91APP: 從 0 開始的 DevOps

2019/10/18

Chief Architect, Andrew Wu

DevOpsDays Taipei 2019

91APP 品牌新零售 虛實融合OMO最佳夥伴

前言: 為何要從“零”開始?

91APP 開發流程與架構改善計畫 執行心得分享 (2019 ~ present)

專案開始之初，首重看見全貌

一旦；當你把眼光投注在哪一個要項的時候，實際上你就只看到那一部分，
你的思緒將被那一部分的內容所牽動，很難再看見其他的事...
所以我們要退後一步，
不! 有時要退後很多步，才能比較清晰地看見全貌。
(好有趣喔! 退遠了卻反而看得更清晰)
所以要調整範圍，試圖把我們在意的事放進視線可及之處，
淡淡的審視它，不帶一點情緒，就是這樣，我們看見了全貌。

https://ruddyblog.wordpress.com/2017/10/27/看見全貌/

> 不是足夠大的公司，不具有這樣的規模量，你還沒機會遭遇此問題，就如我們過去說的，**技術債是屬於那些活下來的公司，至於那些撐不下去的，技術債跟你一點關係也沒有**。換句話說，就是你夠大，你才有機會碰到這樣的問題。

Top highlight

大多數一流的網路公司都曾發生過大規模的系統問題，差別只在於局部崩潰或是全面性的崩潰，但在它們長到這麼大之前，這種異常問題還會少見嗎？一點也不，AWS、Facebook、阿里雲、Netflix、LinkedIn這些公司其實都發生過大規模的異常事件，這些公司的工程師的高水平我想大家都略知一二。

> **一家公司的技術水平，往往都是在遭遇到營運面的困難時，才踏上加速突破的道路。**

https://medium.com/how-gipi-learn/1111-system-reliability-559364b5591d

回顧: 91APP 在 DevOps 面臨的挑戰

虛實融合OMO最佳夥伴

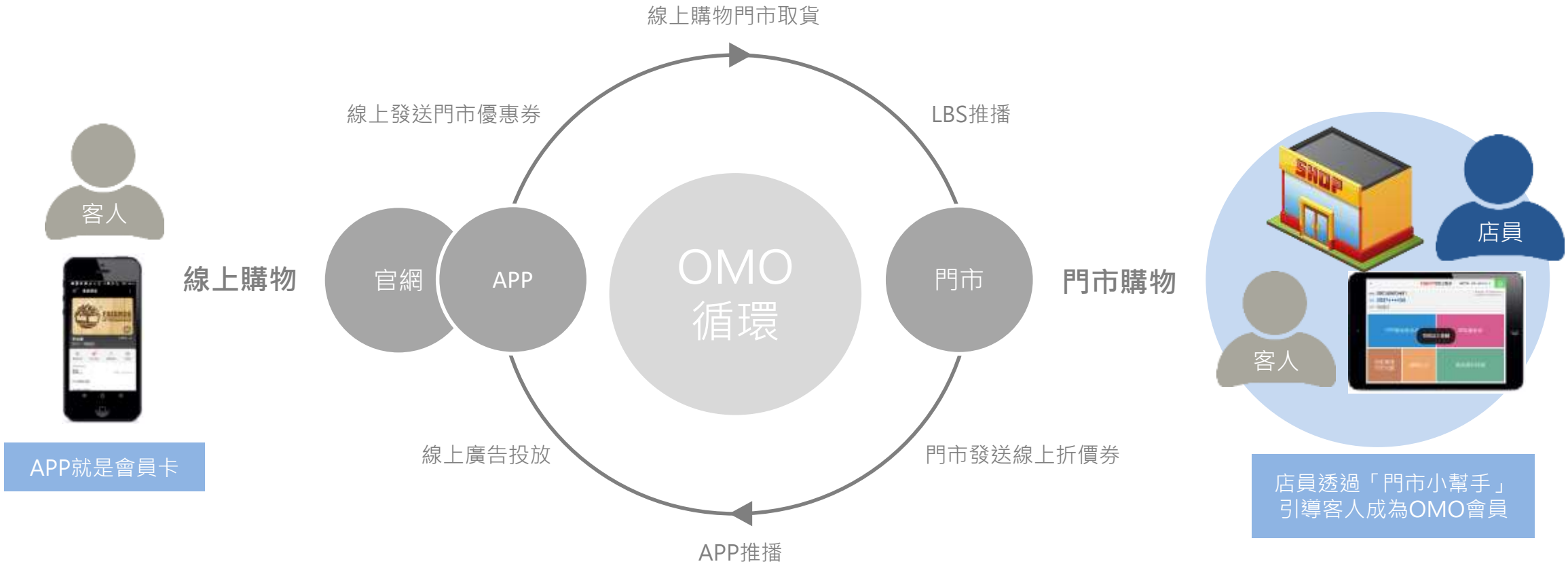
台灣最大&成長最快
品牌新零售解決方案公司

- 2013年成立，前Yahoo!、興奇科技經營團隊創辦
- 總部在台北，馬來西亞/香港分公司
- 員工人數超過400人
- 連續四年榮獲「創新商務獎/最佳商業模式」
- 獲選「勤業眾信亞太區高科技高成長前500強」
  (Ranked 152th,Deloitte Technology Fast 500 Asia Pacific)

91APP 品牌新零售 虛實融合OMO最佳夥伴

協助品牌打造線上電商與門市OMO循環

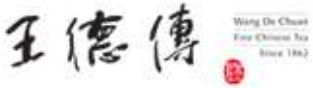
提供一致化購物體驗，打通全通路會員經營，有效掌握全景數據，提升OMO營運效能。



91APP 品牌新零售 虛實融合OMO最佳夥伴

品牌客戶超過10,000家

獲國內外大型實體零售品牌肯定，91APP協助多家企業成功推動OMO變革轉型。



DevOpsDays Taipei 2019

91APP 品牌新零售 虛實融合OMO最佳夥伴

背後代表的是: 龐大的部署規模

AWS: 200+ EC2, 3 regions…

GCP: …

AZURE: …

Local: Mac x N …

背後代表的是: 跨國 (多個市場) 的部署

TAIWAN

MALYSIA

HONG KONG

…

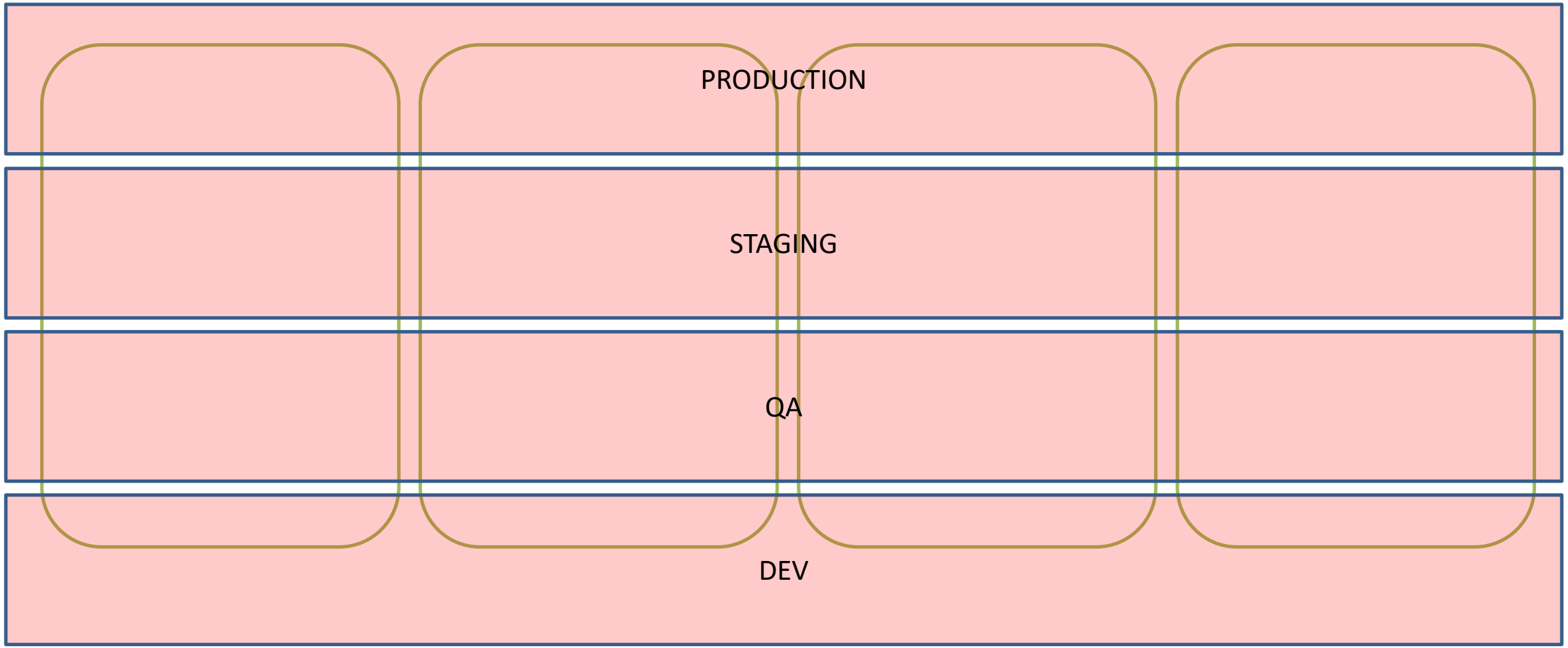
Market: 4+

背後代表的是: 複雜的環境組態

5	類別	子類別	DNS	用途	使用對象	對外公開？	SSL KEY	DNS Record
6	Search		api.search.{Service-Domain}	Search	91APP 內部系統	private		api.search.hk.91app.io
7			write.search.{Service-Domain}	Search	91APP 內部系統	private		write.search.maxis.91app.io
8			read.search.{Service-Domain}	Search	91APP 內部系統	private		read.search.maxis.91app.io
9			ac.search.{Service-Domain}	Search	91APP 內部系統	private		ac.search.maxis.91app.io
10			log.search.{Service-Domain}	Search	91APP 內部系統	private		log.search.maxis.91app.io
11	notification		push.{Service-Domain}	Push(待確認)	91APP 內部系統	private		push.gcp.maxis.91app.io
12			inform.{Service-Domain}	Inform(Email) 可共用	91APP 內部系統	private		inform.maxis.91app.io:19020
13			notify.{Service-Domain}	notify	91APP 內部系統	private		notify.maxis.91app.io
14			inform-cache.{Service-Domain}	ElastiCache Redis, Inform Service 可共用	Inform Service	private		AWS domain
15			voyager.{Service-Domain}	Service Discovery for NS SDK	Consul	private		voyager.maxis.91app.io:8500
16	Proxy		{Delegate-Name}		proxy ELB			gray.lebus.maxis.91app.io
17					proxy api			api.lebus.maxis.91app.io
18	API Gateway		apigw.{Biz-Domain}					apigw.maxis.91app.tw (暫停)
19	CMS		collage.{Biz-Domain}	CMS 後台	客戶	public		collage.maxis.91app.tw
20				CMS API				
21						public		
22						public		
23						private		
24			erp.{INTERNAL_DOMAIN}	ERP	91APP 全公司	private		erp.maxis.91app.biz
25			output-cache.{Service-Domain}		MWeb	private		output-cache.maxis.91app.io
26			image-cache.{Service-Domain}		MWeb	private		image-cache.maxis.91app.io
27			repl-cache.{Service-Domain}		MWeb	private		repl-cache.maxis.91app.io
28			data-cache.{Service-Domain}		MWeb	private		data-cache.maxis.91app.io
29			payment-middleware.{Service-Domain}		91APP內部系統	private		payment-middleware.maxis.91app.io
30			sftp.{Service-Domain}		檔案交換	public		sftp.gs.91app.io

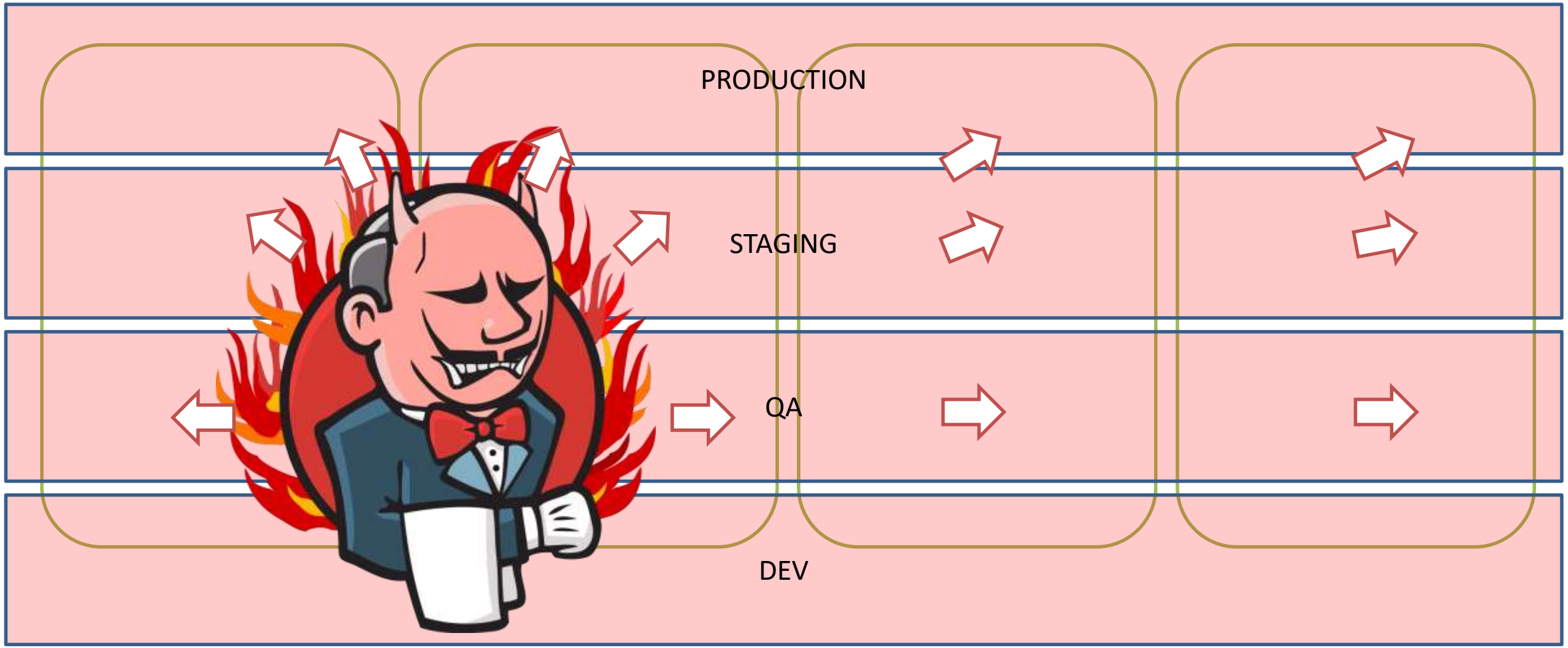
光是對內 & 對外的 DNS Records 就有 60+ ...

背後代表的是: 別忘了還有測試環境...



Environment: 10+

背後代表的是: 快喘不過氣的老爺爺...



Deployment: market x environment x instances (x shop …)

從零開始，邁向下個階段的策略...

組織

- **架構研究團隊:** 偵察部隊, 先行探索，研發合適的武器與科技，告訴團隊最好的作戰方針
- **基礎建設團隊:** 空軍/海軍先行轟炸，替團隊鋪好安全的路徑
- **主力開發團隊:** 有效率，穩健的進攻，按照步調攻下山頭，逐步擴張領土

架構研究團隊:

- 研究最適合 91APP 的流程與系統架構
- 實作與驗證，挑選 PILOT 專案確認成果
- 標準化與推廣，擴大應用範圍

基礎建設團隊:

- 備妥必要的基礎設施，備妥足以支撐架構的環境
- 配合架構團隊，整合最佳的開發環境，加速主力開發團隊的效率與提升服務的品質

關鍵原因: 問題在於系統的複雜度

解決方式: 從架構角度直接降低複雜度的維度（market **x** environment **x** instance **x** shop ... ）

借鏡別人的經驗: SaaS - 12 factor app

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

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

https://www.12factor.net
https://kknews.cc/zh-tw/tech/22k6ke.html

從 12 factor app 衍生出來的決策

1. Codebase – one codebase tracked in revision control, many deploys
- 部署程序一律從 artifact manager 取得 binary code, 確保 one code, one binary, multiple deploy.

2. Dependencies – explicitly declare and isolated dependencies
- 定義內部服務的 service catalog
- 服務間的呼叫一律透過 service discovery 機制來協作。

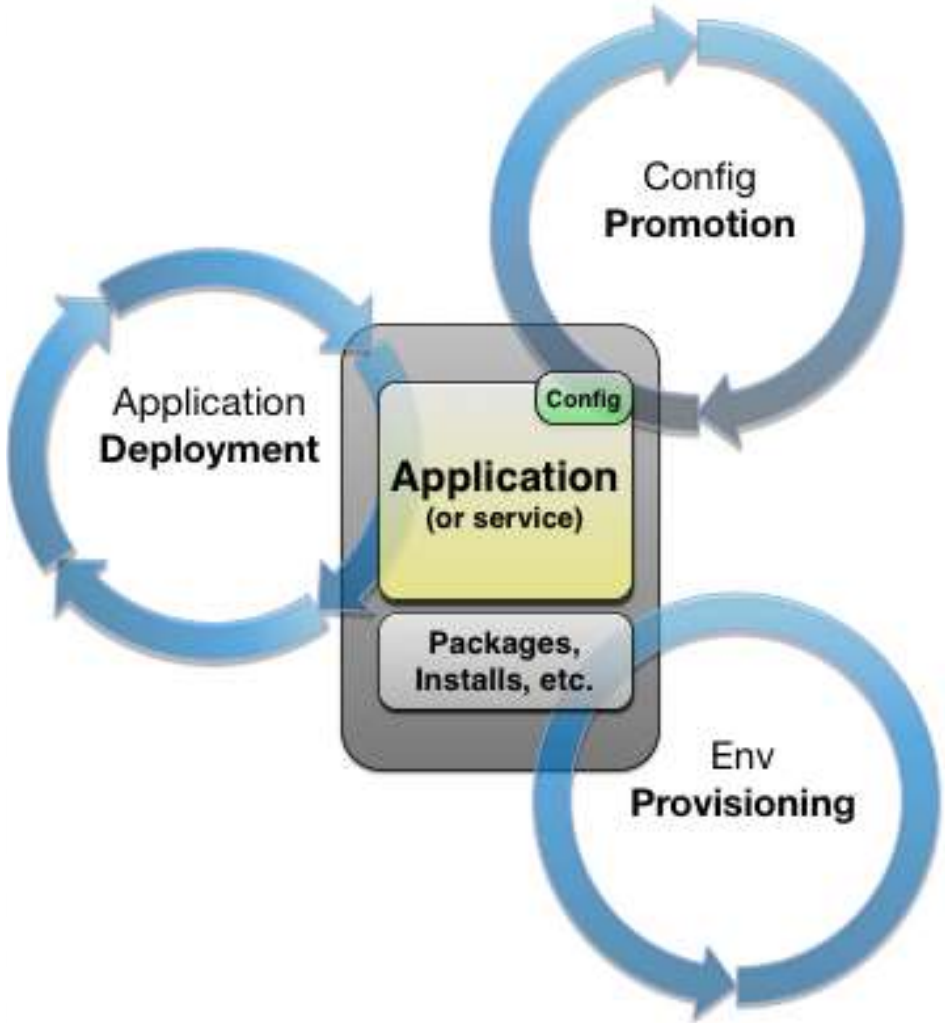
3. Config – store config in the environment
- 一個環境建置一套 config management 的服務 (廣義的 env: market)
- 每個 node 由 infra team 在 env 設置能找到 config manager 的最根本的資訊 (狹義的 env: VM)

5. Build, release and run – strictly separate build and run stages
- 用 artifact management 機制隔離 build & run 流程

7. Port binding – expose services via port binding
- 開放服務一律透過 Reverse Proxy, API Gateway, BFF, Edge Service 等機制。
- 內部則用 service discovery 機制協做

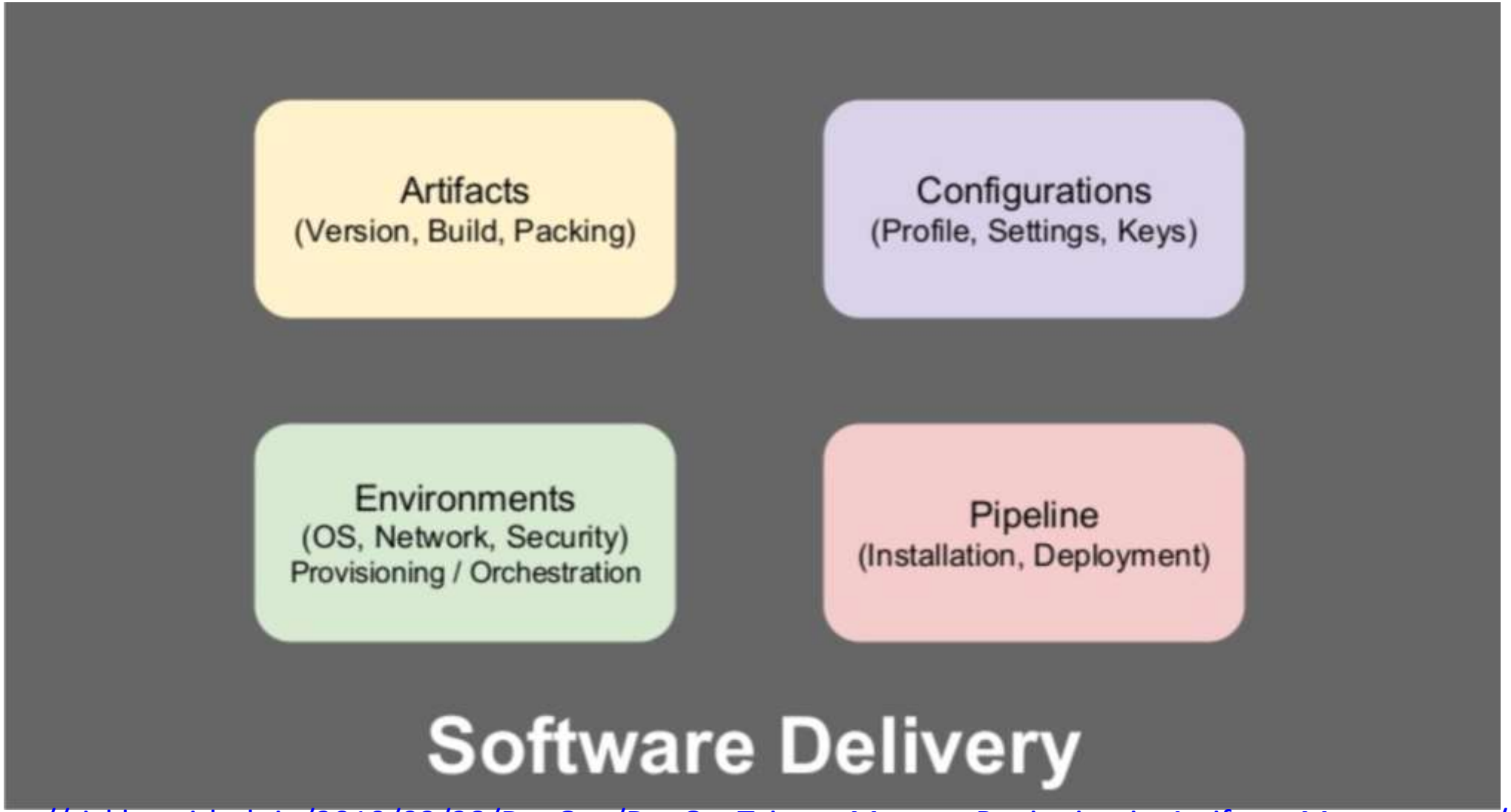
9. Disposability – Maximize robustness with fast startup and graceful shutdown
- 所有服務都應該做好自我管理，讓維運動作簡化成基本的啟動 (startup) 與關閉 (shutdown)。
- 在啟動與關閉時，與 service registry 做好溝通，建立可靠的 service discovery 機制。

解構軟體交付的架構: 三大元素



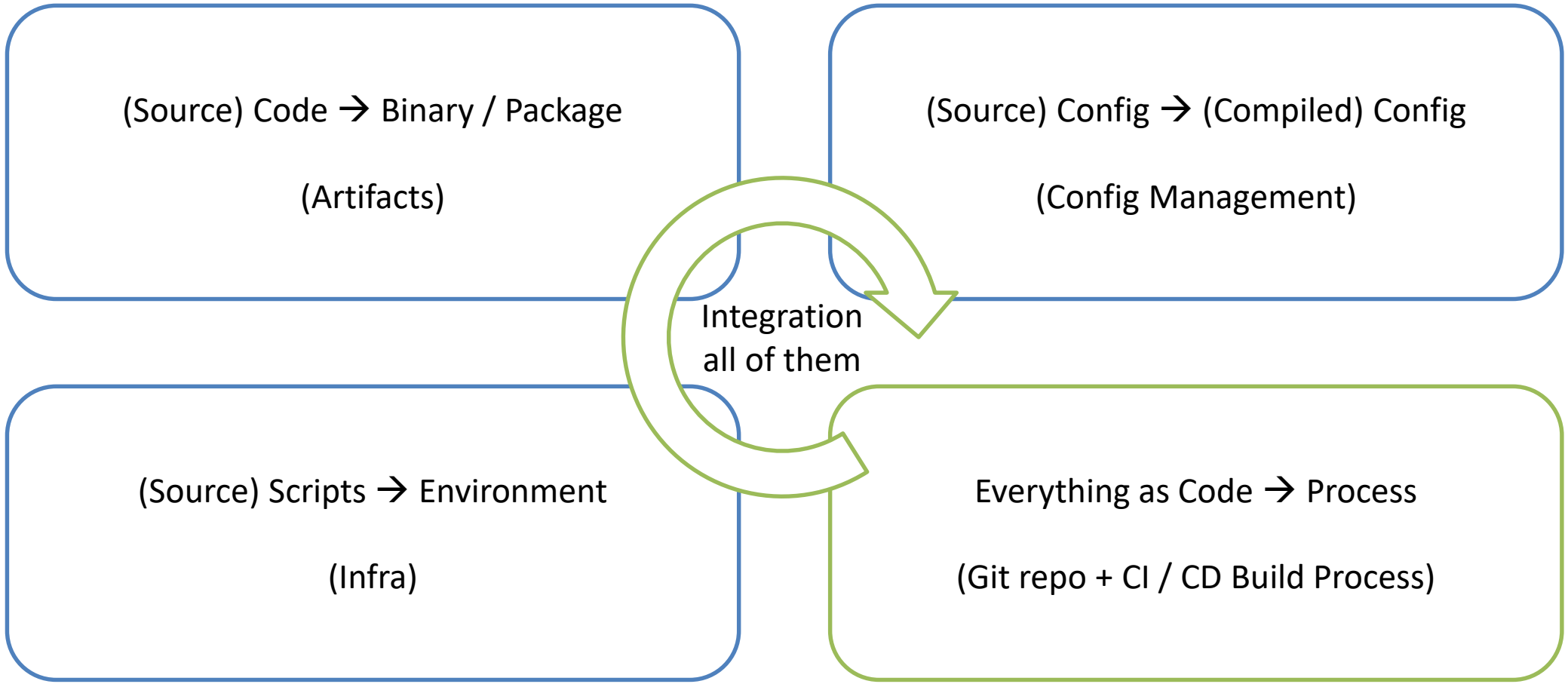
https://paulhammant.com/2014/08/27/provisioning-deployment-and-app-config-cycles/

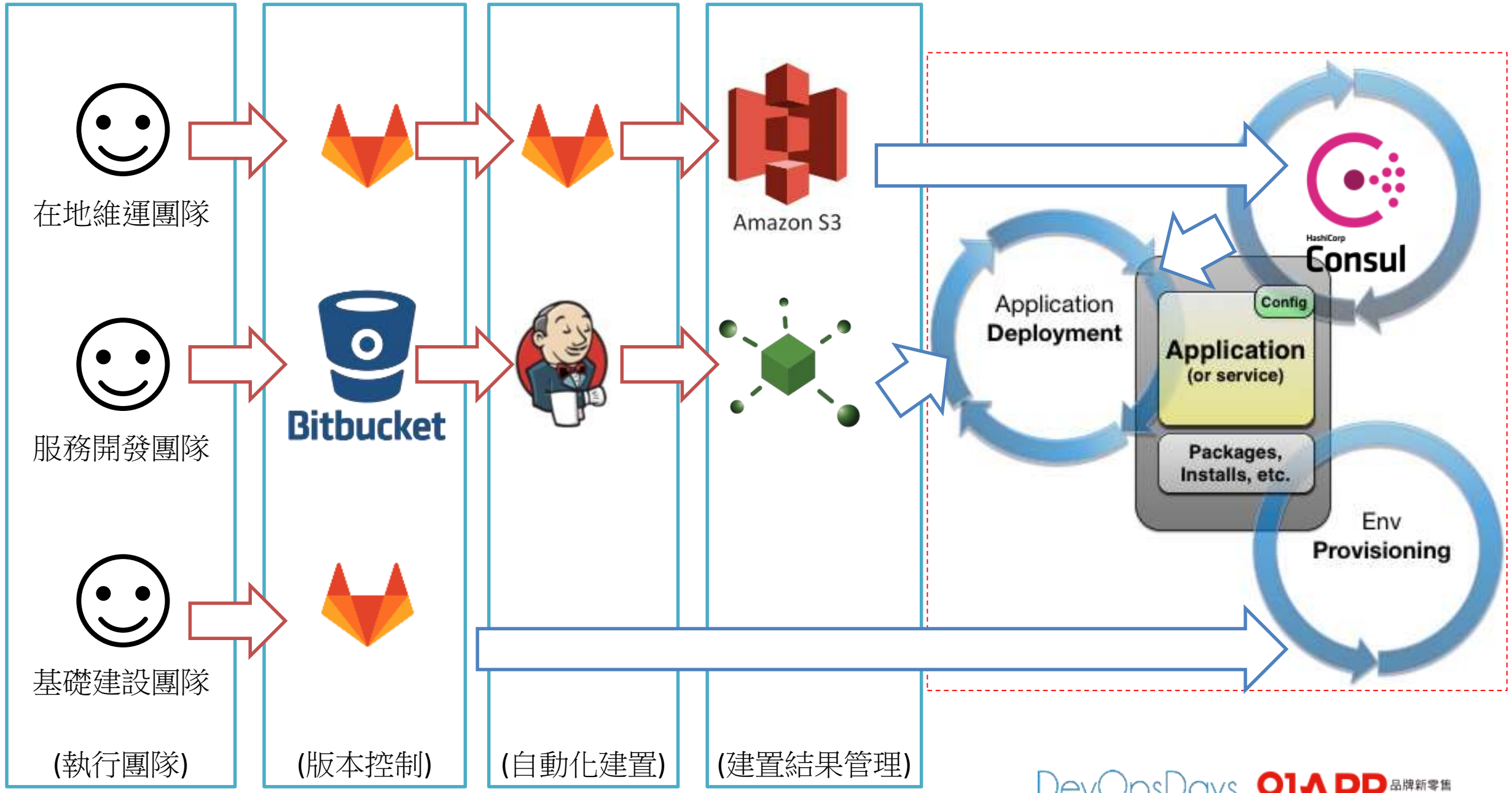
Rick: 軟體交付的四大支柱 (三大元素 + 單一流程)



https://rickhw.github.io/2019/03/28/DevOps/DevOpsTaiwan-Meetup-Beginning-in-Artifacts-Management/

Our View: 3 source + 1 process + integration





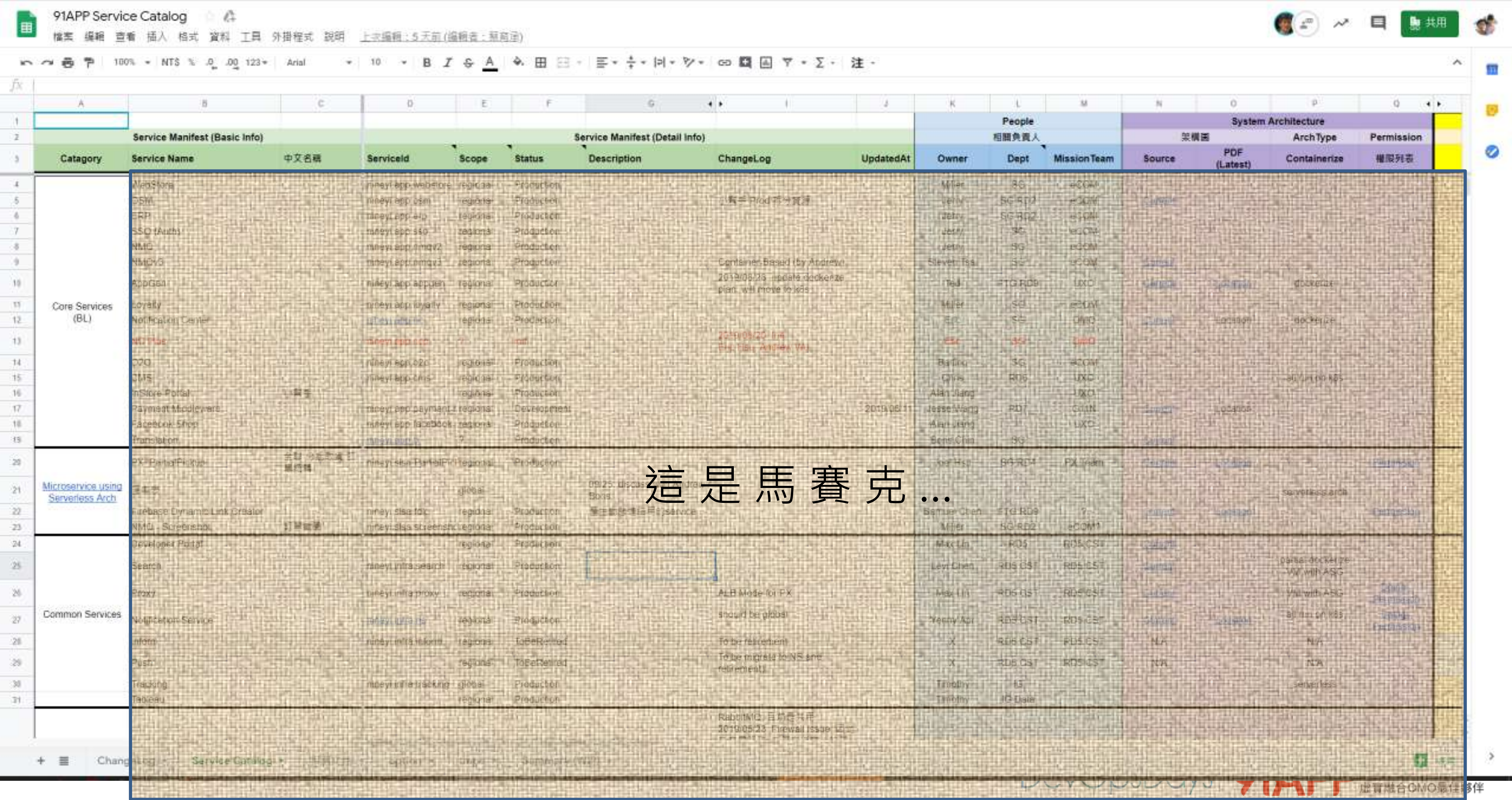
結束了? 挑戰現在才開始...

實際推廣時，我們碰到的挑戰..

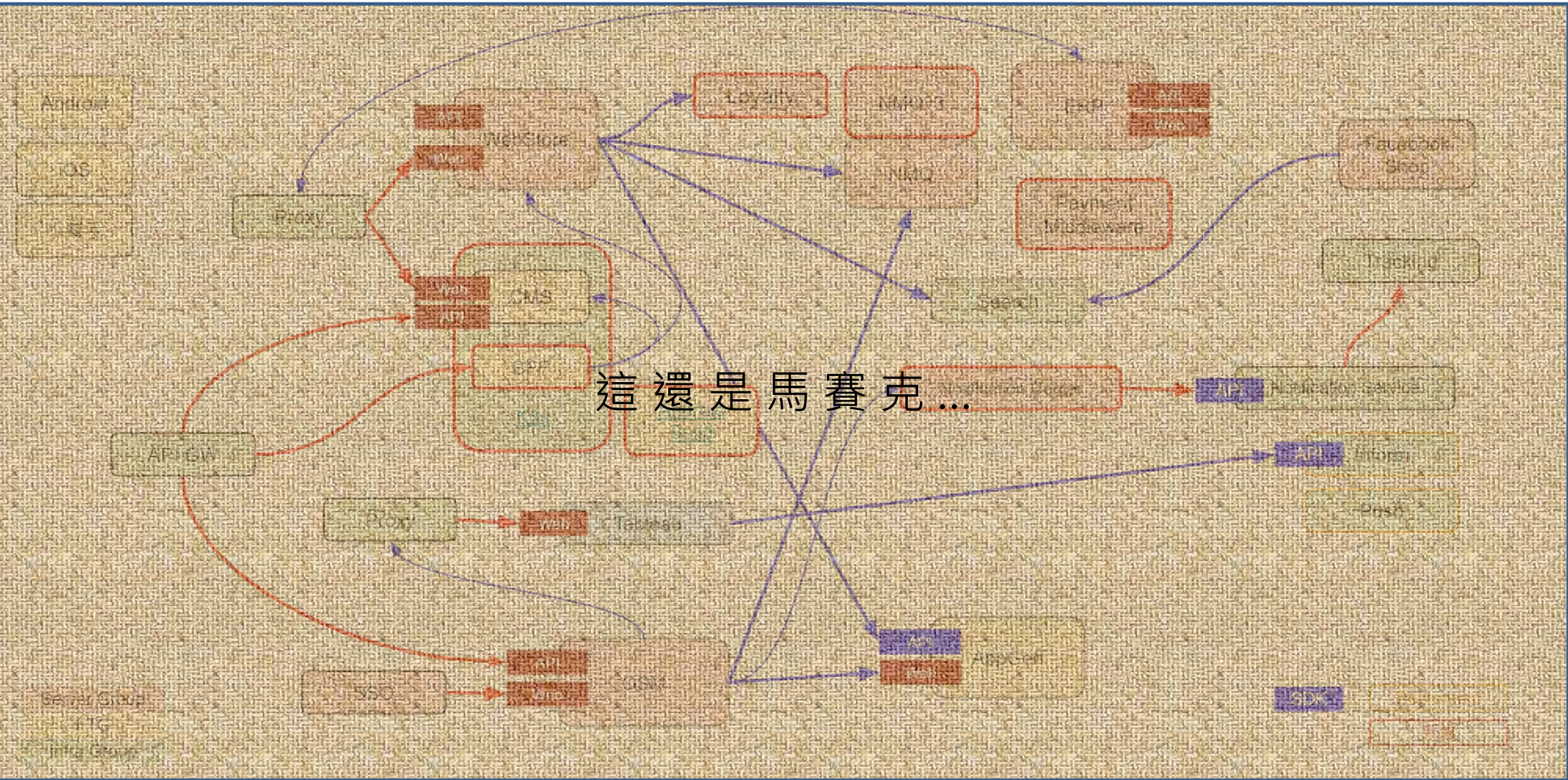
挑戰 #1

基礎建設團隊: 建置符合 91APP 部署需求的 Artifacts Management

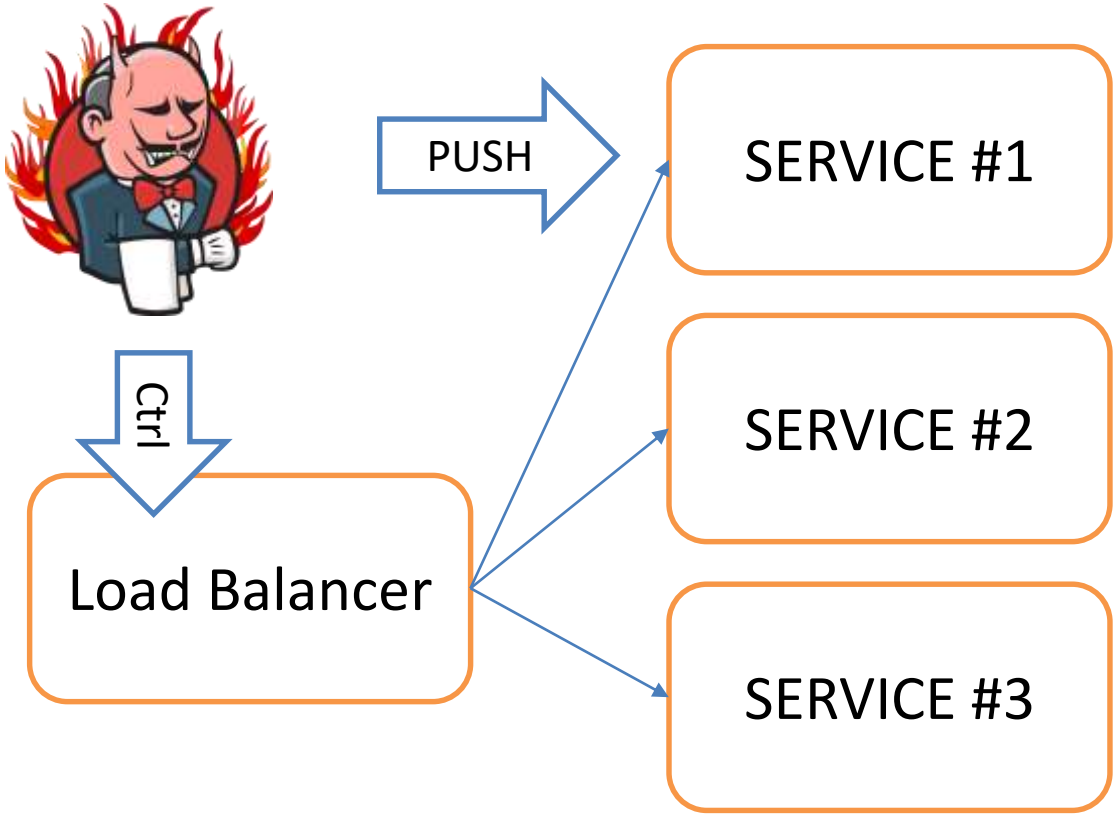
1. 定義有多少服務 (service catalog)
2. 定義服務的相依性 (service dependency)
3. 透過 artifact manager 管理服務的 build result (binary code)
4. 按照 (1) (2) (3) 來設計部署程序


91APP Service Catalog
Service Manifest (Basic Info)
Service Manifest (Detail Info)
People
System Architecture
Catagory
Service Name
中文名稱
ServiceId
Scope
Status
Description
ChangeLog
UpdatedAt
Owner
Dept
Mission Team
Source
PDF (Latest)
Containerize
Permission
Arch Type
Core Services (BL)
Microservice using Serverless Arch
Common Services
這是馬賽克...

Service Dependency (v2019Q3)



現有流程: 不斷升級既有 Instance 的版本。



過去: 太過強調 “自動化”
CI server 必須負責所有上線的程序，包含:

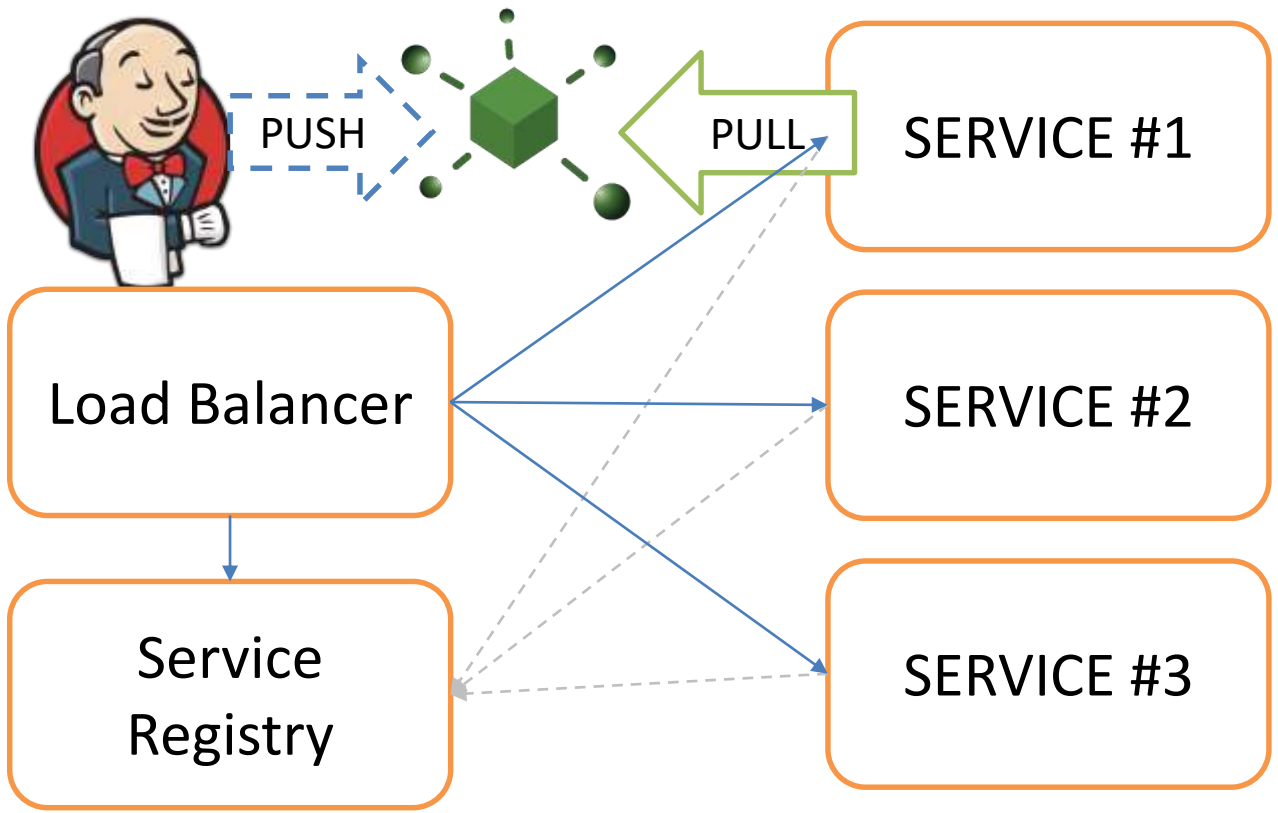
- Build Code
- Deploy (Each Nodes):
  - Offline Service #X
  - Deploy Service #X
  - Testing Service #X
  - Online Service #X

每個環境 (market x environments) 都需要一座有足夠權限的 CI server 來執行...

CI / CD 一次到位, 需要協調 infra 上線流程:

Build / Provision / Deploy / Online
部署架構複雜，效率不佳。
自動化程序難以開發維護

改善流程: Artifacts + Immutable Server



CI / CD 分階段進行, 隔離 Build / Deploy 階段:

Build | Provision (with Pull Artifacts) | Online (Auto Register)
部署架構簡單，效率好 (並行)。

- **流程優化為優先，做好更大規模部署的準備**
- CI 負責 Build Code, 將結果放置 Artifacts
- Deploy 由 Provision Node 階段負責。Node 被建立時從 AM 拉取 (PULL) 指定版本的 binary。要升級直接重蓋一次。
- Online 由 Start 程序負責。Node 啟動後透過 Service Discovery 程序自動跟 Service Registry 註冊, 自動完成上線程序。
- Offline 程序只要正常 Shutdown 即可。

軟體交付的四大支柱 (Four Pillars of Software Delivery)

Like Share 213 people like this. Be the first of your friends.

2019/04/04 23:43:00

上週 (2019/03/28) DevOps Meetup 分享的主題：聊聊軟體交付的濫觴 談產出物管理 (Artifacts Management)，我提出了 軟體交付四大支柱 (Four Pillars of Software Delivery) 的想法，如下圖：

Table of Content

- 延伸閱讀

Categories

DevOps 38

Tags

Artifact Management 12
Continuous Delivery 12
Continuous Integration 9
Configuration 8
Version Control 9

About

- 全站索引
- 關於這裏
- 關於作者
- 學習法則

https://rickhw.github.io/2019/04/04/DevOps/Four-Pillars-of-Software-Delivery/

挑戰 #2

架構團隊: 設計適合 91APP 業務需求的 Config Management

定義: 什麼是 91APP 的 configuration ?

靜態的設定

- 各種 application configuration / settings

Static: Key-Value Configuration

其他服務的連線資訊

- Database connection string
- Service end points
- API Key, Access Key, Access Token

Dynamic: Service Discovery

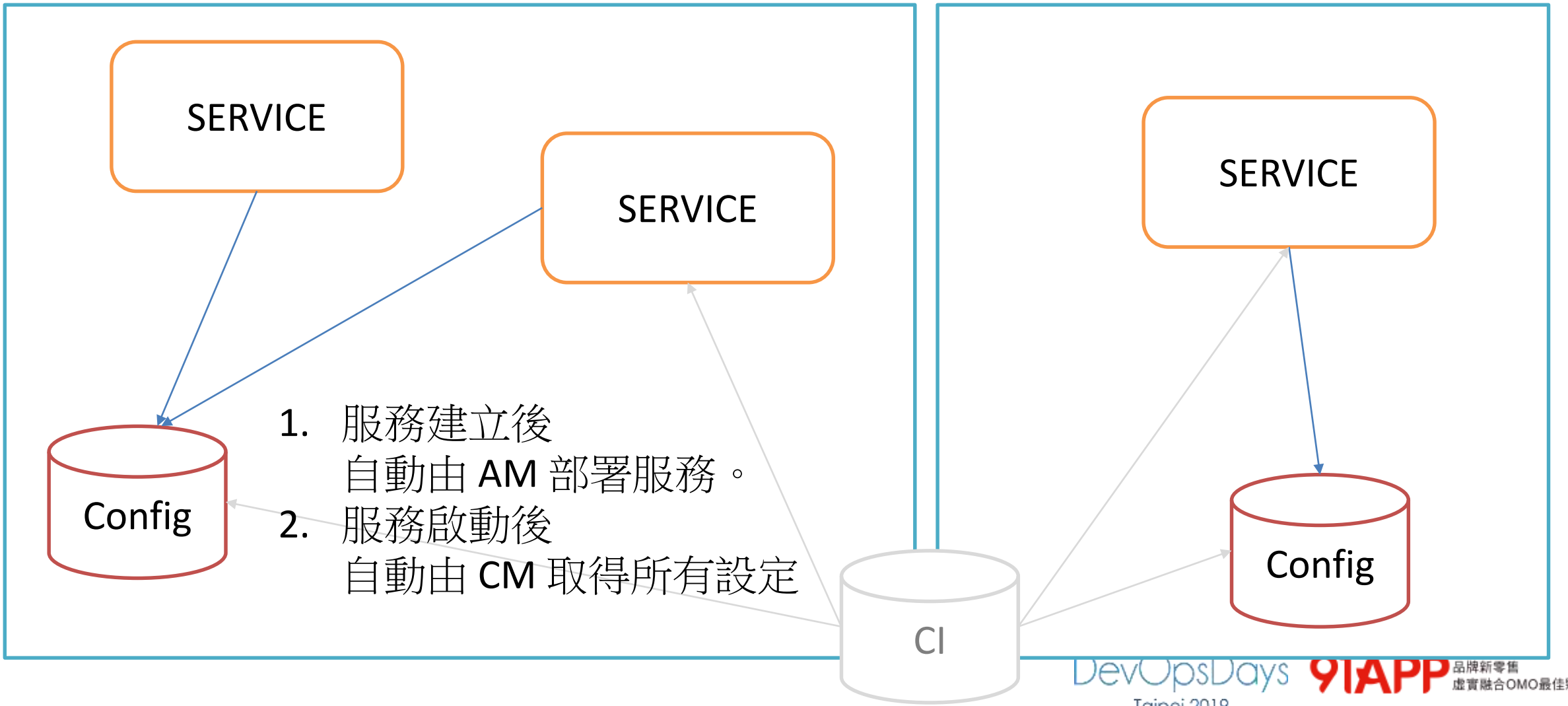
管理需求

- 全 91APP 的通用 (預設) 設定
- 按照地區 (region) 或市場 (market) 不同
- 按照不同客戶 (shop) 不同
- 簡化管理負擔
- 簡化開發方式

How To:

1. Optimize for Maintains Effort
2. Optimize for Development Effort
3. Optimize for Runtime Performance

為91APP量身訂做: 多資料中心 config server

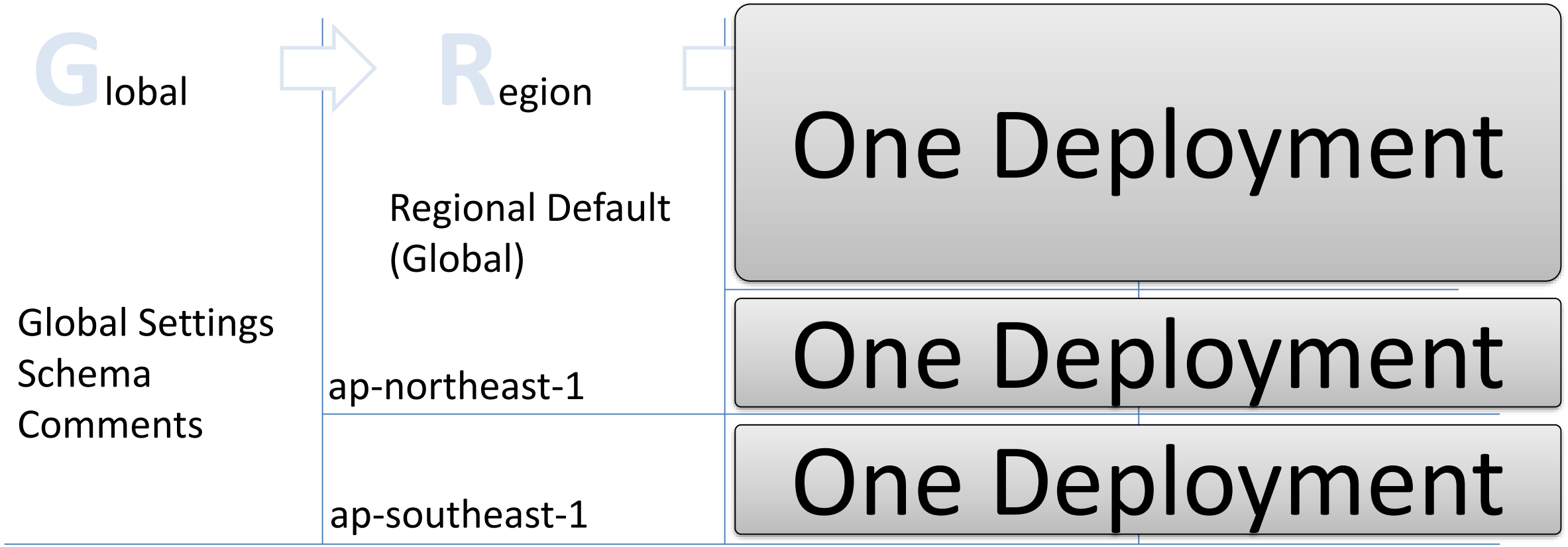


為91APP量身訂做: 階層式 configuration

One Git Repo

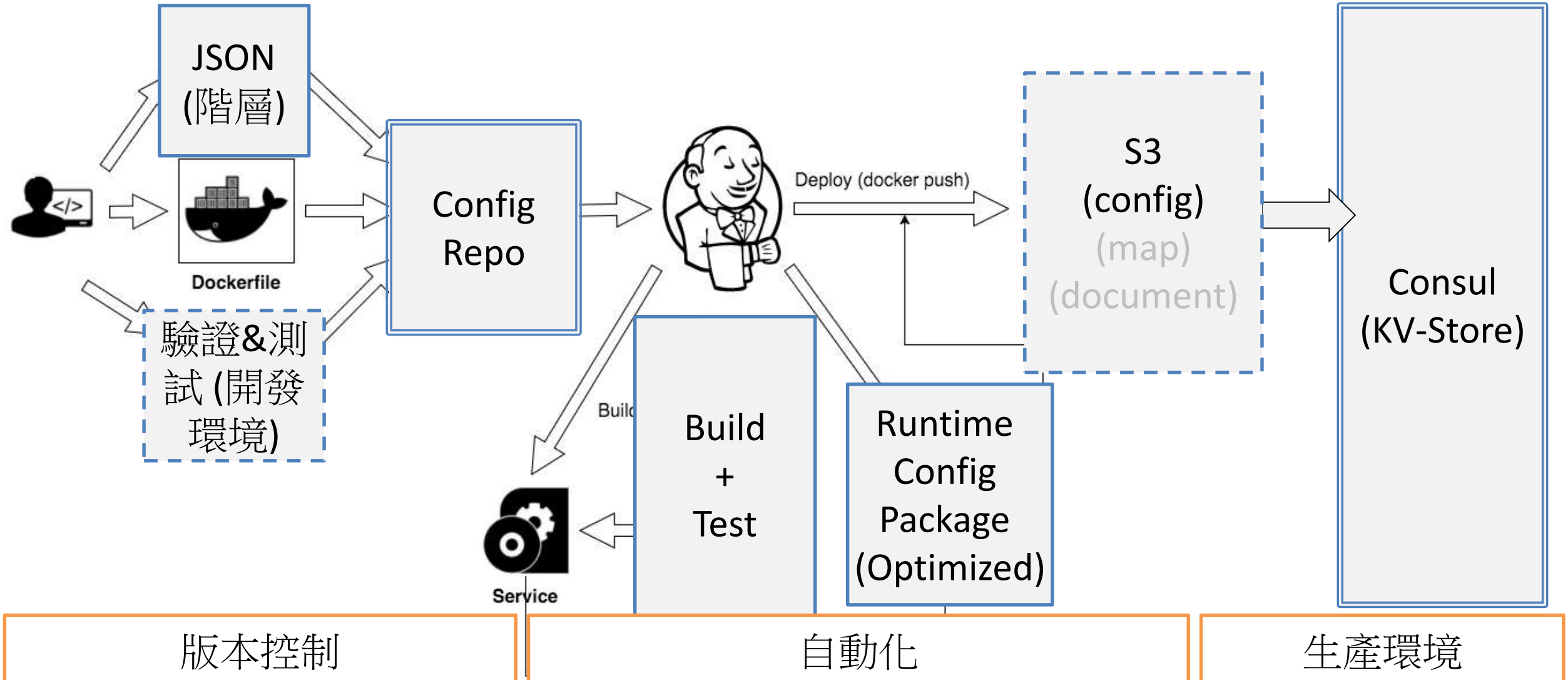
組態 (configuration) 的原始檔目錄結構，以方便管理為設計目標，團隊分工優化為第一
目錄階層直接對應 Global / Region / Market / Shop 四層繼承階層。

為91APP量身訂做: configuration deploy

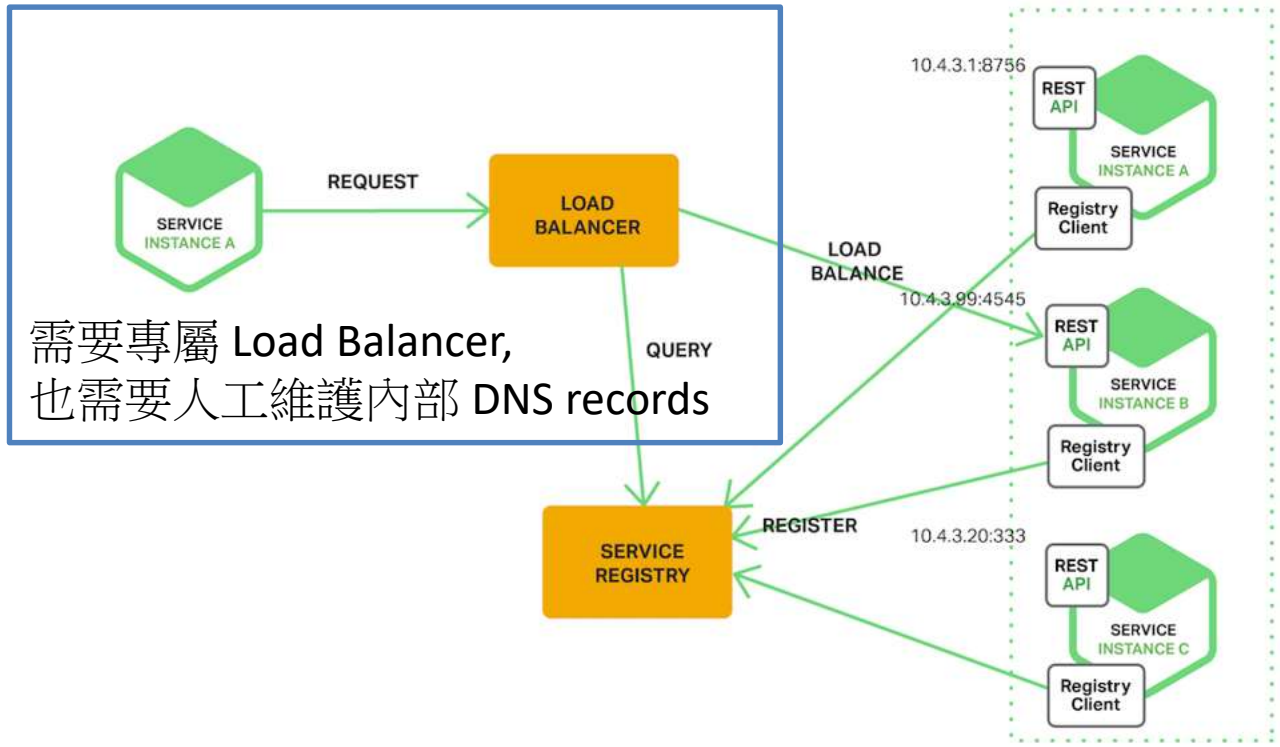


組態 (configuration) 編譯後的結構，以方部署與執行效率最佳化為設計目標。
目錄階層直接攤平，以 Region 為主體, 每個 Market / Shop 都有所有的組態設定，Clien
完成讀取設定值的動作。

Config as code

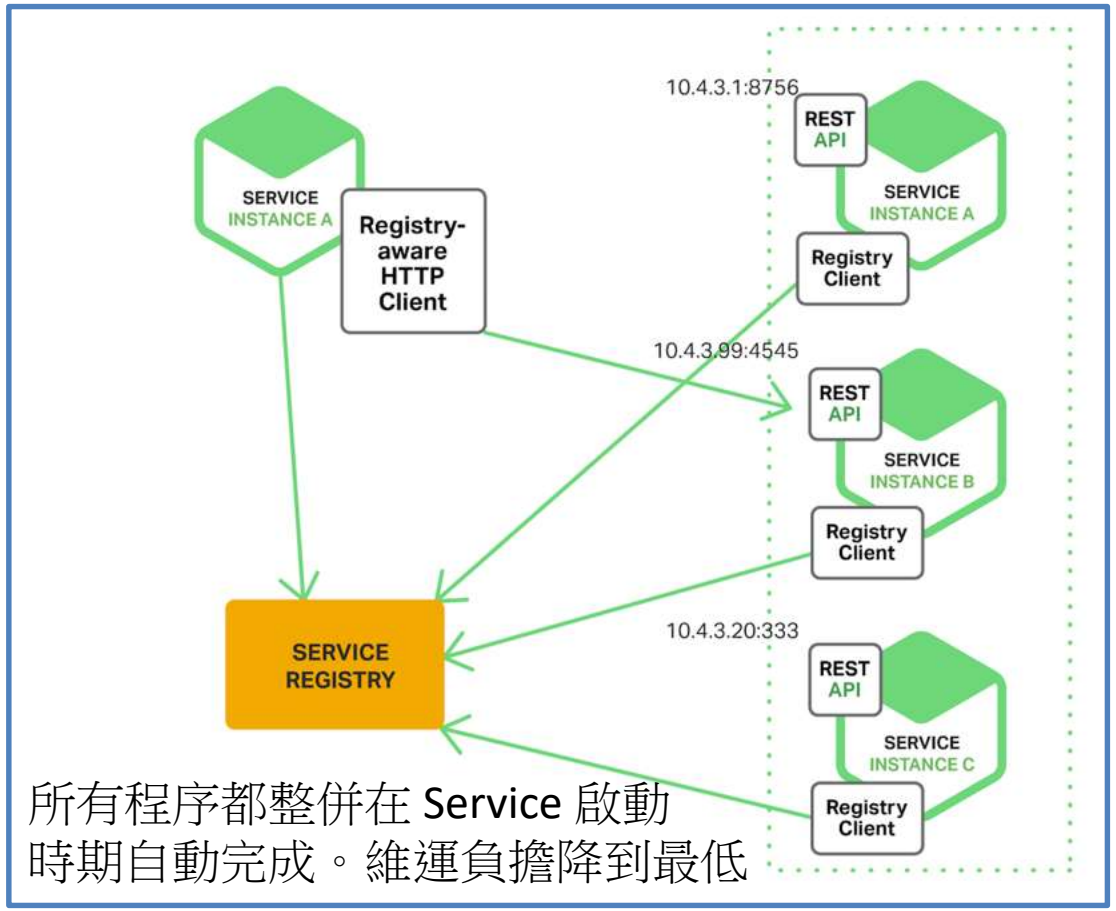


內部服務: 不必再依賴 DNS + Load Balancer



需要專屬 Load Balancer,
也需要人工維護內部 DNS records

The Server-Side Discovery Pattern



所有程序都整併在 Service 啟動
時期自動完成。維運負擔降到最低

The Client-Side Discovery Pattern

10/18 15:30 DE
DevOps Days 2019

從零開始的 Configuration Management

Levi Chen @ DevOpsDays Taipei 2019

91APP

RECAP: DevOps Days 2018

Service Discovery
微服務架構的基礎建設

Andrew Wu, Chief Architect @ 91APP

Sep 11, 2018

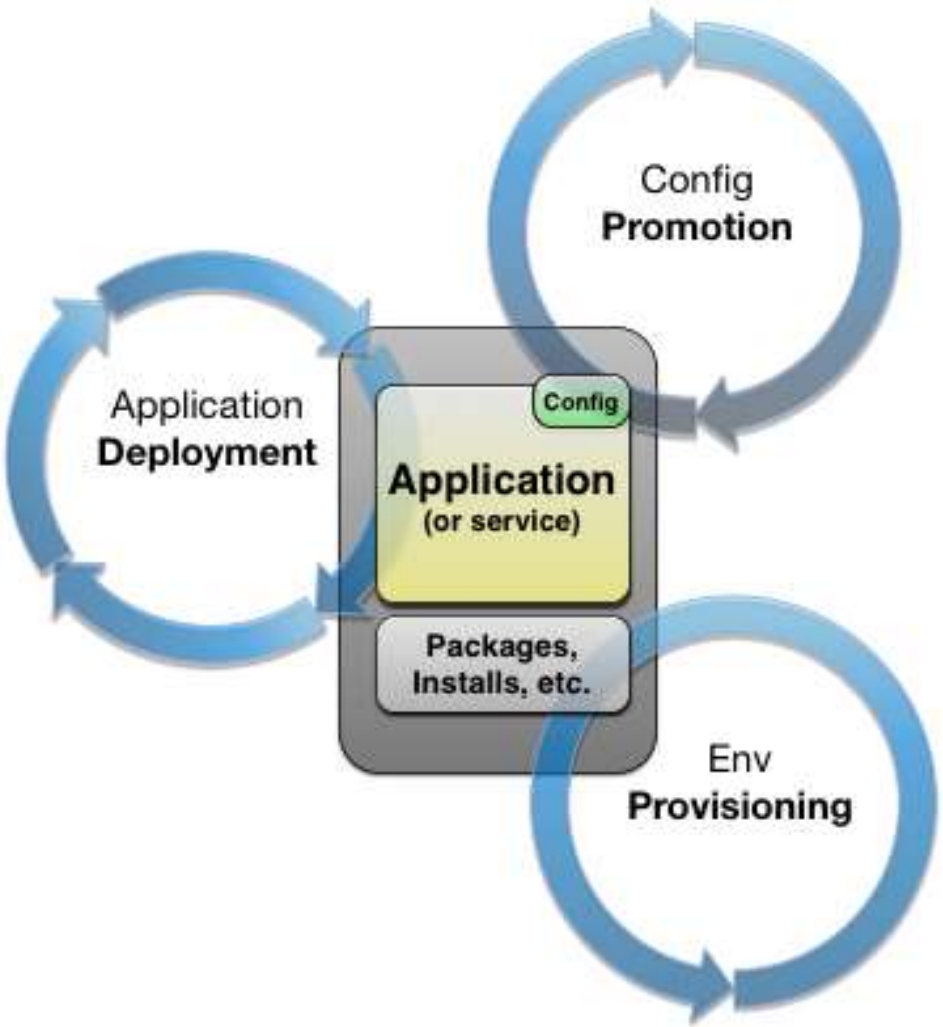
91APP 全通路 x 會員 x 數據 | 新零售最佳夥伴

DevOpsDays
Taipei 2018

挑戰 #3

架構 + 基礎建設團隊: 為 91APP 的開發團隊準備: 環境抽象化 + 基礎建設整合

什麼是 “環境”？



服務執行的環境

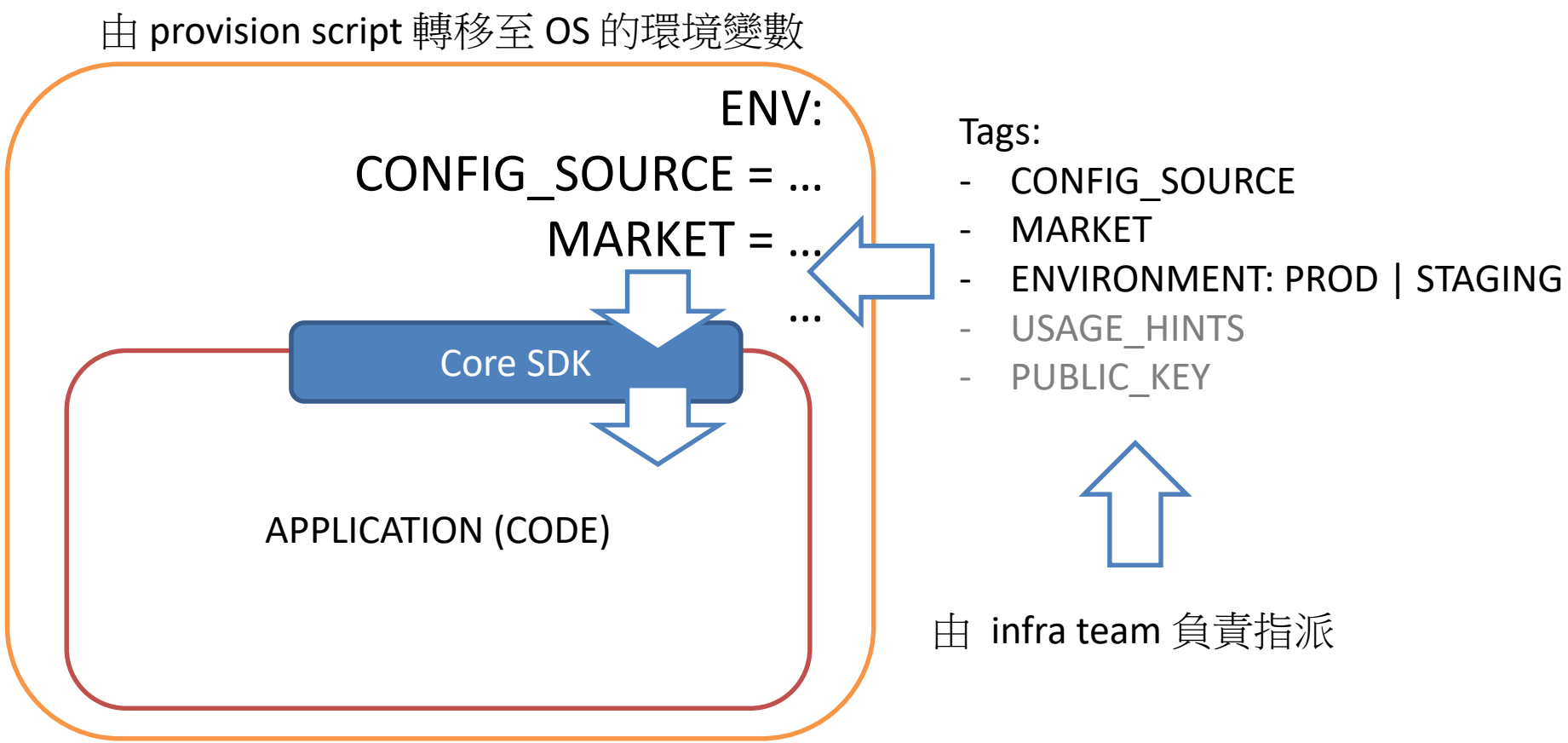
- OS
- Computing Resources
- Network

Configurations

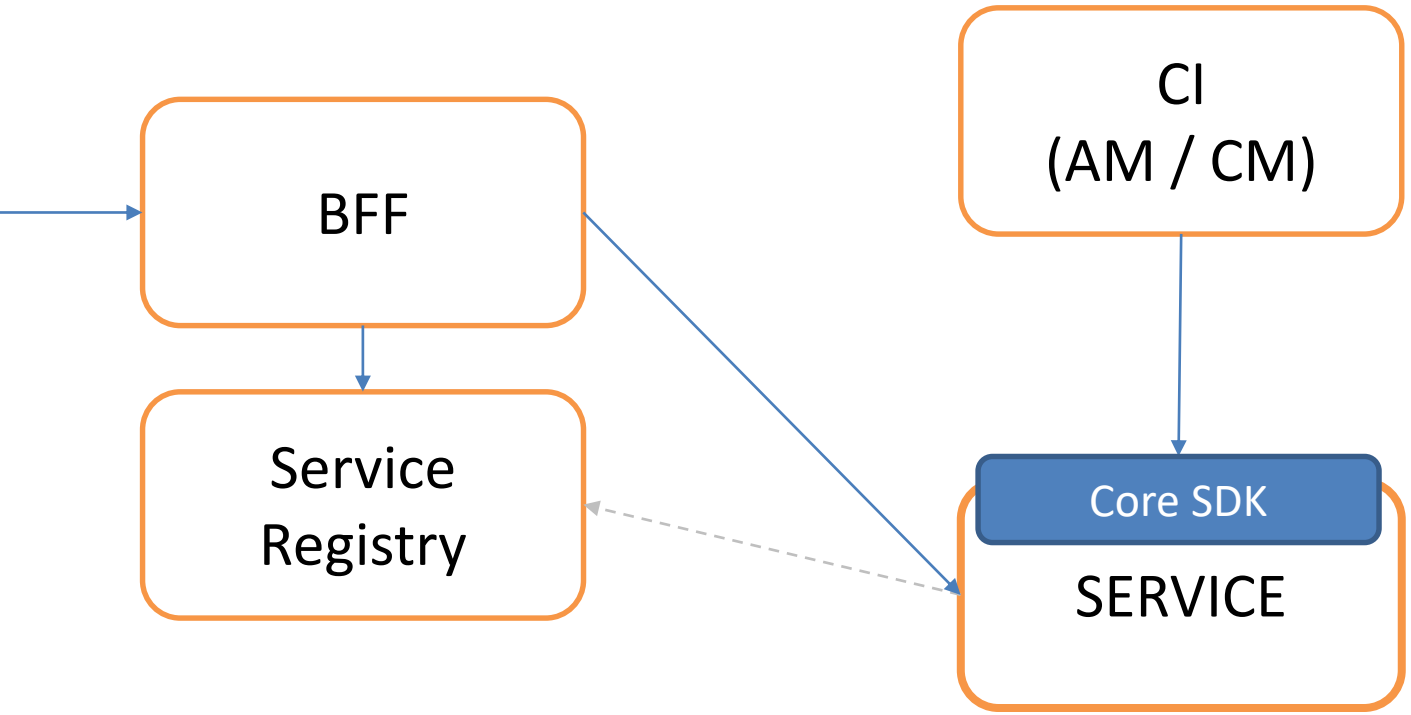
- Basic Config Connect To “Config Manager”
- Basic Environment “Context”
- Hint Information for Core SDK

> **III. Config**
> Store config in the environment

Core SDK 如何找到 Config Server 在哪裡?



服務開發規範: Self Management



目標: 將來 91APP 所有服務，都只要透過標準的 Provision / Start / Stop 程序就能維運。

ENV 第一次啟動時，必須自我組態

- 自動至 AM 部署服務
- 從 TAGS 載入環境變數

ENV 每次啟動時，必須自我註冊

- 從 ENV 尋找 SR / CM
- 服務啟動完成後，至 SR 註冊
- 回應 SR 健康偵測
- 回應來自 Load Balancer 的 Req

按照規範開發的服務，可以大幅降低維運的成本，提升維運的效率。大部分的維運作業，只需要透過開關機就該(能)完成。

Design For Operation, 是開發團隊降低 (自己) 維運負擔的最佳手段。

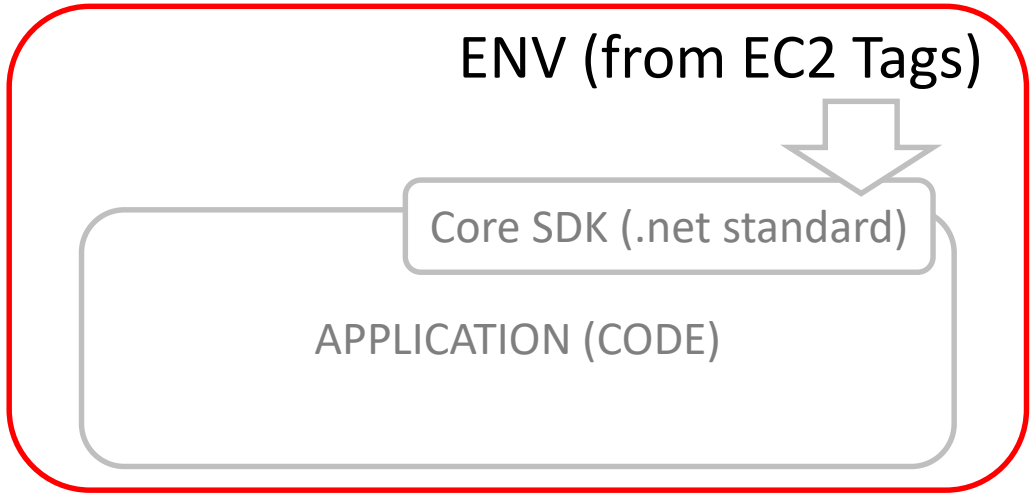
ENV 關閉時

- 回報 SR 註冊移除
- 消化未處理完畢的 Req(s)
- 回報 OS 可正常退出

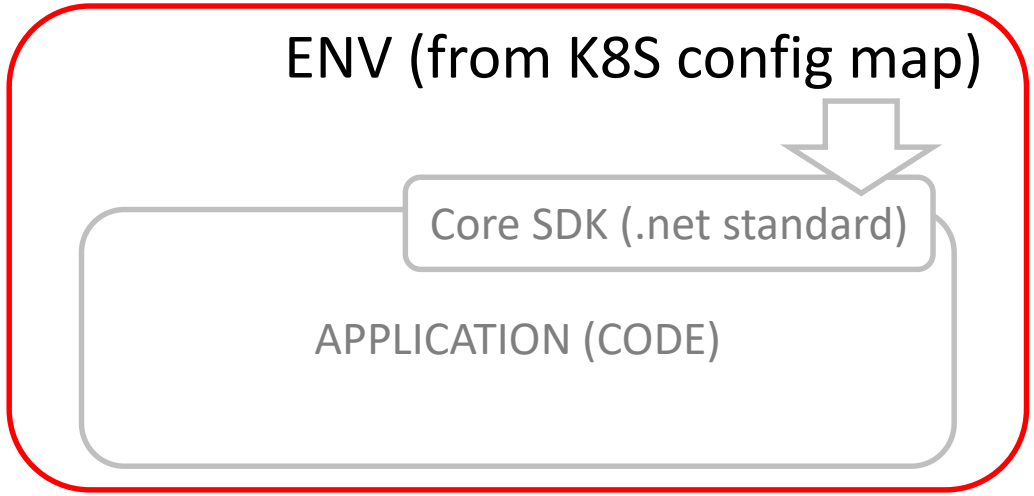
挑戰 #4

最後一理路: Design For Operation, 開發(DEV) 時就要思考該怎麼維運(OPS)。

服務開發規範 #2: Environment Context ...



Virtual Machine (Windows / Linux)

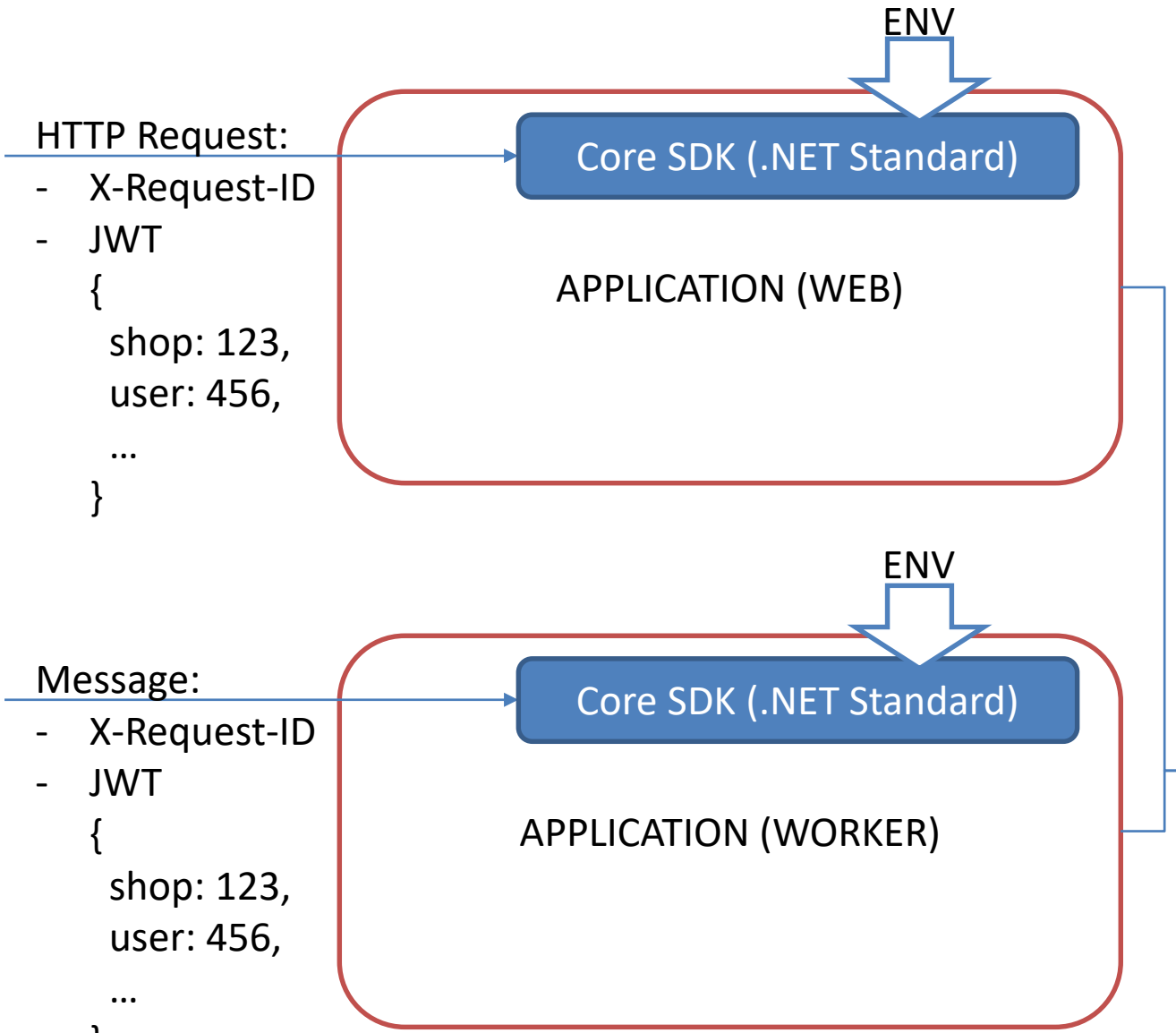


Container (Windows / Linux)

混合環境的挑戰

- **VM / Container**
- **Windows / Linux**
- **.NET Framework / .NET Core**
- **With / Without Kubernetes**

統一一致的環境識別機制，才有一致的監控與紀錄



多租戶的架構

- **Code 如何 偵測 目前的 Req 是哪個客戶?**
- **跨越服務呼叫時，如何正確傳遞 Context?**
- **統一的 Logging**
- **統一的通訊機制與規範**
- **X-Request-ID**

一致的 LOGS 前置欄位定義:

SN	RID	SID	SHOP	USER	…
1	…	…	123	456	…
2	…	…	123	456	…

- **軟體 (code) 必須通用**
- **組態方式必須通用 (ENV)**

- **封裝方式不同 (binary vs container image)**
- **部署方式不同 (install vs docker pull)**
- **執行方式不同 (install as service vs docker run)**

允許團隊逐步由 VM 轉移至 containers (K8S)

DevOps: 將維運也當作需求的來源

DevOps 開發維運一體化的精神：想要怎麼維運，就要怎麼設計。

Question?

Thank You!

Next: 從零開始的 Configuration Management .