

Go 程式設計課 07

Go 與雲原生



大綱

- ❖ 雲原生的定義
 - GCP
 - 12 Factory
- ❖ Container & serverless
- ❖ monitor & deployment
- ❖ network concept
- ❖ Message Queue
- ❖ GCP introduce
- ❖ 實作 MQ / Cluster / Docker



雲原生的定義

GCP (Google Cloud Platform)

雲端原生架構與單體式應用程式的不同在於，後者必須以單一單元的形式建構、測試及部署。而雲端原生架構會將元件分解為鬆耦合的服務，以便管理複雜度、提高速度和靈活性及軟體推送的規模。

簡單來說：盡可能的使用雲端的服務來完成各個系統模塊的建構與運維，不自行完成

目的	服務分類	技術	
❖ 彈性	❖ IaaS	❖ Micro Service	K8S
❖ 分散	❖ PaaS	❖ Containers and orchestration	
❖ 可靠	❖ FaaS	❖ DevOps	
❖ 簡單	❖ SaaS	❖ CI/CD	

ref: <https://cloud.google.com/learn/what-is-cloud-native?hl=zh-tw>



12 Factory (1)

- I. **Codebase** One codebase tracked in revision control, many deploys *git*
 - II. **Dependencies** Explicitly declare and isolate dependencies *go mod*
 - III. **Config** Store config in the environment *自動 set env → env → backend*.
 - IV. **Backing services** Treat backing services as attached resources *所有外部服務都是附加資源*
 - V. **Build, release, run** Strictly separate build and run stages *不可以直接改 prod 程式*
 - VI. **Processes** Execute the app as one or more stateless processes *process 不一定一直在執行
exec and run*
- ref: <https://12factor.net/>



12 Factory (2)

VII. Port binding Export services via port binding

VIII. Concurrency Scale out via the process model

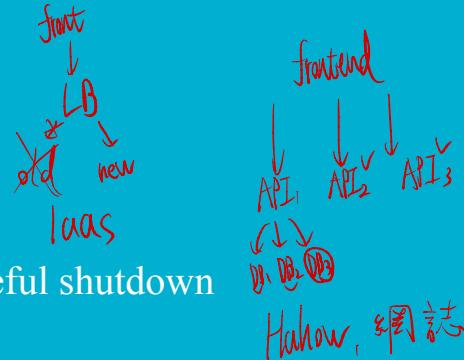
IX. Disposability Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity Keep development, staging, and production as similar as possible

XI. Logs Treat logs as event streams

XII. Admin processes Run admin/management tasks as one-off processes

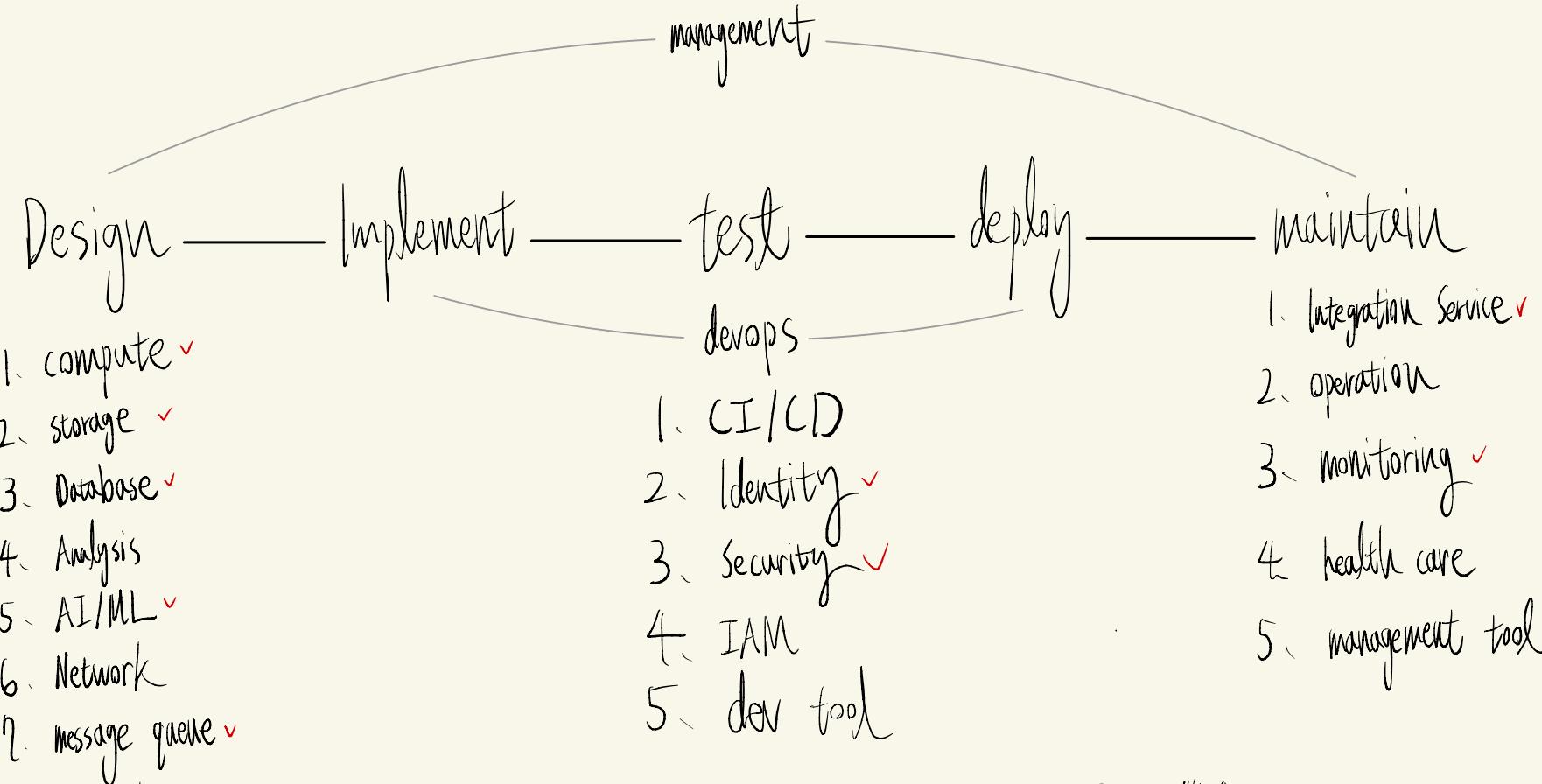
ref: <https://12factor.net/>



要有好方法管理 admin script



我的定義 在整個開發生命周期，盡可能採用雲端元件輔助



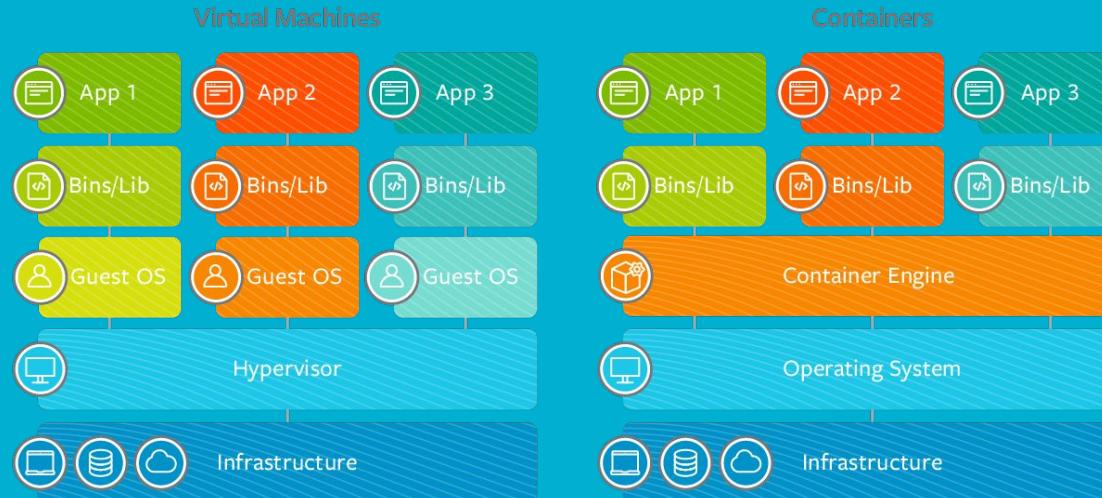
sample: 錄音重點整理



Container & serverless

Container & Serverless

- ❖ 現代應用部署的方式
- ❖ 差異在管理的層次差異



Container

- Container 是一種打包應用程式及其依賴項的方式
- 將應用程式及其運行環境封裝在一個獨立的 "容器" 中
- 容器可以在任何支持的環境中運行,而不受環境差異的影響
- 基於作業系統的權限隔離 API 實作, 可視為 OS 級別的抽象
- 常見的容器技術: Docker

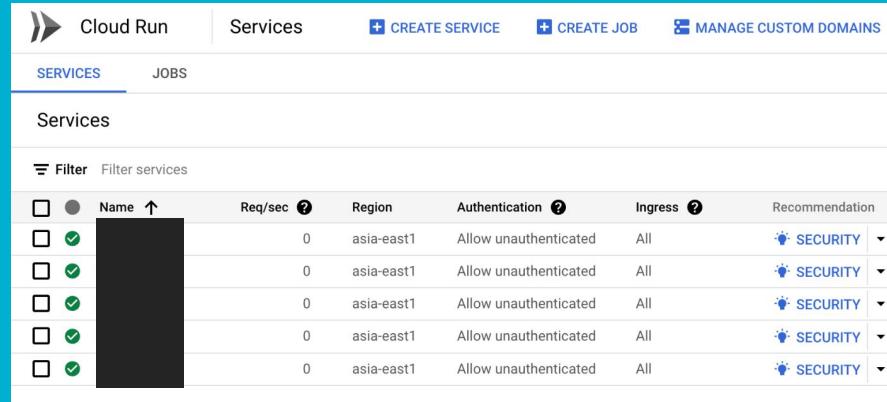
sample code:

<https://github.com/leon123858/tsmc-meal-order/tree/main/user>



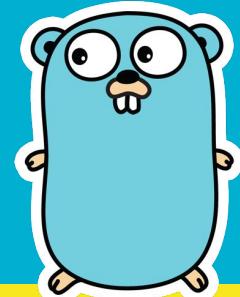
Container 的優勢

- 一致的運行環境: 避免 "在我的機器上可以運行" 的問題
- 輕量級: 容器通常比虛擬機更小, 啟動更快
- 可移植性: 容器可以在不同的系統和雲平台上運行
- 可擴展性: 容器可以輕鬆地複製和擴展以處理更多負載



The screenshot shows the Cloud Run Services interface. At the top, there are navigation links: 'Cloud Run' (with a play icon), 'Services', '+ CREATE SERVICE', '+ CREATE JOB', and 'MANAGE CUSTOM DOMAINS'. Below this is a tab bar with 'SERVICES' (which is selected) and 'JOBS'. A search bar labeled 'Filter Filter services' is present. The main area displays a table of services:

	Name	Req/sec	Region	Authentication	Ingress	Recommendation
<input type="checkbox"/>	[REDACTED]	0	asia-east1	Allow unauthenticated	All	SECURITY ▾
<input type="checkbox"/>	[REDACTED]	0	asia-east1	Allow unauthenticated	All	SECURITY ▾
<input type="checkbox"/>	[REDACTED]	0	asia-east1	Allow unauthenticated	All	SECURITY ▾
<input type="checkbox"/>	[REDACTED]	0	asia-east1	Allow unauthenticated	All	SECURITY ▾
<input type="checkbox"/>	[REDACTED]	0	asia-east1	Allow unauthenticated	All	SECURITY ▾



Serverless

- Serverless 是一種無需管理服務器的應用開發和部署模式
- 開發人員專注於編寫應用程式代碼,而不用擔心底層基礎設施
- 應用程式被拆分為獨立的函數,根據需求自動擴展和運行
- 常見的 Serverless 平台: AWS Lambda, Google Cloud Functions
- note: 在 GCP 的定義中只用 container 不管 VM 也算 Serverless

sample:

<https://github.com/leon123858/tsmc-meal-order/tree/main/functions/storage>



Serverless 的優勢

- 無需管理服務器: 開發人員可以專注於應用程式邏輯
- 自動擴展: 根據請求量自動調整資源, 滿足應用程式需求
- 按使用付費: 只為實際消耗的資源付費, 而不是一直運行的服務器
- 快速部署: 無需配置基礎設施, 更快地將應用程式推向市場

Cloud Functions Functions + CREATE FUNCTION ⚡ REFRESH

Filter functions

	Environment	Name ↑	Last deployed	Region	Recommendation	Trigger	Runtime	Memory allocated
<input type="checkbox"/>	2nd gen	[REDACTED]	Sep 1, 2023, 10:37:50 PM	asia-east1		HTTP	Node.js 18	256 MiB
<input type="checkbox"/>	2nd gen	[REDACTED]	Sep 13, 2023, 1:17:16 AM	asia-east1		HTTP	Node.js 18	256 MiB



 Backend 台灣 (Backend Tw)
Saren Arterius · 56分鐘 · ⓘ

用Vercel一時爽...
用來做hosting的話貴絕全球喔
1mil個每個1s 1gb ram的function request = \$50usd起跳，500k用戶的cara短短5天已經90k usd
這個費用了
隨便估算，200k dau，每個user每日要call 100個api，每個response用1-5秒io (db開始慢) = 3秒，每個fn set 1gb ram = 60mil gbsec = 16666gb hrs = 每日燒\$6666 usd + 流量費用，所以cara 5天燒了90k usd，參考一下
人家cloudflare用cpu time計費，vercel連db/io時間都算進去~還未算100gb 40usd的流量費用

用Vercel一時爽...

用來做hosting的話貴絕全球喔

1mil個每個1s 1gb ram的function request = \$50usd起跳，500k用戶的cara短短5天已經90k usd
這個費用了

隨便估算，200k dau，每個user每日要call 100個api，每個response用1-5秒io (db開始慢) = 3秒，每個fn set 1gb ram = 60mil gbsec = 16666gb hrs = 每日燒\$6666 usd + 流量費用，所以cara 5天燒了90k usd，參考一下

人家cloudflare用cpu time計費，vercel連db/io時間都算進去~還未算100gb 40usd的流量費用

$$200K \times 100 \times 3 = 6 \times 10^7 S = 10^6 min = 1.6 \times 10^4 hr \Rightarrow 2880 USD/dig \\ \times \$0.18 (\text{僅IO})$$

 Jingna Zhang @ cara.a... [Follow](#) ⋮

So freaking speechless right now. Seen many [@vercel](#) functions stories but first time experiencing such discrepancy vs request logs like, this is cannot be real??

This is your daily notification that your team [REDACTED] has used **24166%** of your monthly included Serverless Function Execution amount which has added **\$96,280** to your bill thus far. You'll continue to be charged **\$40 per 100 GB Hrs.**

Managed Infrastructure

Compute

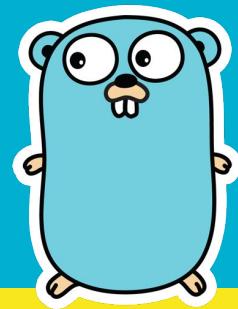
Dynamic, scalable functions to power your framework of choice.

Function Invocations <small>New</small>	100,000 invocations /month included	1M invocations /month included then starting at \$0.60 per million
Function Duration <small>Reduced</small>	100 GB-hour /month included	1,000 GB-hour /month included then \$0.18 per GB-hour
Edge Function Executions <small>Lightweight compute, ideal for long-duration invocations.</small>	500,000 execution units /month included	1M execution units /month included then starting at \$2.00 per million

1. quality
2. management
- 100% 大 ~
- 看8/2 中 10k dau
- ✓ 小 500 dau

Container vs. Serverless

- Container: 打包應用程式及其依賴項,在任何地方運行
- Serverless: 無需管理服務器,專注於編寫應用程式代碼
- 選擇取決於應用程式的需求和架構
- 兩種技術可以結合使用,發揮各自的優勢

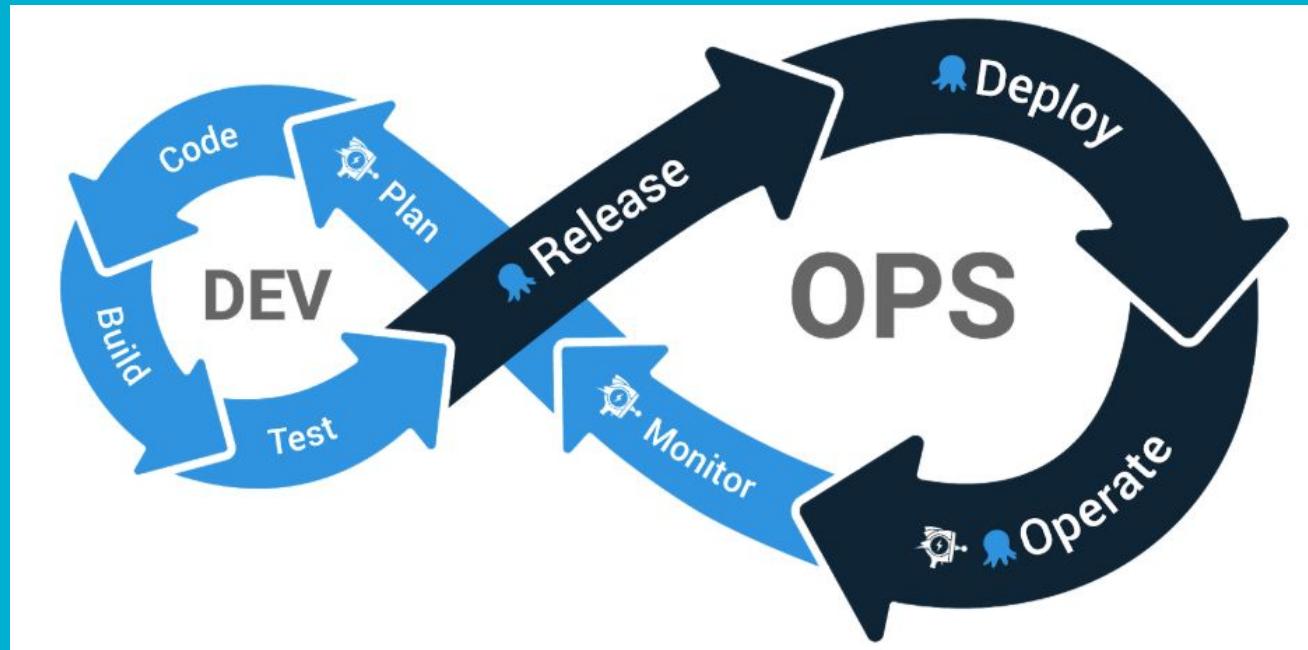


monitor & deployment

Monitor & Deployment

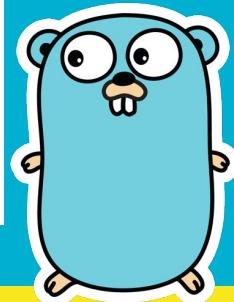
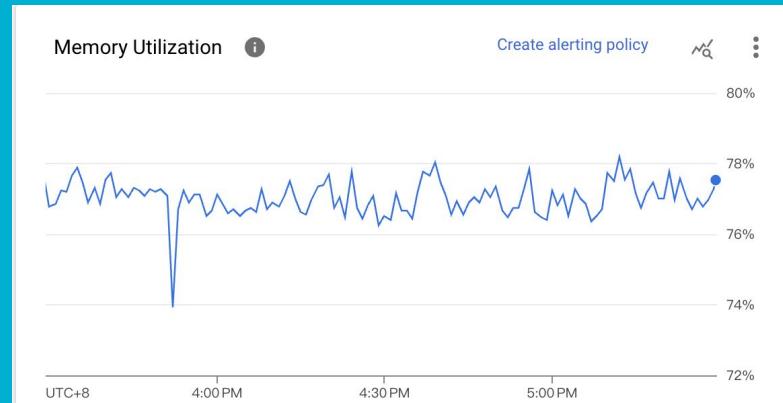
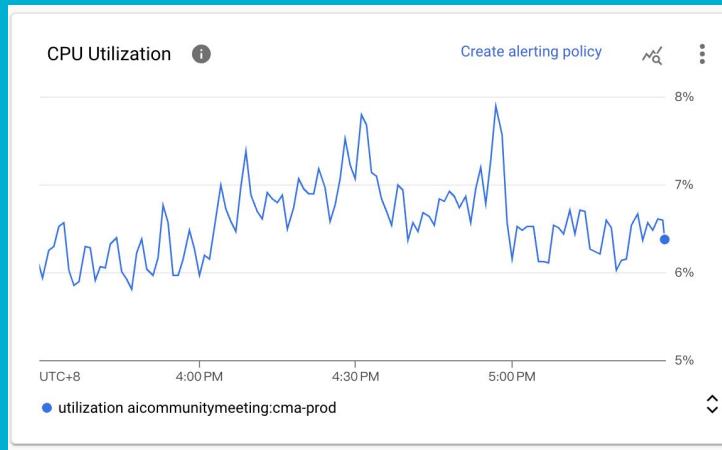
ref:<https://raygun.com/blog/the-art-of-shipping-software/>

軟體運維的兩個關鍵環節



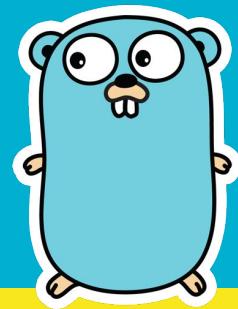
Monitor

- 應用程式監控是持續跟蹤應用程式性能和行為的過程
- 收集關鍵指標,如響應時間、錯誤率、資源使用情況等
- 常見的監控工具: Grafana



為什麼需要應用程式監控？

- 及時發現和解決性能問題,確保應用程式的可用性, ex: CPU 太少, 緊急加機器
- 瞭解用戶體驗和行為,優化應用程式設計, ex: 某時段 CPU 太多, 定期關機器
- 容量規劃和資源優化,控制成本, ex: HPC
- 滿足合規性和安全性要求,減少風險 ex: 及時發現系統異常, 資料外洩
- 確保部分**非功能性需求**



非功能性需求

by Wiki

以下是一些非功能性需求的例子：

- 無障礙
- 審計和控制
- 可用性（參考[服務級別協定](#)）
- 備份
- 目前容量及預估容量
- 認證
- 相容性
- 組態管理
- 部署
- 檔案
- 災難恢復
- 效率（特定負載下消耗的資源）
- 有效性（工作量及其效能表現間的關係）
- 情感因素
- 環境保護
- 履約保證
- 弱點
- 可延伸性（Extensibility，增加機能）
- 故障管理
- 故障容許度（容錯性）
- 法律性或[授權許可](#)問題或避免專利侵權
- 互操作性
- 可維護性
- 可修改性（Modifiability）
- 網路拓撲
- 開放原始碼
- 可操作性
- 效能
- 系統平台相容性
- 隱私權
- 軟體可移植性
- 品質（例如已發現的故障、已交付的的故障、故障排除效力）
- 復原或可復原性（例如平均修復時間MTTR）
- 可靠度（例如平均故障間隔MTBF）
- 報表
- 網路彈性
- 資源限制（處理器、速度、金錢、硬碟容量、網路頻寬等）
- 反應時間
- 強健性
- 可伸縮性（Scalability，水平或垂直的）
- 保安
- 軟體、工具、標準等的相容
- 穩定性
- 可支援性
- 軟體可測試性
- 易用性



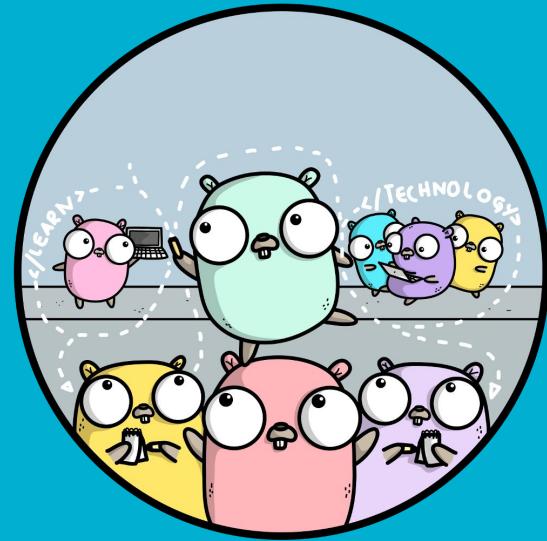
小試身手

DB: 練習設置 cache cluster

為了提高非功能性需求中的可用性和穩定性，
我們會透過硬體的冗余(平行)來達到目標，
最常見的就是後端 server 的平行和資料庫的冗余。

全部試一遍太累了，我們就來練習看看 cache 的冗余吧！

<https://github.com/leon123858/go-tutorial/tree/main/short-url/doc/docker-compose-redis-cluster>



監控的注意事項

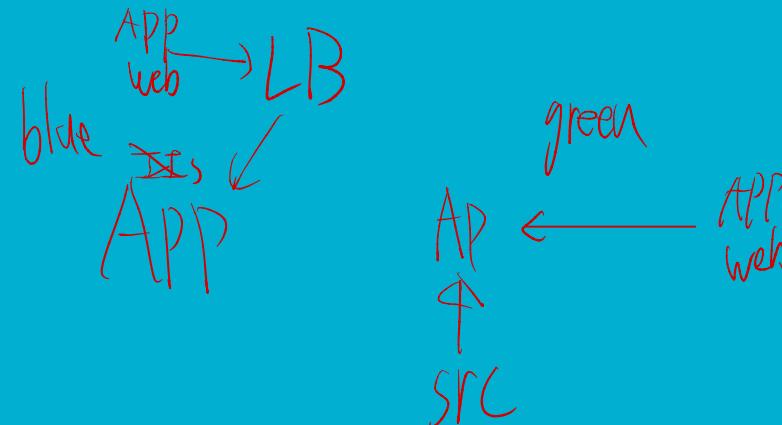
- 確定關鍵指標和閾值,設置合適的警報規則
- 使用儀表板和可視化工具,實時跟蹤應用程式性能
- 收集和分析日誌數據,快速定位問題
- 定期審查和調整監控策略,適應不斷變化的需求
- 詳見 SRE 工程

AWS SRE: <https://aws.amazon.com/tw/what-is/sre/>



Deployment

- 應用程式部署是將新版本的應用程式推送到生產環境的過程
- 涉及代碼發布、配置更新、數據庫遷移等多個步驟
- 常見的部署策略: 滾動更新、藍綠部署、金絲雀發布



好的發布得到的效果

- 快速交付新功能和錯誤修復,滿足用戶需求
- 最小化部署風險和停機時間,確保業務連續性
- 支持頻繁發布和快速迭代,提高開發效率
- 實現自動化和標準化,減少人為錯誤
- 詳見 CI/CD

red hat: <https://www.redhat.com/zh/topics/devops/what-is-ci-cd>



發布的實踐細節

- 採用基礎設施即代碼(Infrastructure as Code),實現部署自動化
- 使用版本控制系統管理配置和部署腳本
- 實施持續集成和持續部署(CI/CD)管道,自動化測試和發布流程
- 監控部署過程和結果,快速回滾失敗的部署

sample:

<https://github.com/leon123858/tsmc-meal-order/tree/main/infra/core>

sample:

<https://ithelp.ithome.com.tw/articles/10319189>



CI 實作

①

[https://github.com/leon123858/go-tutorial/
blob/main/.github/workflows/db-ci.yml](https://github.com/leon123858/go-tutorial/blob/main/.github/workflows/db-ci.yml)

②

[https://github.com/leon123858/go-
tutorial/blob/main/.github/workflows/
short-url-ci.yml](https://github.com/leon123858/go-tutorial/blob/main/.github/workflows/short-url-ci.yml)

network concept

如何從 APP 完成一個網路操作

- 觸發使用者事件
- 調用手機網卡的 DNS 伺服器 IP 地址
- 向 DNS 伺服器依照 APP 內的域名查詢
- DNS 協助取得域名對應的 IP 地址
- 向真實 IP 地址進行 http 協定下的呼叫
- 後端伺服器依據 http 協定解讀呼叫, 完成指令後回傳 http response
- 手機 APP 解讀 http response 後顯示反饋給使用者



DNS 與 IP

- DNS(Domain Name System)和 IP(Internet Protocol)是互聯網通信的兩個基礎要素
- IP 地址用於標識網路上的設備,而 DNS 將易記的域名轉換為 IP 地址
- DNS 使得人們可以使用域名訪問網站,而不必記住 IP 地址



網卡與 APP

- 網路通訊可以分成三大層
 - 應用層
 - 系統層
 - 硬體層
- APP 位在應用層, 作業系統管理系統層, 網卡的韌體管理硬體層
- 簡單來說, 微觀的傳輸流程是
 - APP 把指令打包成高階協議例如 http 後做成字串傳入 OS
 - OS 把字串基於低階協議例如 TCP/IP 拆解成多個封包傳入網卡
 - 網卡把多個封包套上硬體協議例如 WIFI 的 802.1 傳給 router
 - router 透過 IP 找到對應的目標 server
 - 把數據反向傳入目標 server 的應用層



Protocal and Port

- 協議: 定義了網路上設備之間如何通信的規則和格式
- 埠: 用於識別主機上的特定程序或服務的數字標識
- 協議和埠共同確保數據在網路上的正確傳輸和接收

0到1023是眾所周知的埠號 ,分配給
常見的服務和協議

- HTTP: 80
- HTTPS: 443
- FTP: 21
- SMTP: 25
- DNS: 53



Message Queue

MQ 消息隊列

消息隊列是一種在分佈式系統中實現不同組件之間通信的機制

生產者將消息發送到隊列,消費者從隊列中獲取消息

消息隊列提供了異步通信、解耦和可擴展性的優勢

工作原理:

- 生產者將消息發送到消息隊列,包括有效載荷和元數據
- 消息隊列將消息存儲在內部緩衝區中,按照先進先出或其他規則排序
- 消費者從消息隊列中獲取消息,處理消息,並在完成後確認消息
- 消息隊列通過確認機制確保消息的可靠傳遞和處理



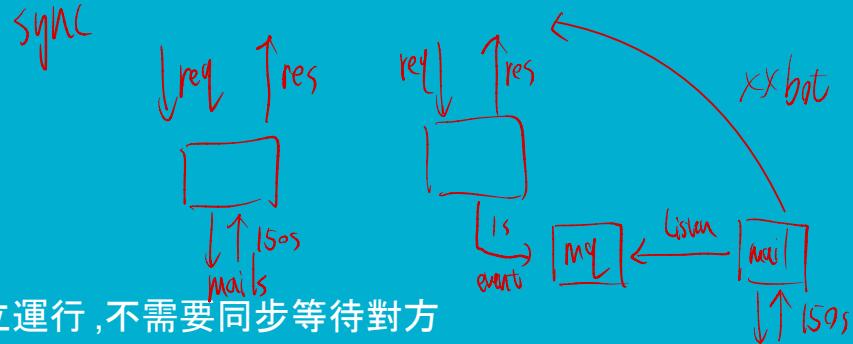
MQ 優勢與場景

優勢:

- 異步通信: 生產者和消費者可以獨立運行,不需要同步等待對方
- 解耦: 生產者和消費者只需要知道消息隊列,不需要了解對方的細節
- 可擴展性: 可以增加多個生產者和消費者,提高系統的吞吐量和處理能力
- 可靠性: 消息隊列提供持久化存儲和確認機制,確保消息不會丟失
- 緩衝: 消息隊列可以緩衝生產者和消費者之間的速度差異,平滑流量波動

應用場景:

- 異步任務處理: 將耗時的任務提交到消息隊列,由後台服務異步處理
- 事件驅動架構: 通過消息隊列實現系統組件之間的事件傳遞和處理
- 分佈式系統集成: 使用消息隊列實現不同系統之間的數據同步和通信
- 流量削峰: 在高併發場景下,使用消息隊列緩衝請求,避免系統過載
- 日誌收集: 將分佈式系統的日誌發送到消息隊列,集中收集和分析

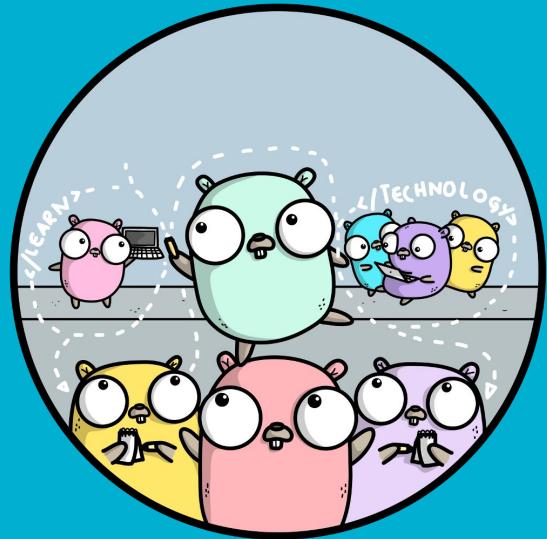


小試身手

MQ: Rabbit MQ 練習

Link

<https://github.com/leon123858/go-tutorial/tree/main/mq>



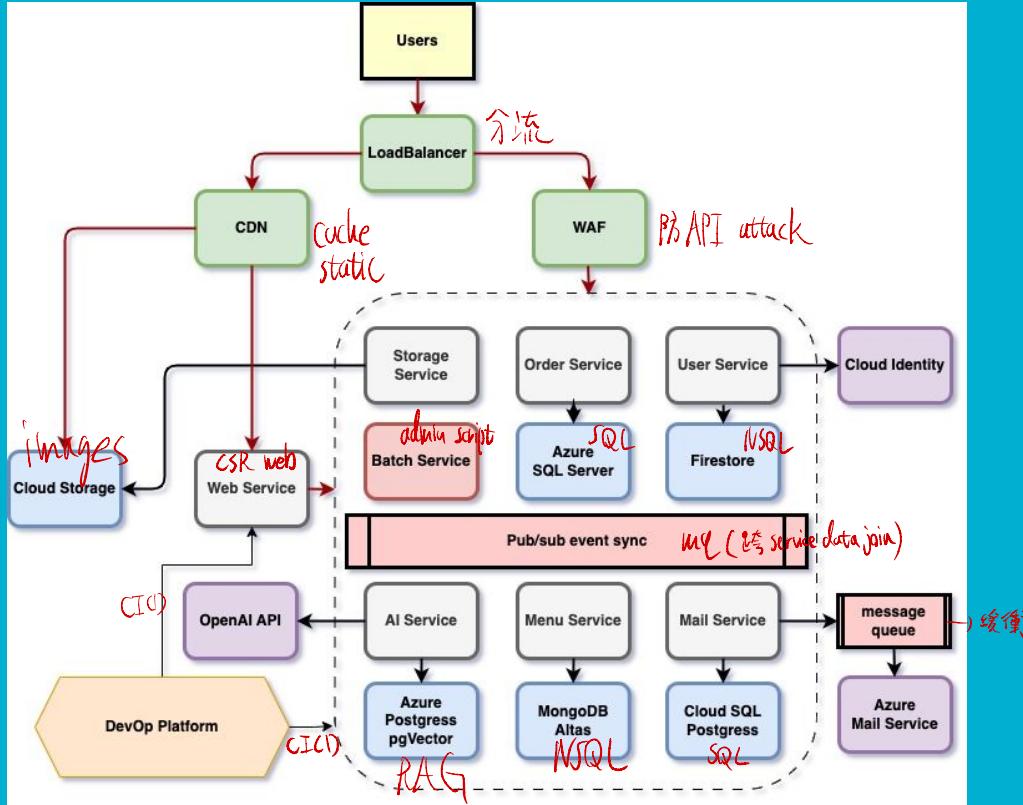
GCP introduce

SPEC

- 一個簡易的訂餐平台
- 消費者可以用帳號密碼登入
- 要產生菜單列表
- 店家可以登入編輯自己的菜單
- 店家可以在介面查看訂單與編輯訂單
- 消費者可以基於菜單點餐
- 可以用 AI 推薦消費者新餐點
- 點餐成功或餐點送達時用 email 通知
- 每個月定期匯總帳單 email 給用戶

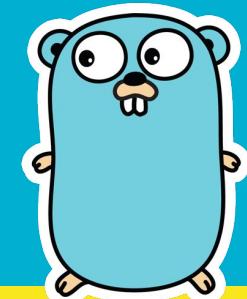
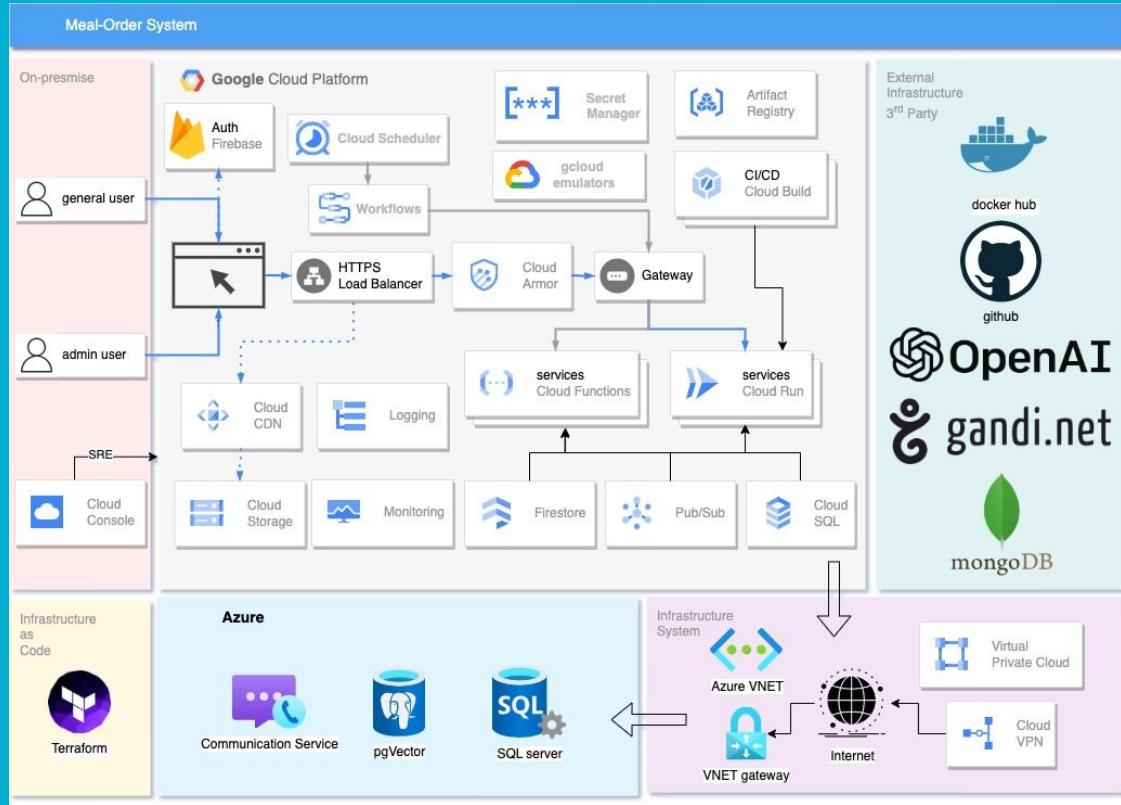


A demo GCP Cloud Native Project

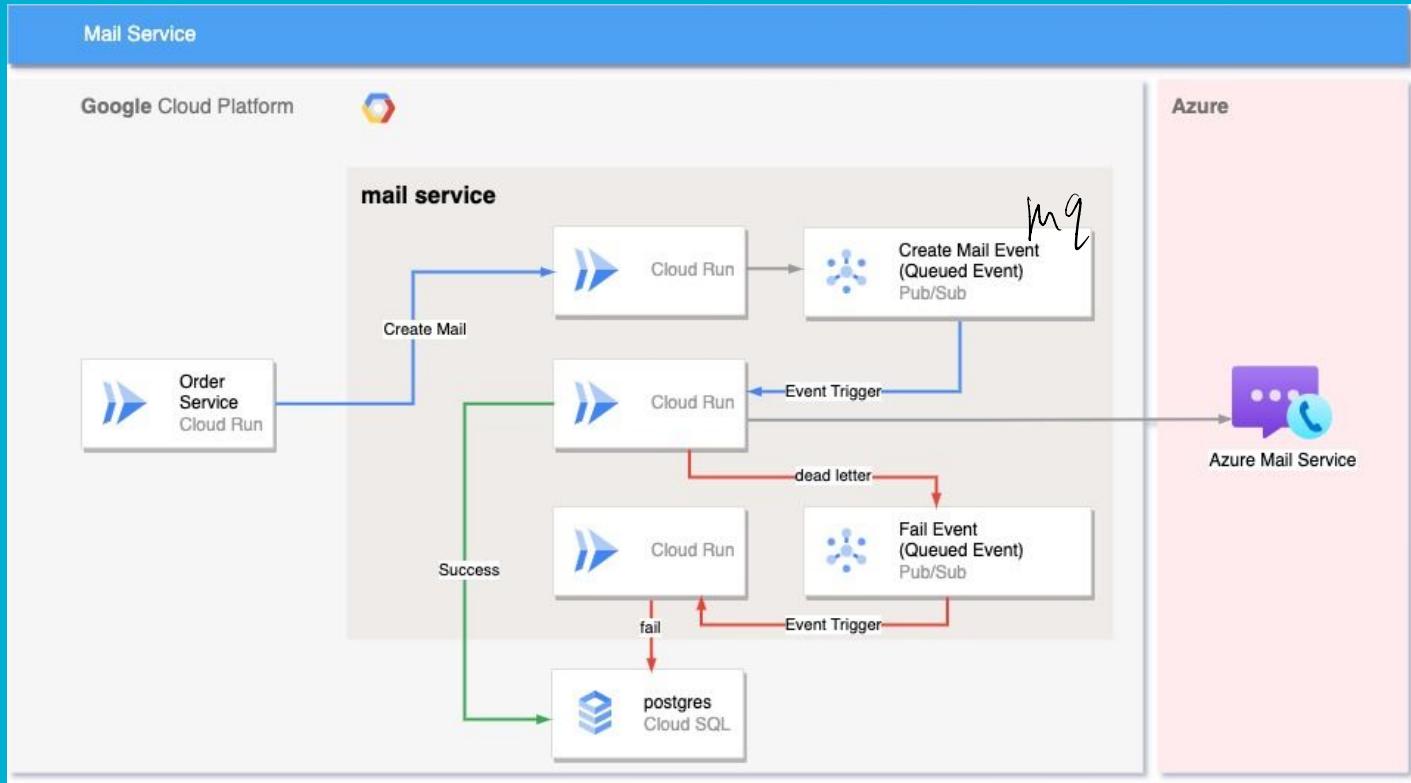


Cloud Arch

- LB
- Scheduler
- Workflow
- Cloud Armor
- Cloud KMS
- Cloud Build
- Registry
- CDN
- Cloud Logging
- Pub/Sub



Cloud Native Mail Service



Lab

設計模式介紹與實戰



MQ: 後端追加 MQ 機制

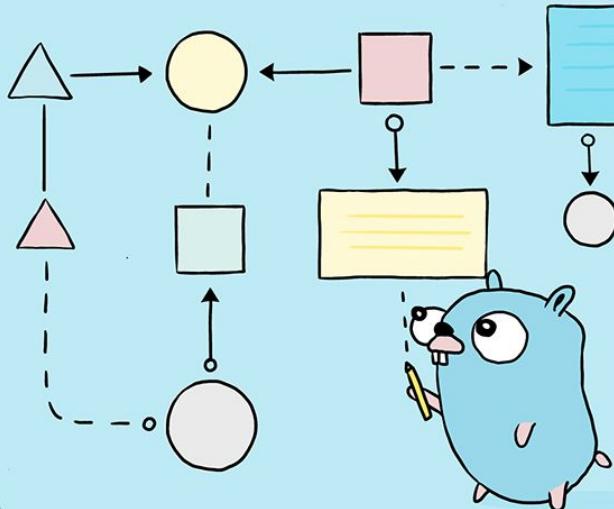
Link

<https://github.com/leon123858/go-tutorial/tree/main/short-url/pkg/mq>

第回 06 設計！



Q&A 時間



作業: 後端容器化

```
FROM python:3.8
ARG APP_HOME=/app
ENV APP_HOME=$APP_HOME
WORKDIR $APP_HOME
COPY requirements.txt .
RUN pip install -r requirements.txt .
COPY . .
CMD python app.py
```

Link: <https://github.com/leon123858/go-tutorial/blob/main/short-url/Dockerfile>

