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 interaktivity
- **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 kepenuhan *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

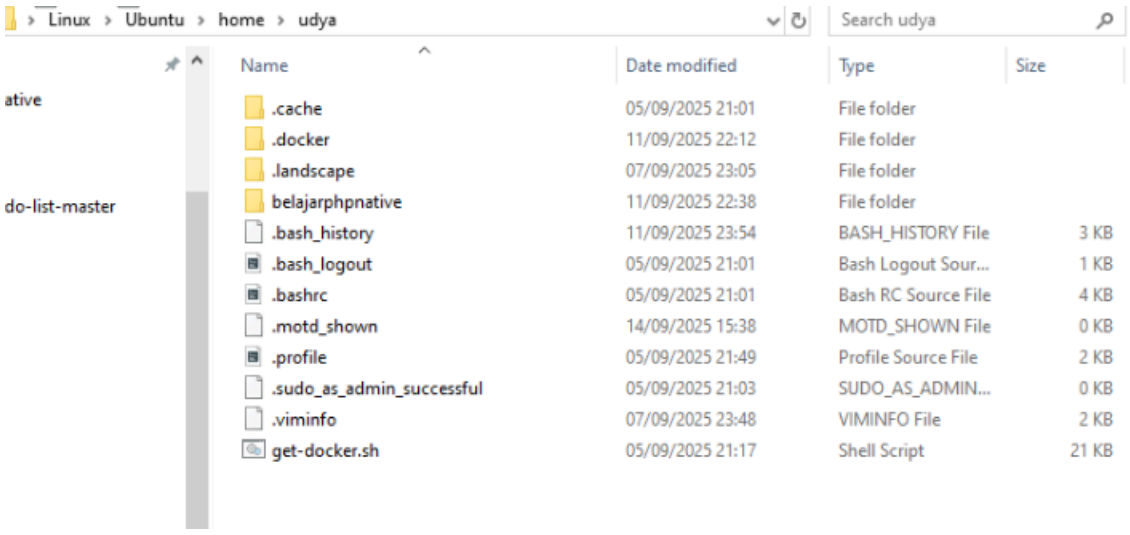
Konsep: Client IP address di-hash untuk menentukan backend server.
Cara Kerja:

- 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**.



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**:

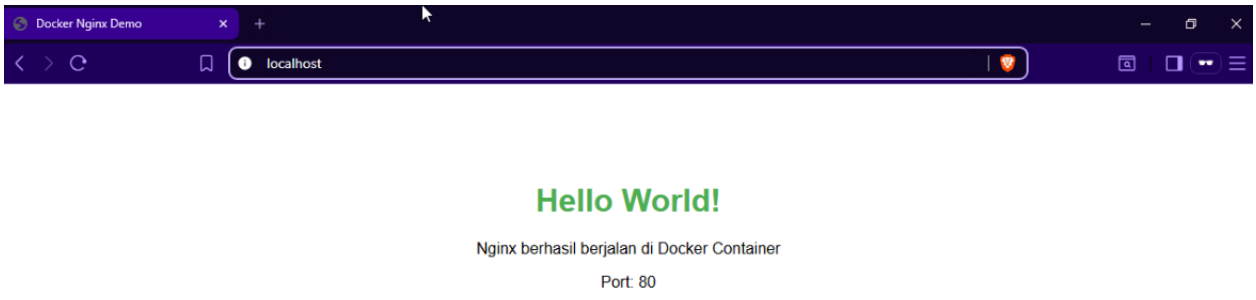
```
version: '3.8'

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

Sekarang jalankan perintah berikut untuk melihat hasilnya.

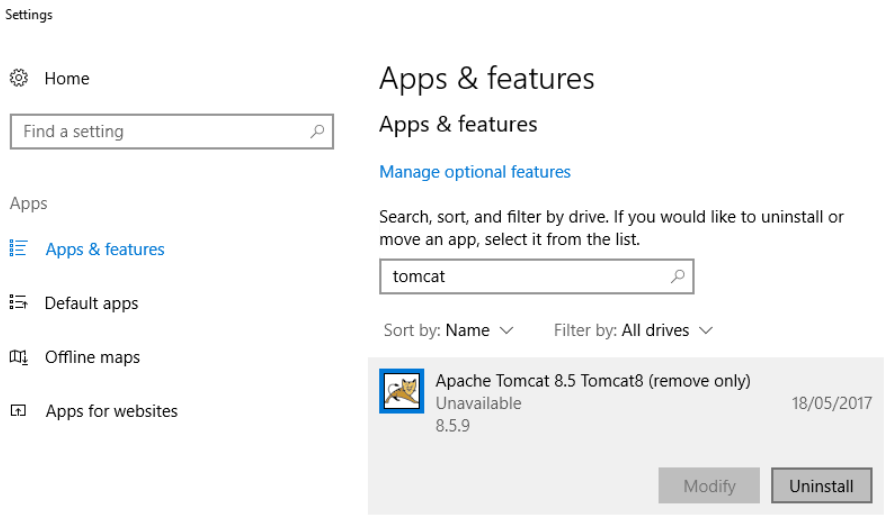
```
docker compose up -d --build
```

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



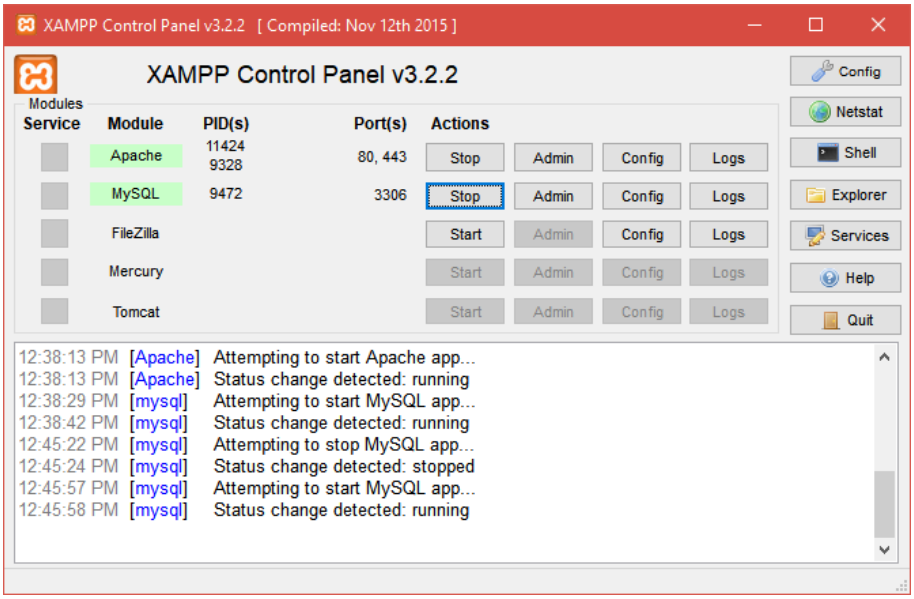
Gambar 3.2 Hasil Nginx Container

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.

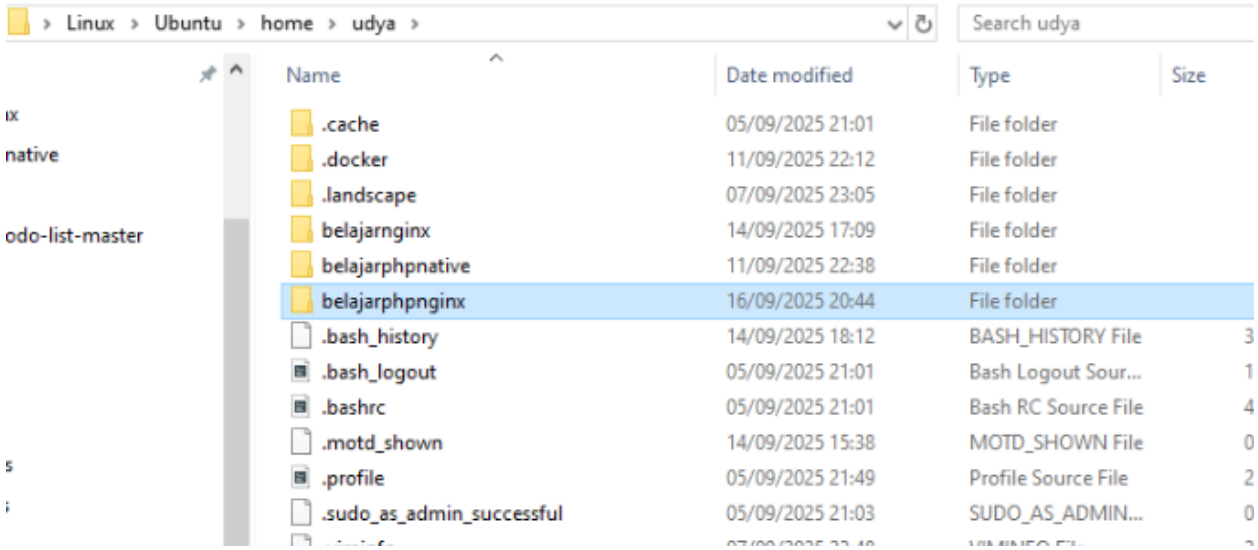
PHP-FPM (FastCGI Process Manager) adalah implementasi FastCGI untuk PHP yang dirancang untuk beban tinggi, berjalan terpisah dari *web server*. PHP-FPM, yang berjalan dalam kontainernya sendiri, bertanggung jawab untuk mengeksekusi *script* PHP dan menghasilkan *output*. Setelah PHP-FPM selesai memproses, 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.



Gambar 3.5 Pembuatan folder belajarphpnginx

Buat file bernama **Dockerfile.nginx** di dalam folder **belajarphpnginx** 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 **belajarphpnginx** 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/mymphpapp`, 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.