Gogolook—全球千萬使用者等級 App 的 DevOps 架構

案例簡介

（一）案例背景

Gogolook（走著瞧股份有限公司）成立於 2012 春天，成員來自臺灣、香港、韓國、巴西等地的軟體創新人才，希望透過快速且不間斷地服務創新以及使用者為中心的設計理念，建立全球行動裝置上最值得信賴的人際溝通網路。

Whoscall 是目前最具代表性的產品，擁有全球超過十億筆電話資料庫資訊，提供 Android 及 iOS 行動裝置使用者最即時的來電辨識服務。Whoscall 於 2012 年獲當時 Google 執行長 Eric Schmidt 公開表揚，並獲數字時代、華人行動應用大賞、Google 年度創新獎等獎項肯定，曾被全球知名科技媒體 TechCrunch、TechinAsia 專題報導，也與臺灣警政機構、韓國金融監督局（FSS）等單位簽訂合作備忘錄。

Whoscall 全球用戶超過六千五百萬，主力市場為臺灣、巴西、香港、日本、韓國等地，可警示惡意與推銷來電，資料庫中也提供豐富的商家營業信息，並已成功的協助數千萬的商家透過來電辨識功能取得行動用戶的信賴。目前亦致力於運用資料庫系統發展人工智慧，藉由長期使用者接收詐騙電話的資料分析模擬詐騙集團的行為，進而預防犯罪。

隨著使用者規模擴大，服務內容增加，對於基礎設施穩定性、研發敏捷性、產品安全性的考驗亦不斷上升。為了適應下一階段的服務挑戰，公司開始進行一連串的 DevOps 大改造。

（二）案例特點

Whoscall 服務的 DevOps 改造案例是依據 DevOps 指導原則逐步進行，流程與工具改造並重，此經驗對其他軟體公司應該很容易借鑒實施，以《鳳凰專案》The Phoenix

Project)「三步工作法」觀點闡述案例特點如下：

- 流動原則：以自動化工具為輔，全面貫通提交源碼、測試、組態配置、佈署等流程。在安全網的基礎上，放心提交源碼進行實驗，加速研發與應變腳步。
- 回饋原則：透過許多層次，透明化監測佈署的任何服務的實際運作狀態，以得到即時且精密的回饋，進而產生對於服務品質及商業目標的洞見。
- 持續學習與實驗原則：系統化改善端到端服務流程，對於服務流程及規格，有宏觀及微觀的掌握。對研發的工程品質，能夠起規範化的積極作用，也有助於加速團隊人才訓練，培育更多即時戰力。

需求分析

Whoscall 核心服務涵蓋行動裝置與雲端系統，為了進行 DevOps 改造，需要技術面與流程面密切配合，故先界定整個改造計畫的總體目標如下：

- Infrastructure as Code：在可能的情況下，應盡力追求配置透明化，利於優化配置、查找問題、經驗傳承，才能進一步演練 chaos engineering 等高級措施；
- 快速佈署：從提交源碼，經由一系列測試、組態配置、佈署程式，需要迅速且穩定的佈署流水線，才能有效支持快速實驗的需求；
- 兼顧安全與彈性：每次佈署都需要通過一系列的編譯期與執行期的檢測基準，確保合規要求；同時也兼顧彈性，能在執行期進行迅速的動態調整；
- 服務監控：針對內部系統及外部依賴系統，需多層次監控以便快速反應、優化服務；
- 端到端服務流程系統化：過去在研發驅動力主導之下，累積了許多技術債，現在已逐漸反噬到研發的步調。因此，從服務頭尾兩端開始系統化檢討整頓服務流程環節，以利於查找問題、累積組織流程資產；

- 透過上述基礎來敏捷地拓展更多服務，發掘更多商業價值。在改造 DevOps 的同時，也持續進行技術選型評估。由於 Gogolook 是小而精的公司，為了集中研發能量，降低導入及學習遷移阻力，降低招聘新人時教育訓練的成本，其技術選型原則有下：

- 避免 lock in，這點非常重要，尤其在技術快速發展、版圖不斷挪移的軟體產業，不被特定工具或廠商 lock in，維持技術選型的自主性，才能敏捷前進；

- 以前瞻眼光選擇活躍的開源軟體，並積極參與線上及線下的社群活動，交流實務經驗；

- 選擇泛用性高、擴充性高的開源軟體，避免技術棧過度碎片化，增加維運基礎設施的困難度；

- 審慎選擇 IaaS/PaaS/SaaS 的服務，善用這些服務，可降低維運基礎設施的精力；但其推陳出新速度，可能沒有對應的開源軟體來得快，這部份需要再加以權衡；

- 若沒有適當的現有組件，則自建，但仍對替代方案保持開放態度。

解決方案

（一）總體技術架構

根據前述技術選型原則，與 DevOps 相關的技術選型如下：

- CI/CD 流水線：Jenkins；
- 行動端平台：Android 及 iOS；
- 行動端 A/B 測試機制：早期自建，近期逐漸改用 Firebase；
- 行動端效能分析：Android vitals、Crashlytics；
- 後端公有雲：Amazon Web Services（AWS）；
- 後端組態管理：Ansible 及 Docker/Kubernetes 生態系；
- 後端效能及日誌分析：New Relic、Sentry、EFK、Prometheus、Grafana；

- 流程與 API 規格介面：Swagger、Cucumber。

（二）具體技術方案

1) Infrastructure as Code（IaC）

- Infrastructure as Code（IaC）是現代從業人員必備的紀律，在可能的情況下，應盡力追求配置透明化、利於優化配置、查找問題、經驗傳承，也才能夠進一步演練 chaos engineering 等高級措施。另外，Ansible 是個泛用性極高的組態管理及遠端執行工具，故選擇以 Ansible 作為 IaC 機制，理由如下：

- 輕量化 IaC 不應有過多的額外開銷（譬如：額外的認證機制、額外的背景程式）。而 Ansible 不需要額外常駐的背景監聽程式，也不需要額外的帳號認證機制，完全沿用 Linux 既有的機制，可降低管理的複雜度；

- 隨插即用：對機器及環境僅有最少的假設。一般來說，只要 Linux 主機有 ssh 及 Python 元件，就能被 Ansible 管理，因此，甚至連 Alpine 這類極輕量的 Linux 系統都能納入列管範圍；

- 易學：架構單純，YAML 配置語法比傳統指令碼語言更簡單、更易複用；

- 易擴充：Ansible 架構單純，可以用正規的 module 形式擴充功能，亦可用陽春的腳本 + stdout/stderr + json 手法擴充，彈性非常大；

- 社群活躍：這個理由尤其在被 RedHat 併購之後，其永續發展更是無虞。

- 後端系統是以 Linux 為主，多半佈署於 AWS，因此以羽量級的 Ansible 為核心的組態配置工具，再搭配其他技術特有的組態配置設施：

- Linux 主機：自行編寫並維護 Ansible roles；

- AWS 系列：用 Ansible 驅動 CloudFormation 等配置機制。

- 新一代的後端技術，多半都直接內建 IaC 機制，如 Docker 生態系，甚至各種

serverless 系列；採用這些技術時，直接套用各自的 IaC 機制即可：

- Docker：用 Dockerfile 及 docker-compose.yml 配置；
- Kubernetes：一堆 YAML 檔案配置。

目前已針對這些組態配置有小規模進行 chaos engineering 的試驗，預計未來會更進一步利用 Netflix 釋出的 Chaos Monkey 系列進行大規模的演練，以確定整個 IaC 配置的完整性。

2) 快速佈署

從提交源碼開始，經由一系列測試、組態配置、佈署程式，都需要迅速且穩定的佈署流水線（deployment pipeline），才能有效支持快速實驗的需求，因此佈署流水線以 Jenkins 為主軸，貫串相關的前端與後端流程：

- Android 源碼→Git→Jenkins→Gradle 腳本；
- iOS 源碼→Git→Jenkins→Xcode 腳本；
- 後端源碼→Git→Jenkins→Ansible 腳本。

重要的提交狀態也都會送到 Slack 及對應的監控儀錶板，讓進度透明化。

在關於行動端軟體的部分，利用 Android 新的 Dynamic Delivery 機制，將行動裝置軟體打包成 App Bundle 格式，各種行動裝置使用者能夠下載最適尺寸的 APK 套裝軟體，也簡化組態配置流程。

後端系統主要採微服務（microservices）架構，將後端服務劃分成更小的獨立佈署單位，每次提交源碼時，只要不涉及重大的架構變動，正常情況下都可不停機持續佈署。

利用前面提到的 Ansible 進行後端服務的打包及佈署步驟。主因 Ansible 是一泛用性極高的組態管理及遠端執行工具，可用一致性的 YAML 語法調用多種不同的技術，減少需要掌握的技術棧：

- 可針對編譯式或直譯式的源碼進行打包、測試、佈署等步驟；
- 可針對本機端、裸機、雲端主機進行組態設定及持續佈署；
- 可針對傳統軟體、Docker 化軟體、serverless 軟體進行持續佈署。

在 AWS 上利用 Ansible 或 Docker 開啟新主機或容器相當簡單，善用這種優勢將後端架構設計成 immutable infrastructure，進一步降低持續佈署的複雜度，也提升後端系統的穩定性。

3) 兼顧安全與彈性

由於與多個公部門有合作關係，為確保品質，每次佈署都會通過一系列的編譯期與執行期的檢測基準；其中自建的檢驗工具委由符合 ISO/IEC 17025 規定的公正第三方定期進行資安檢測。除了上述測試與合規程序外，也須兼顧彈性，確保能在執行期進行迅速的動態調整，系統流程才算完整。

關於行動端軟體的部分則藉由 Google Play Console 的分段回滾(staged rollout) 機制、Apple App Store 的分段發佈 (phased release) 機制，在快速佈署之餘，亦能兼顧風險控制。另外也利用 Firebase Remote Config 及自建的 A/B testing 技術，讓不具軟體研發能力的實驗設計者，也能在不重新佈署行動端軟體的前提下，動態進行多變數的實驗，且同時適用於 Android 及 iOS 兩大平台。

關於後端系統，由於 AWS 提供許多可程式化的動態機制，所以再利 Ansible 或 Docker 進行藍綠佈署(blue/green deployment)或金絲雀佈署(canary deployment) 就相對簡單。隨著 AWS 正式支持 Kubernetes (名為 EKS)，更可直接在儀錶板上進行這方面的操控，降低維運複雜度，提升透明度。

4) 服務監控

內部系統及外部依賴系統需要多層次的監控，以便快速反應、優化服務。故針對行

動端軟體的部分，除了自建的日誌與性能監控機制外，也利用 Crashlytics 服務來監控軟體閃退事件。最近更利用 Android 新的 Vitals 機制，監控行動軟體的種種品質數據：啟動時間、閃退、顯示速度、網路熱點掃描、背景執行行為等等，作為優化產品的依據。後端系統則動用較多層次的監控機制。

- 硬體基礎性能：大部分以 AWS 的 CloudWatch 服務來監控雲端主機群的硬體基礎性能資料。如需更細緻或更廣泛的性能資料，則透過 Prometheus 生態系提供的一系列 exporter；
- 日誌：一般情況下以 AWS 的 CloudWatch 服務來搜集與彙整 layer 7 的日誌，但如需更即時、更細緻、更有彈性的日誌處理，則透過 EFK (Elasticsearch + Fluentd + Kibana) 技術棧來達成；
- 軟體棧性能：以 New Relic 服務掌握軟體在作業系統層次、執行引擎層次、用戶端層次等的分層效能資訊；
- 執行期錯誤：採 Sentry 服務全盤掌握應用程式動態拋出的執行期錯誤；
- 外部健康狀況：採 Pingdom 服務從全球使用者視角理解己方或第三方廠商 API 服務的回應時間與健康狀況；
- 上述的後端系統監控機制多針對特定環節，缺少綜合性、全域性、多層次的視角，因此另外再透過 Prometheus + Grafana 系統，取得以下更多效果：
- 自訂儀錶板：透過 Grafana 針對各服務、各流程所關心的視角，設計儀錶板及統計資訊呈現方式；
- 自訂時間序列運算式：透過 Prometheus 的 PromQL 設計合適的時間序列運算式，精準呈現出關鍵的服務運作指標及趨勢曲線。

5) 端到端服務流程系統化

因過去研發驅動累積的技術債反噬到研發步調，故從服務頭尾兩端系統化地整頓服務流程環節提升整體品質，利於查找問題，累積組織流程資產，逐步進行改革如下：

- 介面思考

多人協作團隊甚至與第三方合作互聯，都非常需要制定良好的技術介面，才能促成順暢合作。為了加速前端後端的並行開發，在研發團隊小規模導入 Swagger 及 RAML，隨著 OpenAPI Specification (OAS) 大勢底定，相關工具逐漸豐富，開始更大規模實施，也推廣到品質監管等職能身上，讓感興趣的人可透過技術介面，理解並實驗相關的後端服務，進而降低教育訓練的成本。這是很重要的基礎，它可以改善後端系統的介面透明度，進一步優化服務流程。

- 流程思考

隨著使用者規模擴大，服務內容增加，流程也漸趨紊亂，因此，開始以系統化角度，重整過去大大小小的服務流程與資料流程，採取以下措施提升服務流程的透明度：

盤點：首先組成特別小組，針對既存的所有服務，進行第一次廣泛的總盤點，先不求深，但力求廣泛無遺漏。

規格表述：由跨職能小組針對高優先度的專案，以類似 BPMN 業務流程模型和標記法（Business Process Management Notation）的形式，深度分析服務的業務與資料邏輯，除做精準表述外，也藉此機會找出可優化流程。

數據埋點：有了透明的流程表述之後，就能進一步在關鍵處埋點，以利於追蹤每一個內部環節的即時狀況。

儀錶板：有以上基礎便可針對各服務、各流程所關心的視角，設計各自的儀錶板，呈現各自的進度狀況，甚至預測未來的走勢，及早預警。

- 規格思考

可執行的軟體規格一直是軟體圈的夢想。以 DevOps 角度而言，MDA（model-driven architecture）之類的軟體塑模作法不見得是最優先的目標，但 SBE（Specification By Example）則有機會以最小的投入產生最大的串連效益：

代表性的測試案例：即使團隊不想轉變成 TDD（Test-Driven Development）或 BDD（Behavior-Driven Development）的研發風格，SBE 仍然可以從端對端測試案例的角度切入，不大幅改變工作習慣，即可從多方面對整個 DevOps 有所貢獻 。

測試自動化：搭配適當的工具可將 SBE 搭上自動測試、回歸測試的流程。

規格：Cucumber 倡議的 GWT（Given-When-Then）語法是一種結構化的規格表述形式，可應用在端到端的黑箱層次，也可應用在 API 服務或高階模組的灰箱、白箱層次。

將上述特性結合起來，SBE 規格就能成為活檔（living document），不僅捕捉業務邏輯，也透過測試自動化機制確保 SBE 檔如實反映最新現況。上述三項措施若實施到位整合就會形成多層次的系統化 DevOps 視角，以端到端的業務服務流程視角來說，BPMN 這類形式可展現出巨觀的流程透明度，SBE 展現出微觀的透明度，且兩者都以非技術人員能夠理解的方式呈現，有助於形成團隊內的共同語言。

以內部技術介面視角而言，Swagger 提供精準的技術語彙，讓稍具技術介面觀念的人，能夠自行掌握介面資訊，並自行做一些操作實驗。

以內部品質管制的視角來說，SBE 搭配測試自動化機制，能降低端到端黑箱或模組白箱的反覆測試人力，進而釋出精力進行更具開創性的探索性測試等品質管制方法。

以內部知識管理的視角來說，上述三項措施，都在累積組織流程資產，減少盲目試誤與爭論的精力。

總結

1) 維運效益

本案例以「三步工作法」相關措施增進的維運效益大致如下：

- 流動原則：Infrastructure as Code 及兼顧安全與彈性的快速佈署程式，積極面在提供催化實驗與緊急應變的溫床；而消極面在提供資訊系統的安全網，避免常見的人為錯誤；
- 回饋原則：逐漸改善服務監控機制就能透過多層次，透明化監測佈署的任何服務的實際運作狀態，作為進一步優化系統效能的根據；
- 持續學習與實驗原則：端到端服務流程的透明度愈高，愈能從流程面發掘更多優化的可能，增進整體服務的效率及可靠度。

2) 研發效益

本案例措施增進的研發效益如下：

- 流動原則：當逐漸改善 Infrastructure as Code 機制之後，研發者免於瑣碎的組態設定問題，得以集中精力在研發任務上，當實施兼顧安全與彈性的快速佈署程式後，研發者可在安全網的基礎上，放心提交源碼進行實驗，加速研發與應變腳步；
- 回饋原則：當逐漸改善服務監控機制，就能透過多層次透明化監測佈署的任何服務的實際運作狀態，研發者可得到即時且精密的回饋，進而產生對於商業目標的洞見；
- 持續學習與實驗原則：當逐漸系統化改善端到端服務流程，研發者對於服務流程及規格可有巨觀及微觀的掌握，對於研發工程品質，能夠起規範化的積極作用，也有助於加速團隊內人才的訓練，培育出更多的即時戰力。

3) 小結

隨著使用者規模擴大，服務內容增加，對於基礎設施穩定性、研發敏捷性、產品安

全性的考驗亦不斷上升，內部進行了一連串 DevOps 改造，以常見的 CALMS（Culture, Automation, Lean, Measurement, Sharing）模型來說，改造不僅在技術面運用許多自動化工具，亦著重流程面的改良及文化面的孵育，務求迅速且安全的流動、即時且多層次的回饋、充分支援持續學習與實驗。

本案例從 Whoscall 的服務背景開始探討，分析 DevOps 改造需求，再進一步逐一探討技術選型思路，並提供具體的建議，這樣改造沒有終止，各種議題也仍持續嘗試反思調整，期以此文共同探索更好的 DevOps 之道。