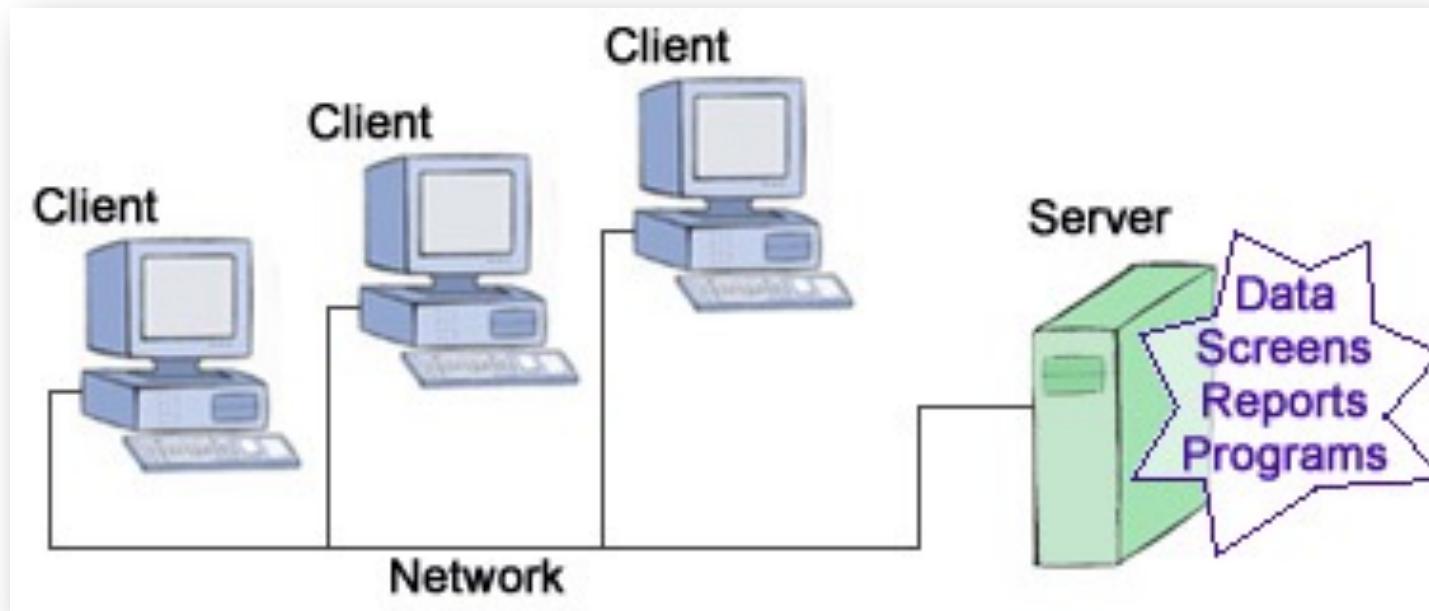
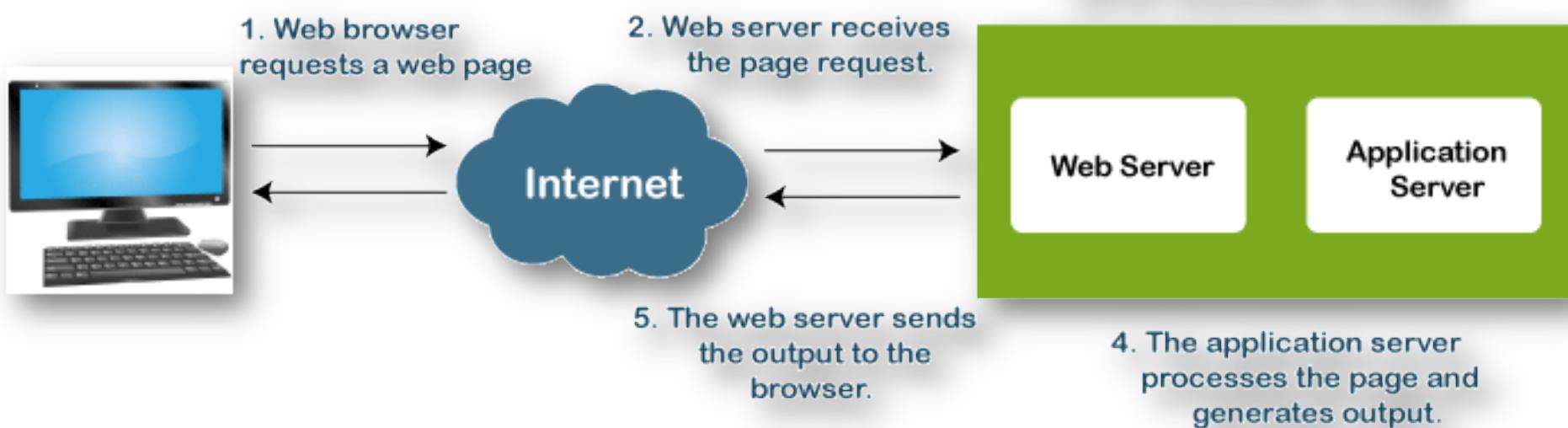


Architectural Design to nFR

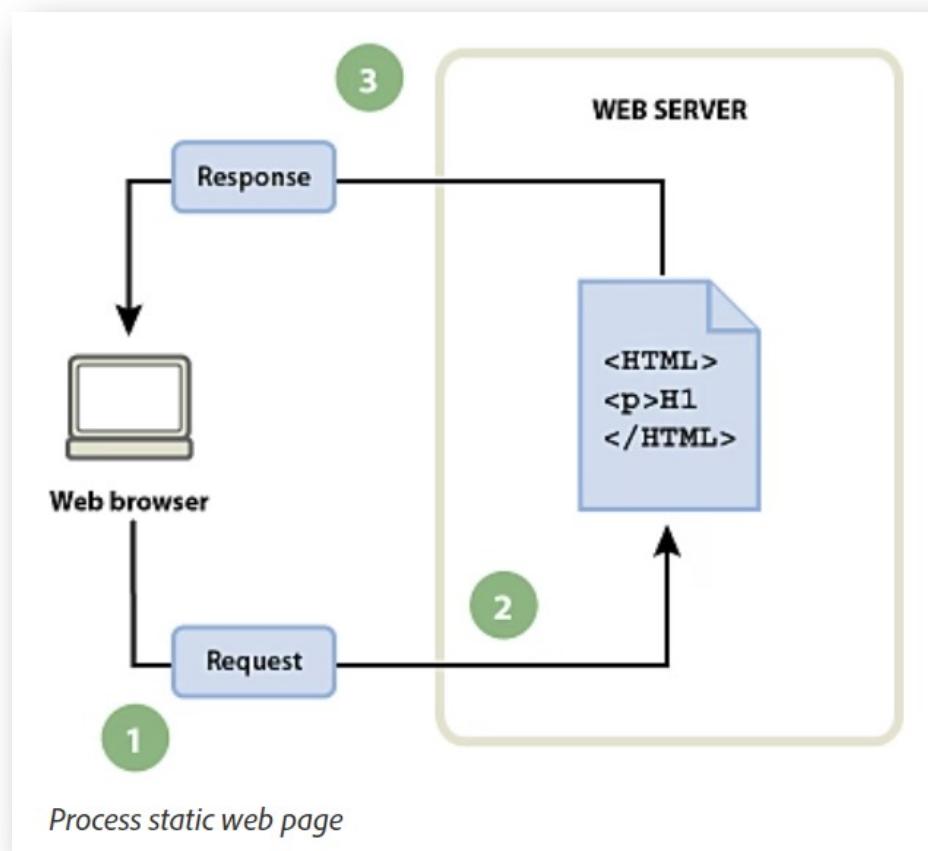
How Client/Server App. Work



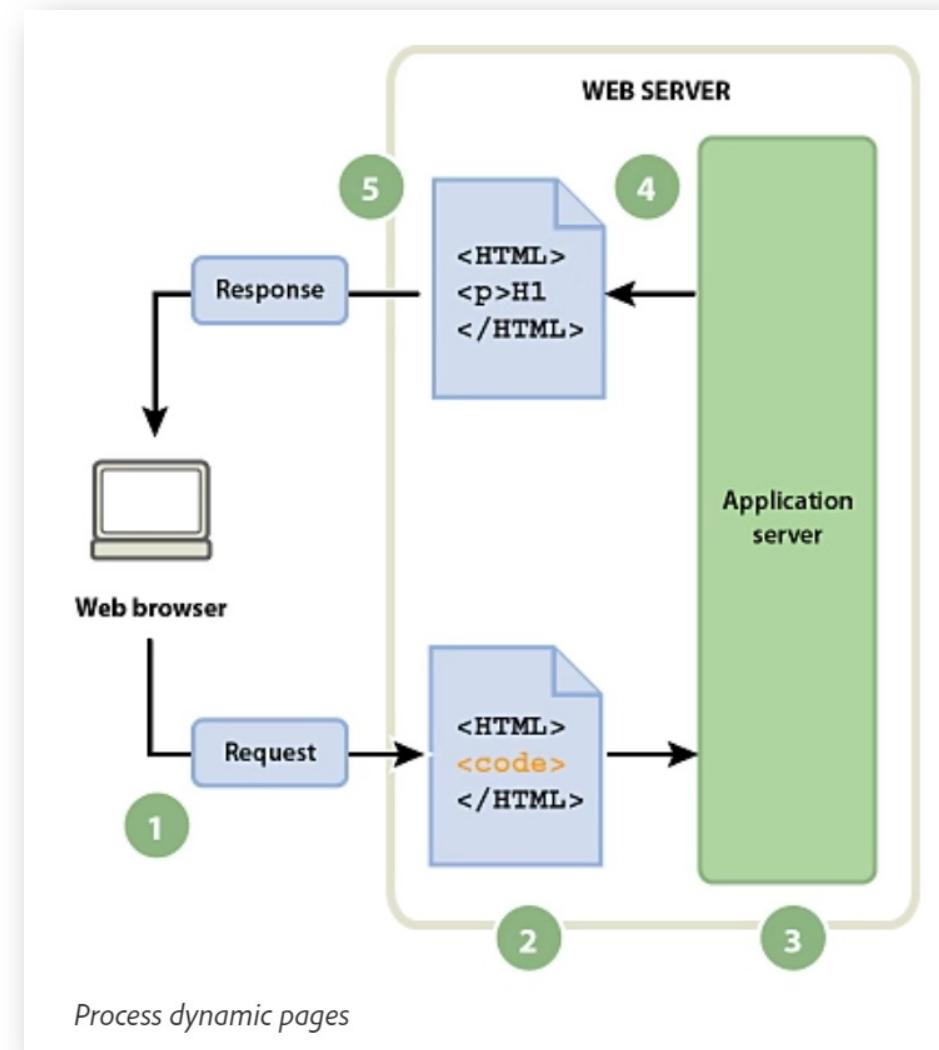
How Web Application Works

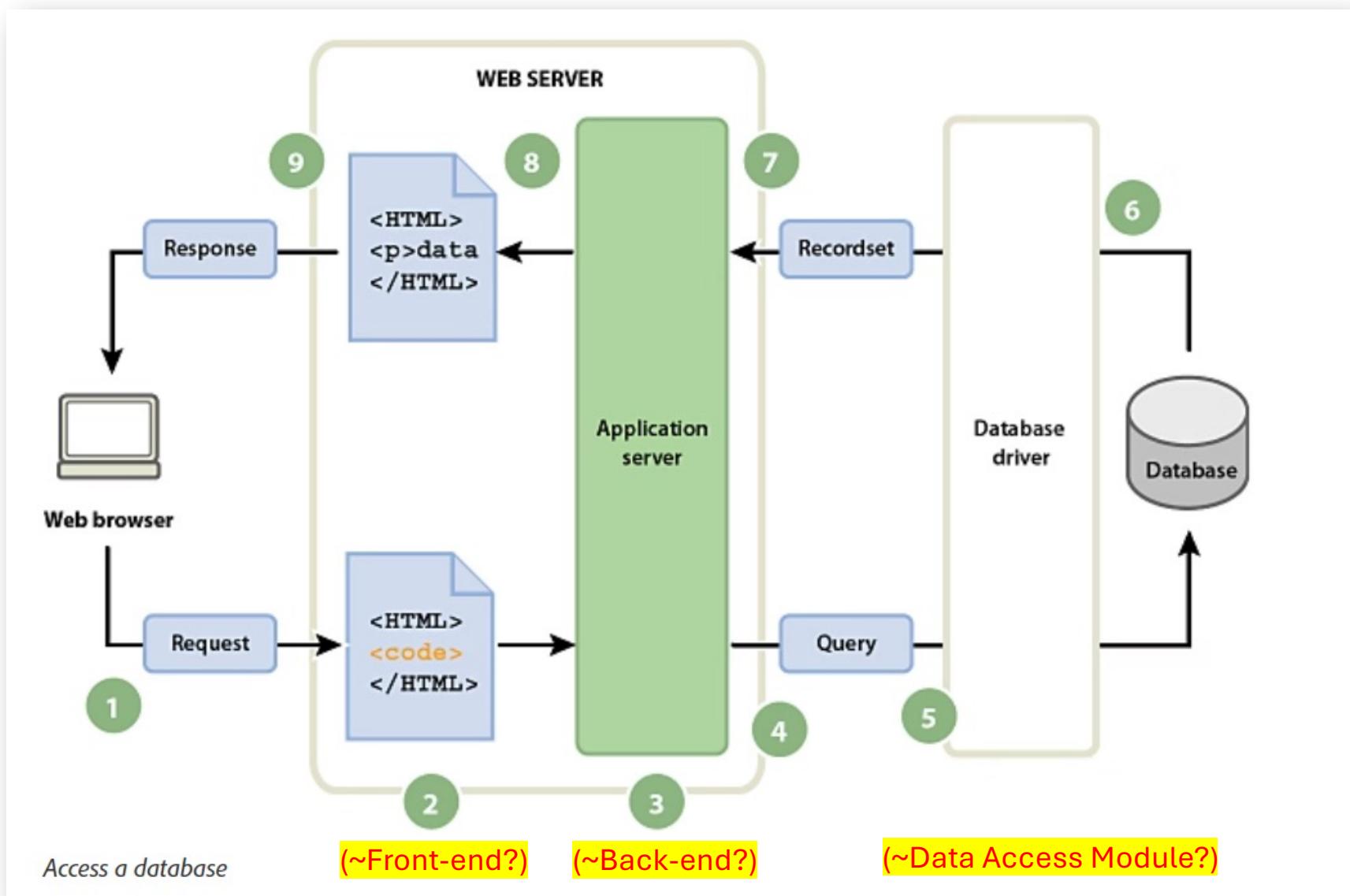


Static Web Page



Dynamic Web Page





HTTP server คือ Web Server

Web Application server คือ App server
ใช่หรือไม่?

ใช่ครับ 

โดยทั่วไปในระบบ Web-based Application สมัยใหม่:

◆ HTTP Server = Web Server

- คือเซิร์ฟเวอร์ที่รับคำขอ (HTTP request) จาก Client (เช่น Web Browser) และส่งกลับ HTML, CSS, JavaScript, หรือ static content อื่นๆ
- มักทำหน้าที่เป็น Reverse Proxy ด้วย เช่น ส่ง request ต่อไปยัง App Server
- ตัวอย่าง:
 - Nginx
 - Apache HTTP Server
 - Microsoft IIS

◆ Web Application Server = App Server

- คือเซิร์ฟเวอร์ที่ประมวลผล Business Logic หรือ Dynamic Content เช่น การคำนวณ การเชื่อมต่อฐานข้อมูล การประมวลผลฟอร์ม ฯลฯ
- รับคำขอจาก HTTP Server (เช่น API request) และตอบกลับเป็น JSON, HTML ที่สร้างแบบ dynamic ฯลฯ
- ตัวอย่าง:
 - Node.js
 - Django / Flask (Python)
 - Spring Boot (Java)
 - .NET Core / ASP.NET

🧠 สรุปเบริยบเทียน

ประเภท	ทำหน้าที่	ตัวอย่าง
HTTP Server (Web Server)	รับ HTTP Request, ให้บริการ Static File, ส่งต่อไปยัง App Server	Nginx, Apache, IIS
Web Application Server (App Server)	ประมวลผล Business Logic, ติดต่อ DB, สร้าง response แบบ dynamic	Node.js, Flask, Spring Boot

HTTP Server กับ Web Application Server

เที่ยบได้กับ front-end กับ back-end หรือไม่?

✓ บางกรณีเที่ยบได้ (แต่ไม่ตรงทั้งหมด)

◆ HTTP Server ≈ Front-end (บางบทบาท)

HTTP Server (เช่น Nginx, Apache) มักทำหน้าที่เป็น Web Server ที่รับคำขอจากผู้ใช้ผ่านเบราว์เซอร์ และตอบกลับหน้าเว็บ (HTML, CSS, JS) หรือทำงานเป็น reverse proxy ส่งคำขอไปยัง Web Application Server

● ในระบบที่ใช้การเรนเดอร์ฝั่ง Server (server-side rendering), HTTP Server อาจส่งหน้าต่างๆ ไปยังผู้ใช้ จึงคล้ายกับ "Front-end" ของระบบ

◆ Web Application Server ≈ Back-end

Web Application Server (เช่น Node.js, Django, Flask, Spring Boot ฯลฯ) มักรับคำขอที่เป็น Logic เช่น API, Business Logic, Database Query

● จึงทำหน้าที่เป็น Back-end ได้ชัดเจนมากกว่า

แนวคิด	อธิบาย	ตัวอย่าง
HTTP Server	ให้บริการไฟล์ Static (HTML/CSS/JS) หรือทำหน้าที่เป็น Reverse Proxy	Nginx, Apache
Web Application Server	ประมวลผล logic, คำนวณ, เรียกฐานข้อมูล ฯลฯ	Django, Express.js
Front-end	ส่วนติดต่อผู้ใช้ที่รับบนเบราว์เซอร์	React, Vue, Angular
Back-end	ส่วนที่อยู่หลังบ้าน ทำงานกับข้อมูลและตระราก	Node.js API, Flask, Laravel

สรุป:

- HTTP Server และ Web Application Server เป็น องค์ประกอบของโครงสร้าง (infrastructure components)
- Front-end และ Back-end เป็น แนวคิดเชิงหน้าที่ (logical or functional roles)

จึงพูดได้ว่า:

- ◆ HTTP Server มักเกี่ยวข้องกับการให้บริการ Front-end
- ◆ Web Application Server ทำงานเป็น Back-end

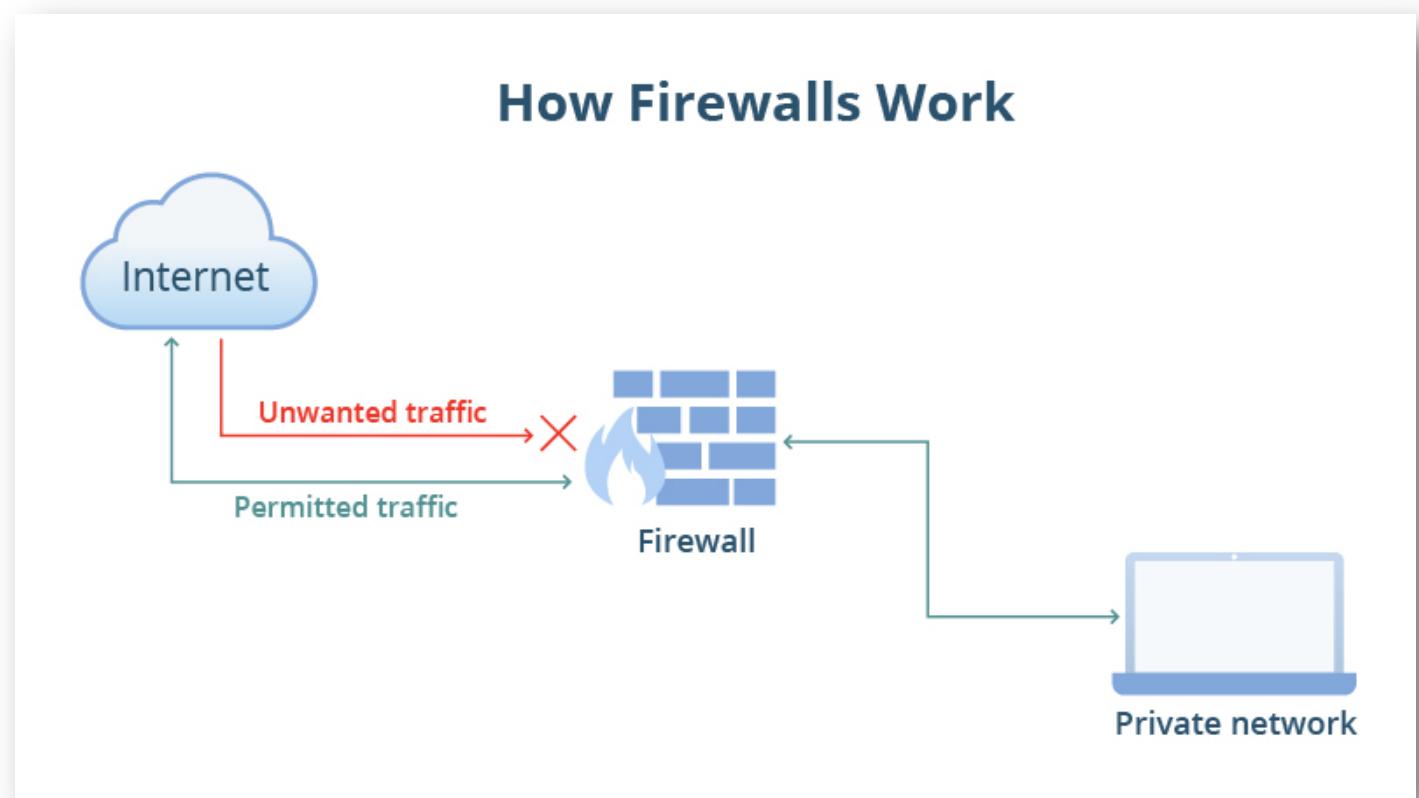
แต่ทั้งคู่ไม่ได้ตรงกับ Front-end / Back-end แบบ 1:1 เสมอไป



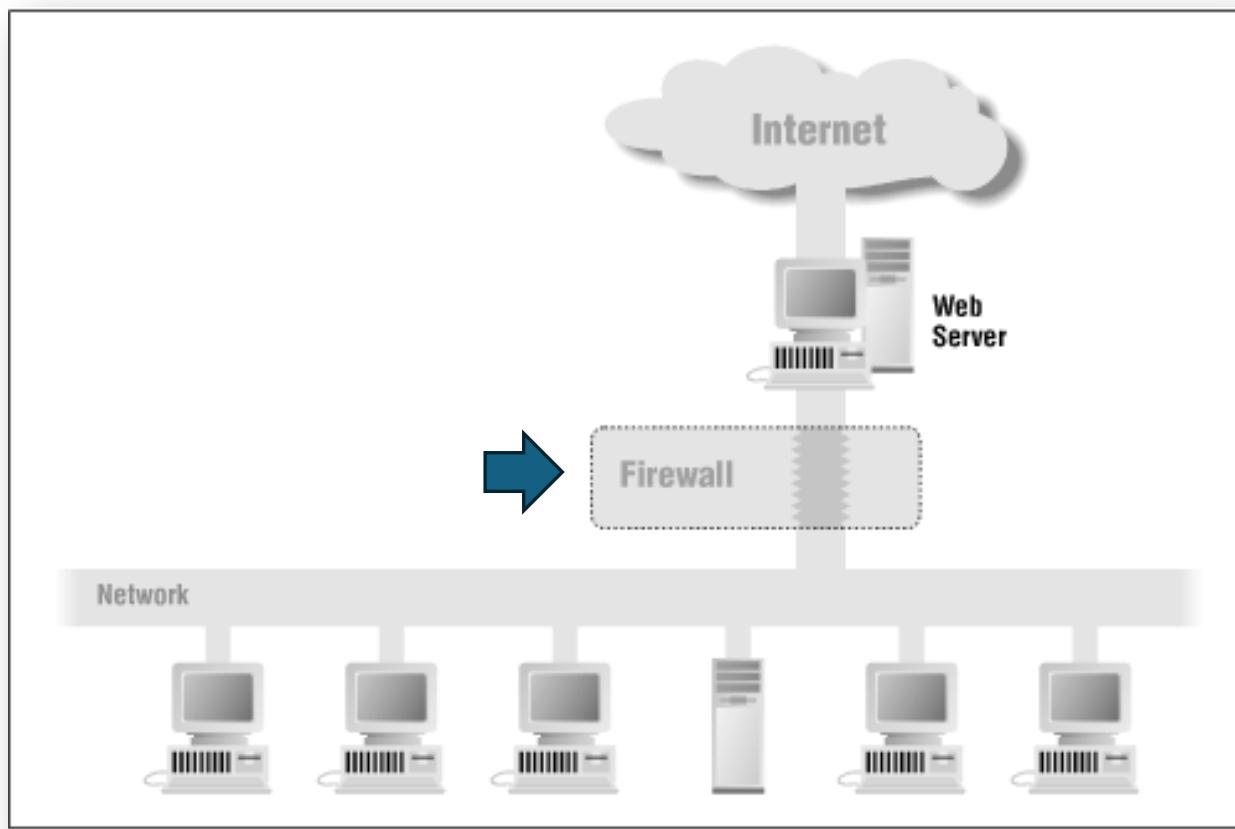
Firewall

Firewall

- A firewall is a network security device that **prevents unauthorized access to a network**
- It inspects incoming and outgoing traffic using a set of security rules to **identify and block threats**
- A firewall can be physical hardware, digital software, software as a service (SaaS) or a virtual private cloud.



Web Server with Firewall



ถ้าเรามี HTTP Server กับ Web Application Server

เราควรร่วง firewall ไว้ที่ตำแหน่งใด?

หากคุณมี HTTP Server และ Web Application Server (หรือ App Server) การร่วง Firewall ควรคำนึงถึงจุดที่ต้องการป้องกันและลักษณะของระบบที่แยกส่วน Frontend และ Backend ดังนี้:

🔧 โครงสร้างระบบโดยสังเขป

 Copy

```
[Client Browser ]  
↓  
[HTTP Server ] ← เปิดรับจาก Internet (Public)  
↓  
[Web Application Server ]  
↓  
[Database Server ]
```

1. ระหว่าง HTTP Server และ Web Application Server

- เพื่อควบคุมไม่ให้ใครก็ตามเรียกใช้งาน Application Server โดยตรง
- Firewall ควรอนุญาตเฉพาะ IP หรือ subnet ของ HTTP Server ให้สามารถเชื่อมต่อกับ App Server ได้

วัตถุประสงค์: ป้องกันไม่ให้ผู้โจมตีจากภายนอก (หรือเครือข่ายอื่น) เข้าถึง Web Application Server โดยตรง

2. ระหว่าง Web Application Server และ Database Server

- เป็นอีกขั้นความปลอดภัยที่ควรมี เพื่อให้เฉพาะ App Server เท่านั้นที่สามารถเข้าถึง Database ได้

วัตถุประสงค์: ป้องกันการโจมตีหรือการเข้าถึงฐานข้อมูลโดยตรงจาก HTTP Server หรือผู้ใช้ภายนอก

3. อาจมี Firewall ด้านหน้า HTTP Server

- ถ้า HTTP Server เป็นจุดที่เปิดรับจากอินเทอร์เน็ต อาจติดตั้ง Web Application Firewall (WAF) เพื่อกรอง Request ที่เป็นอันตราย เช่น SQL injection, XSS ฯลฯ

วัตถุประสงค์: ป้องกันการโจมตีระดับ Web เช่น OWASP Top 10



สรุปภาพรวมตำแหน่ง Firewall ที่แนะนำ

Copy

```
[Client 🌎]
  |
[Firewall (WAF) ✅] ← กรอง HTTP Requests
  |
[HTTP Server 🌐]
  |
[Firewall ✅]           ← อนุญาตเฉพาะ HTTP Server เข้าถึง App Server
  |
[Web App Server 🧠]
  |
[Firewall ✅]           ← อนุญาตเฉพาะ App Server เข้าถึง Database
  |
[Database Server 🗃]
```

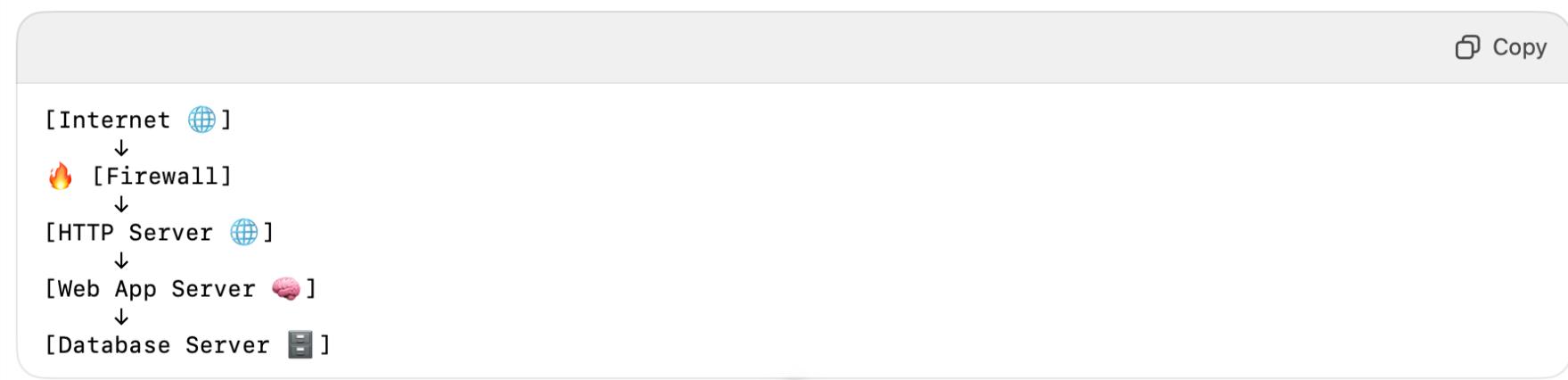
ถ้ามี Firewall ตัวเดียวเท่านั้น วางตรงไหนดีที่สุด

ถ้าคุณมี Firewall เพียงตัวเดียว และต้องเลือก ตำแหน่งเดียวที่ดีที่สุดในการวาง เพื่อรักษาความปลอดภัยของระบบโดยรวม — คำตอบที่เหมาะสมที่สุดคือ:

วาง Firewall ไว้ที่ “ขอบเครือข่าย” ระหว่างภายนอกกับระบบภายใน (Perimeter Firewall)

หรือพูดง่ายๆ ก็คือ ด้านหน้าของ HTTP Server เพื่อกรองทรัพฟิกจากอินเทอร์เน็ตเข้าสู่ระบบทั้งหมด

👉 ตัวอย่างโครงสร้าง:



เหตุผลว่าทำไมควรวางแผนนี้

1. ป้องกันตั้งแต่ขอบเครือข่าย

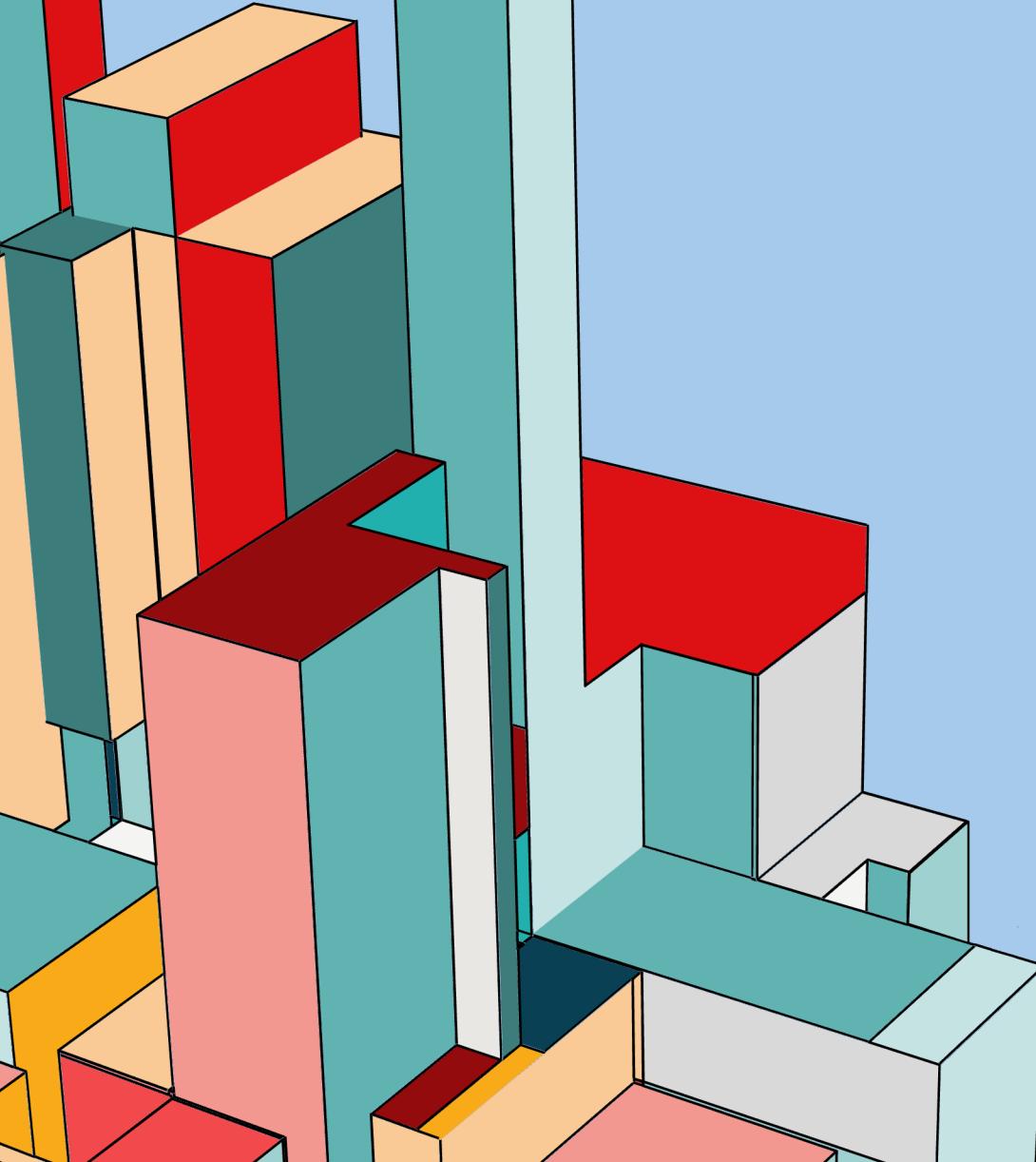
- เป็น แนวป้องกันแรก (first line of defense) ที่กรองทรัพฟิกขาเข้า
- อนุญาตเฉพาะพอร์ตที่จำเป็น เช่น 80 (HTTP) และ 443 (HTTPS)
- ป้องกัน port scanning, brute force, และการโจมตีอื่นๆ

2. จำกัดการเข้าถึงเฉพาะทรัพฟิกที่จำเป็น

- เช่น ป้องกันไม่ให้ผู้โจมตีจากภายนอกเข้าถึง App Server หรือ DB Server โดยตรง

3. ลดความเสี่ยงของระบบอื่นที่อยู่ใน Network เดียวกัน

- โดยอนุญาตแค่ Web Server เท่านั้นที่รับทรัพฟิกจากภายนอก
- ส่วนอื่นๆ อยู่ “หลัง Firewall” และสื่อสารภายในได้อย่างปลอดภัย

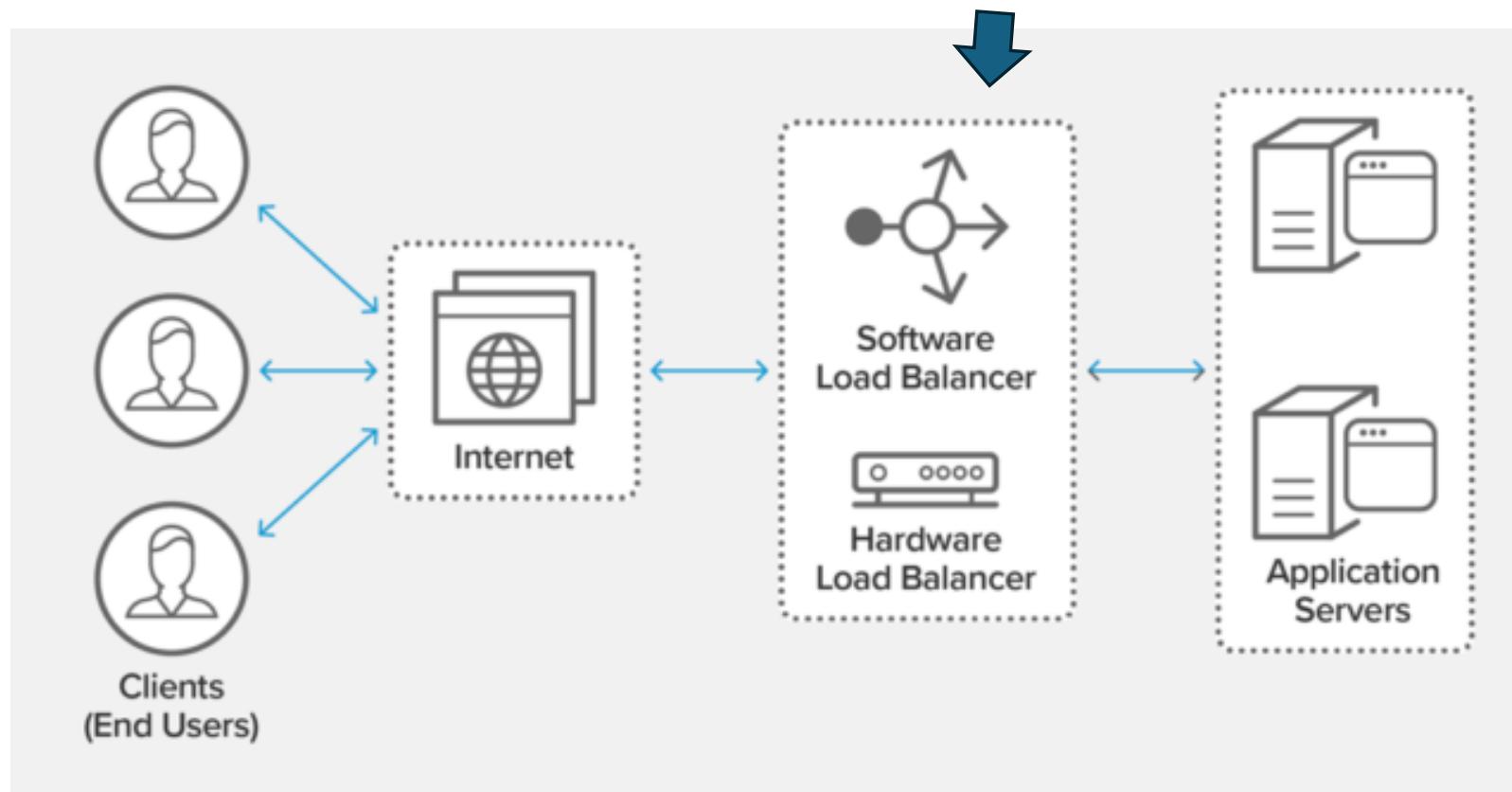


Load Balancer

Load Balancer

- Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*
- Modern high-traffic websites must serve hundreds of thousands of concurrent requests from users or clients
- To cost-effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers
- A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance

How to Deploy Load Balancer



<https://www.nginx.com/resources/glossary/load-balancing/>

Load balance คือวางแผนที่ต้องทำอะไร?

ขอแบบมี HTTP Server และ Web Application Server ในโครงสร้าง

ในระบบที่ประกอบด้วยทั้ง HTTP Server (Web Server) และ Web Application Server (App Server)

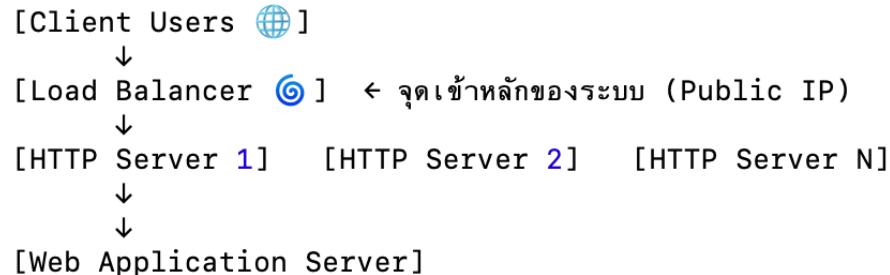
การวางแผน Load Balancer ควรขึ้นอยู่กับว่า:

- คุณต้องการ กระจายโหลดจากผู้ใช้ภายนอกเข้ามาอย่างไร
- และคุณมี หลายเครื่อง สำหรับ HTTP Server หรือ App Server หรือทั้งคู่หรือไม่

🎯 วงศ์ HTTP Server (กรณีมีหลาย HTTP Server)

เป็นกรณีที่พอบนอยู่ที่สุด

 Copy



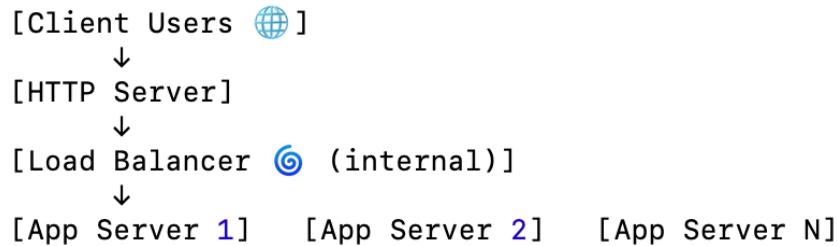
หมายสำคัญ:

- กระจายโหลด HTTP จากผู้ใช้เข้าสู่หลาย HTTP Server
- รองรับผู้ใช้งานจำนวนมากแบบ High Availability
- ให้ HTTP Server ทำ reverse proxy ไปยัง App Server อีกชั้น

⌚ ว่างไว้ระหว่าง HTTP Server และ App Server (กรณีมีหลาย App Server)

กรณีนี้ HTTP Server รับ request แล้วส่งต่อไปยัง Load Balancer อีกตัวที่แจกจ่ายไปยัง App Server หลายตัว

Copy

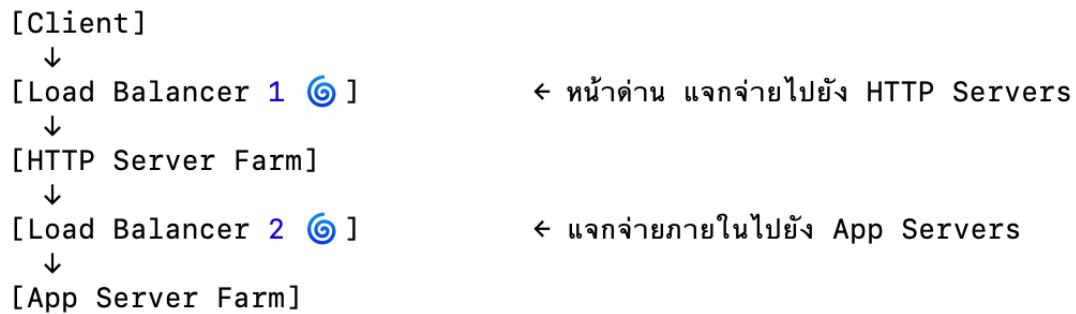


หมายเหตุ:

- ระบบที่ใช้ Business Logic เยอะ หรือโหลดหนักฝั่ง App
- มีหลาย App Server ทำงานเบื้องหลัง
- ต้องการขยายเฉพาะ Layer ของ App Server

⌚ วาง Load Balancer ทั้ง 2 ตำแหน่ง (ถ้าระบบใหญ่และมีหลาย Server ทั้งสองฝั่ง)

 Copy



หมายสำคัญ:

- ระบบใหญ่ระดับ Production ที่ต้องการความเสถียรสูงและรองรับผู้ใช้งานจำนวนมาก



สรุป

กรณี

Load Balancer ควรวางที่ไหน

มีหลาย HTTP Server

หน้า HTTP Server

มีหลาย App Server

หลัง HTTP Server

มีหลายทึ้งคู่

หน้า HTTP Server + หน้า App Server

มี Load Balancer ตัวเดียว

หน้า HTTP Server คือหมายที่สุด

ระบบขนาดเล็ก มีแค่ 1 Server

อาจยังไม่จำเป็นต้องมี Load Balancer

ถ้ามี HTTP server หลายตัว และ Application Server หลายตัว แต่มี load balance ตัวเดียว ทำได้หรือไม่?

คำถาม:

ถ้ามีทั้ง HTTP Server หลายตัว และ Application Server หลายตัว แต่มี Load Balancer เพียงตัวเดียว — แบบนี้ ทำได้หรือไม่?

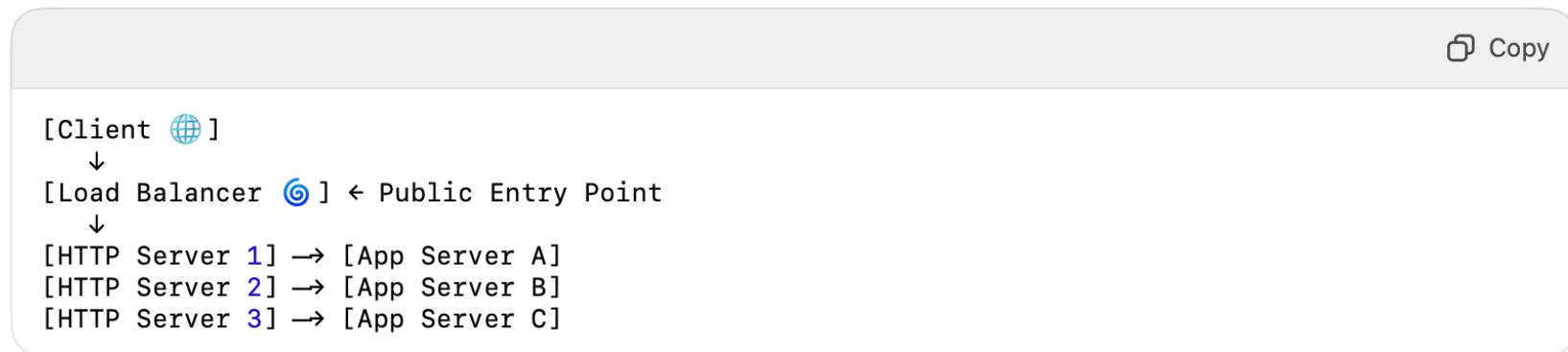
คำตอบ: ทำได้ครับ

แต่ต้องออกแบบ ตำแหน่งของ Load Balancer ให้ชัดเจนว่า

คุณจะให้มันกระจายโหลดไปที่ HTTP Servers หรือ Application Servers เท่านั้น (ไม่สามารถทำทั้งสองได้พร้อมกันจากตัวเดียวแบบตรงๆ)

กรณีที่ 1: วาง Load Balancer ไว้หน้า HTTP Server Farm

คือให้ Load Balancer ทำหน้าที่รับ request จากผู้ใช้ภายนอก แล้วแจกไปยัง HTTP Servers (HTTP Servers ทำหน้าที่เป็น reverse proxy ไปยัง App Servers หลายตัว)



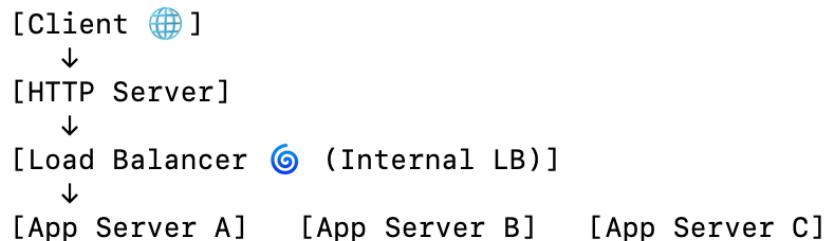
- เหมาะสมสำหรับระบบที่ต้องการควบคุมトラฟฟิกเข้าจากภายนอกเป็นหลัก
- App Servers อาจกำหนด routing ภายใน HTTP Servers (เช่น NGINX → upstream)

- 💡 ข้อดี: เรียนง่าย, ใช้ Load Balancer เดียวควบคุมหน้า Public
- 💡 ข้อเสีย: ถ้า HTTP Server ตัวใดล้ม ระบบบางส่วนจะเสีย เพราะมันผูกกับ App Server เช่น

กรณีที่ 2: วาง Load Balancer ไว้หน้า App Server Farm (ผ่าน HTTP Server เดียวหรือรวม)

ให้ HTTP Server ตัวเดียวหรือรวมกันทำหน้าที่รับ request ก่อน แล้วส่งต่อไปยัง Load Balancer → App Server หลายตัว

 Copy



- เหมาะสมถ้าคุณต้องการควบคุมโหลดที่ระดับ Application Logic เช่น CPU load, Memory ใช้เยอะ
- HTTP Server ต้องเก่งเรื่อง routing และ health-check

- ข้อดี: เหมาะกับงานหนักฝั่ง App เช่น ML Inference, API Heavy
- ข้อเสีย: HTTP Server จะกลายเป็น single point of failure (ถ้าไม่มี redundant)

แนวทางแนะนำ

ถ้ามี Load Balancer แค่ตัวเดียว แนะนำให้เลือก:

- วางไว้หน้า HTTP Server Farm

ถ้าคุณต้องการรองรับผู้ใช้จากภายนอกเยอะๆ มี Static Content หรือเน้น Public Traffic
(ให้ HTTP Server ไปจัดการ App Server ด้านหลังต่อเอง)



คำถาม	คำตอบ
มี Load Balancer ตัวเดียว ใช้ได้ไหม?	<input checked="" type="checkbox"/> ใช้ได้
ควรวางที่ไหน?	วางหน้า HTTP Server (หากต้องรองรับผู้ใช้ภายนอก)
กระจายได้ทั้ง HTTP และ App พร้อมกันไหม?	 ไม่ได้โดยตรง ต้องเลือก layer ที่จะ balance
อยากกระจายทั้งสอง layer ต้องทำไง?	 เพิ่ม Load Balancer ตัวที่สอง หรือใช้ Reverse Proxy ภายใน

ใช้ Reverse Proxy ภายใน ช่วยอธิบายเพิ่ม

แน่นอนครับ! ต่อจากประเด็นก่อนหน้านี้ที่คุณถามว่า:

❓ ใช้ Reverse Proxy ภายใน ช่วยอธิบายเพิ่ม

⌚ Reverse Proxy คืออะไร?

Reverse Proxy คือเซิร์ฟเวอร์ตัวกลางที่รับคำขอ (Request) จากผู้ใช้ หรือจาก HTTP Server และส่งต่อไปยัง Backend Server (เช่น Application Server) และรับคำตอบกลับมาเพื่อส่งคืนผู้ใช้

ในบริบทของภายนอกระบบ (ไม่ใช่ public) Reverse Proxy มักใช้เพื่อ:

- กระจายโหลด (load balancing)
- ช่องโ何况สร้างภายนอกของระบบ
- จัดการ routing ให้ App หลายตัว
- เพิ่มชั้นความปลอดภัย (เช่น ตรวจ token ก่อนส่งให้ App Server)

📦 กรณีใช้งาน: คุณมี Load Balancer ตัวเดียว และใช้ Reverse Proxy ภายใน

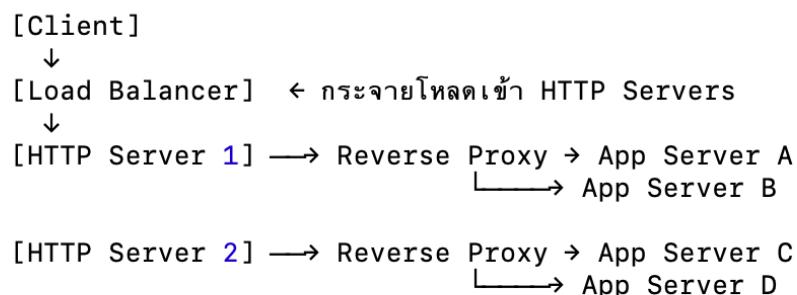
📌 เป้าหมาย:

ให้ Load Balancer แจก request ไปยัง HTTP Servers

จากนั้น HTTP Server (เช่น NGINX) ทำหน้าที่เป็น Reverse Proxy ภายใน เพื่อกระจายไปยัง Application Servers หลายตัว

🏗️ โครงสร้างระบบ

 Copy



ข้อดีของการใช้ Reverse Proxy ภายใน

จุดเด่น

ประโยชน์

 ไม่ต้องใช้ Load Balancer หลายตัว

ลดค่าใช้จ่ายและความซับซ้อน

 ช่อง IP หรือจำนวน App Server

เพิ่มความปลอดภัย

 รองรับการเพิ่ม App Server แบบ dynamic

ใช้ร่วมกับ health check ได้

 รองรับ logging, caching และ rate limit

ช่วยให้ควบคุม traffic ภายในได้ดีขึ้น