Web Service Deployment

莊博傑 (bogay)

Outline

- Introduction
- Deployment
- Architecture

The Evolution of Web Service Development

- CGI (Common Gateway Interface)
- FAMP/LAMP Stack (Linux/FreeBSD, Apache, MySQL, PHP)
- Backend/Frontend Separation (SPA + API service)
- Server-Side Rendering (SSR)
- Microservices
- Backend for Frontend (BFF)
- Serverless

Introduction

Introduction

- Web server
  - Apache, Nginx, Caddy, Traefik, Pingora
- API service
  - HTTP, gRPC, GraphQL
- Storage / Cache
  - SQL: MySQL, PostgreSQL
  - NoSQL: MongoDB, Neo4j, Redis / Valkey
  - Object storage: MinIO
- Monitoring
  - Prometheus, Grafana
  - OpenTelemetry (OTel)

Web Server

- Separation of Concerns
- Performance and Scalability
- Security (e.g. TLS)
- Centralized Request Management
  - log, rate limiting, WAF, etc.
- Integration and Flexibility
  - static files
  - upstream APIs

Web Server: Apache

- Directory-level configuration without restart (.htaccess)
- Built-in scripting support: mod_php, mod_perl
- Modular design
  - auth, rewrite, caching, etc
- Read more
  - 2024 SA: FAMP
  - Confusion Attacks: Exploiting Hidden Semantic Ambiguity in Apache HTTP Server! | DEVCORE 戴夫寇爾

Web Server: Nginx

- Event-driven architecture
  - thousands of concurrent connections efficiently
  - Inside NGINX: How We Designed for Performance & Scale
- Centralized configuration
- Reverse proxy & static content specialist
- Low resource usage
- Fewer functionality
  - people.cs
  - https://people.cs.nycu.edu.tw/~kentang/

Web Server: Traefik

- Dynamic service discovery
  - Docker, K8s label
- Automatic HTTPS
- Rich middleware support
  - rate limiting, authentication, etc.
- Simple configuration
- [Docker - Traefik]

```
services:
  traefik:
    image: "traefik:v3.4"
    container_name: "traefik"
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    networks:
      - proxy
    command:
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--providers.docker.network=proxy"
      - "--entryPoints.web.address=:80"
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
  whoami:
    image: "traefik/whoami"
    restart: unless-stopped
    networks:
      - proxy
    labels:
      - "traefik.enable=true"
      -
"traefik.http.routers.whoami.rule=Host(`whoami.docker.local
host`)"
      - "traefik.http.routers.whoami.entrypoints=web"
networks:
  proxy:
    name: proxy
```

Web Server: Caddy

- Simplest configuration syntax
- Automatic HTTPS out-of-the-box
- Lightweight & secure by default
- Reverse proxy & static file server (doc)

```
example.com {
    root * /var/www
    reverse_proxy /api/* localhost:5000
    file_server
}
```

Web Server: Pingora

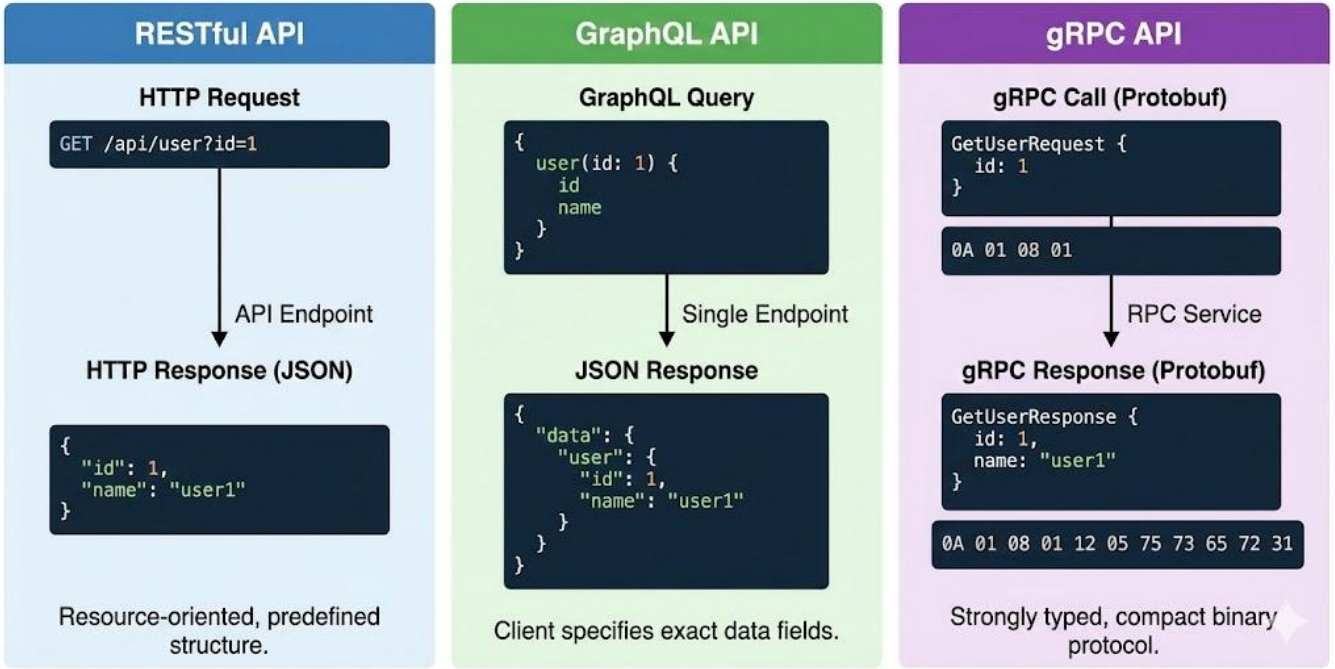
- Modern Rust-based framework built by Cloudflare
- Significantly better performance
- Programmable & extensible
  - custom logic and middleware in Rust

How we built Pingora, the proxy that connects Cloudflare to the Internet

API Service

- Integration and Reusability
  - third-party services, cross platforms
- RESTful, GraphQL, gRPC
- Use programming languages you like
  - REST server written in bash

API Service

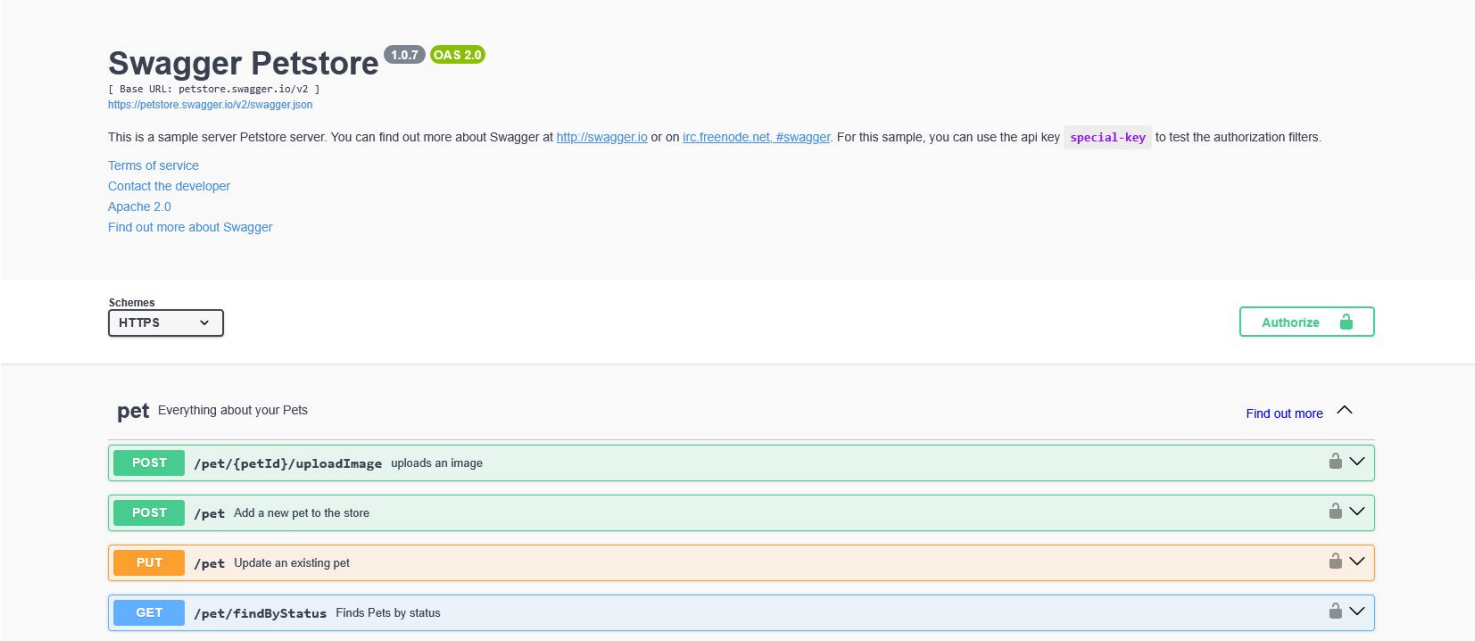


credit: gemini

OpenAPI Spec (formerly Swagger)

- What is OpenAPI?
  - A standard, language-agnostic interface description for REST APIs
  - Usually written in YAML or JSON
- API-First Design
  - Single Source of Truth
  - Documentation: e.g., Swagger UI, Redoc
  - Code Generation (e.g. OpenAPITools/openapi-generator)
- Workflow:
  - Write Code (Rust/Axum) → Extract Spec (utoipa) → Swagger UI
  - Write Spec → Generate Code

OpenAPI Spec (formerly Swagger)



Swagger Petstore

Storage

- Independent scalability
- High availability & fault tolerance
- Performance optimization
- Security & compliance
- Operational flexibility
- Execute the app as one or more stateless processes

Storage: MySQL / MariaDB

- MySQL
  - JSON Support: Native JSON data type
- MariaDB (The Community Fork)
  - Highly compatible with MySQL
    - MariaDB versus MySQL - Compatibility | Release Notes
  - Faster
    - How MariaDB and MySQL performance changed over releases
  - More storage engines (Aria, XtraDB)
  - Mature clustering (Galera Cluster)

Storage: PostgreSQL

- One of most famous open source relational database
  - Object-Relational (ORDBMS)
- Advanced Data Types: Native support for JSON/JSONB
- Extensibility: e.g. PostGIS, pgvector
- Concurrency: Uses MVCC (Multi-Version Concurrency Control)

Storage: MongoDB

- The Leading NoSQL Document Database
  - Schema-less: BSON (Binary JSON)
  - Scalability: Sharding, Replica Sets
    - PostgreSQL: Citus
    - MariaDB: Spider
- License Controversy (SSPL)
  - Moved from AGPL to SSPL (2018)
  - Not Open Source: Rejected by OSI (Open Source Initiative)
  - Repo Removal: Debian, Fedora, and RHEL

Storage: Redis / Valkey

- In-Memory Key-Value Store
  - Latency: Stores data in RAM
- Why do we need it? (vs. SQL)
  - Caching
  - Ephemeral Data: e.g. User Sessions
  - Queues: e.g. Celery, RQ
- Why Valkey?
  - The open-source (BSD) fork of Redis under the Linux Foundation.
  - 針對雲端服務供應商再出手，Redis 7.4轉移使用RSAL與SSPL雙重授權
  - Redis is now available under the AGPLv3 open source license

Storage: MinIO

- High-Performance Object Storage
  - S3 Compatible
- Object Storage
  - Unstructured Data: images, videos, logs, and backups (BLOBs)
  - Make app stateless (No local disk)
- Docker release? · Issue #21647 · minio/minio

Monitoring

- Why Monitoring? (Stop Flying Blind)
  - Visibility: You can't fix what you can't see.
  - Goal: Minimize MTTR (Mean Time To Recovery).
    - Failures are inevitable (e.g., Cloudflare Outage).
- The 3 Pillars of Observability
  - Metrics: "Is there a problem?" (Prometheus)
    - Trends, CPU usage, Request rates.
  - Logs: "What is the problem?" (OpenSearch / Loki)
    - Error messages, Stack traces.
  - Traces: "Where is the problem?" (OpenTelemetry)
    - Request flow across microservices.

Monitoring: Prometheus

- The Industry Standard for Metrics
  - A Cloud Native Computing Foundation (CNCF) graduated project
  - Time-Series Database (TSDB)
- Pull-Based Architecture
  - Periodicallly pulls metrics from HTTP endpoints (e.g., /metrics)
- PromQL (Prometheus Query Language)
  - A powerful functional language to aggregate and analyze data (e.g., `rate(http_requests_total[5m])`)
- Ecosystem:
  - e.g., Node Exporter, Postgres Exporter

Monitoring: Grafana

- The Open Observability Platform
  - Visualizes data stored in Prometheus (and many other sources)
  - Agnostic: Prometheus (Metrics), Loki (Logs), Tempo (Traces), SQL databases
- Key Features:
  - Dashboards: Customizable panels (Graphs, Gauges, Heatmaps)
  - Community: Thousands of pre-built dashboards available
  - Alerting: Visual alert rules management.
- Transforms raw, unreadable PromQL queries into Actionable Insights

Monitoring: OpenTelemetry (OTel)

- The Unified Standard
  - Standardizes the generation of Metrics, Logs, and Traces.
  - CNCF's 2nd most active project (just after Kubernetes).
- Why OTel?
  - Write Once, Send Anywhere: Instrument your code once, export to any backend (Prometheus, Jaeger, Datadog...)
  - Decoupling: Separates how data is generated from where it is stored
- Architecture
  - OTel Collector: A universal proxy to receive, process, and export data.

Deployment

API Service: axum

- Repo: https://github.com/tokio-rs/axum
- `axum` is a web application framework that focuses on ergonomics and modularity

```
❯ cat new my_quote
❯ cd my_quote
❯ cargo add axum
❯ cargo add tokio@1 -F full
```

```
use axum::{Router, routing::get};

#[tokio::main]
async fn main() {
    // build our application with a route
    let app = Router::new()
        // `GET /` goes to `root`
        .route("/", get(root));

    // run our app with hyper,
    // listening globally on port 3000
    let listener =
tokio::net::TcpListener::bind("0.0.0.0:3000")
    .await.unwrap();
    axum::serve(listener, app).await.unwrap();
}

// basic handler that responds with a static string
async fn root() -> &'static str {
    "Hello, My quote!"
}
```

API Service: axum

```
# my_quote/Dockerfile
FROM docker.io/rust:1.91-slim-trixie AS builder

WORKDIR /opt/my_quote
ADD . .

RUN cargo build --release

FROM docker.io/debian:trixie-slim

COPY --from=builder
/opt/my_quote/target/release/my_quote
/opt/my_quote/my_quote

CMD ["/opt/my_quote/my_quote"]
```

- Rust: https://hub.docker.com/_/rust
- Debian: https://hub.docker.com/_/debian

```
# docker-compose.yaml
services:
  quote:
    build: my_quote
    restart: unless-stopped
    init: true
    ports:
      - 3000:3000
```

```
❯ curl -kL http://localhost
Hello, My quote!
```

Web Server: Caddy

- Doc: https://hub.docker.com/_/caddy


```yaml
services:
  caddy:
    image: docker.io/caddy:2.10-alpine
    restart: unless-stopped
    ports:
      - 80:80
      - 443:443
      - 443:443/udp # For HTTP/3
    volumes:
      - caddy_data:/data
  # ...

volumes:
  caddy_data: {}
```

Web Server: Caddy

- Doc: https://caddyserver.com/docs/caddyfile
- Automatic HTTPS: https://caddyserver.com/docs/automatic-https

```
❯ cat Caddyfile
localhost {
        reverse_proxy quote:3000
}
❯ curl -k https://localhost
Hello, My quote!
```

```
services:
  caddy:
    image: docker.io/caddy:2.10-alpine
    restart: unless-stopped
    ports:
      - 80:80
      - 443:443
      - 443:443/udp # For HTTP/3
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - caddy_data:/data
  # ...

volumes:
  caddy_data: {}
```

Storage: PostgreSQL

- Doc: https://hub.docker.com/_/postgres
- volumes
  - default: `/var/lib/postgresql`
  - Can be override by `PGDATA`
- environment variable
  - `POSTGRES_PASSWORD`
  - `POSTGRES_USER`
  - `POSTGRES_DATABASE`

```
❯ docker compose exec db psql \
    -U quote -d quote
psql (18.1 (Debian 18.1-1.pgdg13+2))
Type "help" for help.

quote=#
```

```
services:
  # ...
  db:
    image: docker.io/postgres:18-trixie
    restart: unless-stopped
    volumes:
      - postgres_data:/var/lib/postgresql
    environment:
      - POSTGRES_USER=quote
      - POSTGRES_PASSWORD=quote
      - POSTGRES_DATABASE=quote

volumes:
  # ...
  postgres_data: {}
```

Storage: PostgreSQL

- Hard-coded credential and config in `docker-compose.yaml` is bad
- Alternatives
  - env_file
  - secret
- Read more
  - Store config in the environment
  - Analyzing the Hidden Danger of Environment Variables for Keeping Secrets
  - What is OpenBao?

```
services:
  # ...
  db:
    image: docker.io/postgres:18-trixie
    restart: unless-stopped
    volumes:
      - postgres_data:/var/lib/postgresql
    environment:
      - POSTGRES_USER=quote
      - POSTGRES_PASSWORD=quote
      - POSTGRES_DATABASE=quote
volumes:
  # ...
  postgres_data: {}
```

```
❯ docker compose exec db psql \
    -U quote -d quote
psql (18.1 (Debian 18.1-1.pgdg13+2))
Type "help" for help.

quote=#
```

Storage: PostgreSQL

- Hard-coded credential and config in `docker-compose.yaml` is bad

```
# .env
POSTGRES_USER=quote
POSTGRES_DATABASE=quote
POSTGRES_PASSWORD_FILE=/run/secrets/postgres_password
```

```
services:
  # ...
  db:
    image: docker.io/postgres:18-trixie
    restart: unless-stopped
    volumes:
      - postgres_data:/var/lib/postgresql
    secrets:
      - postgres_password
    env_file:
      - .env

volumes:
  # ...
  postgres_data: {}

secrets:
  postgres_password:
    file: ./secret/postgres_password
```

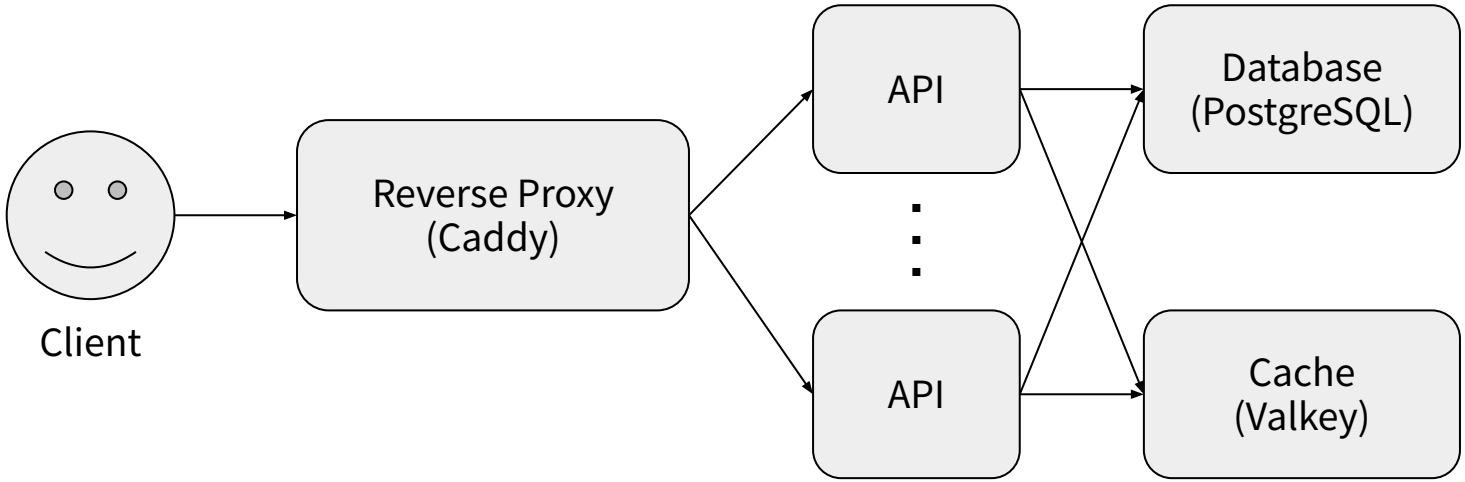
Cache: Valkey

- Doc: https://hub.docker.com/r/valkey/valkey/

```
services:
  # ...
  cache:
    image: docker.io/valkey/valkey:9-trixie
    restart: unless-stopped

volumes:
  caddy_data: {}
  postgres_data: {}

secrets:
  postgres_password:
    file: ./secret/postgres_password
```

```
❯ docker compose exec cache valkey-cli -h
cache
cache:6379> set foo bar
OK
cache:6379> get foo
"bar"
cache:6379>
```

API Service: Connect to Database and Cache



Monitoring: Prometheus

- Doc: https://prometheus.io/docs/prometheus/latest/getting_started/ https://hub.docker.com/r/prom/prometheus
- Example config

```
services:
  # ...
  prom:
    image: docker.io/prom/prometheus:v3.7.3
    restart: unless-stopped
    ports:
      - 9090:9090
    volumes:
      - prom_data:/prometheus
      -
./prometheus.yml:/etc/prometheus/prometheus.
yml

volumes:
  # ...
  prom_data: {}
```

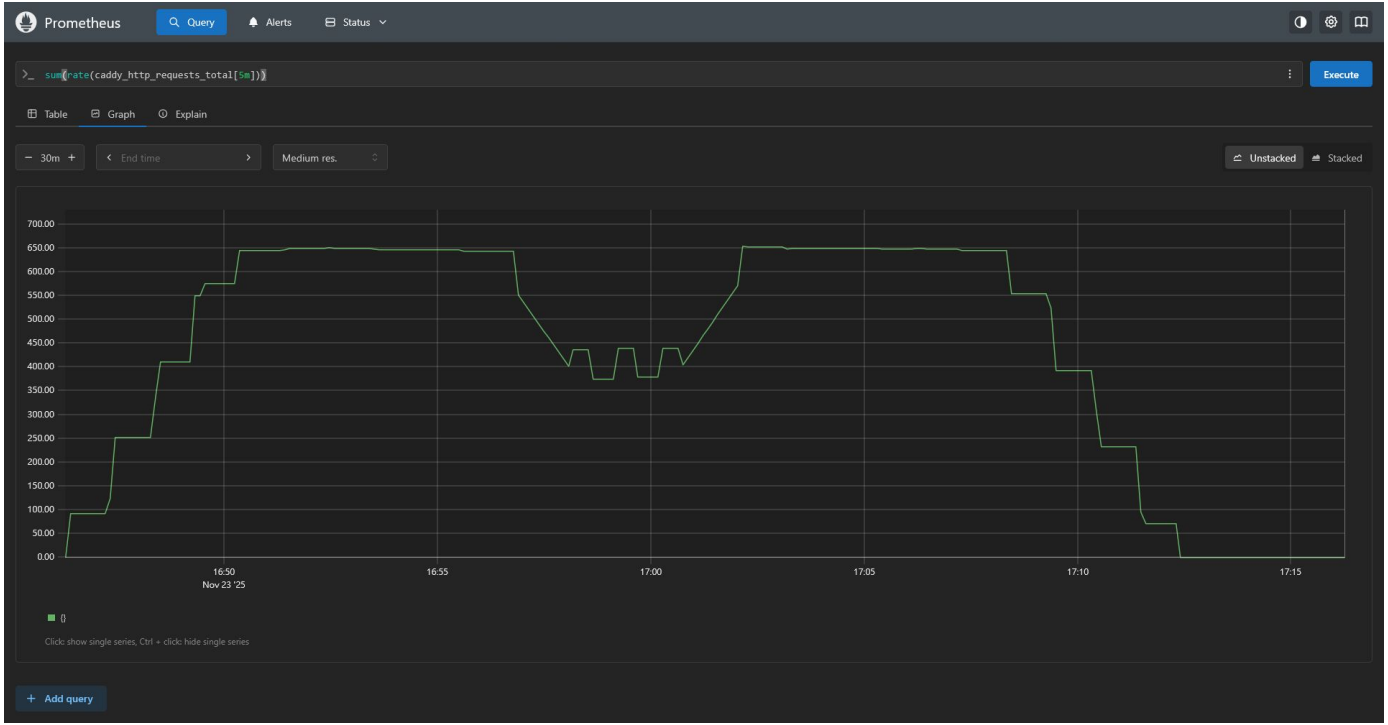
Monitoring: Prometheus

- Doc: https://prometheus.io/docs/prometheus/latest/configuration/configuration/
- Example config
- metrics (Caddyfile directive) — Caddy Documentation

```
# prometheus.yml
scrape_configs:
  - job_name: caddy
    static_configs:
      - targets: [web:9090]
    tls_config:
      insecure_skip_verify: true
```

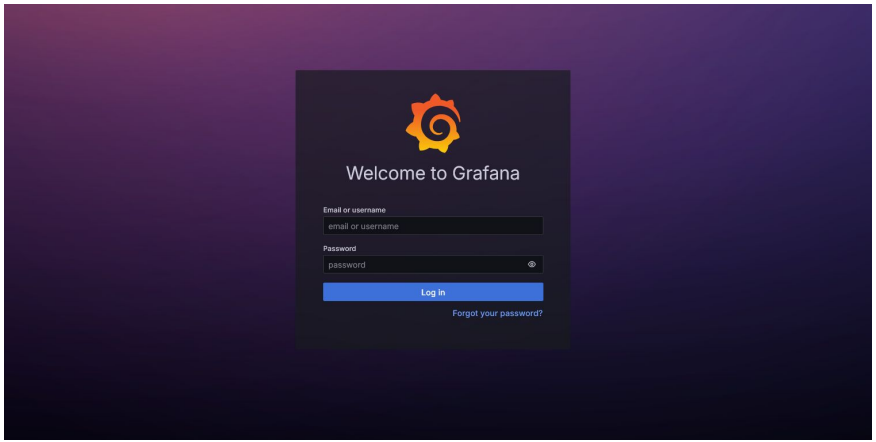
```
{
    metrics
}

:9090 {
    metrics /metrics
}

# ...
```

Monitoring: Prometheus



Monitoring: Grafana

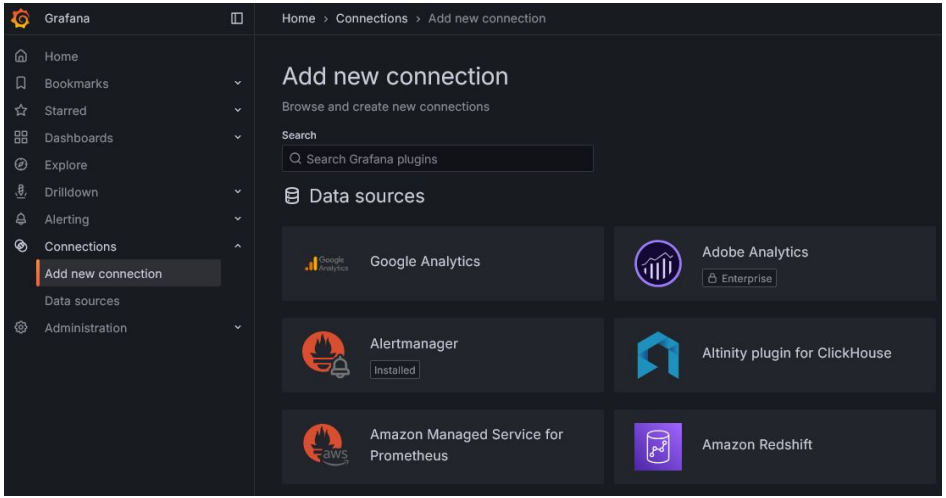
- Doc: https://grafana.com/docs/grafana/latest/setup-grafana/installation/docker/
- Default credential: admin/admin




```
services:
  # ...
  grafana:
    image: docker.io/grafana/grafana:12.1
    restart: unless-stopped
    volumes:
      - grafana_data:/var/lib/grafana
    ports:
      - 3000:3000

volumes:
  # ...
  grafana_data: {}
```

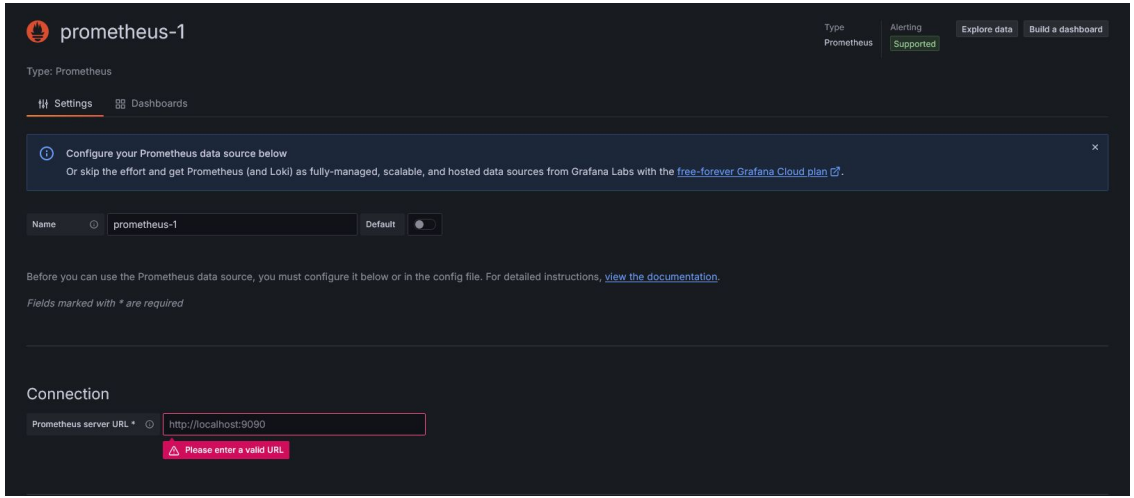
Monitoring: Grafana

- Connections > Add new connection
- Search “Prometheus”
- Select “Prometheus”
- Press “Add new data source”



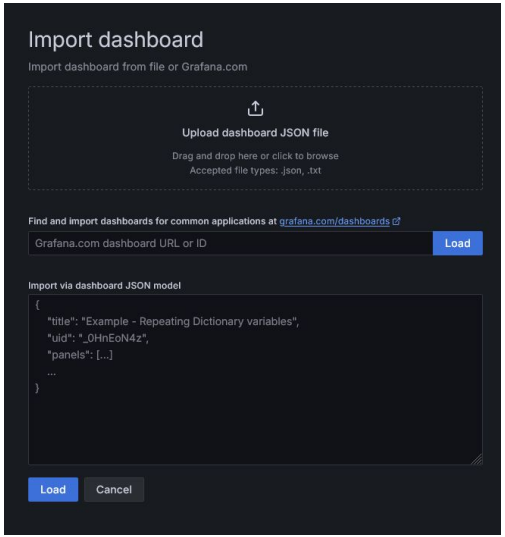
Monitoring: Grafana

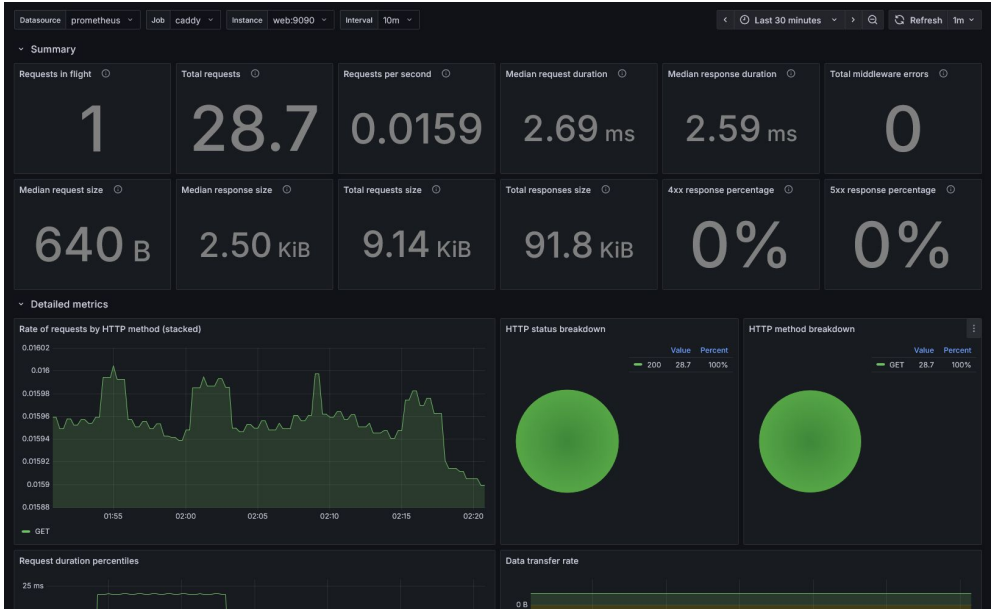
- Fill “Connection”
  - http://prom:9090
- Press “Save & Test”
- Done 🎉



Monitoring: Grafana

- Find ID on [Grafana dashboards | Grafana Labs]
- Dashboards > New > Import

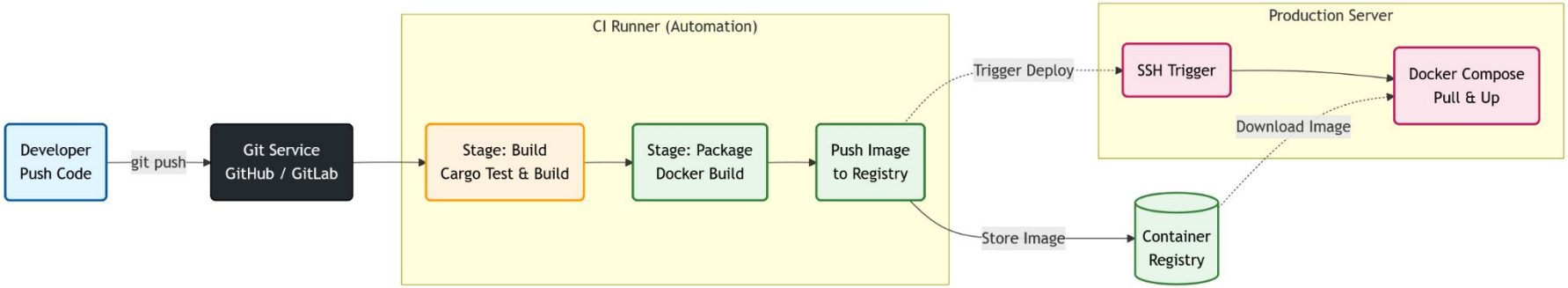




CI/CD (Continuous Integration / Continuous Deployment)

- Automate Everything
  - Stop manually SSH-ing into servers to git pull and restart
  - CI: build and test code every time you push
  - CD: release and deploy the application to production
- Why do we need it?
  - Consistency: Eliminates "It works on my machine" issues
  - Reliability: Prevents human errors during manual deployment
  - Speed: Faster feedback loop for developers

CI/CD


Developer
Push Code
git push
Git Service
GitHub / GitLab
CI Runner (Automation)
Stage: Build
Cargo Test & Build
Stage: Package
Docker Build
Push Image
to Registry
Trigger Deploy
Store Image
Production Server
SSH Trigger
Docker Compose
Pull & Up
Download Image
Container
Registry

CI/CD: GitHub Actions

- Workflow as Code
  - Defined in `.github/workflows/*.yaml`
- Marketplace Ecosystem
  - Thousands of pre-built actions available (e.g., actions/checkout, docker/login-action).
  - Example: A simple Rust build workflow.

```yaml
name: Rust CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v6
      - name: Build
        run: cargo build --verbose
```

CI/CD: GitLab CI

- The Enterprise & Lab Favorite
  - Widely used in private organizations (e.g. NYCU CSIT).
- Configuration
  - Defined in a single `.gitlab-ci.yml` file in the root directory.
- Evolution of Reusability
  - Traditional (include)
  - Modern (CI/CD Components)

```yaml
image: rust:latest

stages:
  - build
  - test

build_job:
  stage: build
  script:
    - cargo build --release

test_job:
  stage: test
  script:
    - cargo test
```

CI/CD: GitLab Auto DevOps

- Zero-Configuration CI/CD
  - Detects your code language and automatically builds, tests, and deploys.
  - No .gitlab-ci.yml needed (by default).
- How it works?
  - Uses Herokuish buildpacks to detect project type
    - Cargo.toml -> Rust
    - package.json -> Node.js
  - Automatically creates a Docker image and deploys to Kubernetes (if connected).
- Pros & Cons
  - Speed: Instant pipeline for prototypes.
  - Magic: Harder to debug or customize specific needs (e.g., specific system dependencies)

IaC (Infrastructure as Code)

- What is IaC?
  - Managing and provisioning resources through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
- Pets vs. Cattle
  - Pets (Manual): Hard to reproduce.
  - Cattle (Automated): Reproducible and scalable.
- Benefits:
  - Version Control
  - Reproducibility
  - Idempotency

IaC: Ansible

- Configuration Management Tool for existing servers
  - installing software
  - managing files
  - configuring services on
- Key Features:
  - Agentless: Uses SSH. No extra agents
  - Declarative: Playbooks -> desired state
  - Push-based

```
- name: "Install Nginx and dependencies"
  become: true
  ansible.builtin.apt:
    name:
      - nginx=1.22.*
      # Install ModSecurity for Nginx
      - libnginx-mod-http-modsecurity
      # Make sure TCP forward works in
SSH(gitlab)
      - libnginx-mod-stream
      - logrotate
    allow_change_held_packages: true
    state: present
  notify: "Restart nginx"
```

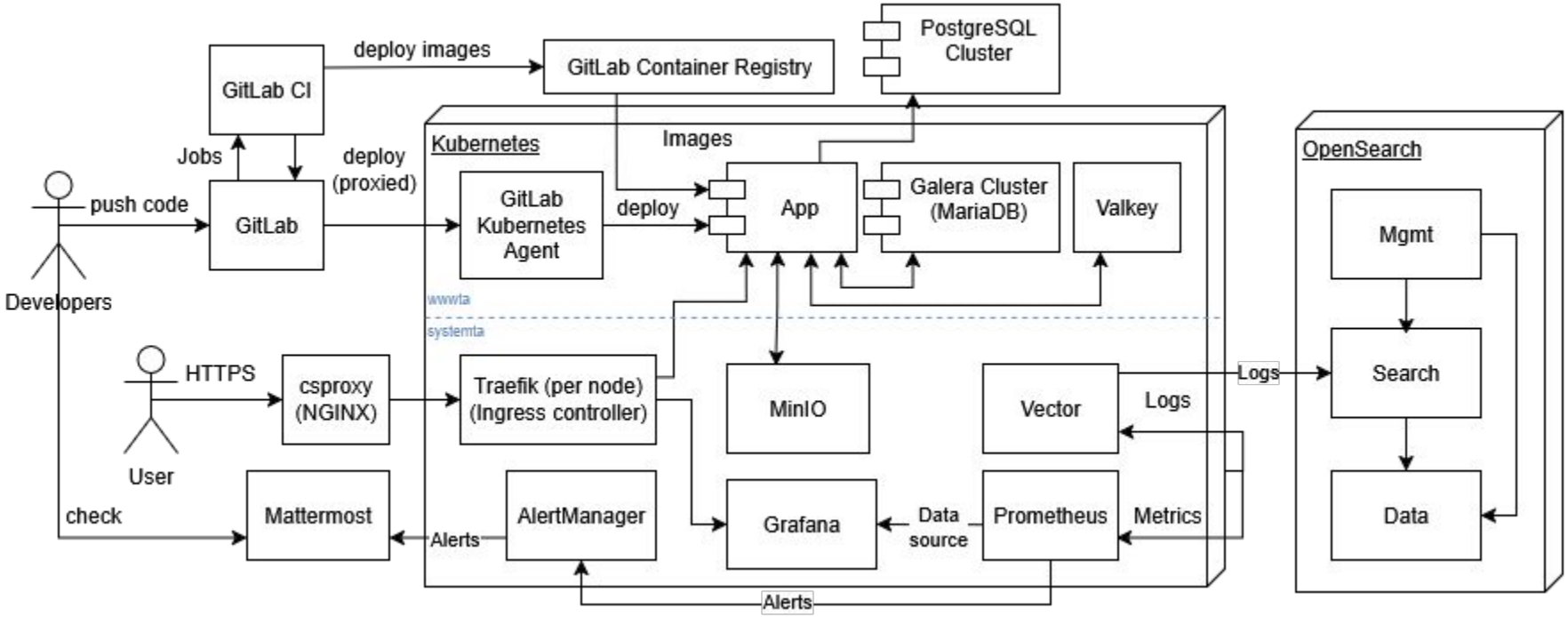
IaC: Terraform

- Infrastructure Provisioning Tool
  - Creating and managing the infrastructure itself (VMs, Networks, Load Balancers, DNS)
- Key Features
  - State Management: Keeps state (.tfstate)
  - Immutable Infrastructure
  - Multi-Cloud Support: AWS, GCP, Azure, Proxmox, VMware, etc
- License
  - HashiCorp adopts Business Source License
  - OpenTofu

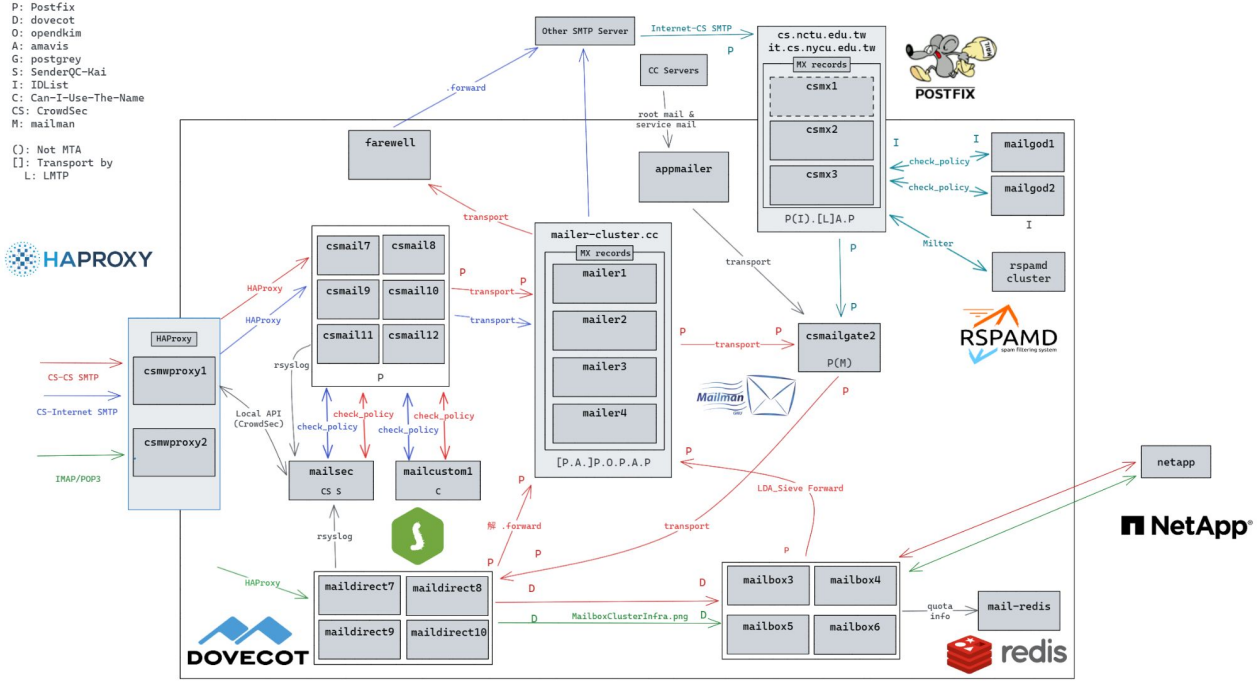
```
resource "minio_iam_user" "rw" {
  name = local.rw_account
  tags = {
    iac = true
  }
}

resource "minio_iam_user_policy_attachment"
"rw" {
  user_name   = minio_iam_user.rw.id
  policy_name = minio_iam_policy.rw.id
}
```

Architecture

Overview: CSIT www/system Group



Overview: CSIT mail Group



[2025 丁組招生線上說明會簡報 公開版]

Go Further

- Multi cluster
  - Zone, Region
  - Service mesh
- Multi cloud
- Auto scaling

114學年度 下學期 網路管理實務 課程綱要

下載課程綱要Word檔　下載課程綱要ODT檔

課程名稱：(中文) 網路管理實務 (英文) Network Administration Practice	開課單位：資科工碩 當期課號：535525
授課教師：蔡孟勳	永久課號：CSIC30210
先修科目或先備能力： 1. 計算機概論 2. 具備 C 或其他程式語言的程式撰寫能力 3. 計算機網路概論 4. 計算機系統管理	學分數：3.00 必/選修：選修 上課時間/教室：M567-EDB27[GF]

課程概述與目標：
本課程將深入淺出地介紹各項計算機網路管理要務，目的是讓學生們可融入系上計算機中心的學習環境，還可建構在計算機系統管理課程的學習基礎上，更進一步探討各種網路應用伺服器的管理，包括Mail System、DNS、VPN、LDAP 等各種常見的網路服務、多伺服器之間的協同作業等課題，授課內容為理論搭配系上工作站常見問題的處理經驗，期望學生能透過此課程學到網路服務管理者所應具備的技術及解決問題的能力，且能夠成為一位可以直接上任的優秀系統及網路服務管理者。

教科書（請註明書名、作者、出版社、出版年等資訊）：
1. Unix and Linux System Administration Handbook (5th Edition) By Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, Dan Mackin, Addison-Wesley Professional, 2017.
2. Slides in class

Appendix

Vector Database

- The Database for the AI Era
  - Stores High-Dimensional Vectors (Embeddings)
- Why do we need it? (Semantic Search)
  - Traditional Search: Matches keywords (e.g., "Puppy" != "Dog")
  - Vector Search: Matches meaning (e.g., "Puppy" ~= "Dog")
- Use Cases:
  - LLM RAG (Retrieval-Augmented Generation)
  - Recommendation Systems
  - Image/Audio Search: Search by content, not metadata.
- Common Solutions:
  - Specialized: Qdrant, Milvus, Chroma.
  - Integrated: pgvector (PostgreSQL extension)

Serverless

- What is Serverless?
  - Misnomer: There are still servers, but you don't manage them
  - Event-driven
  - Pay-as-you-go
- Example: Cloudflare Workers
  - Runs JavaScript/Wasm isolates on the Edge
  - cloudflare/workerd
- Common Use Cases:
  - Middleware / Glue Code
  - Webhooks
  - Lightweight APIs