微服務上線後的救贖

Roberson Liou



歡迎參加我們的每週四固定聚會

關於我

- Backend / Cloud
- Microsoft MVP(2020-2024)
- twMVC 核心成員
- DevOps Taiwan 志工
- BLOG - 工程良田的小球場



大綱

- 微服務簡介
- 上線前的準備
- 上線後的救贖
- 案例分享及總結

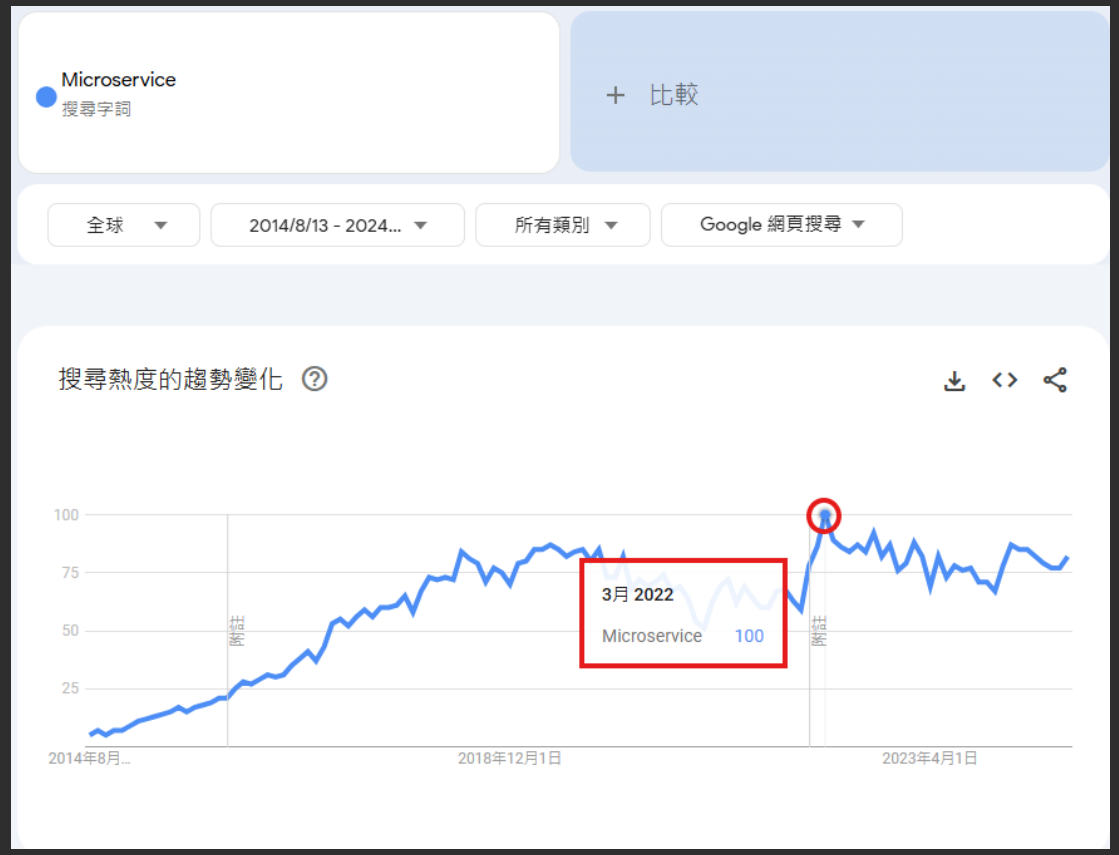
微服務簡介

2024了，你還在微服務的路上嗎?

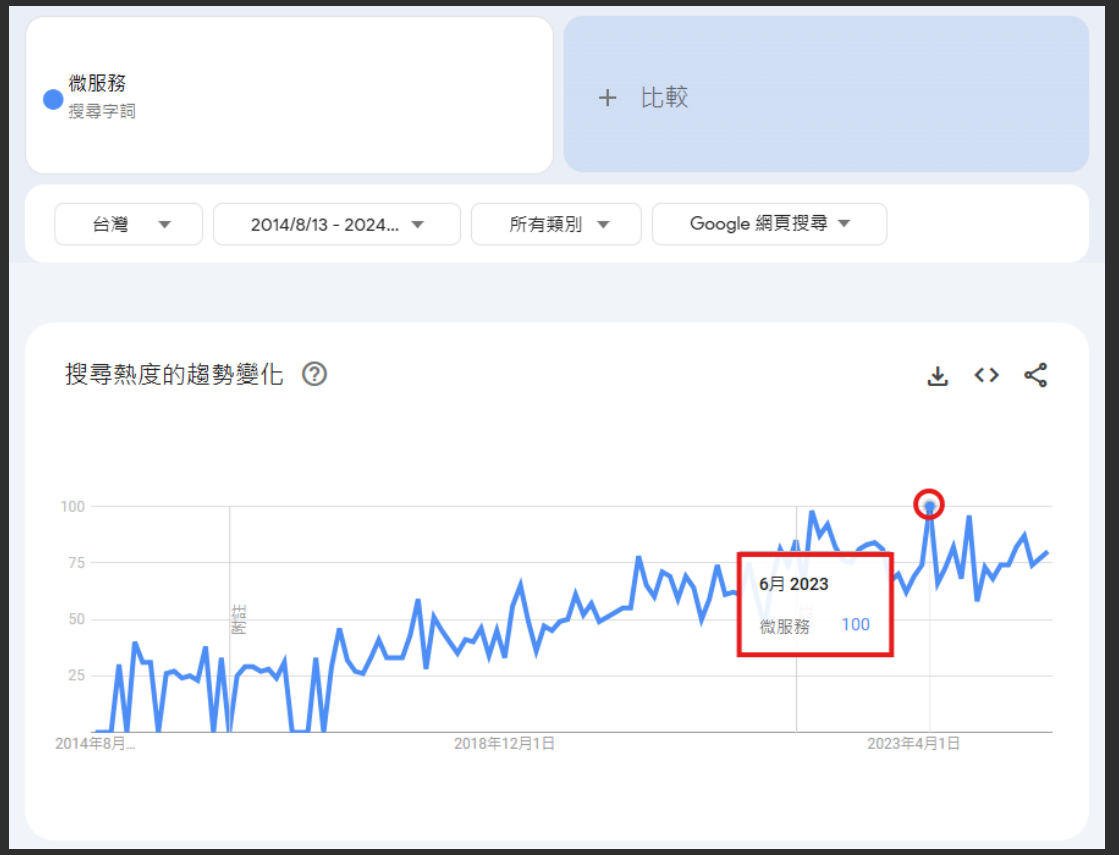


微服務還是很熱衷的被討論嗎?

全球 2014-2024



台灣 2014-2024



當我們談到微服務...

- 微服務就是要 DDD
- 微服務就是要 K8s
- 微服務就是要 CQRS
- 微服務就是要 BFF
- 微服務就是要 gRPC
- (以下省略 1000 字...)



到底什麼才是微服務?

維基百科

微服務 [編輯]

21 種語言

條目 討論 汉 漢 臺灣正體 閱讀 編輯 檢視歷史 工具

微服務（英語：Microservices）是一種軟體架構風格，它是以專注於單一責任與功能的小型功能區塊 (Small Building Blocks) 為基礎，利用模組化的方式組合出複雜的大型應用程式，各功能區塊使用與語言無關 (Language Independent），而且複雜的服務背後是使用簡單 URI 來開放介面，任何服務，任何細粒都能被開放（exposed）。這個設計在 HP 的實驗室被實現，具有改變複雜軟體系統的強大力量。

Microservice – Martin Flower

- “The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms”
- 不專業翻譯
  - 將單體拆為一組小型服務
  - 每個服務跑在獨立的 process
  - 服務間的通訊要能夠輕量化

將上述的微服務定義再延伸一下

- 將單體拆為一組小型服務 => DDD
- 每個服務跑在獨立的 process => Container
- 服務間的通訊要能夠輕量化 => K8s
- 與其說 OOO 就是微服務，我覺得應該說 OOO 適合微服務

導入微服務前，請先想想...

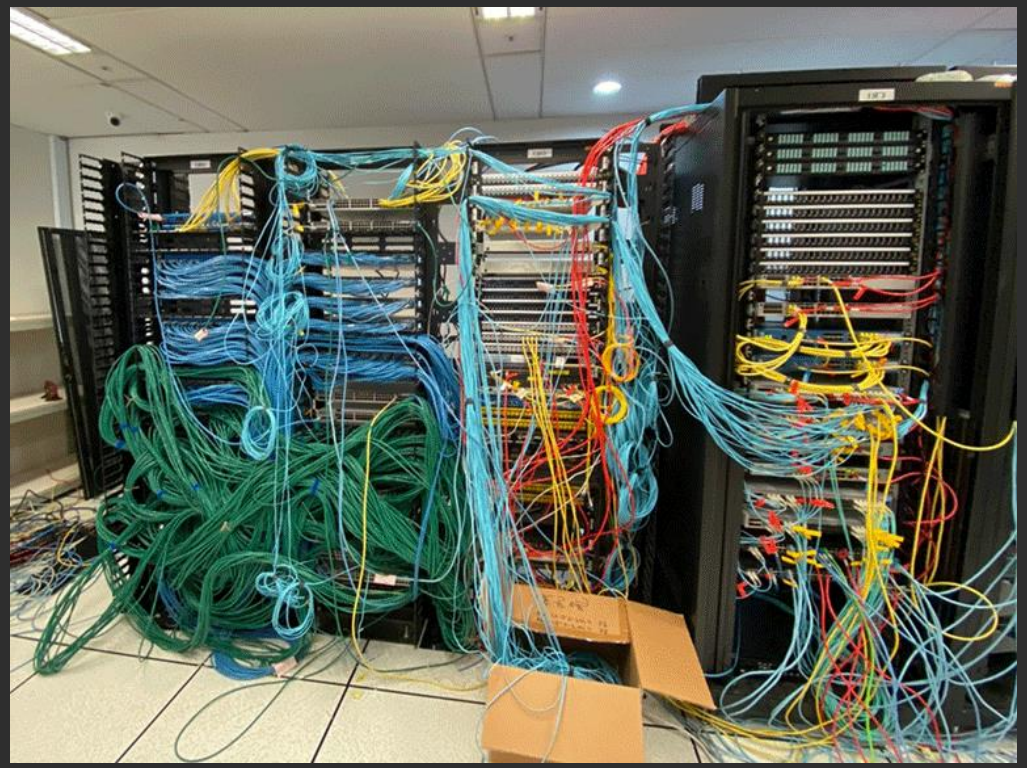
- 它的確帶來許多好處
  - 可擴展性
  - 高容錯性
  - 可獨立部署
  - 技術多樣性
- 但它同時也帶來這些隱憂
  - 分散式架構
  - 最終一致性
  - 維運複雜度
  - 有形跟無形成本

大家都在談微服務

嘴巴講的微服務



實際產出的微服務



tw MVC
https://mvc.tw

常見的微服務症狀

- 單一請求內部服務通訊次數過多
- 服務顆粒度切得太大或太小
- 分散式資料交易常常不一致
- 單一服務需占用大量資源(CPU, Memory)
- 服務間的交易無法被追蹤

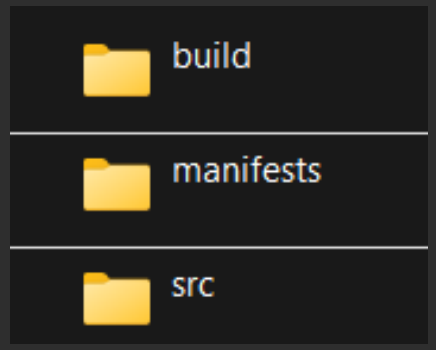
上線前的準備

Repository 管理

- Poly repo vs. Mono repo
- Repo 內放什麼
  - Source code / Pipeline / K8s Manifest
- 統一資料夾命名結構

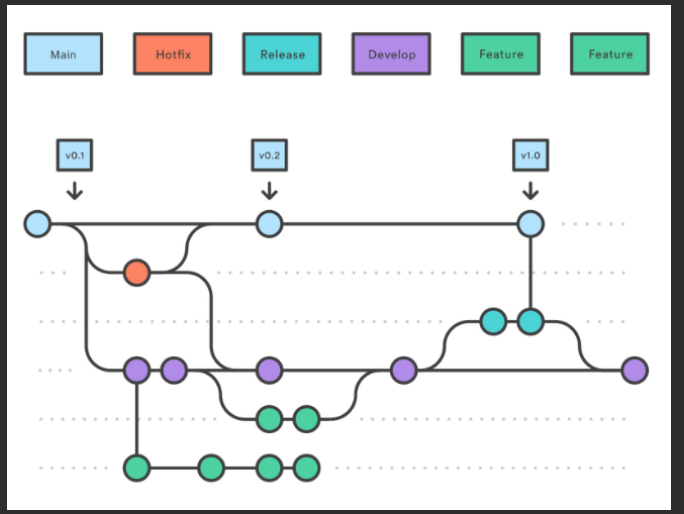






分支策略的選型

- 選定適合團隊使用的分支策略
  - TBD, Git flow, GitHub flow, Release branch…
- Mono repo 需要額外考量
  - 不同服務需能夠同時開發
  - 服務必須能夠單獨 Hotfix



持續交付的能力

- PR Build: Code Quality check
- CI Pipeline: Code Build
- Artifact 管理: Versioning
- CD Pipeline
  - 測試環境全自動，正式環境半自動
- 誰能執行 Pipeline 也是影響交付速度的關鍵

快取設計

- 不要只做 Local Cache，在 K8s 下效益不高
- 可採用分散式快取 Redis
  - 制定 Cache Key 命名規則: 建議包含環境、服務名稱
- 一定要有過期時間
- 快取遺失不能導致服務故障

其它可能需要關注的開發議題

- 服務邊界劃分
- 事件驅動設計：透過非同步模式降低服務間的依賴
- 分散式交易一致性：補償式交易(SAGA)、多階段提交
- API 的冪等性(Idempotency)

上線後的救贖

Pipeline 設計的隱形成本

- 你的 Pipeline 經常需要修改嗎?
  - 改 1 次要多久?
  - 如果每條 pipeline 都要改呢?
- 兩大步驟
  - 將專案先依照應用程式框架類型進行分類
  - 採用範本化來降低 Pipeline 的維運成本

K8s Manifest 也需要範本化

- 服務數量多，配置重複，修改成本大
- 參考工具
  - Helm / Kustomize / Jsonnet…
- 撰寫文件來提升團隊一致性
  - 如目錄結構，命名規範…

Pipeline Agent 數量要充足

- 要能夠持續地頻繁部署，Pipeline Agent 數量必須充足
  - 盡可能讓 Pipeline Agent 也可以自動擴展
  - 如果不行，至少要能在短時間內自動建置新的 Agent
- 還須考慮 Pipeline Server 可同時執行的 Job 數量
- 測試跟正式環境的不要共用 Pipeline Agent
  - 要確保正式環境的執行優先權比較高

微服務間的版本依賴

- 單一請求的背後可能牽涉多個服務所組成
- 需要能夠識別每個微服務的版本，才能快速定義問題
- 為每個服務定義版本，並透過工具整合
- 常見的幾個作法
  - 將服務版本加到分布式追蹤的屬性中
  - 在 Header 中回傳版本號，由上游服務進行整合

微服務的組態管理

- K8s 內建: 環境變數、Config map, Secret(b64)
- 導入 Vault 工具: Azure Key Vault, Hashicorp Vault…
- 組態更新做法
  - 重啟服務
  - Hot reload

第一針的連鎖效應

- 服務部署後，第一次請求總是特別緩慢
- 一個服務慢，所有呼叫它的服務都跟著慢
- 如果你使用的是 K8s，可以實作探針來解決
  - StartupProbe, LivenessProbe, ReadinessProbe
- 若非 K8s， AP 本身要實作 warm up

關於自動擴展

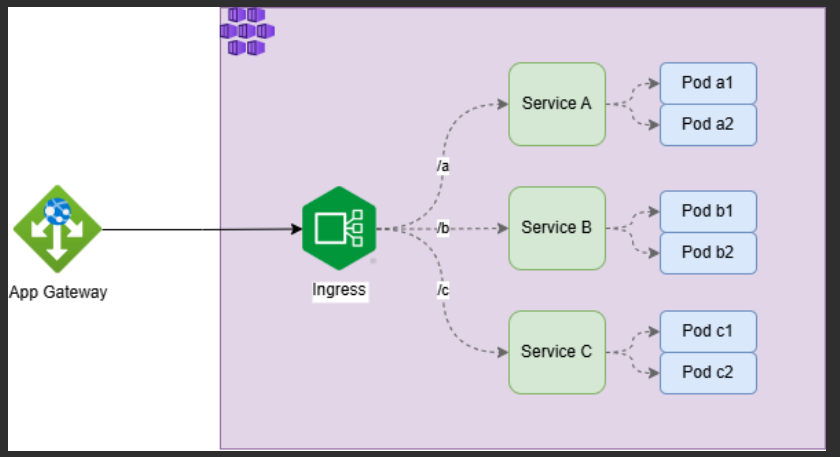
- 設定 HPA 就是萬靈丹嗎?
- 除了 Pod 以外，Node 也要能 Scale
- 多一台 Node 通常需要 5 分鐘
- 建議抓 20% 的 Buffer 做緩衝

Graceful termination

- 當 Pod 被要求刪除後，要等多久才會被移除
  - K8s 預設值為 30 秒
- 什麼時候會收到刪除訊號(SIGTERM)?
  - Pod 被刪除 / 上版 / Scale Down / Cluster Maintenance
- 若現有請求無法在時限內完成，Pod 會強制被終止
  - 可能會導致資料交易的不完整
  - 如果 API 很慢已經是常態，就建議調高 terminationGracePeriodSeconds
- 程式端也可以監聽 SIGTERM，停止接收新的請求
  - API 要實作 CancellationToken

域名及憑證管理

- App Gateway 還是 Ingress?
- 依成本量級區分
  - 經濟艙: 全部給 Ingress 管理
  - 商務艙: 有 DR 需求，域名跟憑證掛 App Gateway，搭配 TLS Offload
  - 頭等艙: 有跨區域 DR 需求，域名指向 L4 LB，搭配多個 App Gateway 管理



微服務的監控範圍

- Application 要具可觀測性
- Kubernetes 監控
  - Cluster 層級: Node CPU / Memory / Network IO, Unscheduled pods…
  - Pod 層級: CPU, Memory, Pod Crash(OOM, Startup error, config…)
- Event Queue: 訊息數量、訊息處理失敗…
- Database: CPU, Memory, DataIO, Log IO…

多租戶架構下的考量

- 資料庫數量增加
  - 監控範圍增加
  - 維護範圍增加
- 資料庫建置是否有自動化?
- 資料合規性議題

漏洞修補

- K8s Node Vulnerabilities
  - OS Level
- Container Vulnerabilities
  - OS Level / AP Level(Nuget)
  - 無法直接在 Cotainer 中進行 Patch
  - 需調整 Dockerfile 並重新 Deploy

團隊之間的交付模式

- 在微服務的架構下，產品團隊很難識別需要部署的服務範疇
- 找一套適合你的組織的交付流程
  - 產品團隊提出發布請求
  - 開發團隊提供交付細節
  - 必要時，開發團隊之間要先決定部署順序
  - 每次的交付都要能夠被回溯檢視

談微服務知識傳遞

- 微服務的去耦合帶來了靈活性，但也讓提高了未來追朔的複雜度
- 當維護這群微服務的不再是同班人馬時...
  - 如何了解服務之間的上下游如何溝通?
  - 如何理解分散式交易一致性是如何實作的?
- 是不需要文件、還是不喜歡寫文件?

案例分享及總結

案例一：微服務開發者

- 三人後端開發團隊
- 服務數量一年內從 3 成長到 > 10
- 採用 SDK，但無專人維護
- 沒有專門的維運團隊

案例二：微服務維運者

- 三人維運團隊
- 維護範圍過大
  - 5 個環境
  - 微服務 > 50 個、資料庫 > 30 個
- 過度設計：GraphQL, gRPC, BFF…

適合微服務的組織模式

- 小規模團隊(5-7人)
- 一組開發團隊不要維護超過 3 個微服務
- SDK 跟共用服務要有專責的團隊維護
- 專門的維運團隊、或跨職能團隊

微服務給我的啟發

- 服務間的故障隔離性提升，但分佈式系統的複雜度增加
- 服務間通訊的複雜度提升，使得可觀測性變得更加重要
- 開發單一功能的速度提升，但團隊間溝通速度反而變慢
- 微服務更需要能夠頻繁、持續且快速交付的能力
  - 真正落實 DevOps 文化的團隊

所以，你要導入微服務嗎?

- 技術實踐是 0 跟 1 的問題，但團隊的技術選型並不只是 0 跟 1
- 你可能還需要考慮...
  - 團隊的技術水平
  - 整體的成本及時間限制
  - 未來的擴展性及維護性
  - 導入後對業務是否有實質利益

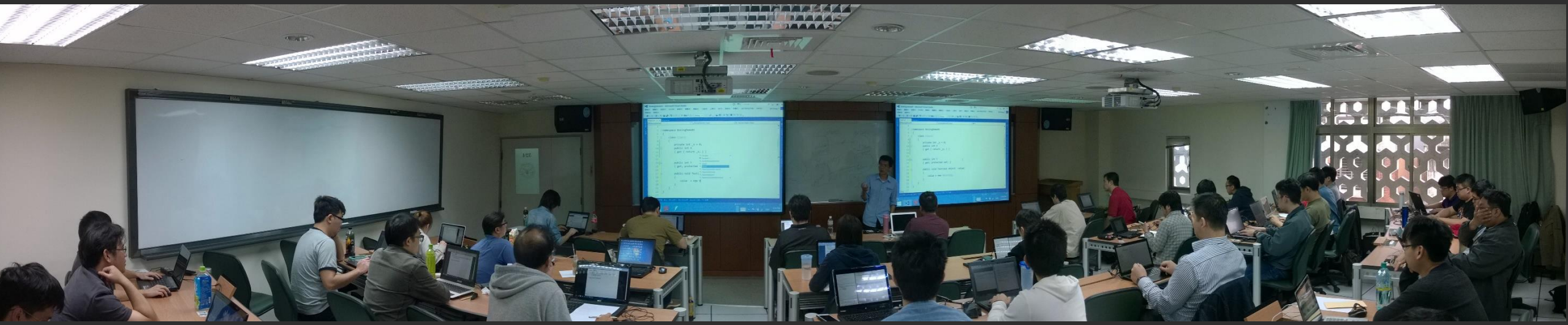
Thank you

Blog 是記錄知識的最佳平台



https://dotblogs.com.tw

SkillTree 為了確保內容與實務不會脫節，我們都是聘請企業顧問等級並且目前依然在職場的業界講師，我們不把時間浪費在述說歷史與沿革，我們並不是教您考取證照，而是教您如何上場殺敵，拳拳到肉的內容才是您花錢想要聽到的，而這也剛好是我們擅長的。



https://skilltree.my

天瓏資訊圖書

