi seperti Ubuntu, Windows, dan macOS, memungkinkan pengembang untuk menguji dan menerapkan kode mereka di lingkungan yang berbeda tanpa perlu menyiapkan infrastruktur sendiri. Selain itu, GitHub Actions juga mendukung **self-hosted runners**. Ini memberikan fleksibilitas tambahan bagi organisasi yang mungkin memiliki persyaratan

khusus, seperti lingkungan kustom dengan perangkat keras atau perangkat lunak tertentu yang tidak tersedia di hosted runners, atau untuk memenuhi kebijakan keamanan internal yang ketat. Dengan self-hosted runners, pengguna dapat menjalankan pekerjaan di server atau mesin lokal mereka sendiri, memberikan kontrol penuh atas lingkungan eksekusi.

5.2.1 GitHub Actions vs Alternatif

Dalam landscape CI/CD tools, terdapat berbagai alternatif populer seperti **Jenkins**, **GitLab CI/CD**, **CircleCI**, **Travis CI**, dan **Azure DevOps**. **Jenkins**, sebagai salah satu pioneer CI/CD tools, sangat powerful dan flexible namun memerlukan setup dan maintenance yang kompleks, termasuk server management dan plugin configuration yang extensive.

GitHub Actions menawarkan significant advantages terutama untuk teams yang sudah menggunakan GitHub sebagai code repository:

Keunggulan GitHub Actions:

- **Native Integration:** Seamless integration dengan GitHub ecosystem tanpa external tools
- **Zero Infrastructure Management:** Hosted runners menghilangkan server maintenance overhead
- **Pay-per-use Model:** Cost-effective dengan generous free tier untuk public repositories
- **Simplicity:** YAML-based configuration yang mudah dipelajari dan maintain
- **Extensive Marketplace:** Thousands of pre-built actions untuk common tasks
- **Excellent Documentation:** Clear examples dan comprehensive guides

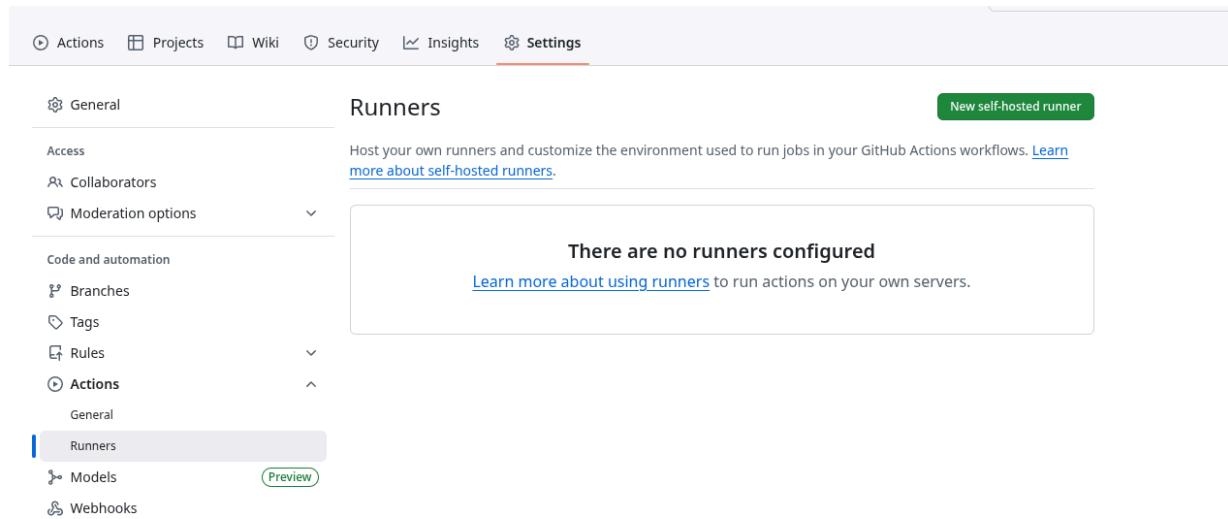
Perbandingan dengan Alternatif:

- **Jenkins:** Lebih powerful tapi complex setup, memerlukan dedicated server dan extensive configuration
- **GitLab CI/CD:** Excellent tapi terikat dengan GitLab ecosystem
- **CircleCI:** User-friendly tapi limited free tier dan separate platform management
- **Azure DevOps:** Comprehensive tapi complex untuk simple projects

Untuk mahasiswa dan pengembang yang baru belajar CI/CD, GitHub Actions memberikan learning curve yang lebih mudah untuk dipahami.

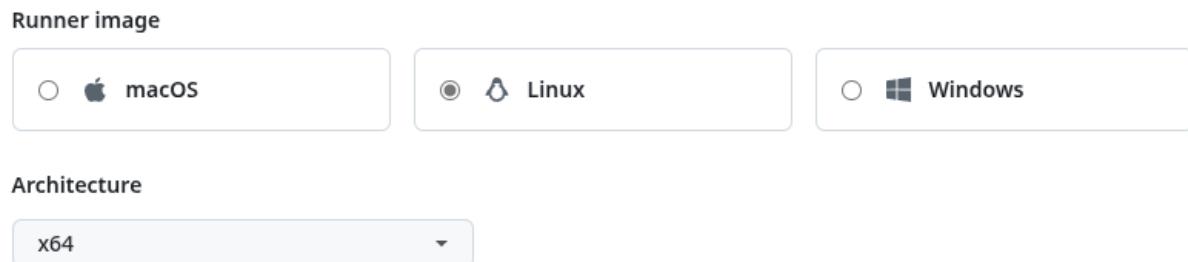
5.2.1 Instalasi GitHub Action Runners

Untuk melakukan instalasi terhadap Github Action Runners, pastikan sudah memiliki repositori terlebih dahulu. Kemudian pergi ke halaman **settings** dan pilih menu **action > runners**, lalu klik **new self hosted runner** pada bagian kanan tombol berwarna hijau.



Gambar 5.3 Konfigurasi Runners

Pada saat memilih jenis OS, pilih saja Linux karena WSL (Windows Subsystem for Linux) adalah Virtual Container Linux yang dijalankan di atas Windows.



Gambar 5.4 Pemilihan OS

Kemudian akan diminta untuk mengeksekusi baris perbaris untuk melakukan instalasi pada Github Actions pada bagian bawahnya.

Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner

# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.328.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.328.0/actions-runner-linux-x64-
2.328.0.tar.gz

# Optional: Validate the hash
$ echo "01066fad3a2893e63e6ca880ae3a1fad5bf9329d60e77ee15f2b97c148c3cd4e" actions-runner-linux-x64-
2.328.0.tar.gz" | shasum -a 256 -c

# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.328.0.tar.gz
```

Configure

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/mhudayaramadhana/hello-world --token
A2HHKXQ465G05K3GI3GXBJ3I265M6

# Last step, run it!
$ ./run.sh
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job
runs-on: self-hosted
```

Gambar 5.5 Skrip Self hosted runner

Pertama buka WSL lalu pastikan direktori berada pada home directory, untuk memindahkannya cukup masukkan perintah `cd`, nantinya dimanapun pada saat itu direktori berada akan pindah ke home directory.

```
udya@DESKTOP-9Q85VU9:~$ cd belajarphpnginx/
udya@DESKTOP-9Q85VU9:~/belajarphpnginx$ cd
udya@DESKTOP-9Q85VU9:~$
```

Gambar 5.6 Kembali ke direktori home

Jalankan kode dari bagian download pada Github Actions baris per-baris secara perlahan pada bagian **Download**.

```

udya@DESKTOP-9Q85VU9:~/actions-runner
udya@DESKTOP-9Q85VU9:~$ cd belajarphnginx/
udya@DESKTOP-9Q85VU9:~/belajarphnginx$ cd
udya@DESKTOP-9Q85VU9:~$ 
udya@DESKTOP-9Q85VU9:~$ mkdir actions-runner && cd actions-runner
udya@DESKTOP-9Q85VU9:~/actions-runner$ curl -o actions-runner-linux-x64-2.328.0.tar.gz -L https://github.com/actions/runners/releases/download/v2.328.0/actions-runner-linux-x64-2.328.0.tar.gz
  % Total    % Received % Xferd  Average Speed   Time     Time   Current
     Dload  Upload Total Spent   Spent    Left  Speed
  0       0      0      0      0      0      0      0      0      0
100  216M  100  216M      0      0  9325k      0  0:00:23  0:00:23  --:--:-- 10.7M
udya@DESKTOP-9Q85VU9:~/actions-runner$ echo "01066fad3a2893e63e6ca880ae3a1fad5bf9329d60e77ee15f2b97c148c3cd4e" | shasum -a 256 -c
actions-runner-linux-x64-2.328.0.tar.gz: OK
udya@DESKTOP-9Q85VU9:~/actions-runner$ tar xzf ./actions-runner-linux-x64-2.328.0.tar.gz

```

Gambar 5.7 Proses ekstraksi folder

Sesudah menjalankan bagian Download, sekarang jalankan juga bagian configure, nantinya Github akan menanyakan beberapa informasi terkait pendaftaran identitas runner, pertanyaan pertama yang ditanyakan adalah *runner group*, lebih baik dibuat default saja jadi cukup tekan **enter**. Pertanyaan selanjutnya adalah nama runnernya, tulis saja namanya **my-laptop**. Pertanyaan terakhir adalah labelnya yang mana label digunakan untuk menentukan aplikasi nanti akan di-build di label apa, untuk menambahkan label ketik saja **my-laptop**. Selanjutnya yang akan dikonfigurasi adalah runner settings, nama folder hasil build dari github bisa diganti, tapi untuk awal disarankan default saja yaitu **_work**, maka cukup tekan **enter**.

```

udya@DESKTOP-9Q85VU9:~/actions-runner
Self-hosted runner registration

# Authentication

✓ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for DESKTOP-9Q85VU9] my-laptop

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip] my-laptop

✓ Runner successfully added
✓ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

✓ Settings Saved.

```

Gambar 5.8 Proses registrasi *runner*

Sekarang refresh halaman github runnernya, apabila konfigurasi berjalan dengan baik maka akan terlihat daftar runner action dari Github.

The screenshot shows the GitHub Actions Settings page for a repository named 'amadhana / hello-world'. The 'Runners' section is displayed, featuring a table with one row. The row contains a icon representing a laptop, the name 'my-laptop', and several status indicators: 'self-hosted', 'Linux', 'X64', and 'my-laptop'. To the right of the table, there is a 'Status' column showing an 'Offline' status with a red dot and three dots for more options. A green button labeled 'New self-hosted runner' is located at the top right of the 'Runners' section.

Gambar 5.9 Tampilan runners yang sudah diregistrasi

Action Runner yang bernama *my-laptop* saat ini dalam keadaan *offline*. Ini menunjukkan bahwa runner tersebut belum diaktifkan atau dijalankan setelah proses konfigurasi dan pendaftaran awal. Status *offline* ini adalah hal yang wajar jika runner baru saja disiapkan dan belum ada pemicu untuk memulai eksekusi *workflow*. Untuk mengaktifkannya, perlu dipastikan aplikasi runner berjalan pada perangkat *my-laptop* dan terhubung dengan platform GitHub Actions. Setelah terhubung, statusnya akan berubah menjadi *idle* dan siap menerima tugas dari *workflow*.

Untuk menjalankannya, kembali ke CLI lalu jalankan dengan perintah:

```
./run.sh
```

Perhatikan gambar di bawah ini:

The terminal window shows the command `./run.sh` being run. The output indicates that the runner is connected to GitHub and is listening for jobs. The current runner version is '2.328.0' and it was last updated on 2025-09-27 at 09:38:04Z.

```
Select udya@DESKTOP-9Q85VU9: ~/actions-runner
✓ Settings Saved.

udya@DESKTOP-9Q85VU9:~/actions-runner$ ./run.sh

✓ Connected to GitHub

Current runner version: '2.328.0'
2025-09-27 09:38:04Z: Listening for Jobs
```

Gambar 5.10 Menjalankan Github Action Runner

Github runner dijalankan secara *manual*, artinya belum di-registrasi untuk berjalan secara otomatis ketika WSL dijalankan. Sekarang kembali ke web github halaman runners dan sekarang mesin dengan label my-laptop statusnya sudah *Idle* alias siap menerima tugas.

The screenshot shows the GitHub Actions Runner settings page for a repository named 'hello-world'. The 'Runners' section displays a table with one row. The row contains the name 'my-laptop', labels 'self-hosted', 'Linux', 'X64', and 'my-laptop', and a status indicator showing a green dot next to the word 'Idle'.

Runners	Status
my-laptop self-hosted Linux X64 my-laptop	Idle

Asdadsa

Sekarang status mesin sudah *Idle* dan siap menerima tugas. Tekan **CTRL + C** untuk mematikannya.

```
udya@DESKTOP-9Q85VU9: ~/actions-runner
✓ Settings Saved.

udya@DESKTOP-9Q85VU9:~/actions-runner$ ./run.sh
✓ Connected to GitHub

Current runner version: '2.328.0'
2025-09-27 09:38:04Z: Listening for Jobs
^CExiting...
Runner listener exit with 0 return code, stop the service, no retry needed.
Exiting runner...
udya@DESKTOP-9Q85VU9:~/actions-runner$
```

Gambar 5.11 Github Runner Action setelah dimatikan

5.3 Konfigurasi Proyek dengan Github Actions

Apabila membuka halaman 'Actions' pada repositori GitHub, bisa dilihat bahwa GitHub telah menyediakan beragam *workflow templates* yang berguna. *Template* ini dirancang khusus untuk menyesuaikan dengan berbagai jenis aplikasi dan teknologi yang sering digunakan dalam pengembangan perangkat lunak.

Sebagai contoh, jika sedang mengembangkan aplikasi web dengan Node.js, ada template yang secara otomatis mengkonfigurasi langkah-langkah untuk membangun (build), menguji (test), dan bahkan

menyebarkan (deploy) aplikasi Node.js yang dibangun. Begitu pula, bagi proyek-proyek Python, Java, .NET, atau aplikasi seluler (mobile apps) seperti Android atau iOS, terdapat *template* khusus yang mempermudah proses integrasi berkelanjutan (Continuous Integration/CI) dan pengiriman berkelanjutan (Continuous Delivery/CD).

Penggunaan templat-templat ini dapat menghemat banyak waktu dan usaha karena pengembang tidak perlu membangun alur kerja dari awal. Pengembang hanya perlu memilih *template* yang paling sesuai dengan proyeknya, kemudian menyesuaikannya sedikit mengikuti kebutuhan spesifik, seperti mengubah versi bahasa pemrograman, menambahkan langkah-langkah pengujian khusus, atau mengonfigurasi lingkungan penyebaran. Ini memungkinkan tim pengembangan untuk fokus pada penulisan kode dan penambahan fitur baru, sementara GitHub Actions mengurus otomatisasi proses pengembangan.

The screenshot shows the GitHub Actions interface. At the top, there's a navigation bar with links for Actions, Projects, Wiki, Security, Insights, and Settings. Below this, a section titled "Get started with GitHub Actions" is displayed, with a sub-section "Suggested for this repository". It lists three templates:

- Laravel**: By GitHub Actions. Test a Laravel project. Status: PHP ●. Includes "Configure" and "Deployment" buttons.
- PHP**: By GitHub Actions. Build and test a PHP application using Composer. Status: PHP ●. Includes "Configure" and "Deployment" buttons.
- Symfony**: By GitHub Actions. Test a Symfony project. Status: PHP ●. Includes "Configure" and "Deployment" buttons.

Below this, a "Deployment" section shows four more templates:

- Deploy a PHP app to an Azure Web App**: By Microsoft Azure. Build a PHP app and deploy it to an Azure Web App. Status: Deployment ●. Includes "Configure" and "Deployment" buttons.
- Deploy to Amazon ECS**: By Amazon Web Services. Deploy a container to an Amazon ECS service powered by AWS Fargate or Amazon EC2. Status: Deployment ●. Includes "Configure" and "Deployment" buttons.
- Build and Deploy to GKE**: By Google Cloud. Build a docker container, publish it to Google Container Registry, and deploy to GKE. Status: Deployment ●. Includes "Configure" and "Deployment" buttons.
- Terraform**: By HashiCorp. Set up Terraform CLI in your GitHub Actions workflow. Status: Deployment ●. Includes "Configure" and "Deployment" buttons.

A "View all" link is located at the top right of the deployment section.

Gambar 5.12 Github Actions Workflow

Sebagai ilustrasi, gambar ini menunjukkan konfigurasi Github Action Workflows dengan tujuan pengujian unit menggunakan PHP yang mengikuti template alur kerja yang disediakan oleh GitHub. Namun, secara teoretis, untuk konteks buku ini, penggunaan template GitHub tersebut tidak diperlukan karena kompleksitas aplikasi yang dibahas belum setinggi kebanyakan template GitHub.

The screenshot shows the GitHub Actions workflow editor interface. At the top, there's a header with a gear icon, the repository name "hello-world / .github / workflows /", the file name "php.yml", and a "master" branch indicator. Below the header is a toolbar with "Edit" (selected), "Preview", and a copy icon. The main area contains the workflow YAML code:

```
2
3   on:
4     push:
5       branches: [ "master" ]
6     pull_request:
7       branches: [ "master" ]
8
9   permissions:
10    contents: read
11
12   jobs:
13     build:
14
15       runs-on: ubuntu-latest
16
17       steps:
18         - uses: actions/checkout@v4
19
20         - name: Validate composer.json and composer.lock
21           run: composer validate --strict
22
```

Gambar 5.13 Sample workflow PHP

GitHub Actions menggunakan format **YAML (YAML Ain't Markup Language)** untuk mendefinisikan workflows. YAML adalah *human-readable data serialization standard* yang menggunakan indentasi untuk menunjukkan struktur hierarki. Template dasar workflow GitHub Actions terdiri dari beberapa komponen mandatory dan optional yang harus dipahami sebelum membuat workflow.

5.3.1 Contoh Workflow Github Actions

Langkah awal konfigurasi alur kerja GitHub Actions adalah memastikan proyek sudah diunggah ke GitHub atau menggunakan repositori yang sudah ada (di-clone). Proses konfigurasi ini akan lebih mudah dilakukan dengan menggunakan editor seperti Visual Studio Code.

Dengan memanfaatkan repositori **hello-world** dari modul sebelumnya, buatlah sebuah workflow sederhana untuk menguji fungsionalitas GitHub Actions. Alur kerja ini hanya akan mencetak teks menggunakan perintah "echo", bertujuan untuk memberikan pemahaman dasar mengenai struktur alur kerja GitHub Actions.

Buka dengan code editor, pada contoh berikut kode akan dibuka dengan VSCode. Untuk membuka VSCode menggunakan WSL, masuk ke direktori yang ingin dibuka lalu tulis kode berikut:

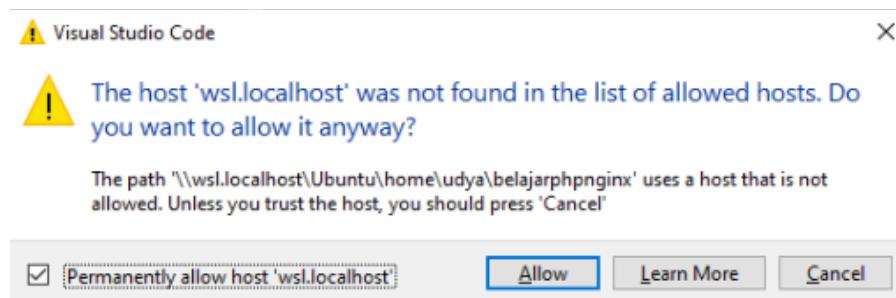
```
code .
```

Perhatikan gambar di bawah ini.

```
udy@DESKTOP-9Q85VU9:~$ git clone https://github.com/mhudayaramadhana/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 19 (delta 6), reused 10 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (19/19), 4.04 KiB | 590.00 KiB/s, done.
Resolving deltas: 100% (6/6), done.
udy@DESKTOP-9Q85VU9:~$ code .
udy@DESKTOP-9Q85VU9:~$ cd hello-world/
udy@DESKTOP-9Q85VU9:~/hello-world$ code .
```

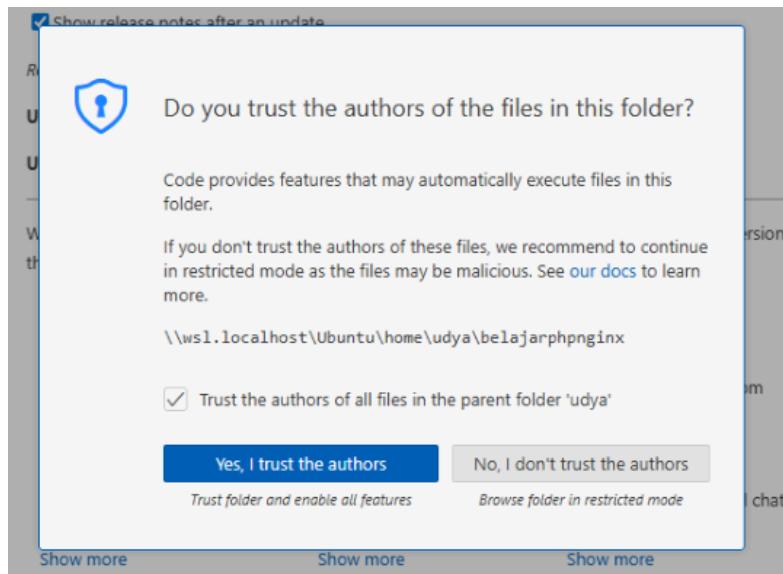
Gambar 5.14 Membuka VSCode di WSL

Apabila ada permintaan izin untuk membuka folder Linux pada VSCode Windows, maka ceklis **Permanently allow host 'wsl.localhost'** lalu klik **allow**.



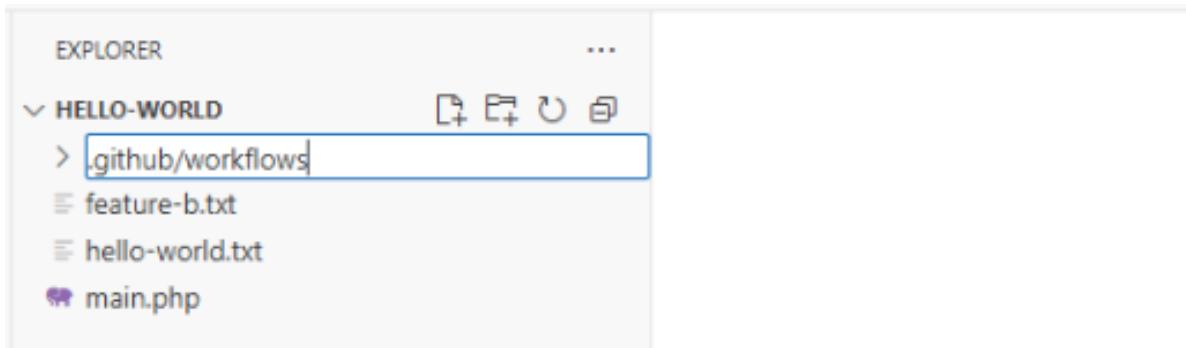
Gambar 5.15 Popup Perizinan wsl.localhost

Kemudian sesaat setelah VSCode berhasil dibuka, akan ada popup lagi yaitu "**Do you trust the authors of the files in this folder?**", centrang kolom **Trust the authors** lalu klik **Yes, I trust the authors**.



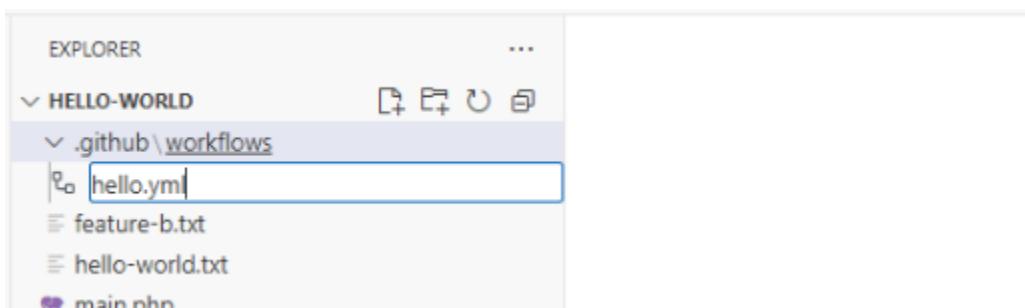
Gambar 5.16 Popup konfirmasi perizinan VSCode

Sekarang buatlah folder baru bernama `.github` lalu buat folder lagi di dalamnya bernama `workflows`, perhatikan gambar di bawah ini:



Gambar 5.17 Pembuatan folder Workflows

Nama folder bisa langsung ditulis `.github/workflows` yang artinya akan ada folder bernama `workflows` di dalam folder `.github`. Buatlah sebuah file baru bernama `hello.yml` pada folder `workflows` lalu masukkan kode berikut:



Gambar 5.18 Pembuatan file hello.yml

```
name: Hello World Workflow

on:
  push:
    branches: [ master ] # Pastikan nama branch sesuai!

jobs:
  greeting:
    runs-on: my-laptop # Ganti dengan label yang dibuat

    steps:
      - name: Say Hello
        run: echo "Hello, World! 🙌"

      - name: Show Date and Time
```

```
run: echo "Current time is $(date) ⏳"  
  
- name: Show GitHub Info  
  run: |  
    echo "Repository: ${{ github.repository }}"  
    echo "Branch: ${{ github.ref }}"  
    echo "Commit: ${{ github.sha }}"
```

5.3.2 Struktur dan Komponen Workflow

Workflow GitHub Actions yang ditunjukkan di atas merupakan contoh sederhana namun lengkap untuk memahami bagaimana automation bekerja menggunakan self-hosted runner di laptop/komputer lokal mahasiswa. Mari kita breakdown setiap bagian untuk pemahaman yang lebih mendalam.

5.3.2.1. Metadata Workflow

Bagian **name** mendefinisikan nama workflow yang akan ditampilkan di GitHub Actions interface. Nama ini akan muncul di tab "Actions" pada repository GitHub dan membantu mengidentifikasi workflow ketika memiliki multiple workflows dalam satu repository. Gunakan nama yang descriptive dan meaningful untuk memudahkan management.

5.3.2.2. Trigger Configuration (on:)

Section **on** menentukan **event triggers** yang akan mengaktifkan workflow ini. Dalam contoh ini, workflow akan dijalankan secara otomatis setiap kali ada **push** ke branch **master**. Ini adalah trigger paling umum dalam CI/CD pipeline Devops Project.

Penjelasan:

- **push**: Event yang terjadi ketika developer melakukan **git push** ke repository
- **branches: [master]**: Membatasi trigger hanya pada branch master
- Jika branch di repository adalah **main**, maka harus diganti menjadi **branches: [main]**

Workflow ini TIDAK akan berjalan jika:

- Push dilakukan ke branch lain (misal: develop, feature-branch)
- Hanya melakukan commit tanpa push
- Melakukan pull request tanpa merge

5.3.2.3. Jobs Definition

Section `jobs` mendefinisikan satu atau lebih pekerjaan yang akan dieksekusi. Setiap job berisi serangkaian steps yang dijalankan secara sequential.

Komponen Jobs:

- **greeting:** Nama job yang sedang didefinisikan (bisa diganti sesuai kebutuhan)
- **runs-on: my-laptop:** Menentukan **runner** yang akan mengeksekusi job ini

Penting tentang runs-on:

- `my-laptop` adalah **label** yang diberikan saat setup self-hosted runner
- Label ini harus **exactly match** dengan yang dikonfigurasi pada runner
- Untuk GitHub hosted runners, biasanya menggunakan: `ubuntu-latest`, `windows-latest`, atau `macos-latest`
- Untuk self-hosted runner, gunakan label custom yang telah dibuat saat instalasi

5.3.2.4. Steps Execution

Steps adalah individual tasks yang dieksekusi dalam job. Setiap step berjalan secara berurutan (sequential) dan menggunakan working directory yang sama.

Step 1: Say Hello

Step pertama ini adalah greeting sederhana yang mendemonstrasikan basic command execution:

- **name:** Label descriptive untuk step ini yang akan muncul di logs
- **run:** Command yang akan dieksekusi di shell/terminal runner
- `echo` adalah command Linux/Unix untuk menampilkan output ke console
- Emoji  dapat digunakan untuk membuat logs lebih friendly dan readable

Output yang dihasilkan:

Hello, World! 

Step 2: Show Date and Time

Step ini mendemonstrasikan penggunaan **command substitution** dalam shell:

- `$(date)` adalah syntax untuk menjalankan command `date` dan menggunakan outputnya
- Perintah `date` menampilkan current date and time dari sistem runner
- Ini berguna untuk tracking kapan workflow dijalankan

Output yang dihasilkan (contoh):

Current time is Tue Sep 30 14:30:45 WIB 2025 

Step 3: Show GitHub Info

Step ini mendemonstrasikan penggunaan **GitHub context variables** dan **multi-line commands**:

Multi-line Commands:

- Symbol `|` (pipe) memungkinkan menulis multiple commands dalam satu step
- Setiap line akan dieksekusi secara berurutan
- Lebih readable dibanding menulis semua commands dalam satu line

GitHub Context Variables:

- `${{ github.repository }}`: Menampilkan nama repository dalam format `owner/repo-name`
 - Contoh: `john-doe/my-php-app`
- `${{ github.ref }}`: Menampilkan full reference dari branch atau tag
 - Contoh: `refs/heads/master` untuk branch master
 - Contoh: `refs/tags/v1.0.0` untuk tag
- `${{ github.sha }}`: Menampilkan full SHA commit hash yang memicu workflow
 - Contoh: `a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0`
 - Hash ini adalah unique identifier untuk setiap commit

Output yang dihasilkan (contoh):

Repository: john-doe/php-webapp

Branch: refs/heads/master

Commit: a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0

Workflow Execution Flow

Ketika workflow ini dijalankan, berikut adalah sequence of events yang terjadi:

1. **Event Detection:** Pengembang melakukan `git push` ke branch master
2. **Workflow Trigger:** GitHub mendeteksi push event dan match dengan trigger configuration
3. **Runner Selection:** GitHub mencari available runner dengan label `my-laptop`
4. **Job Execution:** Job "greeting" mulai dieksekusi pada runner yang dipilih
5. **Step 1 Execution:** Command echo pertama dijalankan
6. **Step 2 Execution:** Command date dengan formatting dijalankan
7. **Step 3 Execution:** Multiple echo commands dengan GitHub context variables dijalankan
8. **Completion:** Workflow selesai dan status dilaporkan ke GitHub

Monitoring dan Debugging

Melihat Workflow Execution:

1. Buka repository di GitHub
2. Klik tab "Actions"
3. Pilih workflow run yang ingin dilihat
4. Expand setiap step untuk melihat detailed logs

Indikator Success/Failure:

- Green checkmark: Step berhasil (exit code 0)
- Red X: Step gagal (exit code non-zero)
- Yellow circle: Step sedang running

Common Issues:

- **Runner not found:** Label `my-laptop` tidak match dengan runner configuration
- **Branch mismatch:** Push dilakukan ke branch selain master
- **Runner offline:** Self-hosted runner tidak sedang running
- **Permission issues:** Runner tidak memiliki permission untuk execute commands

Best Practices

1. **Descriptive Naming:** Gunakan nama yang jelas untuk workflow, jobs, dan steps
2. **Comments:** Tambahkan comments untuk explain complex logic
3. **Error Handling:** Selalu test workflow dengan simple commands dulu
4. **Incremental Development:** Build workflow secara bertahap, test setiap addition
5. **Version Control:** Commit workflow changes dengan descriptive messages

Workflow sederhana ini menjadi foundation untuk memahami konsep GitHub Actions sebelum masuk ke workflows yang lebih complex seperti building Docker images atau deploying applications.

Apabila sudah dibuat, sekarang *commit* lalu *push*. Pastikan nama label sesuai dengan label yang dibuat pada runners.

Runners	Status
my-laptop self-hosted Linux X64 my-laptop	● Offline ...

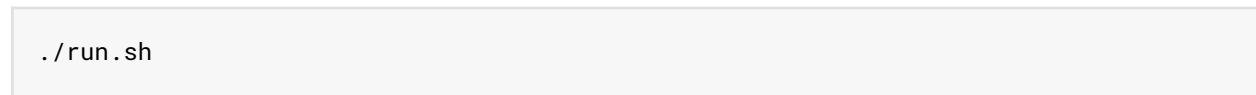
Gambar 5.19 Runner yang sudah dimatikan

Tulisan **my-laptop** di samping ikon adalah nama runner, dan tulisan **self-hosted,linux,X64,my-laptop** di dalam badge adalah **label**. Pastikan nama pada **runs-on** berisi salah satu nilai dari label runner.

The screenshot shows the GitHub Actions interface. At the top, there are navigation links: Actions, Projects, Wiki, Security, Insights, and Settings. Below this, a search bar says "Filter workflow runs". The main area is titled "All workflows" and shows "Showing runs from all workflows". A single workflow run is listed under "1 workflow run": "Hello World Workflow #1: Commit 787b886 pushed by mhudyaramadhan". It has a status of "master", was triggered "1 minute ago", and is currently "Queued". There are dropdown menus for Event, Status, Branch, and Actor, and a three-dot menu icon.

Gambar 5.20 Proses Berjalan Workflow

Setelah melakukan *commit* dan *push*, buka halaman *actions*. Pada halaman workflow akan terlihat *workflow run* dengan status *queued*. Hal ini terjadi karena *action-runner* pada komputer lokal belum berjalan secara otomatis dan perlu diinisiasi secara manual. Untuk melakukannya, pergi ke folder *action-runners* dan jalankan perintah yang telah disediakan.



Perhatikan gambar berikut:

```
udya@DESKTOP-9Q85VU9: ~/actions-runner
udya@DESKTOP-9Q85VU9:~$ cd actions-runner/
udya@DESKTOP-9Q85VU9:~/actions-runner$ ./run.sh

✓ Connected to GitHub

Current runner version: '2.328.0'
2025-09-30 05:19:19Z: Listening for Jobs
2025-09-30 05:19:25Z: Running job: greeting
```

Gambar 5.21 Action Runners yang dijalankan

Setelah dijalankan dan sambil menunggu bisa dilihat pada halaman *actions* statusnya berubah menjadi **in progress**. Klik *detail* dari *workflow* untuk melihat proses secara rinci.

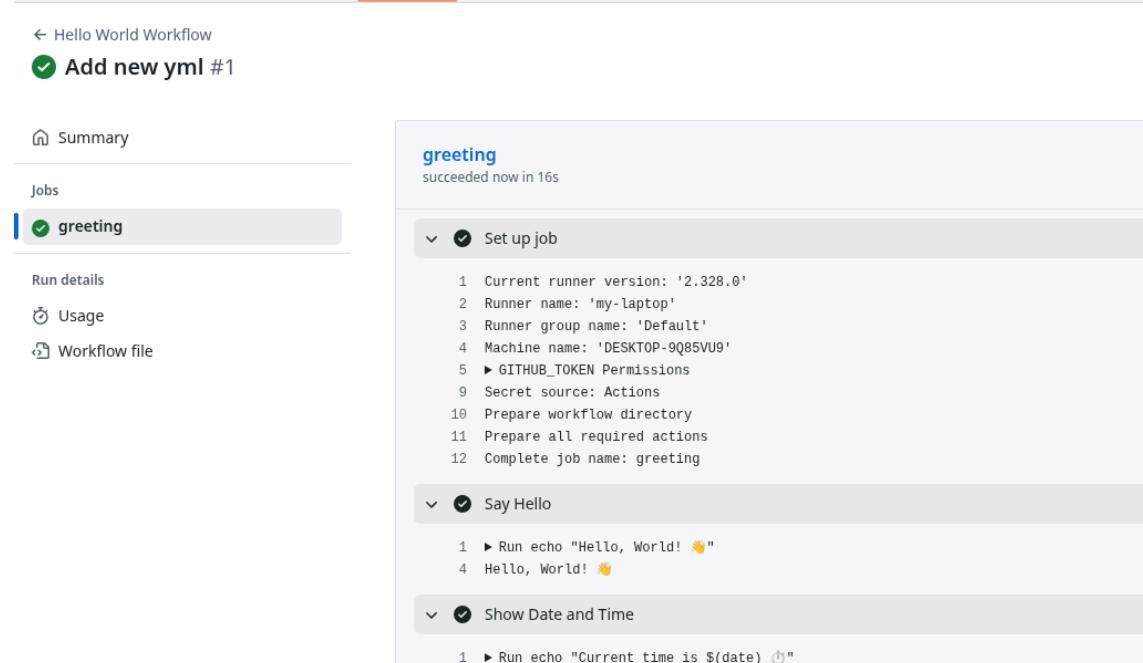
The screenshot shows the GitHub Actions interface for the "Hello World Workflow". It displays a summary of the workflow run, which was triggered via push 3 minutes ago by mhudyaramadhan on branch master. The status is "In progress" with a total of 2 jobs. One job, "greeting", is listed under "Jobs". Below the summary, there are sections for "Run details" (hello.yml on: push) and "Usage".

Gambar 5.22 Halaman action runners

Nantinya, hasil akhir dari setiap *workflow* akan menyajikan laporan lengkap mengenai langkah-langkah yang telah dieksekusi selama proses berlangsung. Laporan ini berfungsi sebagai catatan audit yang transparan, memungkinkan tim untuk memahami secara detail setiap aksi yang dilakukan oleh sistem. Sebagai contoh, pada ilustrasi di atas, *workflow* tersebut dirancang untuk melakukan beberapa tugas dasar, yaitu:

1. **Mencetak Tulisan "Hello World"**: Langkah ini sering digunakan sebagai uji coba awal untuk memastikan bahwa sistem dapat menjalankan perintah dasar dan berinteraksi dengan *output* konsol. Ini juga merupakan indikator bahwa lingkungan eksekusi telah diatur dengan benar.
2. **Menampilkan Tanggal Saat Ini**: Tugas ini berfungsi untuk mencatat waktu eksekusi *workflow* secara akurat. Informasi tanggal dan waktu sangat krusial untuk pelacakan, *logging*, dan pemecahan masalah, terutama dalam lingkungan pengembangan yang dinamis.
3. **Menampilkan Hasil Commit Terakhir**: Langkah ini mengintegrasikan *workflow* dengan sistem kontrol versi seperti Git. Dengan menampilkan detail dari *commit* terakhir, tim dapat segera mengetahui versi kode mana yang sedang diproses atau di-deploy. Informasi ini meliputi *hash commit*, pesan *commit*, nama penulis, dan tanggal *commit*, yang semuanya esensial untuk menjaga konsistensi dan integritas kode dalam proyek.

Secara keseluruhan, laporan akhir ini tidak hanya menjelaskan apa yang telah dilakukan, tetapi juga memberikan konteks penting yang membantu tim dalam memverifikasi keberhasilan setiap tahapan, mengidentifikasi potensi masalah, dan memastikan bahwa setiap perubahan telah diimplementasikan sesuai dengan harapan. Ini adalah komponen kunci dalam praktik DevOps yang efektif, dimana otomatisasi dan transparansi menjadi prioritas utama.



Gambar 5.23 Workflow yang berhasil

5.4 Github Actions Runner untuk Pemula

Meskipun GitHub Actions Runner mudah digunakan bagi pengembang pemula, kekurangannya adalah pembatasan penggunaan runner per-proyek untuk setiap akun. Untuk menjalankan runner pada beberapa proyek sekaligus, dibutuhkan sebuah organisasi. Namun, untuk tujuan pembelajaran, hal ini tidak perlu dilakukan. Fokus utama adalah memahami struktur pembuatan *workflow* dan potensi yang bisa dikembangkan dengan *workflow action runner*.

Kelebihan:

- 1. Mudah untuk Memulai (Low Barrier to Entry)** GitHub Actions memiliki learning curve yang relatif mudah dibandingkan CI/CD tools lainnya. Pengembang pemula tidak perlu setup server atau infrastructure yang kompleks - cukup membuat file YAML di repository dan *workflow* sudah bisa berjalan. Integrasi dengan GitHub yang sudah familiar membuat proses pembelajaran menjadi lebih mudah tanpa perlu mempelajari platform baru.
- 2. Gratis untuk Pembelajaran** GitHub menyediakan free minutes yang cukup untuk *public repositories*, dan pengembang pemula bisa menggunakan *self-hosted runner* pada laptop pribadi tanpa biaya sama sekali. Ini memungkinkan pengembang pemula untuk melakukan eksperimen dan belajar sebanyak mungkin tanpa khawatir tentang biaya, berbeda dengan CI/CD platforms lain yang memiliki *limited free tier*.
- 3. Langsung Applicable untuk Dunia Kerja** GitHub Actions adalah tool yang widely used di industry, sehingga memahami actions runner dapat digunakan pada dunia profesional. Pengalaman dengan *workflow automation* dan *CI/CD concepts* menjadi *valuable portfolio items* yang dicari oleh *employer*, memberikan *real-world value* dari pembelajaran di kampus.

Kekurangan:

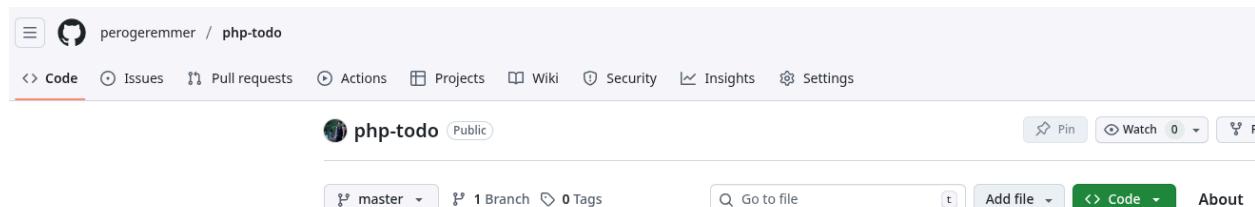
- 1. Pembatasan Runner per Proyek** Meskipun GitHub Actions Runner mudah digunakan bagi pengembang pemula, kekurangannya adalah pembatasan penggunaan runner per-proyek untuk setiap akun. Self-hosted runner hanya dapat terdaftar pada satu repository pada satu waktu. Untuk menjalankan runner pada beberapa proyek sekaligus, dibutuhkan sebuah organisasi.
- 2. YAML Syntax and Debugging yang Challenging** YAML format yang digunakan GitHub Actions sangat sensitive terhadap indentation dan syntax. Mahasiswa yang baru pertama kali menggunakan YAML sering mengalami kesulitan dengan formatting errors yang tidak immediately obvious. Ketika workflow gagal, debugging juga challenging karena tidak bisa direct access ke runner environment - hanya bisa rely pada logs, yang kadang tidak cukup detail untuk identify root cause masalah.
- 3. Maintenance and Network Dependency** Self-hosted runners memerlukan laptop/komputer pengembang untuk always running and connected to internet. Jika laptop shutdown, hibernate, atau kehilangan koneksi internet, workflows tidak dapat execute.

Namun, untuk tujuan pembelajaran, hal-hal di atas tidak perlu menjadi penghalang. **Fokus utama adalah memahami struktur pembuatan workflow dan potensi yang bisa dikembangkan dengan workflow action runner.** Kelemahan-kelemahan yang disebutkan lebih merupakan considerations untuk penggunaan *production*, bukan batasan untuk dalam mempelajari CI/CD. Dengan mindset pembelajaran yang tepat, keuntungan GitHub Actions untuk pengembang pemula melebihi kelemahannya dalam membangun fondasi tentang praktik DevOps pada sebuah proyek.

5.5 Kolaborasi Pengembang dengan Actions Runner

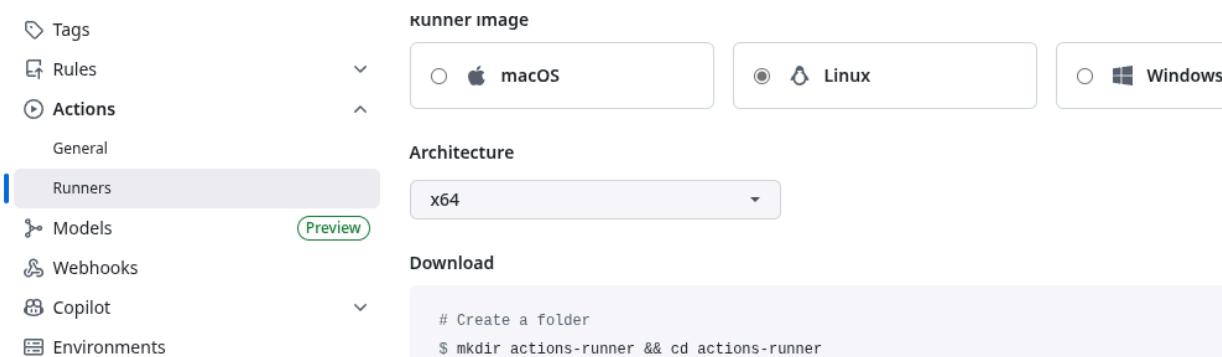
Github Actions Runner memfasilitasi kolaborasi antar pengembang. Dalam skenario ini, Pengembang A memiliki repositori dan mengundang Pengembang B sebagai kolaborator. Apabila ingin melihat proyeknya, maka bisa diakses pada URL berikut:

<https://github.com/perogeremmer/php-todo>



Gambar 5.24 Contoh Repositori

Karena Pengembang B tidak memiliki akses untuk membuat *self-hosted runner*, Pengembang A dapat memberikan akses melalui *runner* yang akan dipasang pada komputer lokal. Semua kode pada bagian **Download** dan **Configure** harus disalin oleh pengembang A dan diberikan kepada Pengembang B.

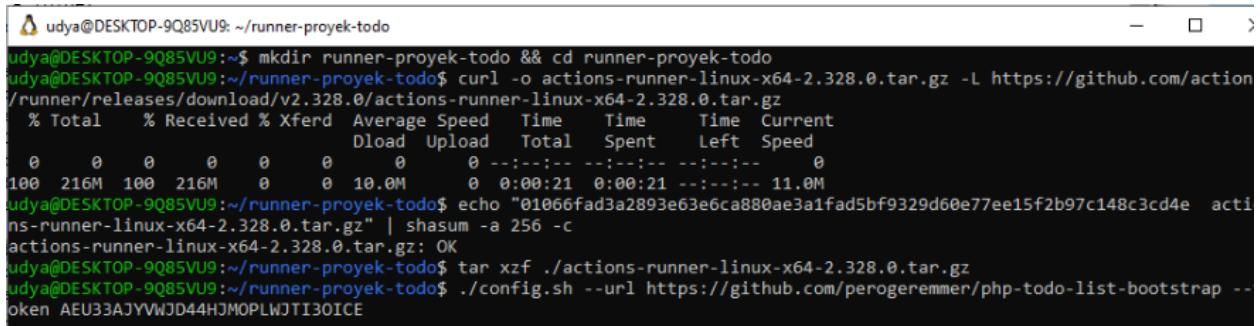


Gambar 5.25 Proses Pembuatan Runner

Perlu diingat apabila komputer lokal pengembang B sudah memiliki Actions Runner sebelumnya, maka pada baris pertama bagian **mkdir actions-runner** harus diganti dengan nama folder lain, misalnya **runner-proyek-todo**. Sehingga kodenya menjadi sebagai berikut:

```
mkdir runner-proyek-todo && cd runner-proyek-todo
```

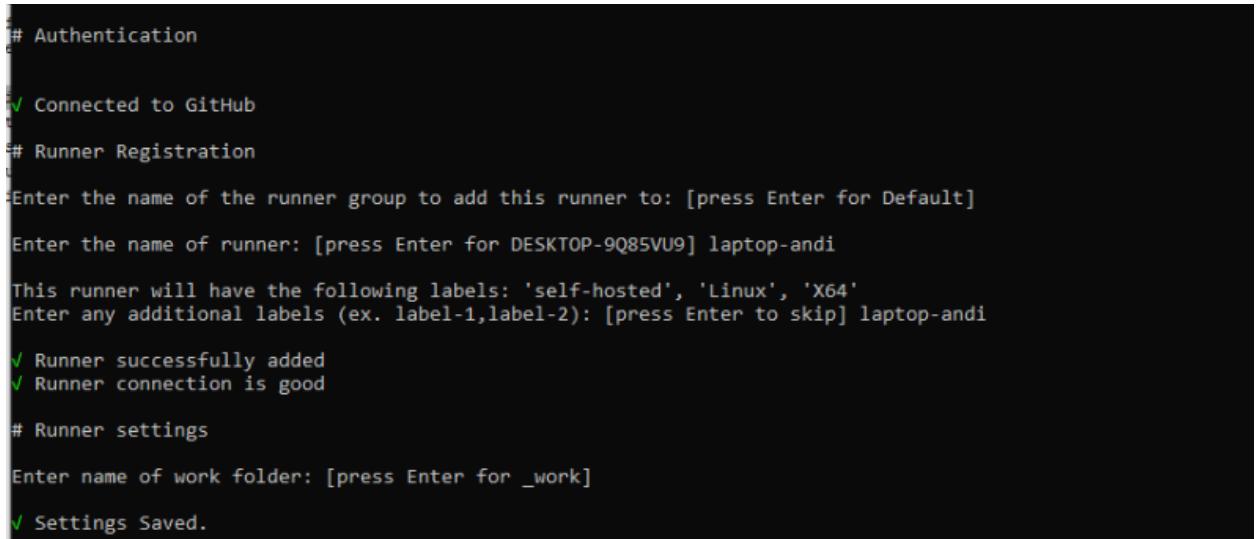
Perhatikan gambar berikut, untuk konfigurasinya masih sama seperti modul sebelumnya, yang membedakan hanyalah nama foldernya saja.



```
udya@DESKTOP-9Q85VU9:~/runner-proyek-todo
udya@DESKTOP-9Q85VU9:~/runner-proyek-todo$ curl -o actions-runner-linux-x64-2.328.0.tar.gz -L https://github.com/actions/runners/releases/download/v2.328.0/actions-runner-linux-x64-2.328.0.tar.gz
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total Spent   Left Speed
  0     0    0     0       0      0   0:00:21  0:00:21    0:00:21  0
100  216M  100  216M    0     0  10.0M      0  0:00:21  0:00:21    0:00:21  11.0M
udya@DESKTOP-9Q85VU9:~/runner-proyek-todo$ echo "01066fad3a2893e63e6ca880ae3a1fad5bf9329d60e77ee15f2b97c148c3cd4e" | shasum -a 256 -c
actions-runner-linux-x64-2.328.0.tar.gz: OK
udya@DESKTOP-9Q85VU9:~/runner-proyek-todo$ tar xzf ./actions-runner-linux-x64-2.328.0.tar.gz
udya@DESKTOP-9Q85VU9:~/runner-proyek-todo$ ./config.sh --url https://github.com/perogeremmer/php-todo-list-bootstrap --token AEU33AJYWWJD44HJMOPLWJTI3OICE
```

Gambar 5.26 Proses pembuatan runner baru

Saat mendaftarkan *GitHub Action Runner*, pastikan untuk memberi nama *runner* dengan jelas. Misalnya, jika pengembang bernama Andi, nama *runner* dan labelnya adalah **laptop-andi**. Hal ini akan memudahkan proses identifikasi ketika runner berjalan.



```
# Authentication
✓ Connected to GitHub
# Runner Registration
Enter the name of the runner group to add this runner to: [press Enter for Default]
Enter the name of runner: [press Enter for DESKTOP-9Q85VU9] laptop-andi
This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip] laptop-andi
✓ Runner successfully added
✓ Runner connection is good
# Runner settings
Enter name of work folder: [press Enter for _work]
✓ Settings Saved.
```

Gambar 5.27 Registrasi runner baru

Apabila berhasil maka runner akan berisi nama runner dari **laptop-andi**.

Runners

New self-hosted runner

Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. [Learn more about self-hosted runners.](#)

Runners	Status
 laptop-andi self-hosted Linux X64 laptop-andi	● Offline ...

Gambar 5.28 Penambahan runner baru

Sekarang ketika pengembang A (Hudya) melakukan push maka pengembang B (Andi) dimana runnernya bernama **laptop-andi** akan mendapatkan permintaan untuk melakukan proses CI/CD. Perhatikan kode **workflows.yml** pada repositori **php-todo** di atas:

```
# Workflow GitHub Actions untuk Auto Deploy PHP Todo App
# Workflow ini akan berjalan otomatis ketika ada push ke branch master
# Fungsinya: pull kode terbaru dan rebuild semua Docker container

name: CI/CD Auto Pull and Rebuild

# Kapan workflow ini akan dijalankan:
# 1. Ketika ada push ke branch master (otomatis)
# 2. Manual trigger melalui GitHub Actions tab
on:
  push:
    branches: [ master ] # Hanya berjalan ketika push ke master
    workflow_dispatch: # Bisa dijalankan manual juga

jobs:
  deploy-andi:
    runs-on: [laptop-andi]

    steps:
      # Step 0: Detect project path dynamically
      - name: Set project path
        run: |
          PROJECT_PATH=$(find /home -name 'php-todo' -type d 2>/dev/null | head -1)
          if [ -z "$PROJECT_PATH" ]; then
            echo "✖ Error: Folder php-todo tidak ditemukan di /home"
            echo "Pastikan project sudah di-clone ke salah satu user directory"
            exit 1
          fi
```

```

echo "PROJECT_PATH=$PROJECT_PATH" >> $GITHUB_ENV
echo "✅ Found project at: $PROJECT_PATH"

# Step 1: Ambil kode terbaru dari branch master
- name: Pull latest changes
  working-directory: ${{ env.PROJECT_PATH }}
  run: |
    git fetch origin master      # Download update terbaru
    git reset --hard origin/master # Reset ke versi terbaru (hapus perubahan
lokal)

# Step 2: Stop semua container yang sedang berjalan
- name: Stop existing containers
  working-directory: ${{ env.PROJECT_PATH }}
  run: |
    docker compose down || true # Stop container (|| true artinya lanjut
walaupun error)

# Step 3: Hapus image lama untuk menghemat storage
- name: Remove old images
  working-directory: ${{ env.PROJECT_PATH }}
  run: |
    docker system prune -f    # Hapus container/network yang tidak terpakai
    docker image prune -a -f # Hapus semua image yang tidak terpakai

# Step 4: Build ulang dan jalankan semua container
- name: Build and start containers
  working-directory: ${{ env.PROJECT_PATH }}
  run: |
    docker compose build --no-cache # Build image baru tanpa cache
    docker compose up -d           # Jalankan container di background

# Step 5: Tunggu sampai semua service siap
- name: Wait for services to be ready
  working-directory: ${{ env.PROJECT_PATH }}
  run: |
    sleep 30                      # Tunggu 30 detik
    docker compose ps              # Lihat status semua container

# Step 6: Test apakah aplikasi berjalan dengan baik
- name: Health check
  run: |
    curl -f http://localhost:8080 || exit 1 # Test akses ke aplikasi

# Step 7: Kasih tahu hasil deployment

```

```

- name: Notify deployment status
  if: always() # Selalu jalankan step ini (sukses atau gagal)
  run: |
    if [ ${{ job.status }} == 'success' ]; then
      echo "✅ Deployment berhasil di laptop-andi! Aplikasi sudah update ke
versi terbaru"
    else
      echo "❌ Deployment gagal di laptop-andi! Cek log error di atas"
    fi

```

Pada file `workflows.yml`, terdapat dua blok `deploy` yang identik, yaitu `deploy-andi` dan `deploy-hudya`. Penggunaan dua blok ini dimaksudkan agar setiap kode dapat dibangun (build) dan diterapkan (deploy) pada komputer lokal yang berbeda. Dalam skenario nyata, hal ini merepresentasikan lingkungan server pengembangan (development) dan server produksi (production).

Runners

New self-hosted runner

Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. [Learn more about self-hosted runners.](#)

Runners	Status
laptop-andi <small>self-hosted Linux X64 laptop-andi</small>	● Active ...
laptop-hudya <small>self-hosted Linux X64 laptop-hudya</small>	● Active ...

Gambar 5.29 Dua runner yang aktif

Kemudian pastikan masing-masing pengembang sudah memasang actions runner. Pengembang A (Hudya) sebagai pemilik repositori juga memasang runners pada komputer lokal.

Selanjutnya masing-masing pengembang harus menjalankan kode dengan kode `./run.sh` pada direktori runner komputer lokal. Hal ini bertujuan untuk menangkap perintah dari Github Actions dalam melakukan proses CI/CD

```

udyah@DESKTOP-9Q85VU9:~$ cd runner-proyek-todo/
udyah@DESKTOP-9Q85VU9:~/runner-proyek-todo$ ./run.sh

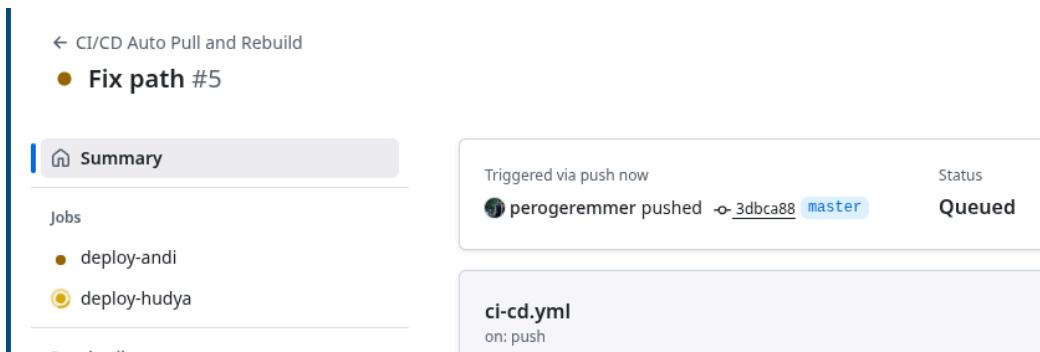
✓ Connected to GitHub

Current runner version: '2.328.0'
2025-09-30 08:34:12Z: Listening for Jobs
2025-09-30 08:34:16Z: Running job: deploy-andi
2025-09-30 08:40:27Z: Job deploy-andi completed with result: Succeeded

```

Gambar 5.30 Proses listening job pada runner laptop andi

Setelah dijalankan, maka masing-masing komputer pengembang bisa terlihat proses CI/CD-nya melalui menu **actions** pada repositori Github.



Gambar 5.31 Halaman Workflows pada Github

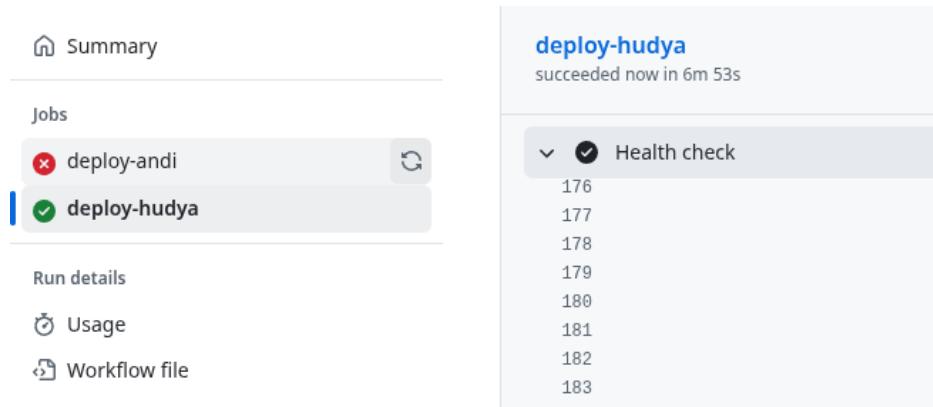
Ketika pengembang A atau pengembang B melakukan *commit* dan *push* ke branch master, secara otomatis *workflows* akan mendeteksi bahwa kode akan dijalankan pada label *laptop-andi* dan *laptop-hudya*.

This screenshot shows a failed workflow run titled 'Fix project path for both deploy #4'. On the left, there's a sidebar with 'Summary', 'Jobs' (showing 'deploy-andi' failed and 'deploy-hudya' pending), 'Run details', 'Usage', and 'Workflow file'. The main area shows the 'deploy-andi' job status as 'failed now in 21s'. It details the steps: 'Set up job' (success), 'Set project path' (failure), and three log entries. The first entry is a note about finding 'php-todo'. The second entry is an error message: 'Error: Folder php-todo tidak ditemukan di /home'. The third entry is a note about ensuring the project is cloned to a user directory. The final entry is an error message: 'Error: Process completed with exit code 1'.

Gambar 5.32 Proses Gagal pada Github Action Runners

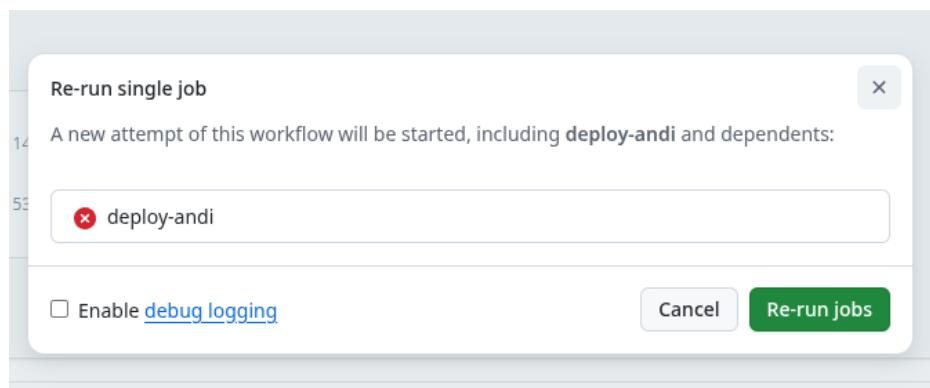
Ketika proses *build* dan *deploy* aplikasi dilakukan dalam alur DevOps, seringkali kita dihadapkan pada kemungkinan terjadinya kegagalan. Kegagalan ini bisa bersumber dari berbagai tahap, baik pada saat *build*, *deploy*, atau bahkan pada salah satu atau kedua *runner* yang bertugas menjalankan perintah-perintah tersebut. Penting untuk melakukan pemantauan yang cermat untuk mengidentifikasi akar masalah dengan cepat.

Gambar di atas menunjukkan kegagalan *workflow* "deploy-andi". Penelusuran mengungkap penyebabnya: komputer Andi belum meng-*clone* projek **php-todo**, bagian integral dari sistem yang akan di-*deploy*. Dalam `workflows.yml`, langkah pertama validasi memeriksa keberadaan folder **php-todo** di **/home** (direktori *home* pengguna di WSL). Kegagalan validasi ini berarti repositori proyek tidak ditemukan, menghentikan *workflow* dan menyebabkan kegagalan *deploy*. Ini menekankan pentingnya persiapan *runner* sebelum melakukan *build* atau *deploy*.



Gambar 5.33 Tombol re-run proses workflows

Apabila terjadi kegagalan, maka runner yang mengalami kegagalan bisa diulang tanpa harus mengulang keseluruhan proses pada komputer / server lain.



Gambar 5.34 Popup konfirmasi single job

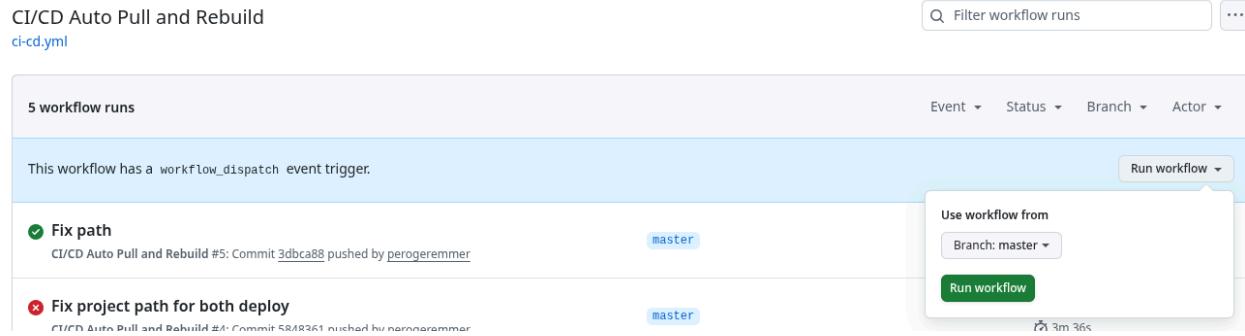
Pada saat menjalankan CI/CD yang gagal kembali, pengembang bisa mengaktifkan debug logging untuk melihat proses yang terjadi secara *realtime* pada saat proses CI/CD berjalan.

Setelah proses berhasil diselesaikan pada masing-masing komputer pengembang, secara alur *workflows* maka container akan dijalankan secara otomatis.

```
dochudya@perogeremmer-pc:~/actions-runner$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
NAMES
72b1c4f725e1   php-todo-nginx      "/docker-entrypoint..."  42 seconds ago
tcp             php-todo-nginx
fb9de62804b   phpmyadmin/phpmyadmin:latest  "/docker-entrypoint..."  42 seconds ago
tcp             php-todo-phpmyadmin
3d34a8815334   php-todo-php      "docker-php-entrypoi..."  42 seconds ago
php-todo-php
cac1c346c3aa   mysql:8.0        "docker-entrypoint.s..."  43 seconds ago
306/tcp, 33060/tcp  php-todo-mysql
```

Gambar 5.35 Tampilan list container setelah workflow berjalan

Selanjutnya, alur kerja yang telah dibuat dapat dipicu secara manual, tidak hanya bergantung pada proses *push* ke *branch master*, berkat adanya perintah *dispatch workflow*. Fitur ini memberikan fleksibilitas lebih bagi pengembang untuk menguji dan menerapkan perubahan tanpa harus selalu melakukan *commit* ke *branch* utama. Dengan begitu, proses pengembangan menjadi lebih efisien dan memungkinkan iterasi yang lebih cepat dalam siklus pengembangan perangkat lunak.



Gambar 5.36 Tombol Manual Workflow

5.6 Menjalankan Actions Runner Secara Otomatis

Pada modul sebelumnya, *Actions Runner* dapat dijalankan secara manual melalui eksekusi perintah `./run.sh`. Proses ini memungkinkan pengembang untuk menguji dan memverifikasi alur kerja secara langsung. Namun, dalam lingkungan pengembangan dan produksi yang lebih dinamis, menjalankan Runner secara manual setiap kali diperlukan bukanlah pendekatan yang efisien.

Terdapat mekanisme konfigurasi yang membuat Actions Runner untuk berjalan secara otomatis. Dengan mengonfigurasi Runner agar berjalan secara otomatis, setiap kali ada perubahan kode yang didorong ke repositori atau *trigger* lain yang telah ditentukan, Runner akan secara otomatis mengambil tugas, menjalankan alur kerja (misalnya, pengujian, pembangunan, deployment), dan melaporkan hasilnya tanpa intervensi manual. Ini tidak hanya menghemat waktu dan sumber daya, tetapi juga mengurangi potensi kesalahan manusia dan memastikan konsistensi dalam setiap siklus pengembangan.

Sebagai catatan, actions runner yang dikonfigurasi untuk berjalan otomatis lebih baik dijalankan pada server khusus yang ditugaskan untuk selalu build dan deploy agar mendapat kode terbaru yang dapat diakses oleh seluruh tim, misalnya **server development**.

Untuk mendaftarkan runner sebagai service otomatis yang berjalan setiap WSL menyala, masukkan perintah berikut pada direktori action runner yang ingin dijalankan otomatis:

```
sudo ./svc.sh install
```

Nantinya *github runner* akan dipasang sebagai service otomatis yang berjalan ketika WSL dinyalakan.

```
udya@DESKTOP-9Q85VU9:~/actions-runner
[udo@DESKTOP-9Q85VU9:~/actions-runner$ sudo ./svc.sh install
[sudo] password for udya:
Creating launch runner in /etc/systemd/system/actions.runner.mhudyaramadhana-hello-world.my-laptop.
Run as user: udya
Run as uid: 1000
gid: 1000
Created symlink /etc/systemd/system/multi-user.target.wants/actions.runner.mhudyaramadhana-hello-wor
→ /etc/systemd/system/actions.runner.mhudyaramadhana-hello-world.my-laptop.service.
udo@DESKTOP-9Q85VU9:~/actions-runner$
```

Gambar 5.37 Instalasi service Github Action Runner

Selanjutnya masukkan perintah berikut untuk menjalankan *action runner*:

```
sudo ./svc.sh start
```

Sistem github runner akan dijalankan secara *background*.

```
+ /etc/systemd/system/actions.runner.mhudyaramadhana-hello-world.my-laptop.service.
udo@DESKTOP-9Q85VU9:~/actions-runner$ sudo ./svc.sh start
/etc/systemd/system/actions.runner.mhudyaramadhana-hello-world.my-laptop.service
● actions.runner.mhudyaramadhana-hello-world.my-laptop.service - GitHub Actions Runner (mhudyaramadhana-hello-world.my-laptop)
  Loaded: loaded (/etc/systemd/system/actions.runner.mhudyaramadhana-hello-world.my-laptop.service; enabled; preset: enabled)
  Active: active (running) since Sat 2025-09-27 16:43:34 WIB; 29ms ago
    Main PID: 1365 (runsvc.sh)
      Tasks: 1 (limit: 4687)
     Memory: 860.0K (peak: 864.0K)
        CPU: 5ms
       CGroup: /system.slice/actions.runner.mhudyaramadhana-hello-world.my-laptop.service
               └─1365 /bin/bash /home/udo/actions-runner/runsvc.sh
                  ├─1368 ./externals/node20/bin/node ./bin/RunnerService.js

Sep 27 16:43:34 DESKTOP-9Q85VU9 systemd[1]: Started actions.runner.mhudyaramadhana-hello-world.my-laptop.service...aptop).
Sep 27 16:43:34 DESKTOP-9Q85VU9 runsvc.sh[1365]: .path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/us...
Hint: Some lines were ellipsized, use -l to show in full.
```

Gambar 5.38 Proses memulai service github action runner

Untuk menghapus *service github* yang sudah berjalan otomatis, masukkan perintah berikut:

```
sudo ./svc.sh uninstall
```

6.1 DevOps dengan Kecerdasan Buatan

GitHub, sebagai platform pengembangan perangkat lunak terkemuka, telah mengintegrasikan fitur kecerdasan buatan (AI) canggih yang dikenal sebagai GitHub Copilot. Fitur ini dirancang secara fundamental untuk merevolusi cara pengembang menulis kode, namun perannya meluas jauh melampaui bantuan pengkodean dasar, terutama dalam konteks manajemen repositori dan praktik DevOps.

Secara garis besar, GitHub Copilot berfungsi sebagai asisten kode berbasis AI yang dapat menyarankan baris kode lengkap atau bahkan seluruh fungsi secara real-time. Ini dilakukan dengan menganalisis konteks kode yang sedang ditulis, pola pengkodean yang ada, dan miliaran baris kode yang tersedia secara publik. Manfaat utamanya meliputi:

- **Peningkatan Produktivitas:** Pengembang dapat menulis kode lebih cepat karena Copilot mengotomatiskan tugas-tugas pengkodean yang berulang dan mengurangi waktu yang dihabiskan untuk mencari sintaks atau implementasi.
- **Pengurangan Kesalahan:** Dengan menyarankan kode yang terbukti benar dan mengikuti praktik terbaik, Copilot membantu meminimalkan bug dan kesalahan ketik.
- **Pembelajaran dan Eksplorasi:** Bagi pengembang yang kurang berpengalaman, Copilot dapat menjadi alat pembelajaran yang hebat, membantu mereka memahami cara mengimplementasikan fitur atau pola tertentu. Bagi pengembang berpengalaman, ini dapat memicu ide-ide baru atau menunjukkan cara-cara alternatif untuk mendekati masalah.
- **Konsistensi Kode:** Copilot dapat membantu menjaga konsistensi gaya dan struktur kode di seluruh proyek, yang sangat penting dalam tim pengembangan.

Selain bantuan pengkodean, GitHub Copilot juga menunjukkan potensi signifikan dalam area manajemen repositori dan praktik DevOps, misalnya:

- **Memahami Konteks Repositori:** Copilot dapat dilatih untuk memahami struktur, dependensi, dan tujuan dari sebuah repositori. Ini memungkinkannya untuk memberikan saran yang lebih relevan dan sesuai dengan arsitektur proyek. Misalnya, ketika seorang pengembang ingin menambahkan fitur baru, Copilot dapat menyarankan lokasi terbaik untuk kode tersebut atau dependensi yang mungkin diperlukan berdasarkan pemahaman proyek secara keseluruhan.
- **Pemeriksaan Cela Keamanan Kode (Security Vulnerability Checks):** Dengan kemampuannya menganalisis pola kode, Copilot dapat diintegrasikan untuk mengidentifikasi potensi celah keamanan. Ini dapat mencakup identifikasi pola kode yang rentan terhadap injeksi SQL, *cross-site scripting (XSS)*, atau kesalahan konfigurasi keamanan lainnya. Meskipun bukan pengganti alat keamanan khusus, ini dapat berfungsi sebagai lapisan pertahanan pertama, membantu pengembang menulis kode yang lebih aman sejak awal.
- **Pembuatan Dokumentasi Otomatis:** Berdasarkan analisis kode, Copilot berpotensi untuk menghasilkan draft awal dokumentasi untuk fungsi, kelas, atau modul, mempercepat proses

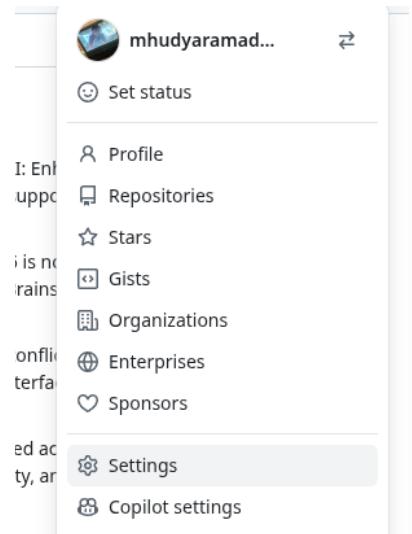
dokumentasi yang seringkali memakan waktu.

- **Bantuan Penulisan Commit Messages:** Dengan menganalisis perubahan yang dibuat dalam kode, Copilot dapat menyarankan *commit messages* yang deskriptif dan informatif, memastikan riwayat proyek tetap bersih dan mudah dipahami.
- **Optimasi Kode dan Refactoring:** Copilot dapat menyarankan cara-cara untuk mengoptimalkan kinerja kode atau melakukan refactoring untuk meningkatkan keterbacaan dan pemeliharaan, berdasarkan praktik terbaik yang dikenal.
- **Manajemen Konfigurasi:** Dalam konteks DevOps, Copilot dapat membantu dalam penulisan skrip untuk otomatisasi infrastruktur atau manajemen konfigurasi, seperti skrip Dockerfile, Kubernetes, atau Terraform, dengan menyarankan sintaks dan praktik terbaik.
- **Analisis Log dan Pemecahan Masalah:** Meskipun masih dalam tahap pengembangan, potensi Copilot untuk membantu menganalisis log kesalahan dan menyarankan solusi untuk masalah operasional adalah bidang yang menarik untuk eksplorasi di masa depan.

Dengan terus berkembangnya kemampuan AI, integrasi fitur seperti GitHub Copilot ke dalam platform seperti GitHub akan terus mengubah metode pengembangan perangkat lunak, tidak hanya dalam kecepatan pengkodean tetapi juga dalam efisiensi manajemen proyek dan implementasi praktik DevOps yang lebih kuat dan aman.

6.2 Aktivasi Github Copilot

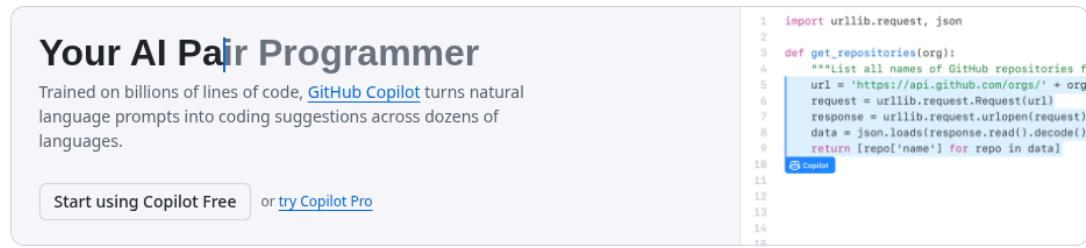
Untuk mengaktifkan Github Copilot, pergi ke pojok kanan menu profil, lalu klik ***copilot settings***.



Gambar 6.1 Menu Navbar Profil

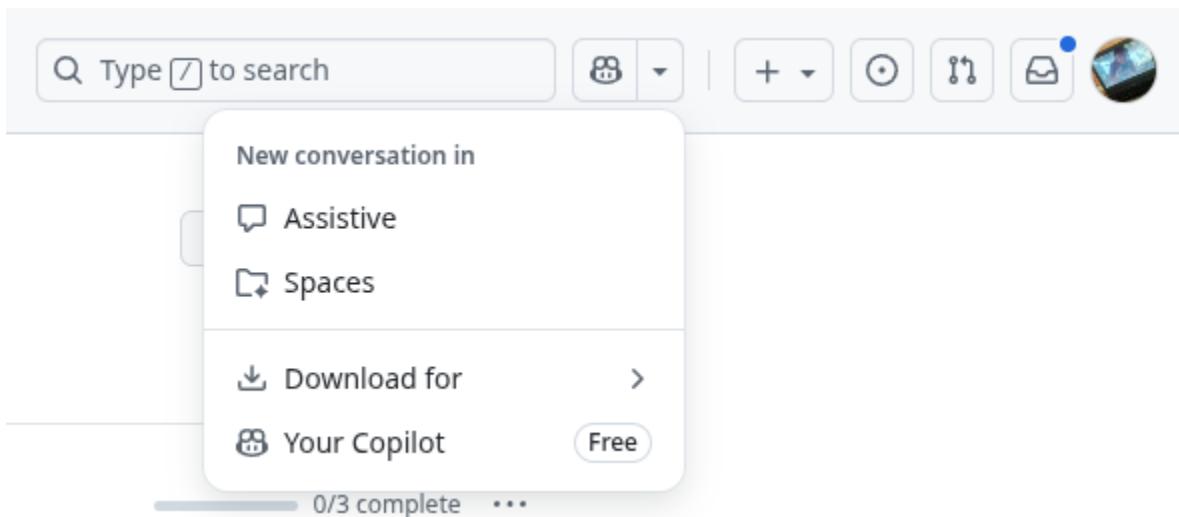
Selanjutnya klik ***Start using Copilot Free*** untuk mengaktifkan Github Copilot pada akun Github.

GitHub Copilot



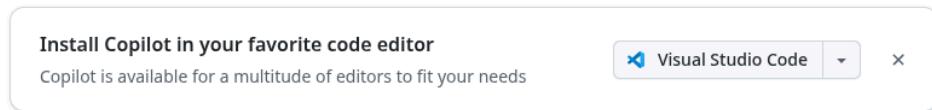
Gambar 6.2 Tampilan Github Copilot Saat Memulai

Selanjutnya Github Copilot sudah bisa digunakan, terdapat menu pada bagian atas (navigation bar) berlogo Github Copilot. Klik dan pilih menu **spaces**.



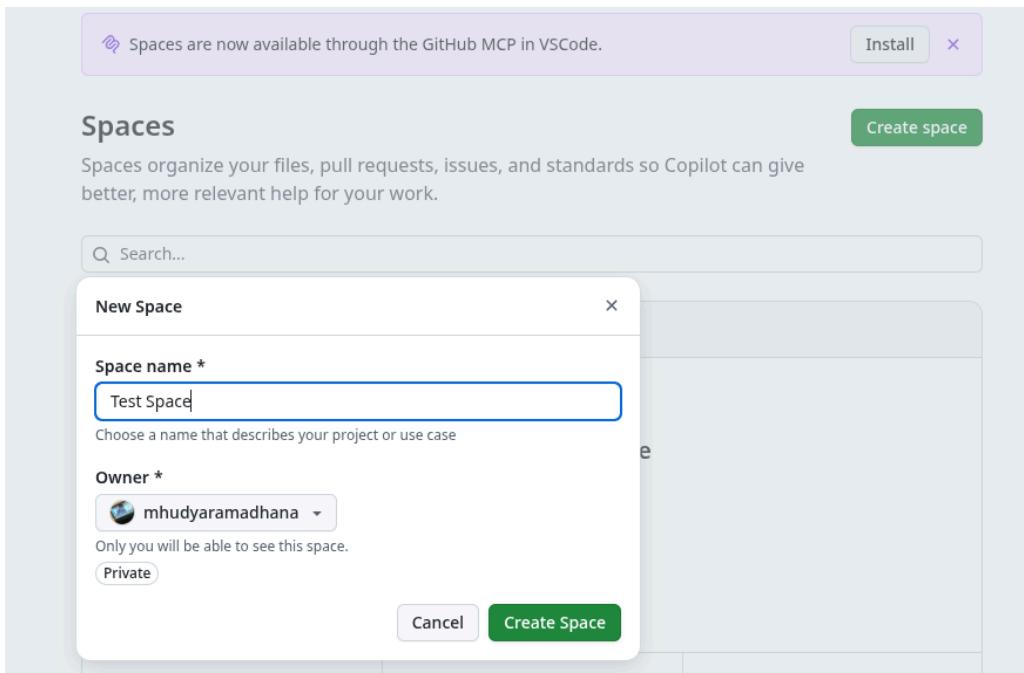
Gambar 6.3 Navbar Github Copilot

Nantinya halaman **spaces Github Copilot** sudah terbuka dan bisa digunakan sebagai *asisten code*.



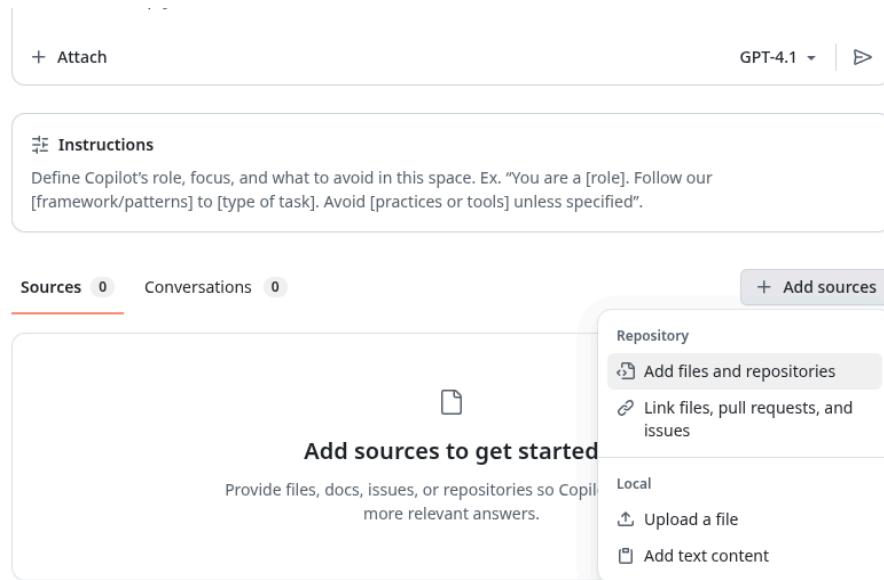
Gambar 6.4 Tampilan Github Copilot Spaces

Selanjutnya, klik spaces pada menu sebelah kiri. Spaces pada Github copilot adalah AI yang bisa digunakan untuk berkomunikasi dan bertanya terkait apa saja konteks yang ada di dalam repositori sehingga pengembang bisa memahami kode di dalam repositori tersebut.



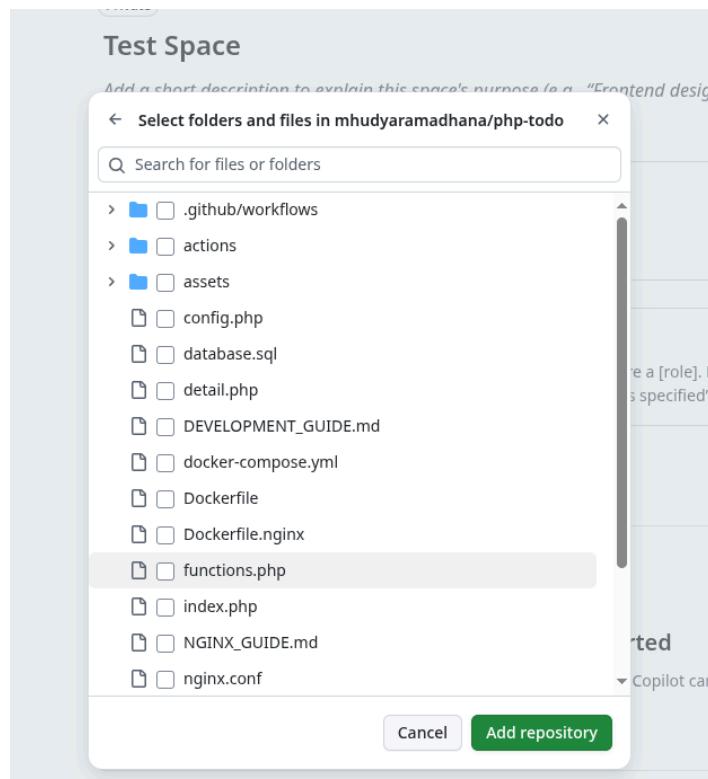
Gambar 6.5 Nama Space Github Copilot

Buatlah space baru, pada contoh di atas space bernama **Test Space**. Setelah terbuka, klik **add sources** dan pilih **add files and repositories**.



Gambar 6.6 Tombol Add Sources untuk Github Copilot Spaces

Kemudian pilih repositori yang diinginkan lalu klik add repository tanpa menceklis satupun file, hal ini bertujuan agar **space** bisa mengakses keseluruhan file.



Gambar 6.7 Pemilihan Repotori pada Github Copilot Spaces

Setelah repositori dibuat space sudah bisa digunakan. Gunakan space untuk berinteraksi dengan AI terkait apa yang ada di repositori, atau meminta saran dalam improvisasi kode. Model juga bisa dipilih, namun untuk menggunakan model lain perlu mengaktifkan fiturnya terlebih dahulu, misalnya menggunakan Gemini 2.0 Flash. Nantinya akan ada window popup, cukup klik *enable* saja.

Kenapa harus ada file functions.php secara terpisah?

- 💡 Pertanyaan yang bagus! Secara umum, memisahkan fungsi-fungsi ke dalam file **functions.php** adalah praktik yang umum dalam pengembangan web, khususnya di PHP. Tujuannya adalah untuk menjaga **keteraturan kode, reusabilitas, dan kemudahan pemeliharaan**.

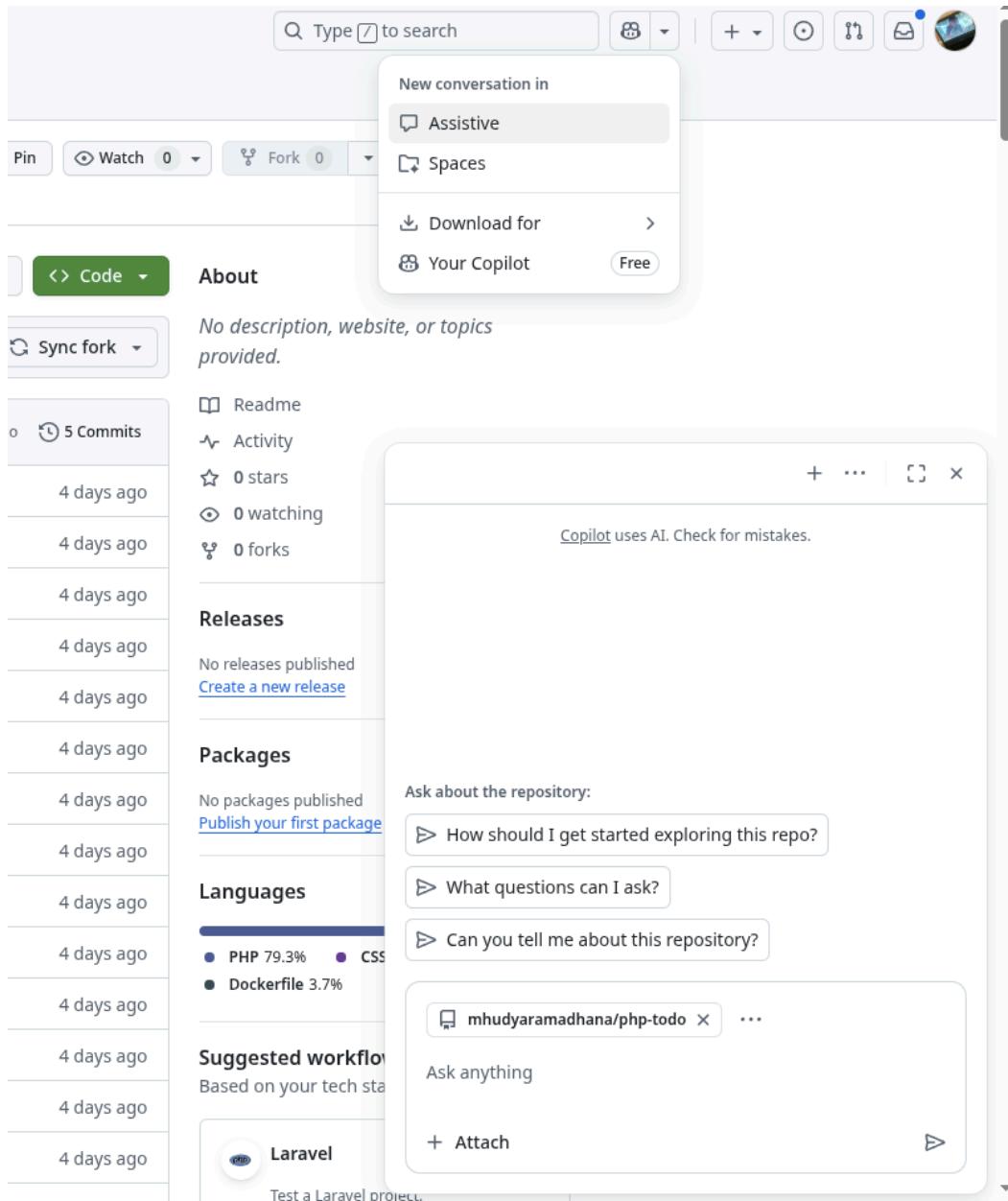
Kenapa dipisah?

1. *Organisasi Kode*

Gambar 6.8 Hasil pertanyaan AI Github Copilot Spaces

Pertanyaan pada Spaces bisa diajukan menggunakan bahasa Indonesia dan bisa ditanyakan tujuan dari kode-kode di dalam repositori yang telah dipilih. Github Spaces yang telah diizinkan mengakses repositori akan memahami konteks pertanyaan yang terhubung dengan repositori tersebut.

Selain melalui *spaces*, GitHub Copilot juga dapat diakses secara asistif langsung pada jendela yang sedang dibuka. Untuk melakukannya, pilih repositori yang ingin diakses, klik logo Copilot di *navbar*, lalu pilih "assistive". Sebuah *popup chat* akan muncul di bagian kanan bawah.



Gambar 6.9 Tampilan Assistive Github Copilot Spaces

6.2.1 Perbedaan Spaces dan Assistive

Perbedaan mendasar antara mode *assistive* dan *spaces* dalam konteks pengelolaan repositori terletak pada cakupan dan fleksibilitas konteks yang dapat diakses. Memahami perbedaan ini sangat penting untuk memilih pendekatan yang paling efisien dalam berbagai skenario pengembangan proyek.

Mode Assistive

Mode *assistive* dirancang untuk skenario di mana fokus diperlukan pada satu repositori tunggal. Ketika menggunakan mode ini, hanya satu repositori yang dapat dipilih dan dijadikan sebagai konteks kerja. Ini berarti bahwa semua operasi, analisis, atau bantuan yang diberikan akan sepenuhnya terfokus pada konten, histori, dan struktur dari repositori yang dipilih tersebut.

- **Keuntungan:**

- **Fokus yang Jelas:** Memungkinkan pengguna untuk berkonsentrasi penuh pada satu bagian kode atau proyek tanpa gangguan dari repositori lain.
- **Efisiensi Sumber Daya:** Dalam beberapa kasus, memproses satu repositori mungkin membutuhkan sumber daya komputasi yang lebih sedikit dibandingkan dengan banyak repositori secara bersamaan.
- **Simplicitas:** Antarmuka dan alur kerja mungkin terasa lebih sederhana dan lugas karena cakupannya yang terbatas.

- **Kekurangan:**

- **Keterbatasan Konteks:** Tidak ideal untuk proyek-proyek yang memiliki dependensi antar-repositori atau memerlukan pemahaman holistik dari ekosistem kode yang lebih besar.
- **Kerja Manual Lebih Banyak:** Jika pekerjaan melibatkan beberapa repositori, pengguna harus beralih konteks secara manual, yang bisa menjadi tidak efisien.

Spaces (Ruang Kerja)

Sebaliknya, *spaces* atau ruang kerja menawarkan cakupan konteks yang jauh lebih luas dan lebih fleksibel. Dengan menggunakan *spaces*, pengguna memiliki kemampuan untuk memilih beberapa repositori secara bersamaan dan menjadikannya sebagai satu kesatuan konteks kerja. Ini sangat powerful untuk proyek-proyek yang terdistribusi atau memiliki arsitektur *microservices*, di mana berbagai komponen proyek mungkin tersebar di beberapa repositori yang berbeda.

- **Keuntungan:**

- **Konteks Repotori yang Lebih Luas:** Memberikan pandangan komprehensif tentang seluruh ekosistem proyek, memungkinkan pemahaman yang lebih baik tentang bagaimana berbagai bagian berinteraksi.
- **Kolaborasi yang Lebih Baik:** Memfasilitasi kolaborasi pada proyek-proyek besar yang melibatkan banyak tim dan repositori.
- **Pencarian dan Analisis Lintas Repotori:** Memungkinkan pencarian, analisis

- dependensi, atau otomatisasi tugas di seluruh repositori yang tergabung dalam satu *space*. Ini sangat berguna untuk identifikasi masalah lintas modul atau refactoring skala besar.
- **Manajemen Proyek yang Efisien:** Memudahkan pengelolaan dan pemantauan kemajuan di berbagai bagian proyek dari satu lokasi terpusat.
 - **Kekurangan:**
 - **Kompleksitas:** Mengelola beberapa repositori sekaligus bisa jadi lebih kompleks, terutama untuk pengguna baru.
 - **Intensif Sumber Daya:** Memproses dan menjaga konteks untuk banyak repositori secara bersamaan mungkin membutuhkan sumber daya komputasi yang lebih besar.

Kapan Menggunakan Spaces?

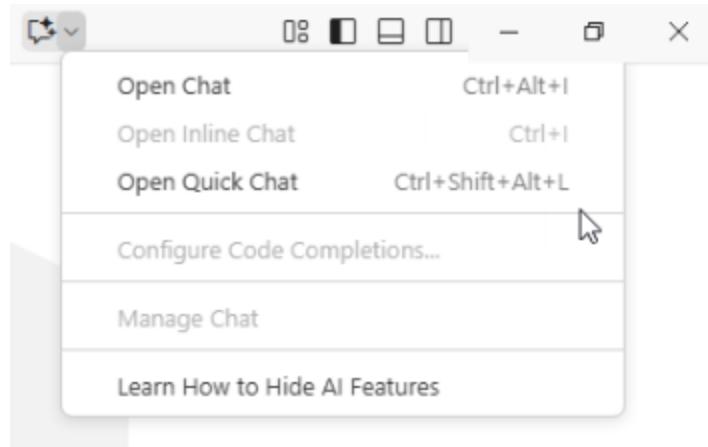
Pada saat membutuhkan konteks yang lebar yang terdiri dari beberapa repositori, menggunakan *spaces* adalah pilihan yang sangat baik karena memungkinkan pengguna untuk melihat gambaran besar, mengidentifikasi hubungan antar-repositori, dan melakukan operasi yang memerlukan pemahaman holistik terhadap seluruh proyek. Contoh skenario di mana *spaces* sangat bermanfaat meliputi:

- **Proyek Microservices:** Setiap layanan mungkin berada di repositori terpisah, dan *spaces* memungkinkan pemantauan dan pengelolaan seluruh arsitektur layanan secara terpadu.
- **Pengembangan Full-Stack:** Ketika *front-end*, *back-end*, dan *shared libraries* mungkin berada di repositori yang berbeda, *spaces* membantu menjaga koherensi antar komponen.
- **Refactoring atau Migrasi Berskala Besar:** Ketika perubahan kode memengaruhi banyak repositori, *spaces* memungkinkan pelacakan dan implementasi perubahan tersebut secara terkoordinasi.
- **Analisis Dependensi Kompleks:** Untuk memahami bagaimana perubahan di satu repositori dapat memengaruhi repositori lain.
- **Pencarian Kode atau Pelacakan Isu Lintas Repositori:** Memungkinkan pencarian masalah atau fitur di seluruh basis kode proyek yang tersebar.

6.2.2 Aktivasi pada VSCode

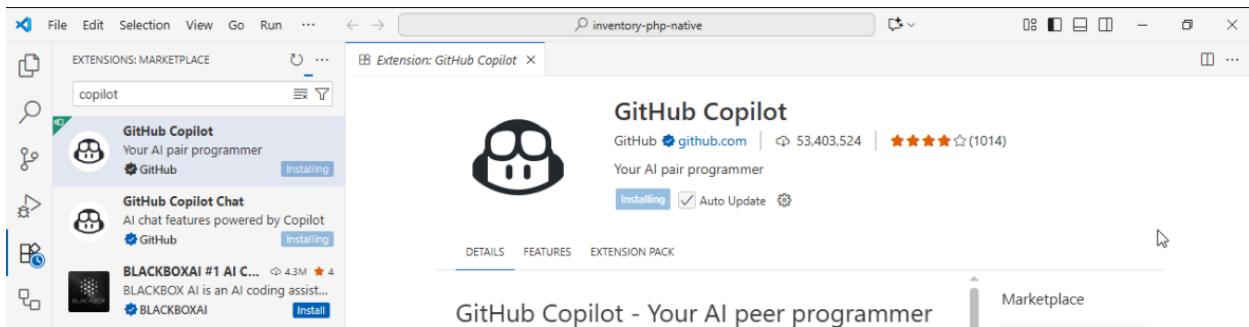
Github Copilot adalah alat bantu pemrograman berbasis kecerdasan buatan yang dikembangkan oleh GitHub dan OpenAI. Untuk memulai menggunakan Github Copilot pada Visual Studio Code (VSCode), pengguna perlu mengunduh dan memasang ekstensi (extension) Github Copilot yang tersedia melalui Marketplace VSCode.

Secara default pada Visual Studio Code versi terbaru, Github Copilot Chat sudah tersedia pada navigation bar bagian atas. Navigation menu ini memungkinkan untuk membuka chat dengan Github Copilot. Namun apabila belum tersedia, bisa mengunduh pada ekstensi VSCode.



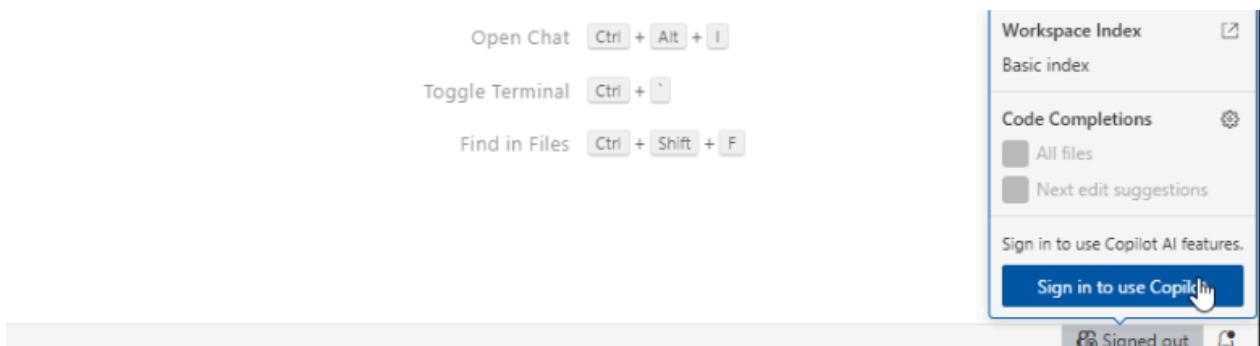
Gambar 6.10 Menu Github Copilot pada VSCode

Untuk menambahkan ekstension, pilih icon extension pada bagian kiri VSCode lalu ketik **copilot**.



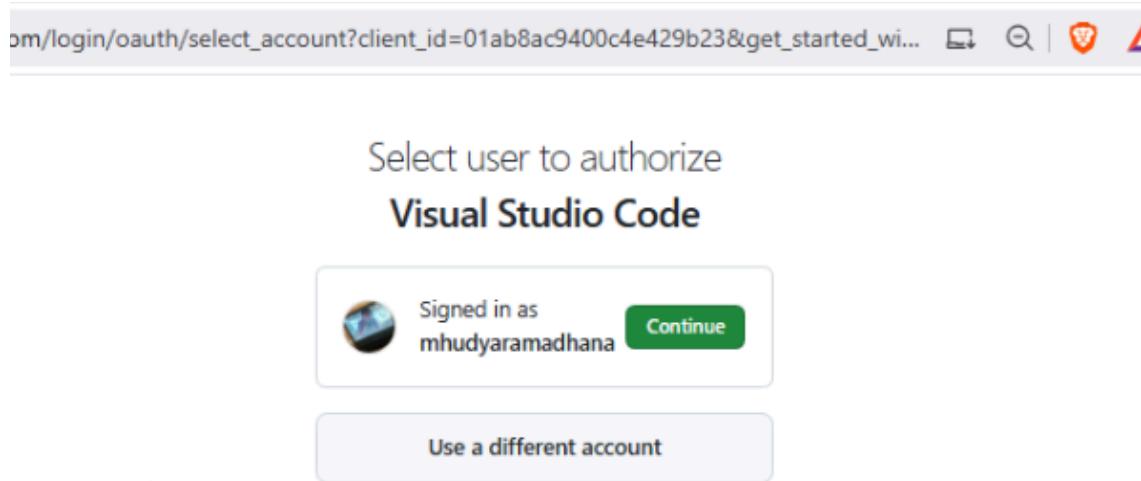
Gambar 6.11 Ekstensi Github Copilot

Setelah ekstensi terpasang, pengguna perlu melakukan otentikasi dengan akun GitHub. Menu ini terletak pada bagian bawah sebelah kanan dan klik Sign in to use Copilot.



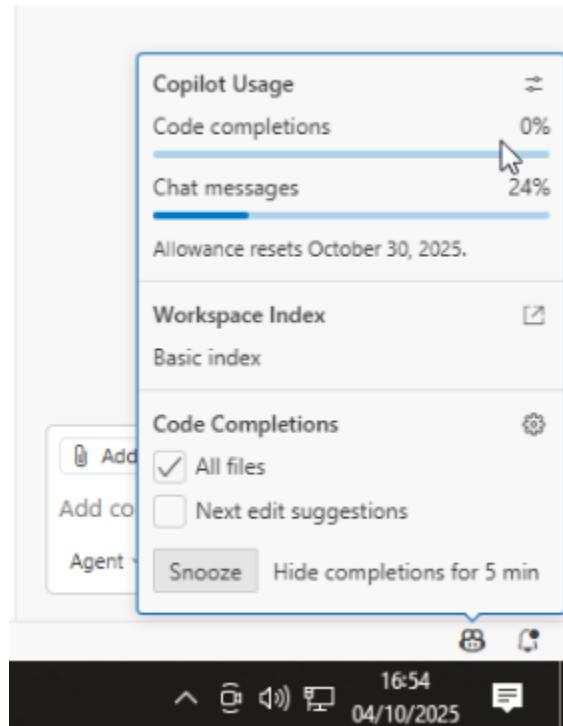
Gambar 6.12 Menu sign in ke Github Copilot

Pilih Sign in with Github dan VSCode akan membuka browser secara otomatis untuk memilih user dalam autorisasi akses Github Copilot ke VSCode.



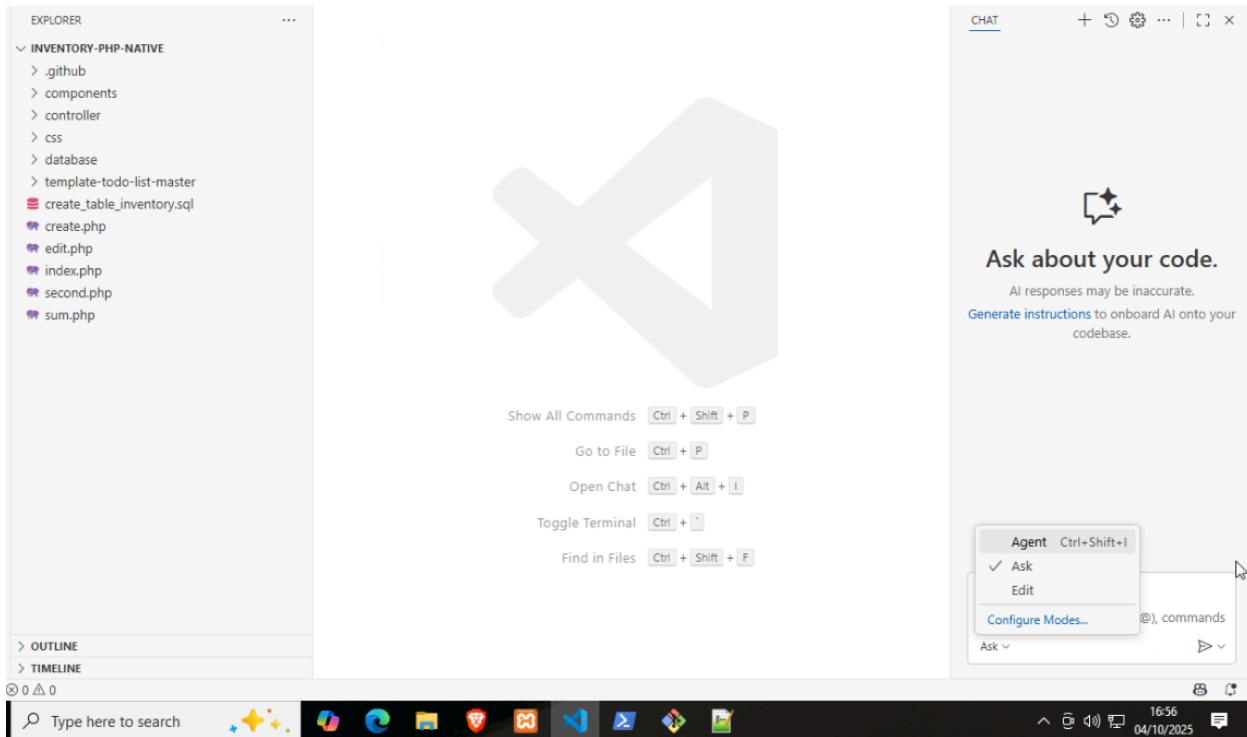
Gambar 6.13 Tampilan pemilihan akun

Klik **continue** pada halaman tersebut lalu klik tombol **Authorize Visual Studio Code** pada tombol berwarna hijau untuk mengizinkan akses.



Gambar 6.14 Tampilan Usage Github Copilot

Sekarang menu sudah berubah dan pengguna bisa melihat seberapa banyak penggunaan Github Copilot. Secara *default*, Github Copilot dapat diakses secara gratis dengan batasan tertentu.



Gambar 6.15 Tampilan mode Github Copilot

GitHub Copilot, sebagai asisten AI dapat diakses dan diintegrasikan ke dalam lingkungan pengembangan Visual Studio Code. Ada dua mode utama yang memungkinkan pengembang berinteraksi dengan Copilot: mode **Agent** dan mode **Ask**.

Mode Agent: Dalam mode agen, GitHub Copilot bekerja secara proaktif di latar belakang saat pengembang menulis kode. Copilot akan menganalisis konteks kode yang sedang dikerjakan, mengidentifikasi pola, dan menyarankan pelengkapan kode secara *real-time*. Saran-saran ini dapat berupa baris kode tunggal, blok kode lengkap, atau bahkan seluruh fungsi, yang disesuaikan dengan gaya dan bahasa pemrograman yang digunakan. Mode agen dirancang untuk meningkatkan efisiensi dan kecepatan pengembangan dengan mengurangi kebutuhan untuk mengetik berulang-ulang dan mencari sintaks.

Mode Ask: Mode Ask memberikan pengalaman yang lebih interaktif dan eksplisit. Daripada menunggu saran muncul secara otomatis, pengembang dapat secara langsung mengajukan pertanyaan atau memberikan instruksi kepada Copilot. Ini memungkinkan pengembang untuk:

- **Meminta penjelasan:** Misalnya bisa bertanya "Bagaimana cara kerja fungsi ini?" atau "Apa tujuan dari variabel ini?".
- **Menghasilkan kode berdasarkan deskripsi:** Pengembang dapat memberikan deskripsi fungsionalitas yang diinginkan, seperti "Buatkan saya fungsi Python untuk menghitung faktorial" atau "Hasilkan kode JavaScript untuk mengelola event click pada tombol."
- **Melakukan refactoring atau perbaikan kode:** Pengembang bisa meminta Copilot untuk

menyederhanakan kode yang kompleks, memperbaiki *bug*, atau mengoptimalkan kinerja.

- **Mencari informasi atau contoh:** Jika pengembang tidak yakin tentang cara menggunakan suatu API atau library tertentu, pengembang dapat bertanya kepada Copilot untuk mendapatkan contoh penggunaan atau dokumentasi yang relevan.

Kedua mode ini melengkapi satu sama lain, memberikan fleksibilitas bagi pengembang untuk memilih cara berinteraksi dengan GitHub Copilot yang paling sesuai dengan alur kerja dan preferensi.

The screenshot shows a portion of an HTML file in a code editor. The code includes PHP logic for including components like header.php and navbar.php. A tooltip from GitHub Copilot provides a comment about the use of include statements. Below the code editor, a sidebar displays a summary of the suggestion, including a reference count and a tip about modularization.

```
> ...
ctype html>
l lang="en">
d>
<?php include('components/header.php') ?>
ad>

y>
<?php include('components/navbar.php') ?>

<main role="main" class="container">
    <?php include('components/welcome_message.php') ?>
    <?php
        include('database/inventory.php');

        $data = new Inventory();
    ?>
```

Pada bagian index.php terdapat include components, kenapa harus di-include dan dipisah seperti itu ya?

Used 1 reference

Memisahkan dan meng-include file komponen seperti header.php, navbar.php, footer.php, dan lain-lain di index.php bertujuan untuk:

1. Modularitas & Reusabilitas

Komponen yang sering digunakan di banyak halaman (misal: header, navbar, footer) cukup dibuat satu kali, lalu di-include di setiap halaman yang

Gambar 6.16 Hasil chat dengan Github Copilot pada VSCode

6.3 Github Copilot sebagai Asisten Pengembang

Pengembang dapat secara signifikan meningkatkan efisiensi alur kerja dengan memanfaatkan GitHub Copilot sebagai asisten AI untuk secara otomatis menghasilkan pesan *commit*. Proses ini dirancang agar mudah dan terintegrasi secara mulus dalam lingkungan pengembangan.

Untuk memulai, pengembang hanya perlu membuka bagian kontrol sumber Git dalam *Integrated Development Environment* (IDE) atau antarmuka Git pilihan mereka. Di sana, mereka akan menemukan tombol khusus yang terletak di samping kolom pesan *commit* yang biasa. Mengklik tombol ini akan mengaktifkan GitHub Copilot.

Setelah diaktifkan, GitHub Copilot akan secara otomatis melakukan tinjauan menyeluruh terhadap semua perubahan kode yang telah diubah pengembang. Ini mencakup analisis mendalam terhadap penambahan, penghapusan, dan modifikasi baris kode, serta konteks sekitarnya. GitHub Copilot kemudian akan menyusun pesan *commit* yang relevan dan deskriptif. Pesan *commit* yang dihasilkan tidak hanya merangkum perubahan teknis tetapi juga berusaha untuk menjelaskan tujuan dan dampak dari *commit* tersebut, mirip dengan bagaimana seorang pengembang manusia akan menulisnya. Pendekatan ini memastikan bahwa riwayat *commit* tetap bersih, informatif, dan mudah dipahami, baik untuk pengembang saat ini maupun untuk anggota tim di masa depan yang mungkin perlu memahami kode.

```

index.php M
index.php > html > body > div.container.mt-4 > div.row
35   <html lang="en">
36     <head>
37       <!-- CUSTOM CSS STYLES - LOCAL ASSET MANAGEMENT -->
38       <link href="assets/style.css" rel="stylesheet"
39     </head>
40   <body>
41     <div class="container mt-4">
42       <div class="row"> You, 4 days ago F
43         <div class="col-md-12">
44           <h1 class="mb-4">Todo List Application</h1>
45           <?php if ($message): ?>
46             <div class="alert alert-?>= $message ?>
47               <?= escape($message) ?> messa

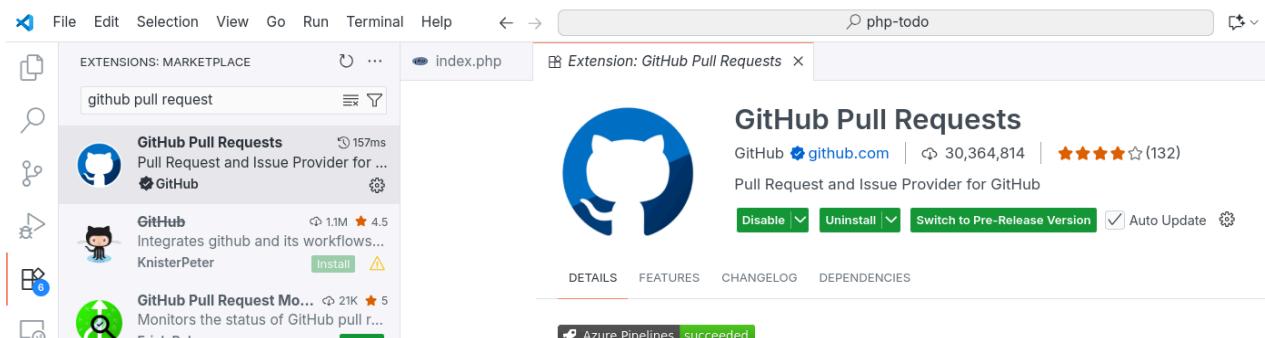
```

Gambar 6.17 Tampilan commit message yang dibuat oleh Github Copilot

Gambar di atas mengilustrasikan bahwa pesan commit yang dihasilkan oleh Github Copilot sangat mudah dipahami dan deskriptif, menjelaskan perubahan ekstensif yang telah dilakukan.

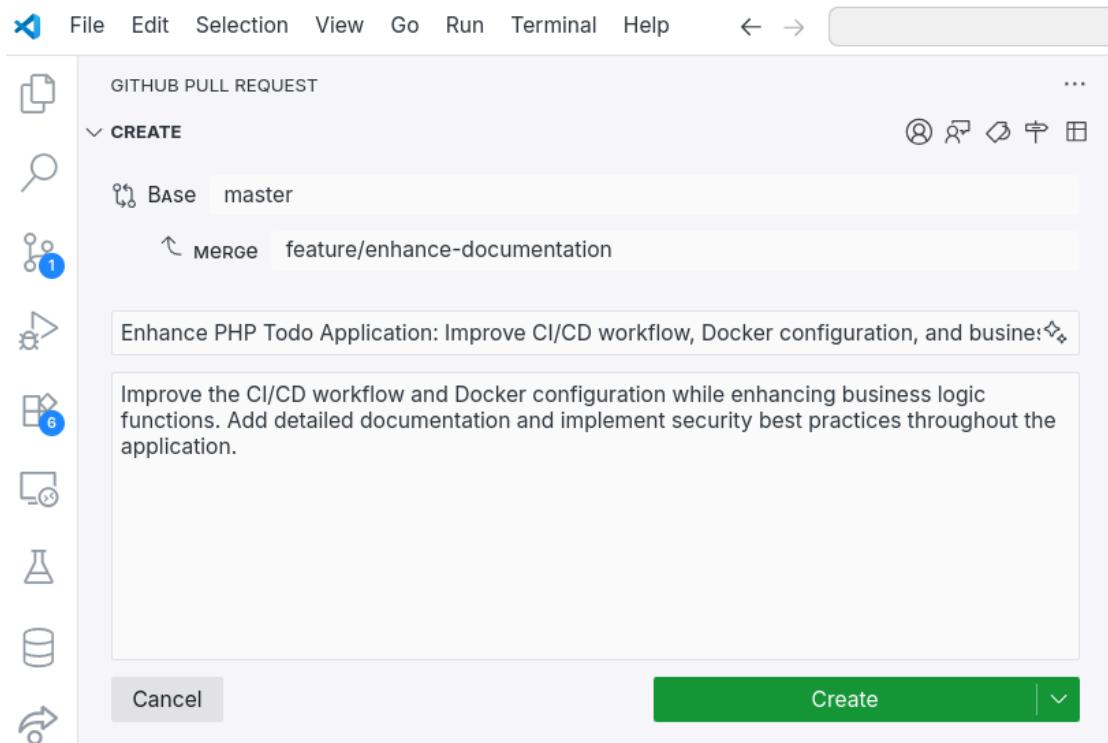
Github Copilot untuk Pull Requests

Github Copilot bisa digunakan untuk membuat pesan *pull requests* secara otomatis, pertama pastikan ekstensi Github Pull Requests telah terpasang pada Visual Studio Code.



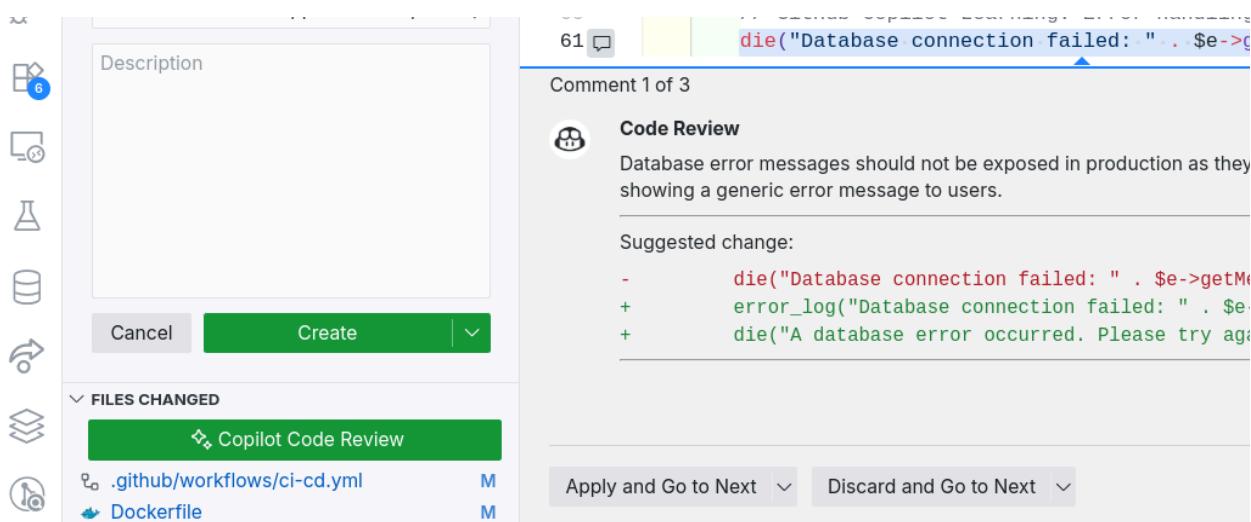
Gambar 6.18 Ekstensi Github Pull Requests

Setelahnya, pastikan sudah memiliki branch baru yang digunakan untuk membuka pull requests ke master. Pada contoh di bawah ini, branch yang digunakan adalah **feature/enhance-documentation**. Buka ekstensi Github Pull Request, dapat dilihat bahwa pada kotak judul pull requests terdapat tombol untuk meminta Github Copilot membuat pesan.



Gambar 6.19 Tampilan Github Pull Request pada VSCode

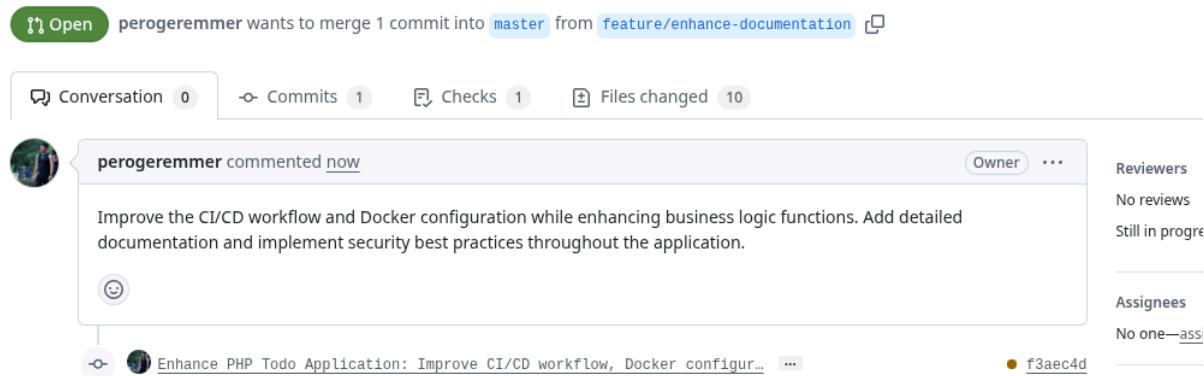
Pada bagian **Files changed** terdapat sebuah tombol berwarna hijau bertuliskan **Copilot Code Review**. Tombol ini merupakan fitur Github Copilot untuk meninjau kode yang telah dibuat. Dengan menekan tombol ini, Github Copilot akan secara otomatis menganalisis kode, mengidentifikasi potensi masalah, bug, atau area yang bisa dioptimalkan, serta memberikan saran perbaikan. Proses review otomatis ini dapat sangat membantu dalam meningkatkan kualitas kode, mempercepat proses pengembangan, dan memastikan praktik terbaik dalam *coding*.



Gambar 6.20 Tampilan Code Review Github Copilot

Pesan *commit* yang dihasilkan oleh GitHub Copilot akan ditampilkan pada halaman *pull request*. Pesan ini memberikan deskripsi perubahan file yang jelas yang dilakukan oleh pengembang.

Enhance PHP Todo Application: Improve CI/CD workflow, Docker configuration, and business logic functions #1



Gambar 6.21 Hasil Pull Requests

6.4 Kecerdasan Buatan dalam Actions Workflow

Memanfaatkan GitHub Actions Workflow, kecerdasan buatan (AI) dapat diintegrasikan untuk menjalankan berbagai fungsi dalam pengembangan perangkat lunak. Salah satu fungsi adalah pemindaian kerentanan kode (*vulnerability scanning*) secara otomatis pada repositori proyek. GitHub telah menyediakan fitur-fitur yang mendukung kapabilitas ini melalui *workflow* khusus seperti **CodeQL**.

- **CodeQL:** Menawarkan analisis keamanan yang lebih mendalam dan komprehensif. CodeQL adalah mesin analisis semantik yang memungkinkan pengembang menulis *query* untuk menemukan kerentanan, *bug*, dan varian pada kode. Ketika diimplementasikan melalui GitHub Actions, CodeQL dapat melakukan pemindaian terjadwal atau berdasarkan *event* tertentu (misalnya, *push* ke cabang utama) untuk mengidentifikasi pola kerentanan kompleks yang mungkin terlewatkan oleh metode lain. Hasil pemindaian CodeQL disajikan dalam bentuk *alert* langsung di antarmuka GitHub, seringkali dengan saran perbaikan, yang mempermudah tim untuk meninjau dan mengatasi masalah keamanan secara efisien.

Pemanfaatan AI dalam *workflow* ini tidak hanya terbatas pada pemindaian kerentanan. AI juga dapat digunakan untuk:

- **Analisis Kualitas Kode:** Mengevaluasi kualitas, keterbacaan, dan kepatuhan kode terhadap standar tertentu.
- **Pengujian Otomatis Cerdas:** Mengidentifikasi kasus uji yang paling relevan atau menghasilkan *test case* baru berdasarkan perubahan kode.
- **Otomatisasi Tinjauan Kode:** Memberikan saran otomatis atau mendeteksi *bug* potensial selama tinjauan kode oleh rekan kerja.

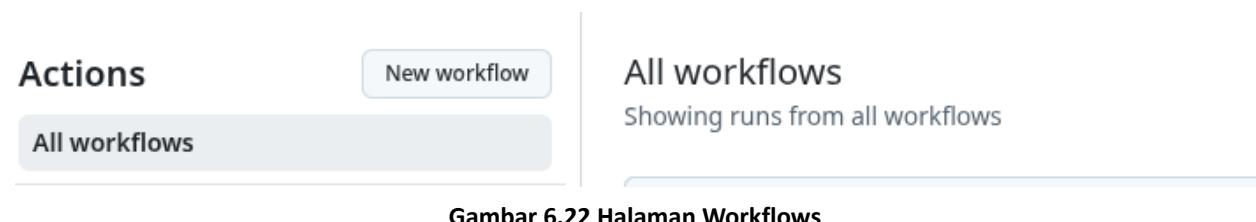
- **Deteksi Anomali Log:** Memantau log aplikasi dan sistem untuk mendeteksi perilaku abnormal yang mungkin mengindikasikan serangan atau masalah operasional.

Integrasi AI ke dalam GitHub Actions Workflow secara signifikan meningkatkan praktik DevOps dengan mengotomatisasi aspek-aspek keamanan dan kualitas, hal ini membuat tim pengembang bisa fokus kepada inovasi sambil memastikan integritas dan keamanan produk mereka.

Instalasi Workflow AI Security

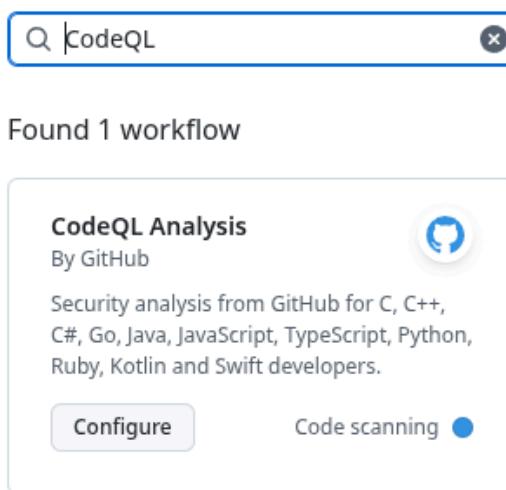
Pertama, pastikan repositori yang digunakan sudah memiliki actions workflow sebelumnya yang telah dibahas pada modul 5.3.

Kedua, pergi ke halaman **Actions** pada *repository*, lalu klik **new workflow**.



Gambar 6.22 Halaman Workflows

Ketiga, cari CodeQL pada kotak pencarian, lalu klik **configure**.



Gambar 6.23 Kotak pencarian CodeQL

Selanjutnya akan dipindahkan ke halaman konfigurasi, pastikan tulisannya **codeql.yml** in **master** lalu klik tombol hijau bertuliskan **commit changes** pada bagian sebelah kanan.

The screenshot shows the GitHub repository 'perogeremmer / php-todo'. In the 'Code' tab, under '.github/workflows', the file 'codeql.yml' is selected. The code content is as follows:

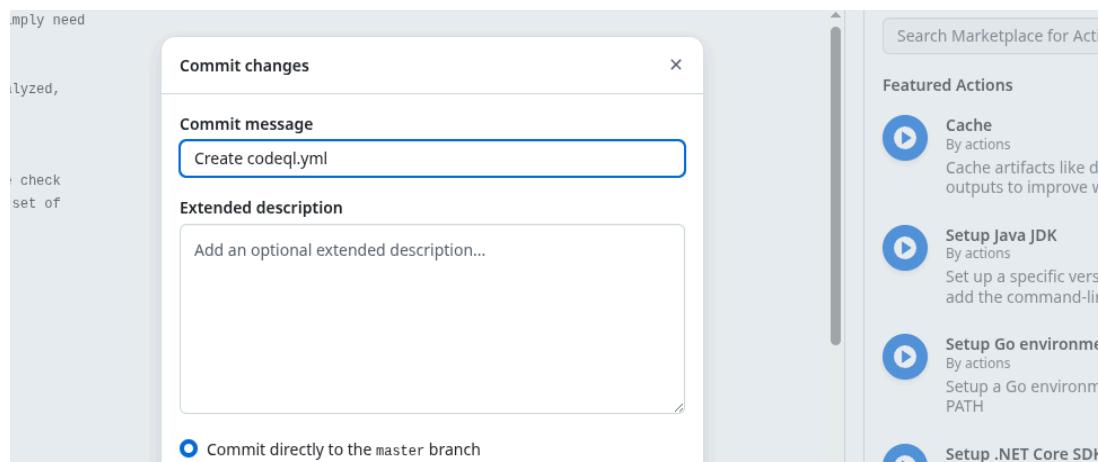
```

1  # For most projects, this workflow file will not need changing; you simply need
2  # to commit it to your repository.
3  #
4  # You may wish to alter this file to override the set of languages analyzed,
5  # or to provide custom queries or build logic.
6  #
7  # ***** NOTE *****
8  # We have attempted to detect the languages in your repository. Please check
# the 'languages' matrix defined below to confirm you have the correct set of

```

Gambar 6.24 Konfigurasi file codeql.yml

Nantinya akan ada popup sekali lagi, pilih **commit directly to the master branch** lalu klik **commit changes**.



Gambar 6.25 Confirmation popup message commit

Setelah itu, secara otomatis CodeQL akan berjalan pada halaman actions workflow dan tidak perlu ditunggu hingga prosesnya selesai.

The screenshot shows the GitHub Actions page for the 'CodeQL Advanced' workflow. The sidebar on the left has 'Actions' selected, with 'CodeQL Advanced' highlighted. The main area shows '3 workflow runs' for the 'CodeQL Advanced' workflow. The first run is a pull request for 'Enhance PHP Todo Application: Improve CI/CD workflow, Docker configur...' with status 'feature/enhance-documentation'. The second run is for 'CodeQL Advanced' with status 'master'. The third run is for 'CodeQL Advanced' with status 'Scheduled'.

Gambar 6.26 Hasil CodeQL setelah dijalankan

Hasil dari CodeQL bisa dilihat pada halaman **Security**, lalu pilih **Code Scanning**. Nantinya akan ada hasil yang dideteksi oleh AI CodeQL.

The screenshot shows the GitHub Security tab with a search bar at the top containing "is:open branch:master". Below the search bar, there are filters for "Open" (6) and "Closed" (0) issues, and dropdown menus for Language, Tool, Rule, Severity, and Sort. A table lists two issues under the "Workflow does not contain permissions" rule:

Issue	Severity	Branch
Workflow does not contain permissions	Medium	master
Workflow does not contain permissions	Medium	master

#6 opened now • Detected by CodeQL in .github/workflows/ci-cd.yml :82

#5 opened now • Detected by CodeQL in .github/workflows/ci-cd.yml :17

Gambar 6.27 Tab Code Scanning pada halaman Security

Hasil ini akan menjelaskan bagian mana yang memiliki kekurangan pada file di repositori Github pengembang.

The screenshot shows a GitHub Actions workflow file with the following code:

```
71 # Step 7: Kasih tahu hasil deployment
72 - name: Notify deployment status
73   if: always() # Selalu jalankan step ini (sukses atau gagal)
74   run: |
75     if [ ${job.status} ] == 'success' ; then
76       echo "Deployment berhasil di laptop-andi! Aplikasi sudah update ke versi terbaru"
77     else
78       echo "Deployment gagal di laptop-andi! Cek log error di atas"
79     fi
80
81 deploy-hudya:
```

Actions job or workflow does not limit the permissions of the GITHUB_TOKEN. Consider setting an explicit permissions block, using the following as a minimal starting point: {}

CodeQL

```
82 runs-on: [laptop-hudya]
83
84 steps:
```

The screenshot shows the results of a CodeQL scan. It includes a table with the following data:

Tool	Rule ID	Query
CodeQL	actions/missing-workflow-permissions	View source

If a GitHub Actions job or workflow has no explicit permissions set, then the repository permissions are used. Repositories created under organizations inherit the organization permissions. The organizations or repositories created before February 2023 have the default permissions set to read-write. Often these permissions do not adhere to the principle of least privilege and can be reduced to read-only, leaving the write permission only to a specific types as issues: write or pull-requests: write.

Show more ▾

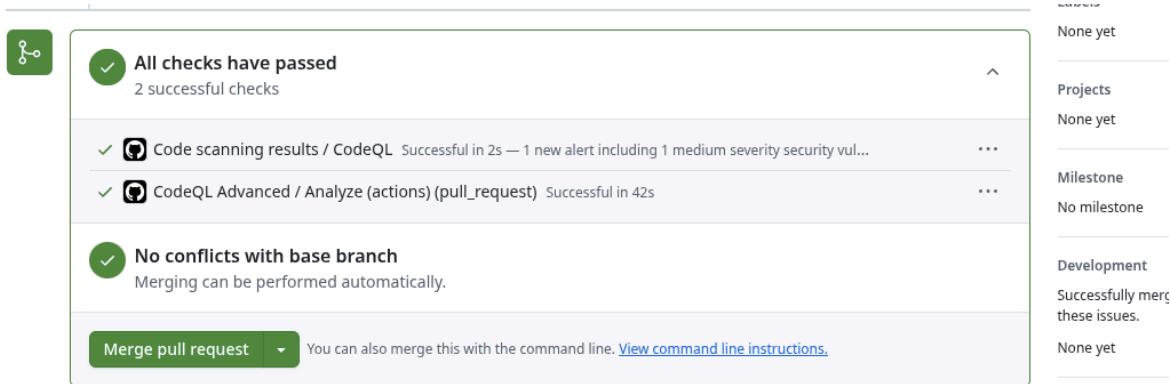
- ⌚ First detected in commit 5 days ago
- [Create codeql.yml](#) Verified ✅ ab0b908
- .github/workflows/ci-cd.yml:17 on branch master

Gambar 6.28 Hasil dari temuan CodeQL

Pada halaman *pull request*, khususnya di bagian *detail*, terlihat adanya pengecekan kode otomatis yang dilakukan oleh *workflow* CodeQL. Gambar di bawah menunjukkan bahwa CodeQL telah berhasil memindai seluruh kode dalam *pull request* tanpa menemukan masalah atau kerentanan.

Keberhasilan ini mengindikasikan bahwa penggunaan ekstensi CodeQL dalam *Actions Workflow* sangat membantu dan efektif dalam proses peninjauan kode. Dengan adanya pemindaian otomatis ini, pengembang dapat memastikan kualitas dan keamanan kode mereka sejak awal siklus pengembangan. Proses ini tidak hanya mempercepat identifikasi potensi masalah, tetapi juga mengurangi beban kerja tim peninjau kode, memungkinkan pengembang untuk fokus kepada aspek-aspek fungsional dan pekerjaan yang lebih kompleks.

Secara keseluruhan, integrasi CodeQL dalam pengembangan merupakan langkah yang baik untuk menjaga integritas dan keamanan proyek perangkat lunak.



Gambar 6.29 Hasil pengecekan oleh CodeQL