Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

DevOps: An Introduction

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

National Chengchi University

Meet

WEALTHGRID

A successful, mid-size financial company

They have competition, but it’s not much to worry about.
The players are all familiar and competing within a stable and steady market.

WealthGrid is good company with a good market. Until one day…

同業轉型

‘We want to be a tech company with a banking license’ – Ralph Hamers

08 August 2017 1 min read Listen

8 August 2017

CEO Ralph Hamers has told The Banker that he wants ING to be seen as a tech company with a banking license.

ING亦名荷蘭國際集團，是一個國際金融服務私營企業，成立於1991年，由荷蘭最大的保險公司Nationale-Nederlanden，與荷蘭的第三大銀行NMB PostBank Group合併而成。

異業切入

FINANCE

Amazon could become the third-biggest US bank if it wants to: Bain study

PUBLISHED TUE, MAR 6 2018•4:23 PM EST | UPDATED WED, MAR 7 2018•9:23 AM EST

Thomas Franck
@TOMWFRANCK

SHARE

KEY POINTS

- Bain writes that Amazon's banking services could grow to more than 70 million U.S. consumer relationships over roughly five years, rivaling Wells Fargo.
- Amazon could evade more than $250,000,000 in credit card interchange fees every year if it finds a bank willing to partner.
- The Bain report finds one-quarter of those using voice assistants like Amazon's Alexa would consider using them for everyday banking.

新創競爭

STARLING BANK

full production bank was built in a year

2014 Founded by Anne Boden

June 2014 Kick-off with Regulators

September 2015 Technical prototypes

2 engineers

January 2016 Raise $70m – start build

July 2016 Banking licence & first account in production AWS account

October 2016 Mastercard debit cards

November 2016 Alpha testing mobile app

December 2016 Direct debits live

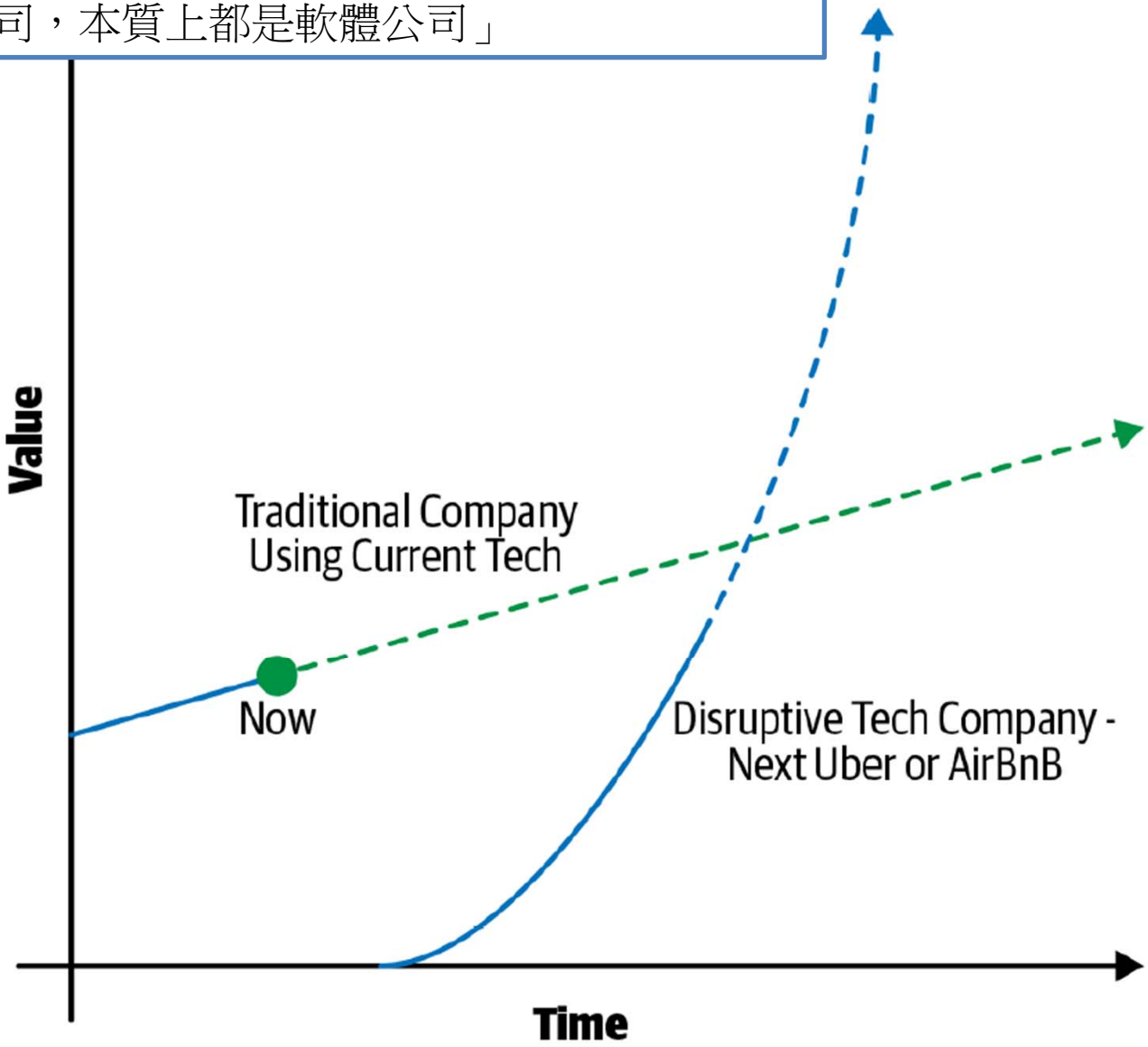
January 2017 Faster payments live

February 2017 Launched beta testing program

20 engineers

May 2017 Public App Store Launch

IT技術近10年來正急遽影響日常生動的各個層面
Marc Andreessen(2011): 軟體正在吃下全世界,未來「未來大部份公司，本質上都是軟體公司」



穩定中暗藏的危機

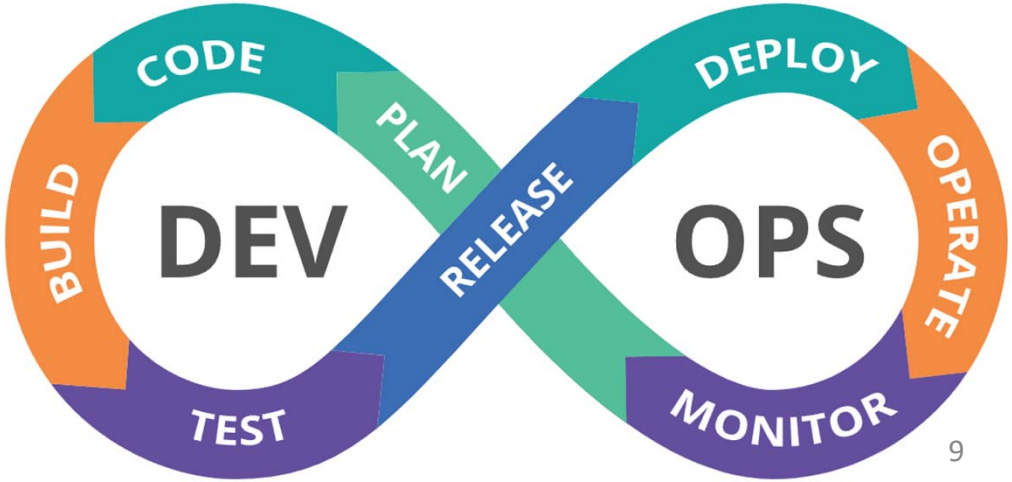
- Disrupt or Be Disrupted (Sharma, 2017)
  - Avg. life expectancy for fortune 500 company
    - 1960: 75 years
    - 2010: 15 years or less
- IT組織追求目標 (Kim et al., 2016)
  - 回應快速變動的競爭格局
  - 提供穩定、可靠、安全的服務
- IT組織惡性循環三部曲 (Kim et al., 2014)
  - 有限時間內，要處理問題、承諾新功能
  - 累積技術債 (受限於技術能力或時程壓力的變通處置)
  - 組織中每個人變得更忙、需要更多溝通、協調和批准

2013-2016 由Kim and Humble進行的實證研究顯示DevOps 為打破此惡性循環的重要實踐

https://www.devops-research.com/research.html

什麼是DevOps?

- **定義** (CMU 軟體工程學院)
  - A set of practices, which (實踐方法)
  - reduces the time between change to production (快速回應)
  - while ensuring high quality （保持穩定、可靠、安全）
- **關注開發者Dev與維運者Ops順暢的協作模式**
  - 開發者: 程式開發、品保、測試人員
  - 維運者: SP、OP

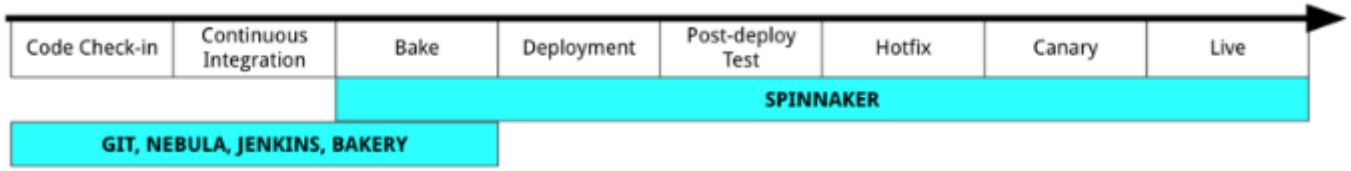


DevOps是long-term evolution guideline
不是立竿見影的特效藥

Move fast without breaking things

Google State of DevOps Reports https://www.devops-research.com/research.html

- Metrics for software delivery and operational performance
  - Deployment frequency (DevOps: High)
  - Lead time : 開發人員commit程式到上線的時間 (DevOps: Short)
  - MTTR (Mean time to recover)
  - Change failure percentage: 由於改動導致的系統失效
- Examples
  - Amazon deploys changes into production every 11.6 seconds
  - Netflix lead time = 16 minutes

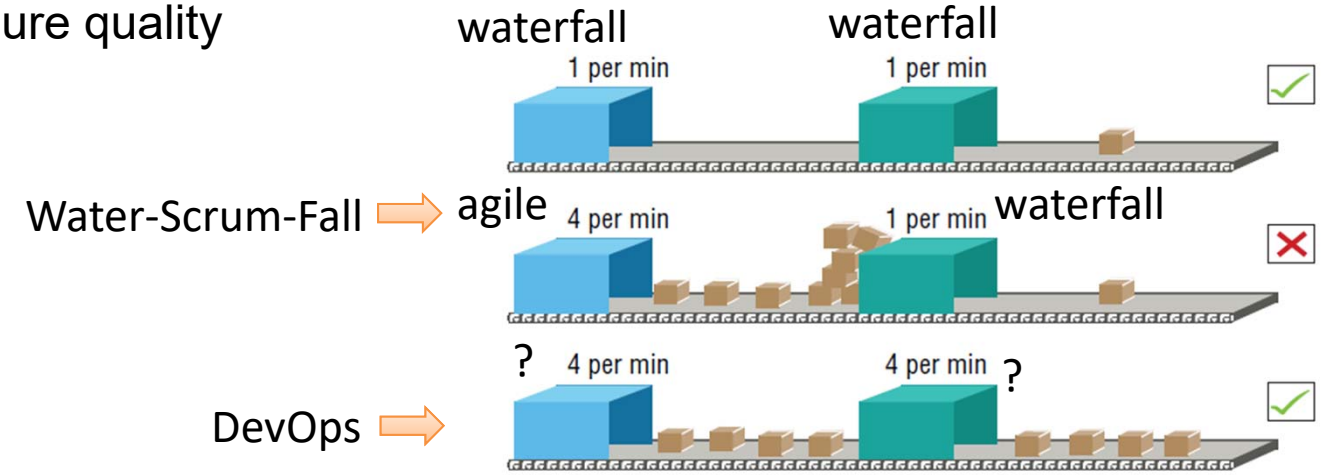


https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15

Dev vs. Ops: 為何會不順暢?

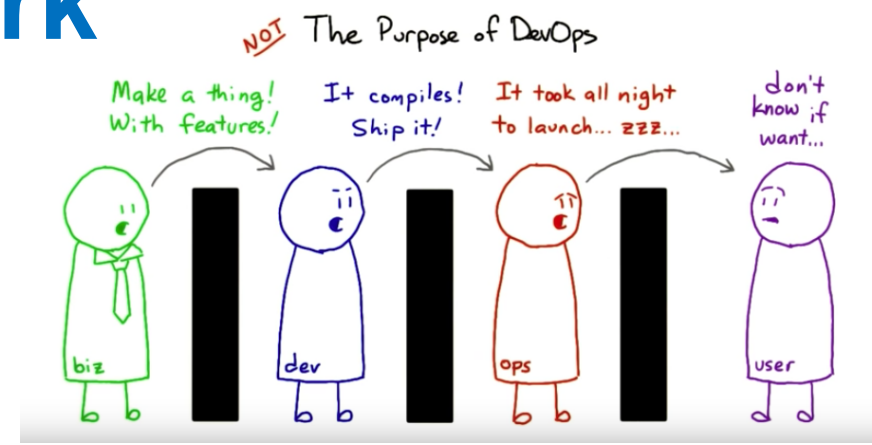
- Dev vs. Ops (Sharma, 2017)
  - Dev's task
    - Create innovation
    - Deliver ASAP
  - Ops's task
    - Provide stable, fast, and secure service
    - Ensure quality

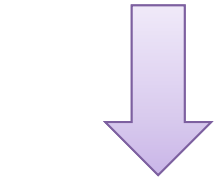
思考: 為何看似tradeoff的Fast和Quality可以並存?
工業革命教我們的事
1. 什麼東西做產品比起人來說更快?
2. 什麼東西做產品做一次和做1000次幾乎一樣品質?



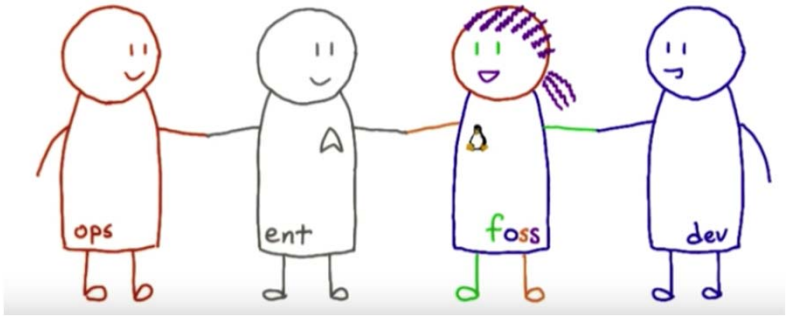
CAMS Framework

- DevOps 要領
  - Culture
  - Automation
  - Measurement
  - Sharing
- 分類
  - 技術: A, M
  - 組織: C, S





The Purpose of DevOps



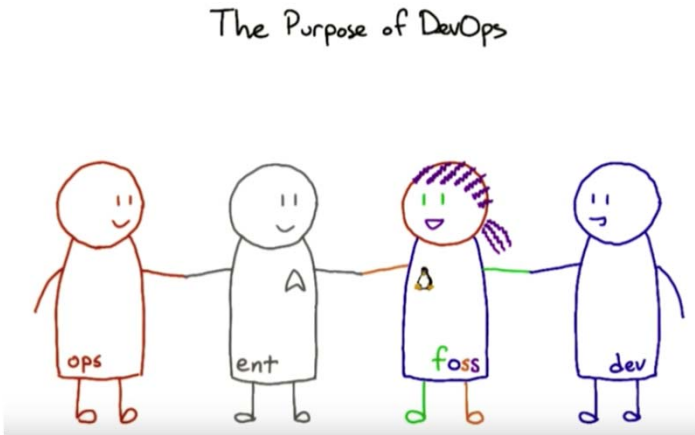
Foss = Free and open-source software

Share

- Dev
  - 向Ops學習
    - 了解平台 (部署環境)的限制
    - 以容器思維封裝程式: 讓開發/部署環境盡量保持一致 (Dev/prod parity)
    - 了解Monitor技巧 and Metrics
  - 與Ops有更好的溝通與合作
    - 與Ops合作建構CI/CD Pipeline

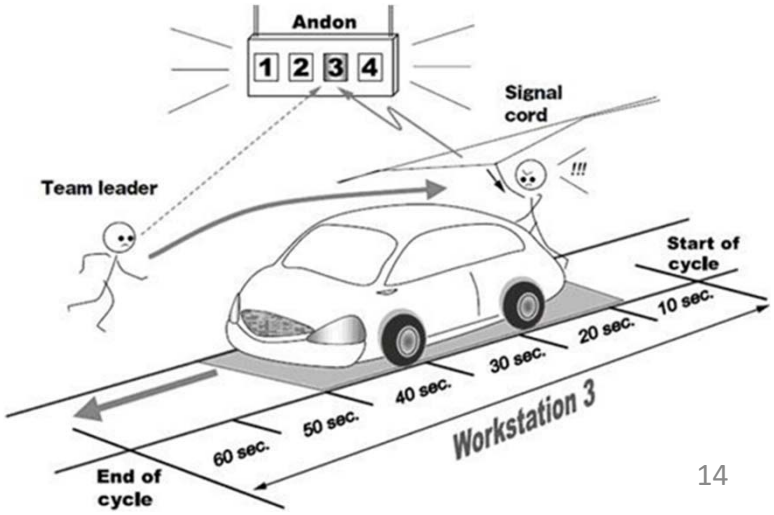
CI/CD = Continuous Integration / Continuous Delivery and Deployment

- Ops
  - 向Dev學習
    - 了解規格、了解code → 知道程式如何影響平台
  - 維運管理方式變革
    - 容器化: 以一致、獨立的方式管理、部署軟體
    - 自動化: 所有工作應編寫為程式
    - 版本化: 環境變動納入版控管理
  - 與Dev有更好的溝通與合作
    - 監控CI/CD Pipeline，即時反映錯誤



Culture

- 目的
  - 早日發現問題，在問題尚未擴大惡化就予以解決 (Shift-Left)
  - 視錯誤、問題為組織學習機會，而非究責根據
- 範例
  - Toyota Andon Cord 安燈索
    - 在生產線上，每個工作站上方都有一根繩索
    - (受訓後的)工人在特定狀況(零件出現瑕疵、損壞、所花時間過多)拉繩
    - 示警後全團隊必須立刻「共同」解決問題，如果一定期間內無法解決，則中止生產線，直到問題解決為止
  - 觀察點: 立刻解決、一起解決
    - 不是有空時再解決



回饋

- 有關Logo
  - 日常有「重視過程而非只重視結果的意義」→這個意義不錯
- 目前學校造成問題的根源與反向標題
  - 包容
    - 目前: 只任用「同溫層」、學門歧視
  - 共創
    - 目前: 資源分配、重大決策未基於一定程度共識
  - 真誠
    - 目前: 許多溝通充滿交易、權謀、算計
- 如何因應新時代的挑戰
  - 強韌性→Adaptive(適應新挑戰)→Generative (演進型)
  - 當前挑戰: 後疫情、後五年五百億(極端學術產出)
- 有關SDG

Culture and Share: 組織變革

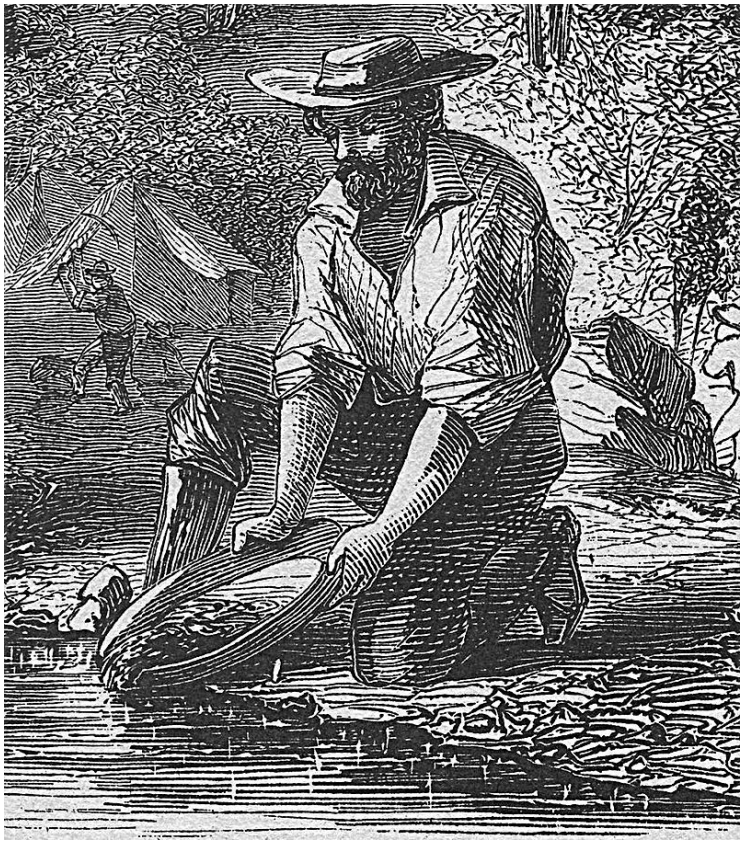
	病態型 Pathological	官僚型 Bureaucratic	演進型 Generative
資訊	Information is hidden	Information may be ignored	Information is actively sought
提出問題者	Messengers are "shot"	Messengers are tolerated	Messengers are trained
責任規屬	逃避 Responsibilities are shirked	切割 Responsibilities are compartmented	Responsibilities are shared
跨團隊合作	Bridging between teams is discouraged	Bridging between teams is allowed but discouraged	Bridging between teams is rewarded
發生失敗時	Failure is covered up	Organizatoin is just and merciful	Failure causes inquiry
新的想法	New ideas are crushed	New ideas create problems	New ideas are weclomed

Figure 8: *The Westrum organizational typology model: how organizations process information (Source: Ron Westrum, "A typology of organisation culture," BMJ Quality & Safety 13, no. 2 (2004), doi:10.1136/qshc.2003.009522.)*


A NEW AND MAGNIFICENT CLIPPER FOR SAN FRANCISCO.
MERCHANTS' EXPRESS LINE OF CLIPPER SHIPS!
Loading none but First-Class Vessels and Regularly Dispatching the greatest number.
THE SPLENDID NEW OUT-AND-OUT CLIPPER SHIP
CALIFORNIA
HENRY BARBER, Commander, AT PIER 13 EAST RIVER.
This elegant Clipper Ship was built expressly for this trade by Samuel Hall, Esq., of East Boston, the builder of the celebrated Clippers "SURPRISE," "GAMECOCK," "JOHN GILPIN," and others. She will fully equal them in speed! Unusually prompt dispatch and a very quick trip may be relied upon. Engagements should be completed at once.
Agents in San Francisco, Messrs. DE WITT KITTLE & CO.
RANDOLPH M. COOLEY, 88 Wall Street, Tontine Building.
NESBITT & CO., PRINTERS.

1848 加州淘金熱

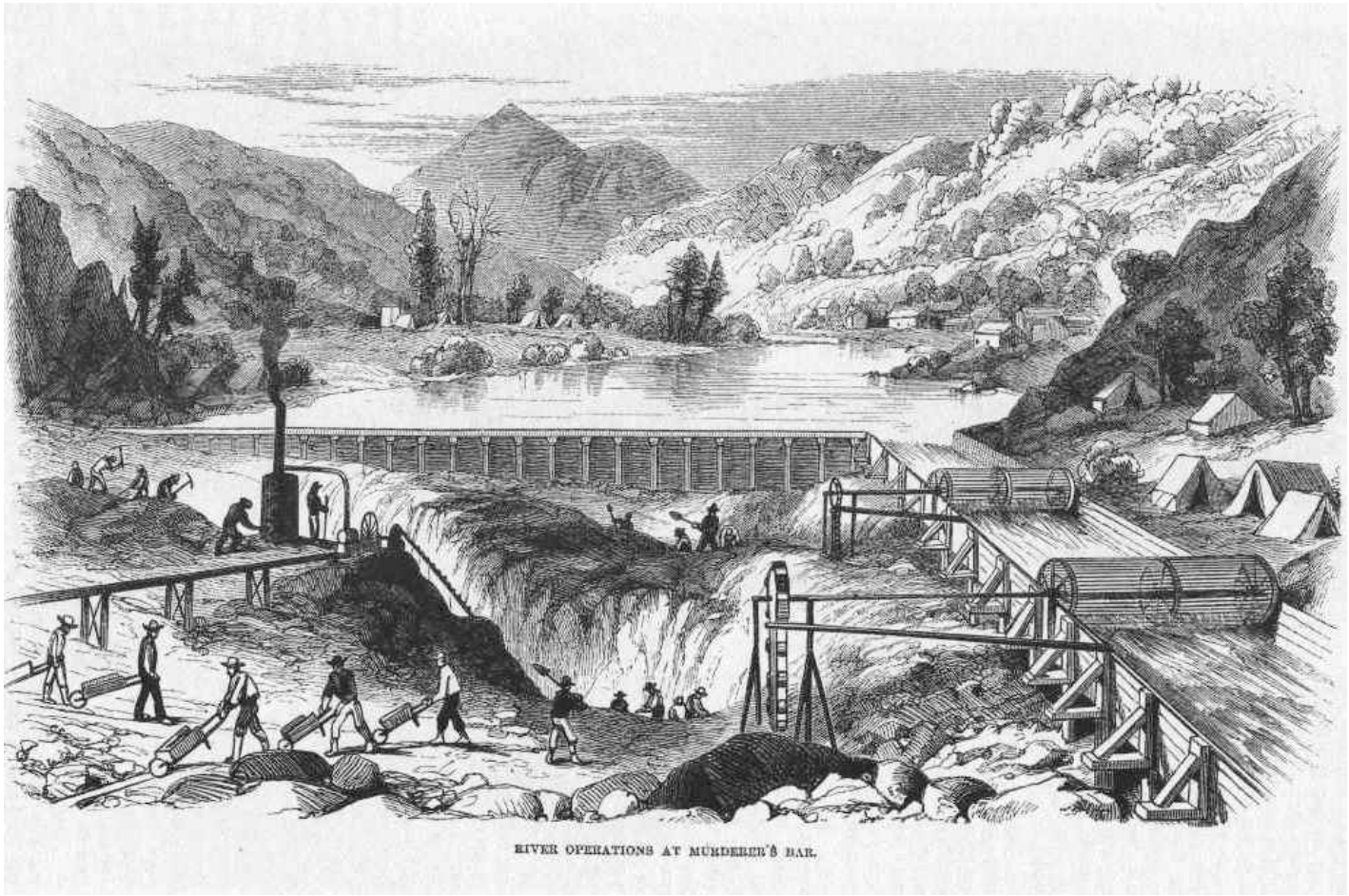
- At first, loose gold and gold nuggets could be picked up off the ground.
- Later, gold was recovered from streams and riverbeds using simple techniques, such as panning.



California, 1848

- More sophisticated methods were developed and later adopted elsewhere.

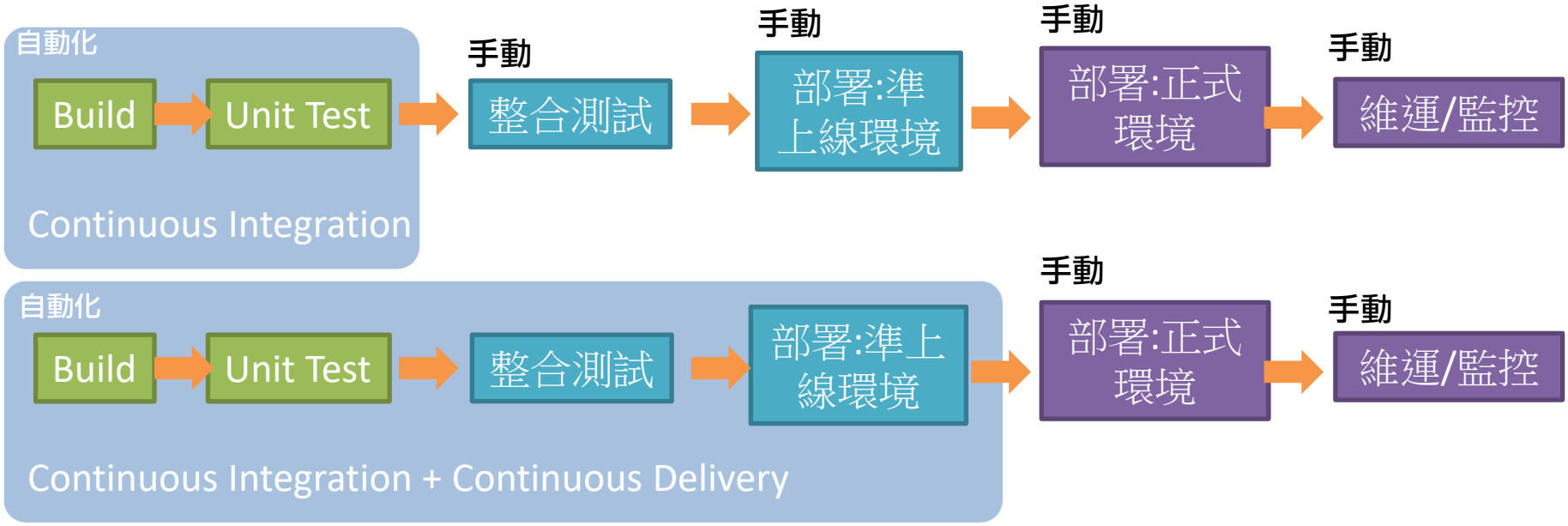
- At its peak, technological advances reached a point where significant tooling and the automated pipeline was required, and collaboration among individuals become important.



RIVER OPERATIONS AT MURDERER'S BAR.

Automation and Measurement: the CI/CD Pipeline

CI = Continuous Integration 持續整合
CD = Continuous Delivery / Continuous Deployment 持續交付/持續部署



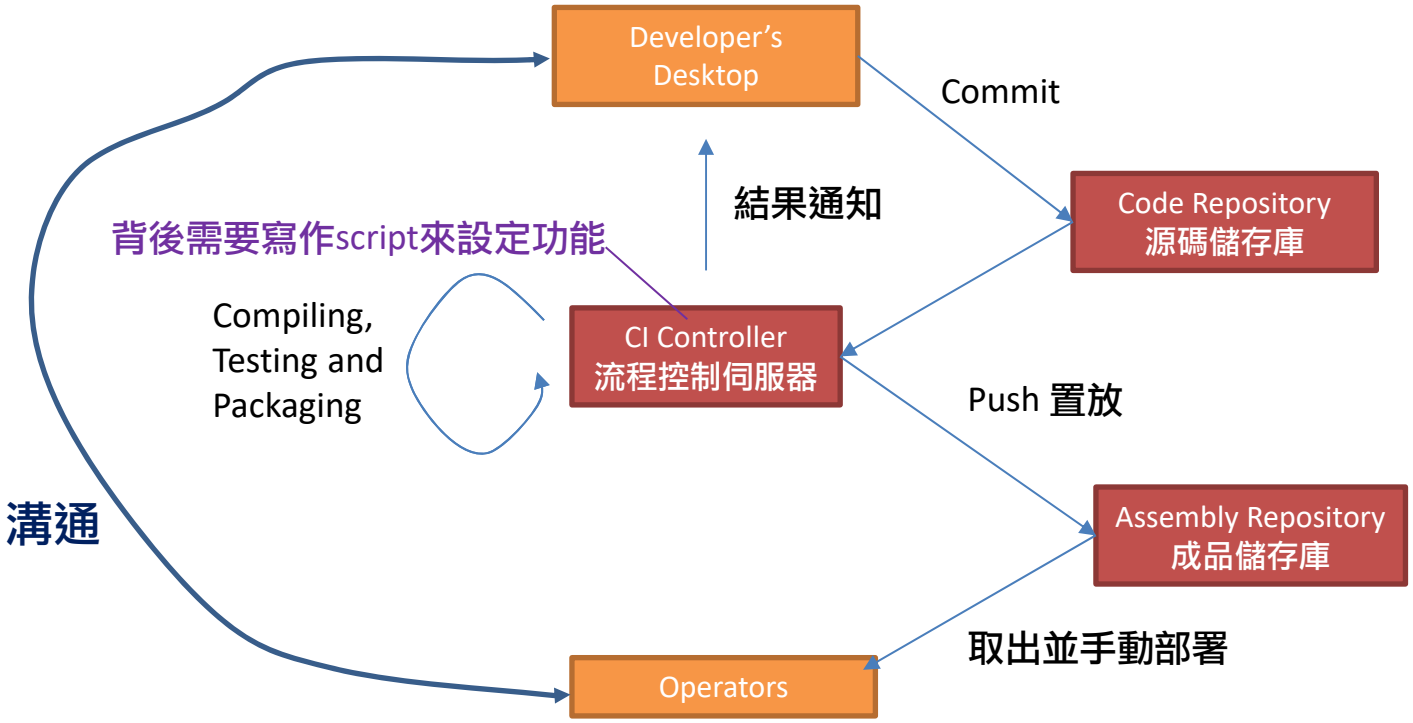
Continuous Integration

- 定義
  - frequently integrating new or changed code with the existing code repository
  - merging all working copies to a shared mainline regularly
- 代價
  - 編寫自動化測試碼
  - 維護額外設施 (CI Server)
- 利益
  - 整合時不易出現重大錯誤
  - 開發人員能專心於開發/除錯 (開發到一半被叫去修大錯誤)

Continuous Integration

- 實現CI的核心技術
  - Controller (Jenkins)
    - 依設定好的流程自動組建、派送各項資訊
  - 分散式版控 (Git)
    - Github flow (Github官方)
      - Create a branch → Add commits → Open a Pull Request → Code Review → Test → Merge
    - Git flow (Vincent Driessen, 2010)
      - 定義五種branch: master、develop、hotfix、release 以及 feature
      - 所有的變動統一merge到develop
      - 正式發佈版本或hotfix才merge到master
      - 2020/3/5 作者建議，由於技術進步，當前應改用Github flow簡化版控

The CI Pipeline

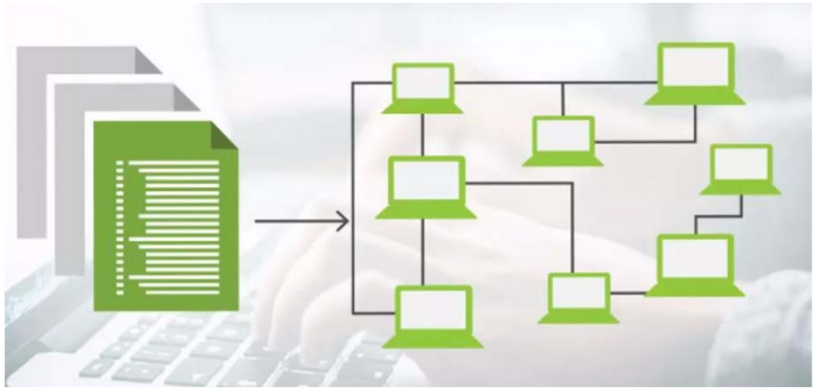


Continuous Delivery/ Deployment

- 定義
  - 藉由自動化讓軟體產品的產出過程在短週期內完成
  - 讓軟體可持續的保持在隨時可以釋出的狀況
- 和CI的差異
  - 涉及多個團隊之間的合作（開發、運維、QA、管理部門等）

IaC: Infrastructure as Code

- 定義
  - 將IT日常維運工作，寫成程式，以自動化腳本(as Code)方式運行
  - 基於既有軟體工程品保原則來管理自動化腳本
  - 所有工作化成自動化腳本→ 管理好腳本=做好維運工作
- 好處
  - 降低負擔
    - 自動化提高了效率
  - 降低風險
    - 每個改動都經品保與版控
    - 維運工作的可重現性大幅提高
    - 自動化→不易出錯→品質提高
  - 維運工作內容具體化
    - 不再「只有一個人會」，輪替、交接變容易



維運工作: 安裝、更新、配置、遷移、監控、修復

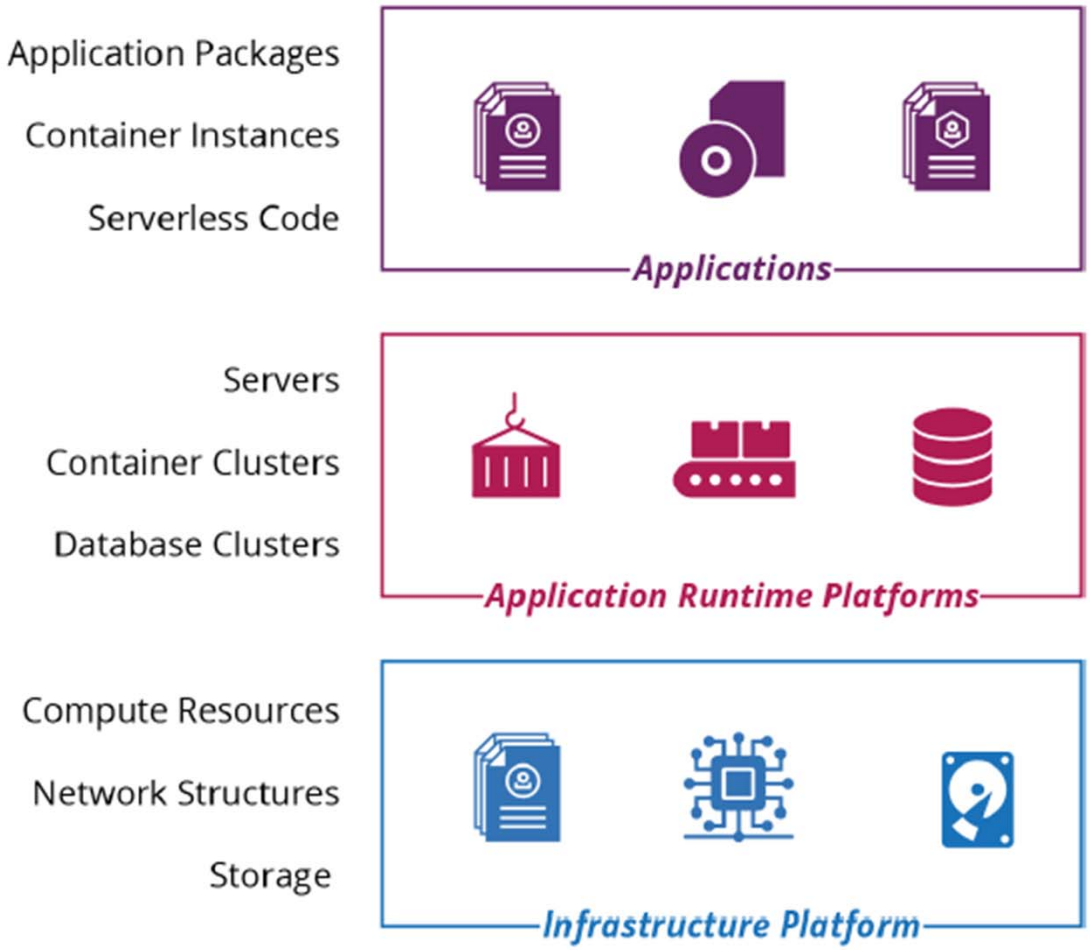
Infrastructure as Code

- Principles
  - Assume systems are unreliable
    - Create disposable things
    - Cattle, Not Pets
      - “CERN Data Centre Evolution” by Gavin McCance
  - Make everything reproducible
    - Ensure that you can repeat any process
    - Pitfall: Snowflake systems
      - 系統管理時，經常為了偶發事件一次調一點設定
        - » 如: 加裝一些模組在OS中、調整一些底層參數
      - 最後管理員本身也不敢輕易更動設定
      - 系統很難重建
  - Minimize variation
    - “管理100台配置相同的server比管理5台配置完全不同的server容易”

Infrastructure as Code

- 主軸
  - Stability comes from making changes?
  - Stability comes from making “disciplined” changes
- Core practices
  - Define Everything as Code
    - Benefits: Reusable, consistent, and transparent
  - Continuously Test and Deliver All Work in Progress (WIPs)
    - Make sure all WIPs are production ready
  - Build Small, Simple Pieces That You Can Change Independently
    - The code should be modularized

那些配置可以As Code?



Server Configuration Code Tools

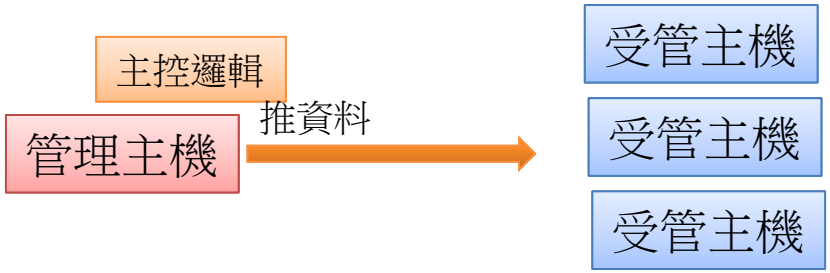
- Tools
  - Ansible
  - CFEngine
  - Chef
  - Puppet
  - Saltstack
- Configuration management
  - Ansible Tower (提供UI, 儲存playbooks)
  - Chef Server (儲存config files)

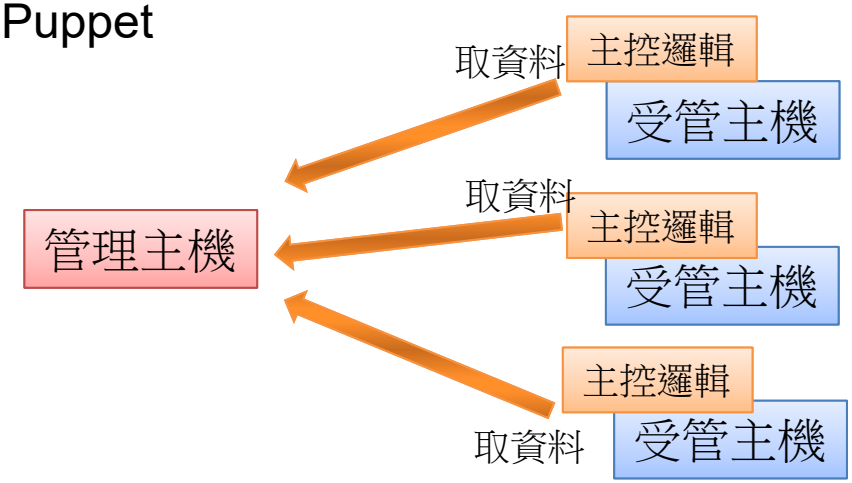
Define Everything as Code

- Two approaches
  - Declarative (functional, “what”)
    - Defining “What”
    - Typically idempotent
    - Ex: Ansible, Terraform, CFEngine
  - Imperative (procedural, “how”)
    - Defining “How”
    - Ex: Ansible, Chef, Puppet

Define Everything as Code

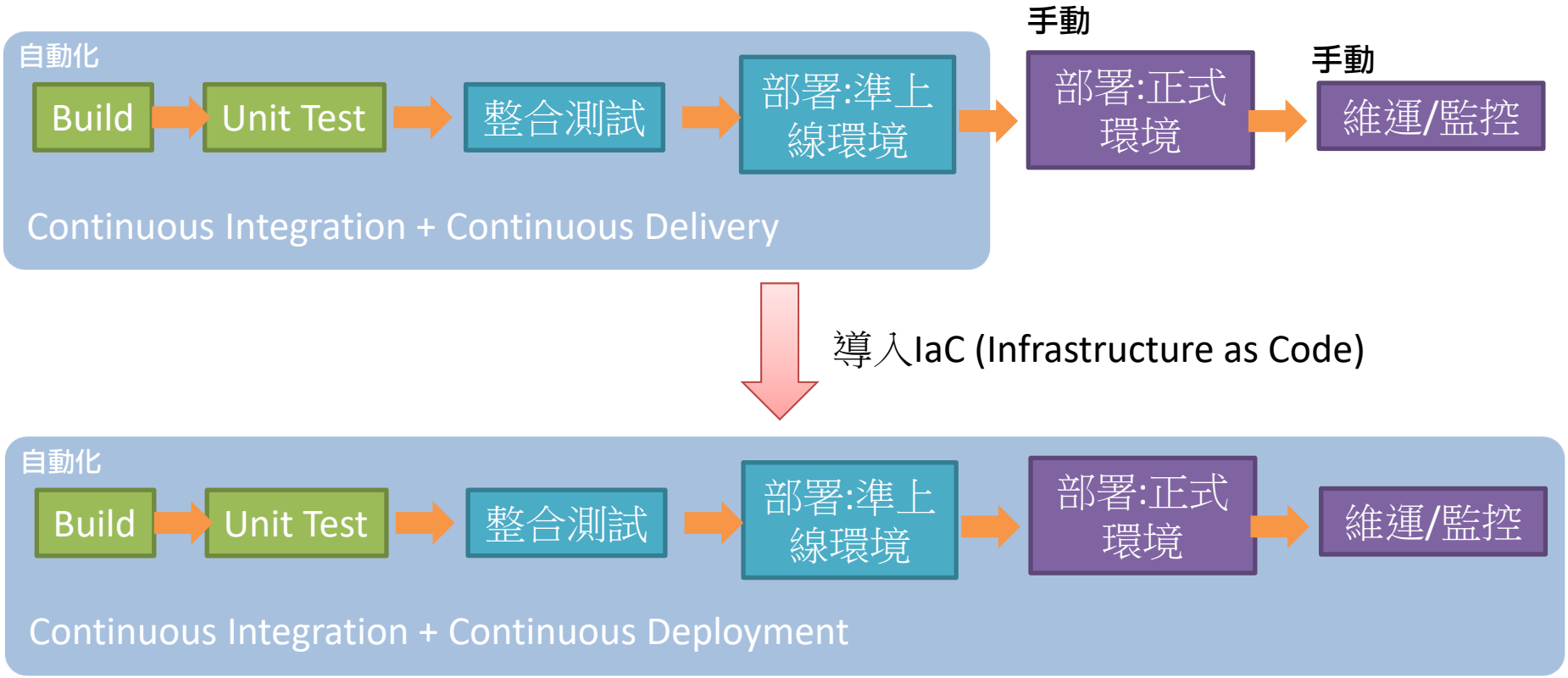
- Two operation modes
  - Push
    - The config scripts are pushed from config server to the target
    - Ex: Ansible、Terraform
  - Pull
    - The config scripts are pulled from config server to the target
    - Chef, Puppet



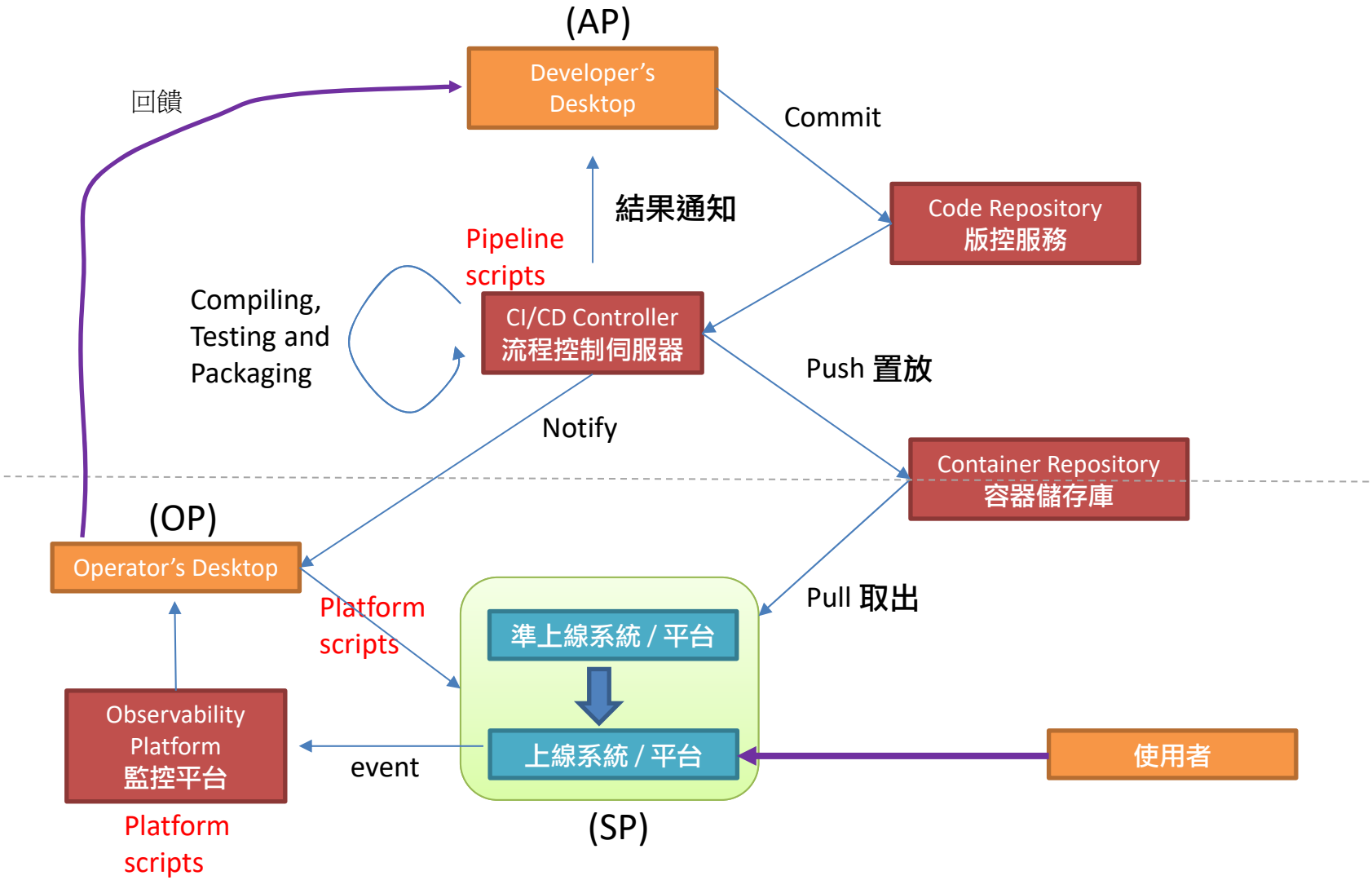


願景: 軟體產線全面自動化

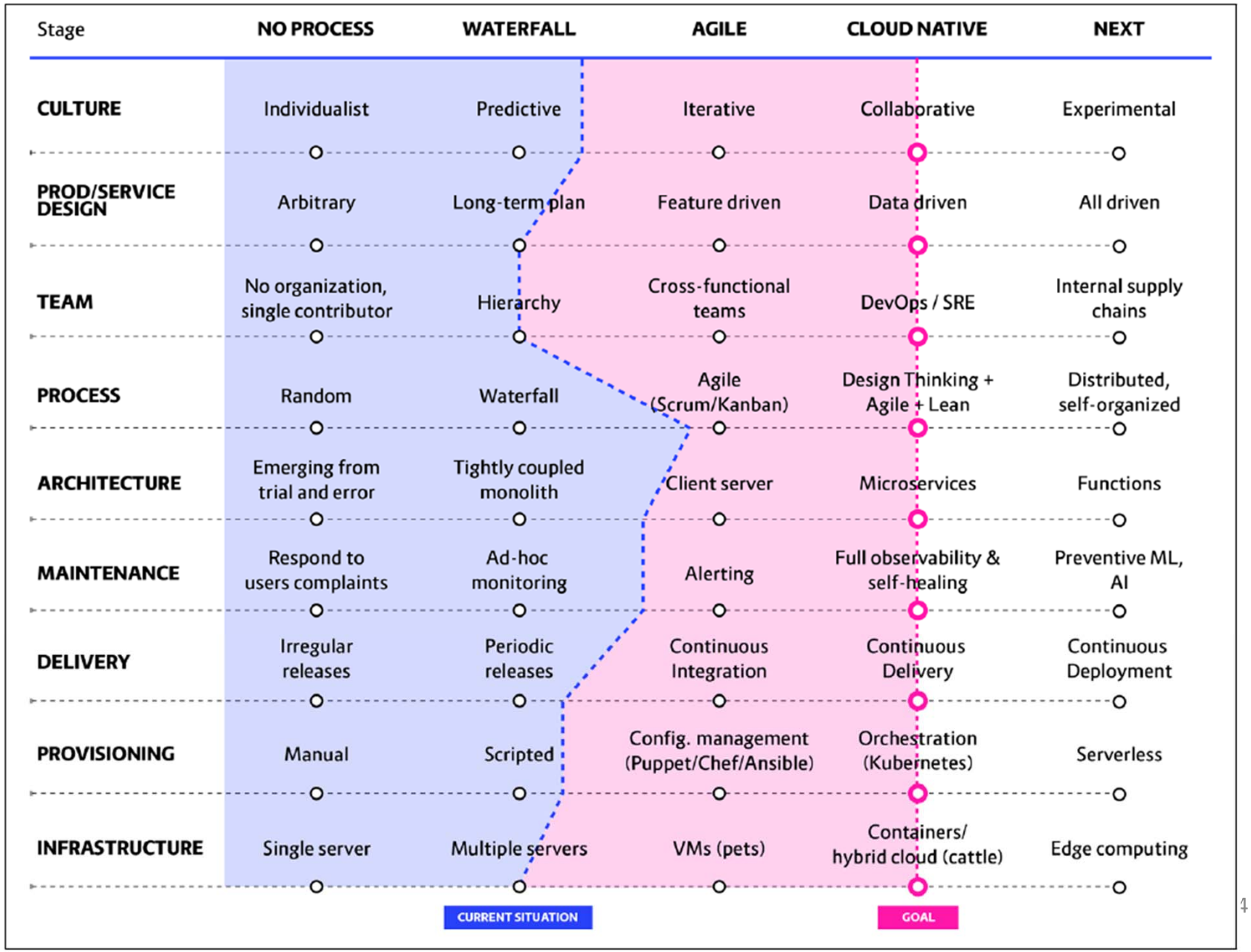
CI = Continuous Integration 持續整合
CD = Continuous Delivery / Continuous Deployment 持續交付/持續部署



Full CI/CD Pipeline



Cloud Native 成熟度量表



Q & A