從混亂到有效治理
Policy as Code
在跨團隊 CI/CD 的實踐與應用

DevOpsDays Taipei 2025

吳明倫 (Allen Wu)

- 玉山銀行 / 智能金融處 / DevOps Engineer
- Centralized CI/CD / DevOps 推廣
- Obsidian 狂熱者
- Blog : Byte and Ink

演講經歷

- HWDC 2024
  Centralized CI/CD 策略的探索與實踐
- DevOpsDays Taipei 2024
  以開發者資料驅動的 CI/CD 優化策略
- DevOpsDays Taipei 2023
  以產品思維驅動的 DevOps 策略：金融業 AI 團隊的實踐之旅

Blog

智能金融處：一群讓資料活起來的人

Platform User

Machine Learning Engineer

Platform Team

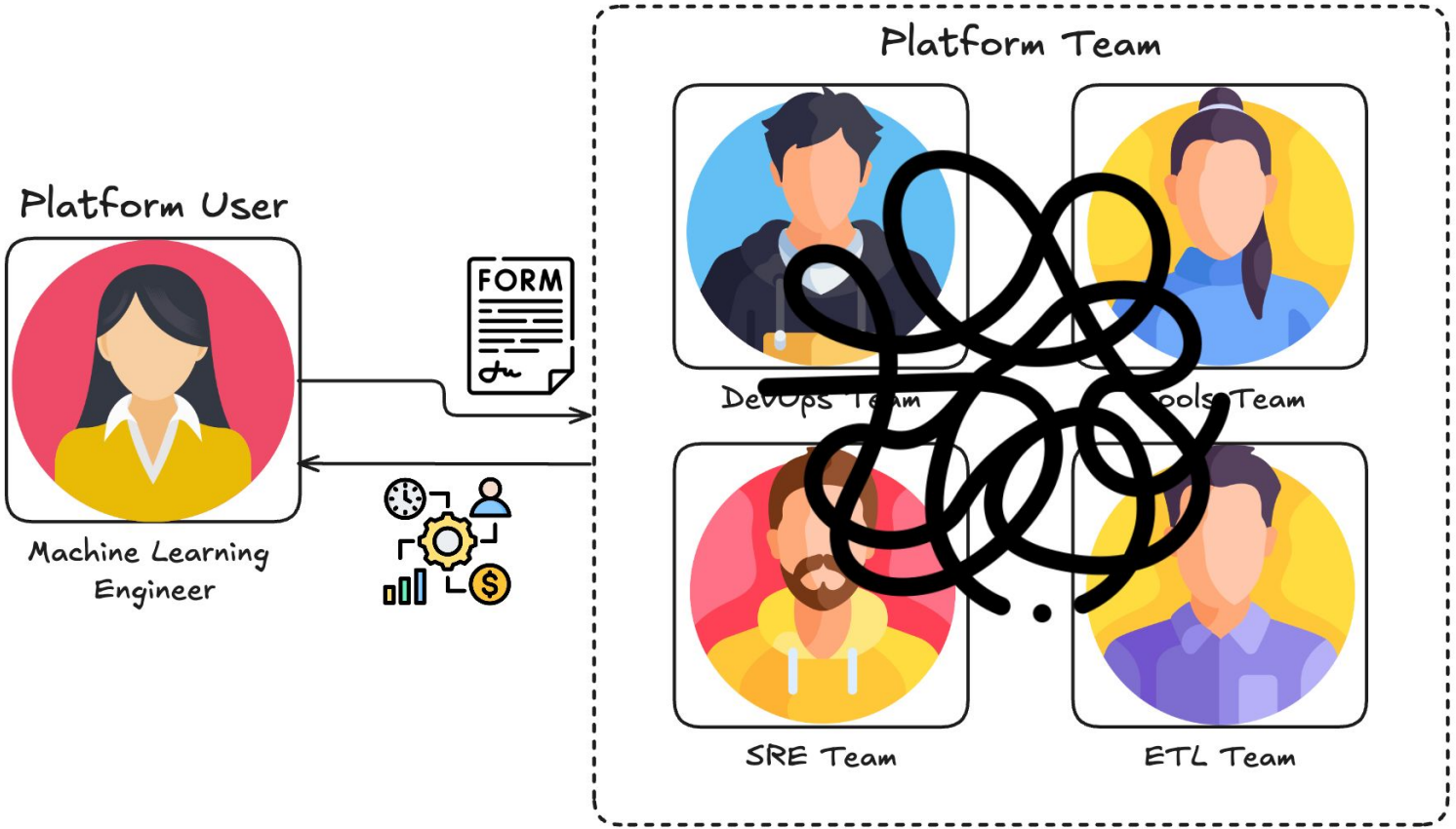
DevOps Team

Tools Team

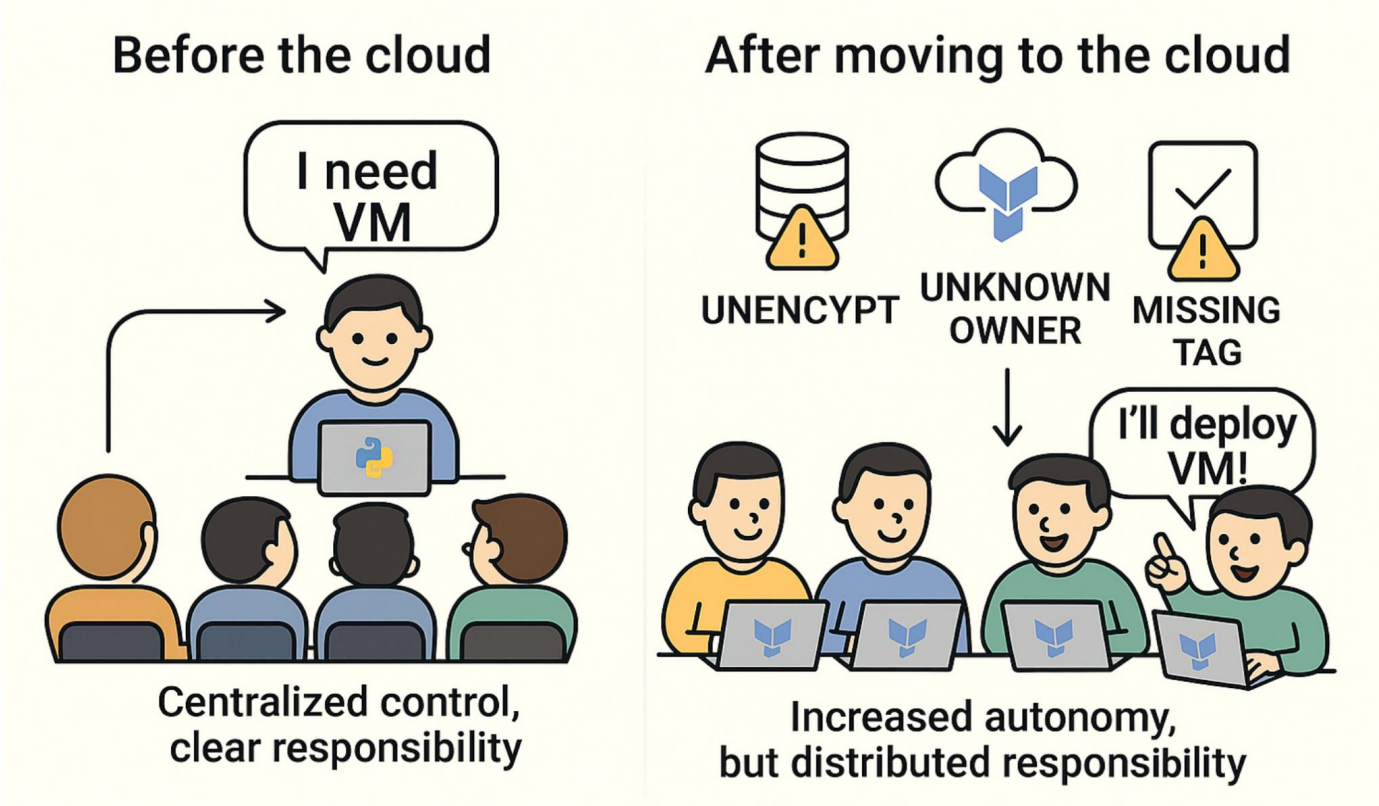
SRE Team

ETL Team

地端思維：沒有建資源的自由
不需承擔治理的責任

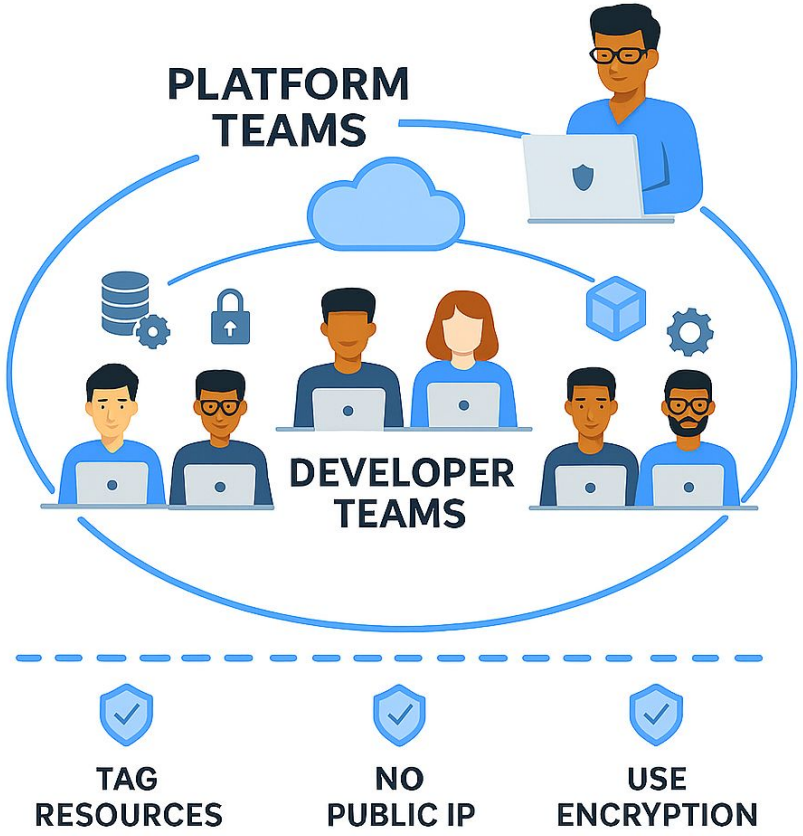


上雲讓開發者有更高的自主性
也把治理變成每個人的責任



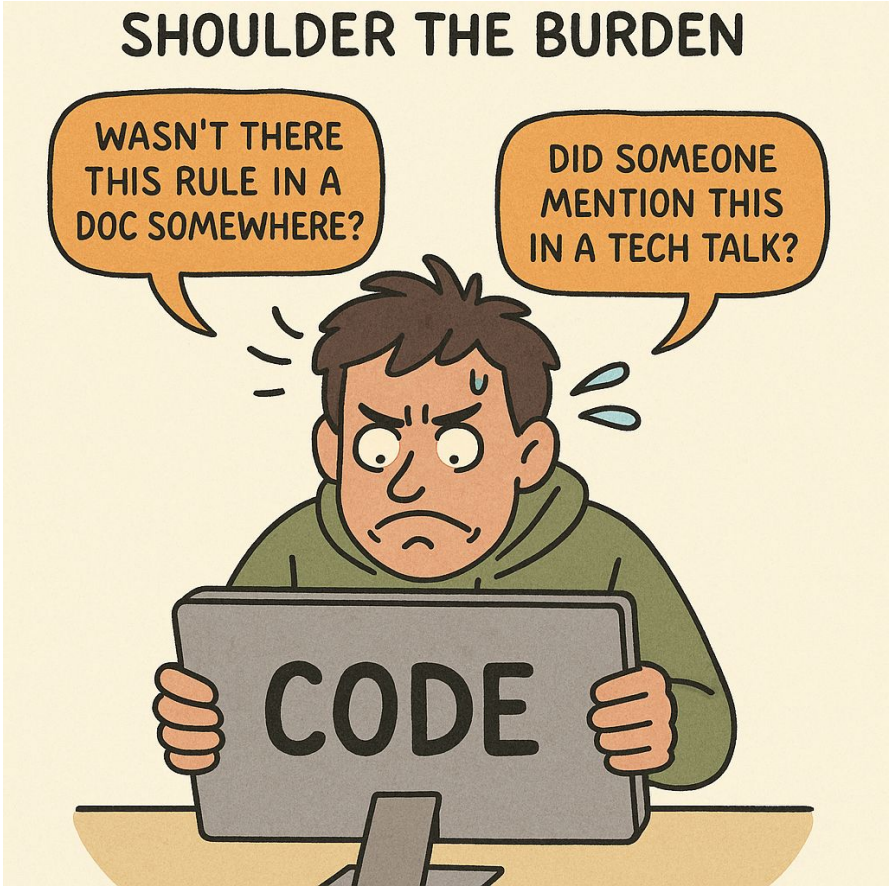
自由沒有邊界，錯誤及責任無從追蹤

自由擴張後，更需要清楚可見的邊界



各團隊將規範收攏在不同文件中

Reviewer 的無力與模糊地帶



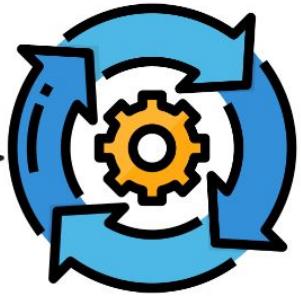
當基礎建設去中心化

治理不能再靠「記憶」及「默契」

PaC 把共識變成程式碼，把責任從人交給流程

檢查主體

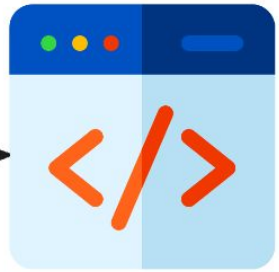
Senior Dev
/ Tech Lead



機器

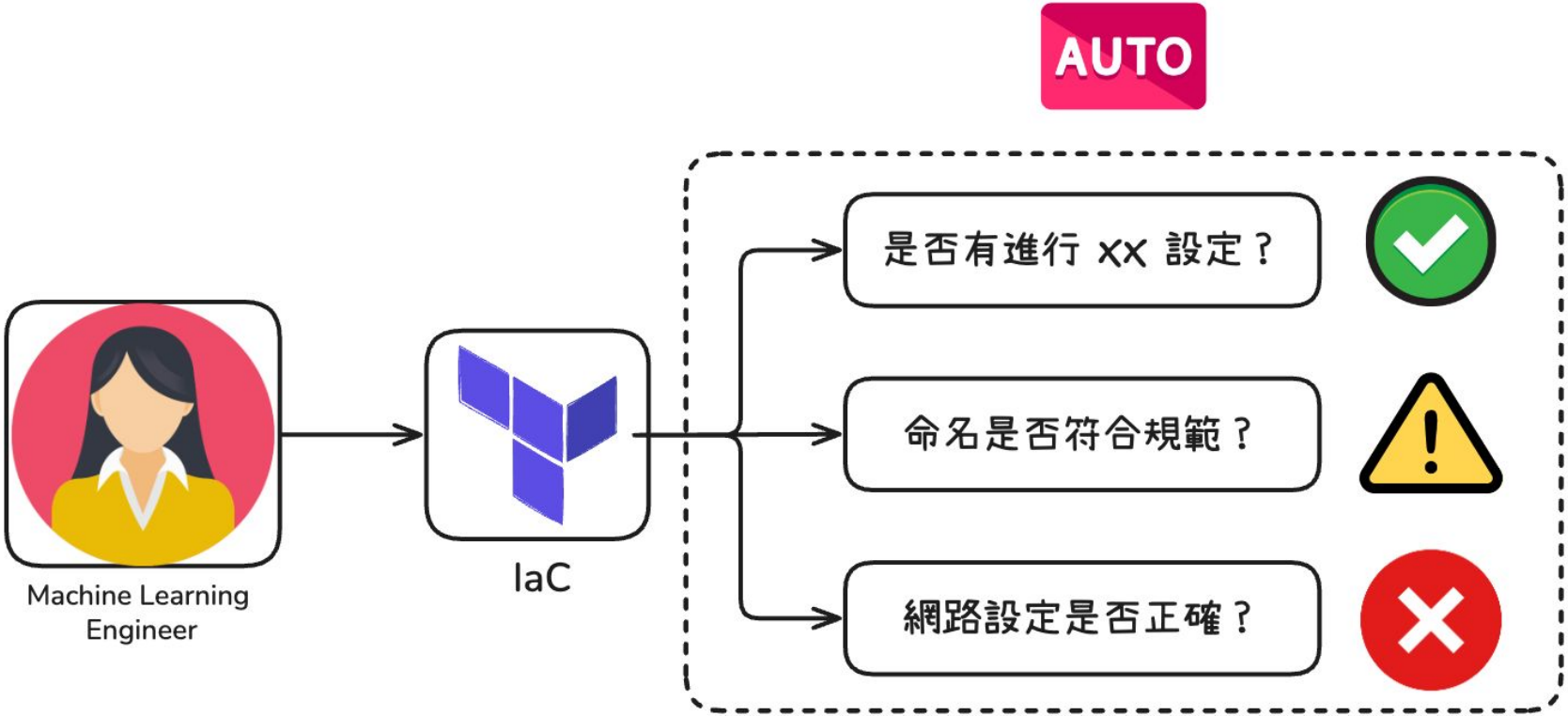
規範形式

會議記錄/文件



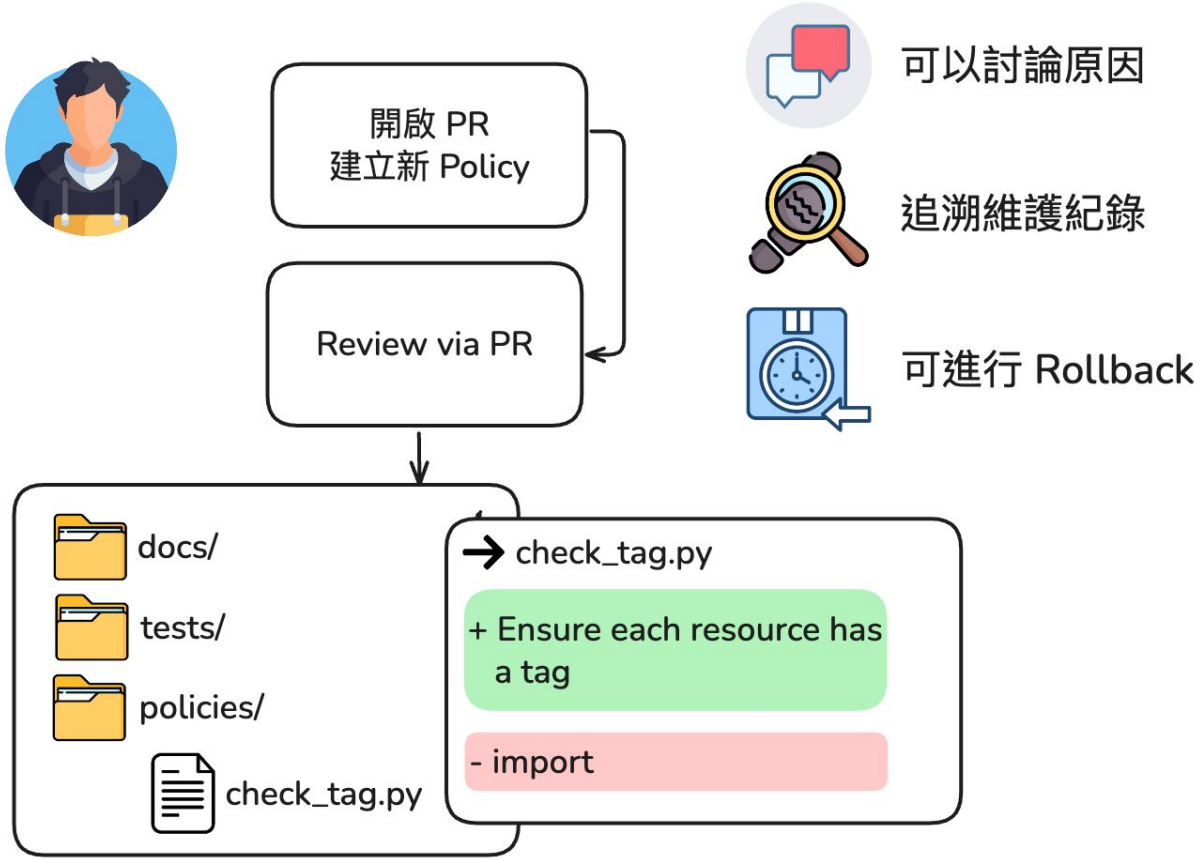
程式碼

規範寫下來沒有用，除非它能被自動執行

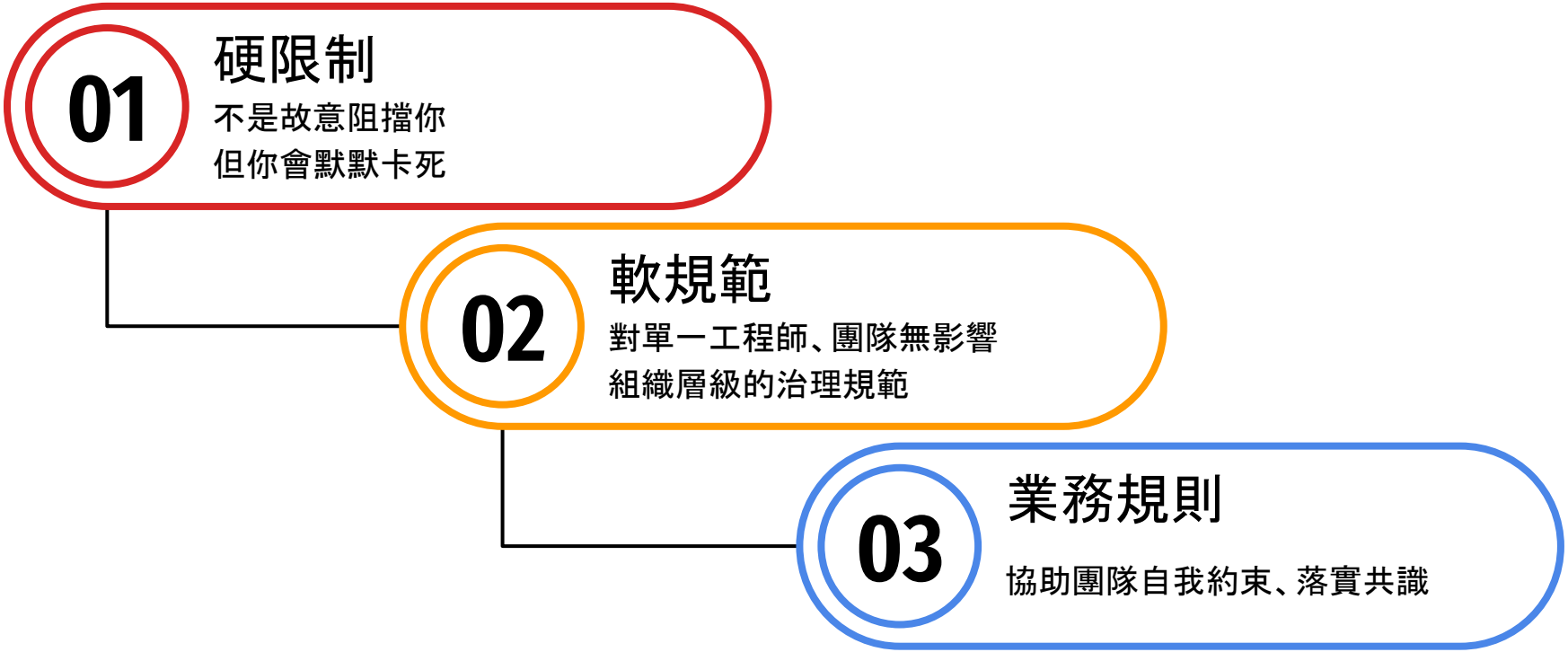


規範不能只被寫下來，還要能被「維護」

用 Git 管理規範，就像管理程式碼一樣自然



PaC 不只是寫規則，而是幫組織守住這三個死角



硬限制 - 部署成功 ≠ 真的能用

“資源建好了，但下一步你根本走不下去”

這些限制通常藏在底層 infra，像是 Org Policy、IAM、VPC 防火牆，
踩到會直接斷線、權限被拒、無法連網——但在部署過程完全沒人提醒你。

啟動 VM 後
發現不能裝套件
(Proxy 沒有開通白名單)

IAM 設定錯誤
API Call
全部失敗

軟規範 - 對單一工程師無感，整體治理影響巨大

“命名、標籤這種小事，最後都會變成大事”

這些規則看似無關緊要，但當你想查帳單、維運、清查責任時，
才發現資源亂七八糟、標籤缺一堆，沒人知道哪台是誰建的。

沒有適當的 Tag
帳單無法分攤

Bucket 命名不一致
自動化報表失效

業務規則 - 靠共識沒問題，只是會忘記

“團隊默契不能只靠記憶，它需要被執行。”

這類規則不是由組織上層規定，而是特定服務、角色、情境下的**自我規範**。
PaC 能幫助這些團隊**自我約束**與**落實共識**。

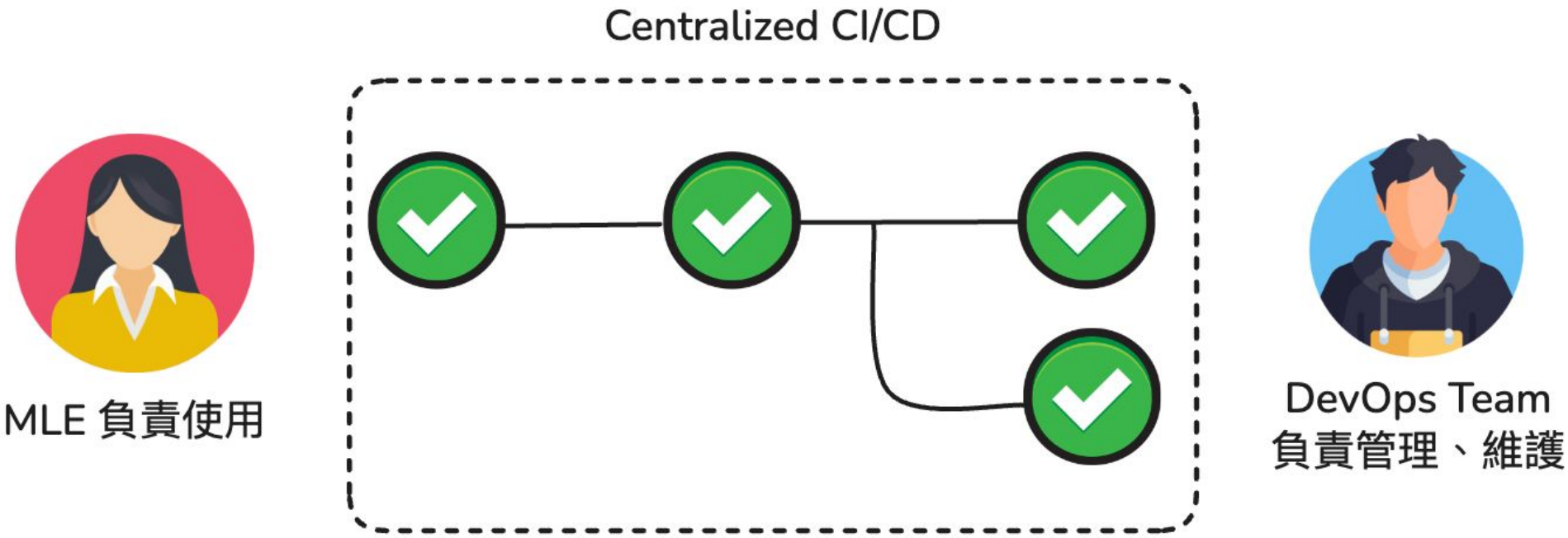
資料表沒有註記
來源 Label

模型訓練任務要標註
dataset version

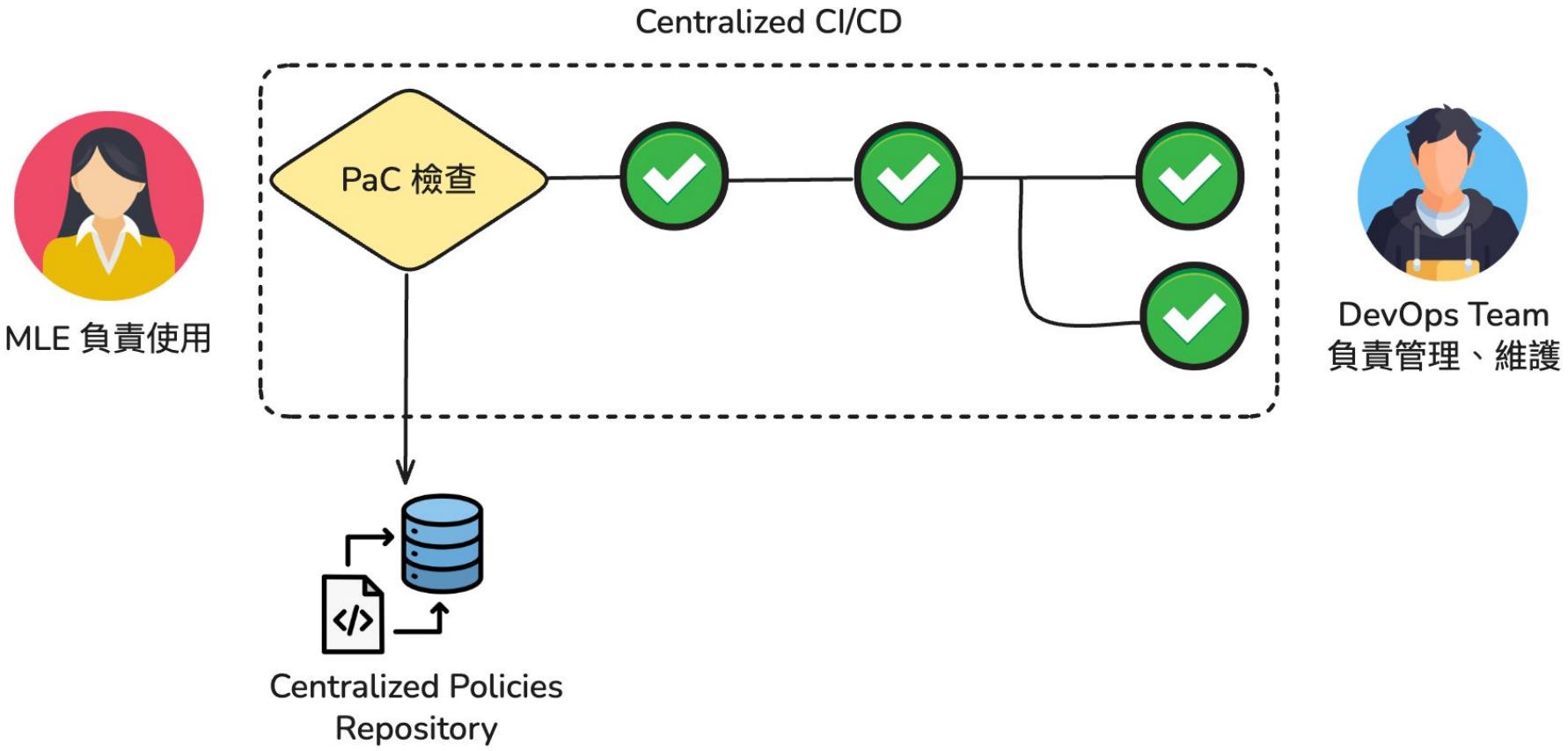
類型不難區分，真正的難題是：

「怎麼落地？」

導入前 - 穩定的 Centralized CI/CD



我們試著讓治理往左走了一步



我們是怎麼找到第一批 Policy 的？

借力社群

許多 PaC 工具皆提供社群貢獻的 Policy

訪談內部治理角色

尋求 Platform、SRE、管理小組的痛點

CI/CD Observability

從 CI/CD 紀錄中探索失敗場景
(以開發者資料驅動的 CI/CD 優化策略)

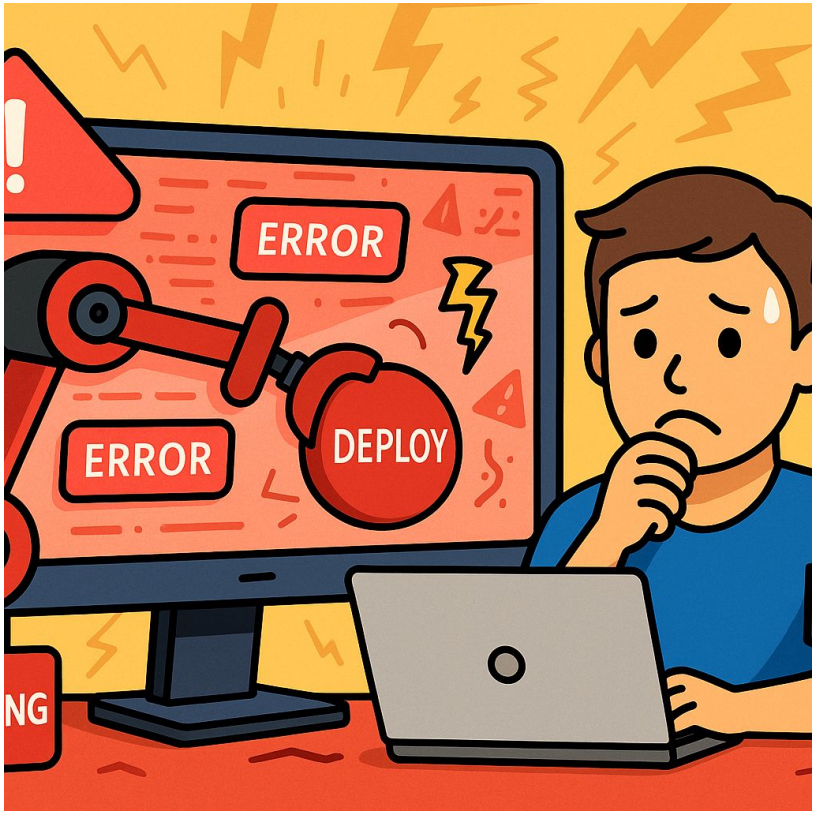
找到 Policy 然後呢？

怎麼讓大家開始接受它 ？

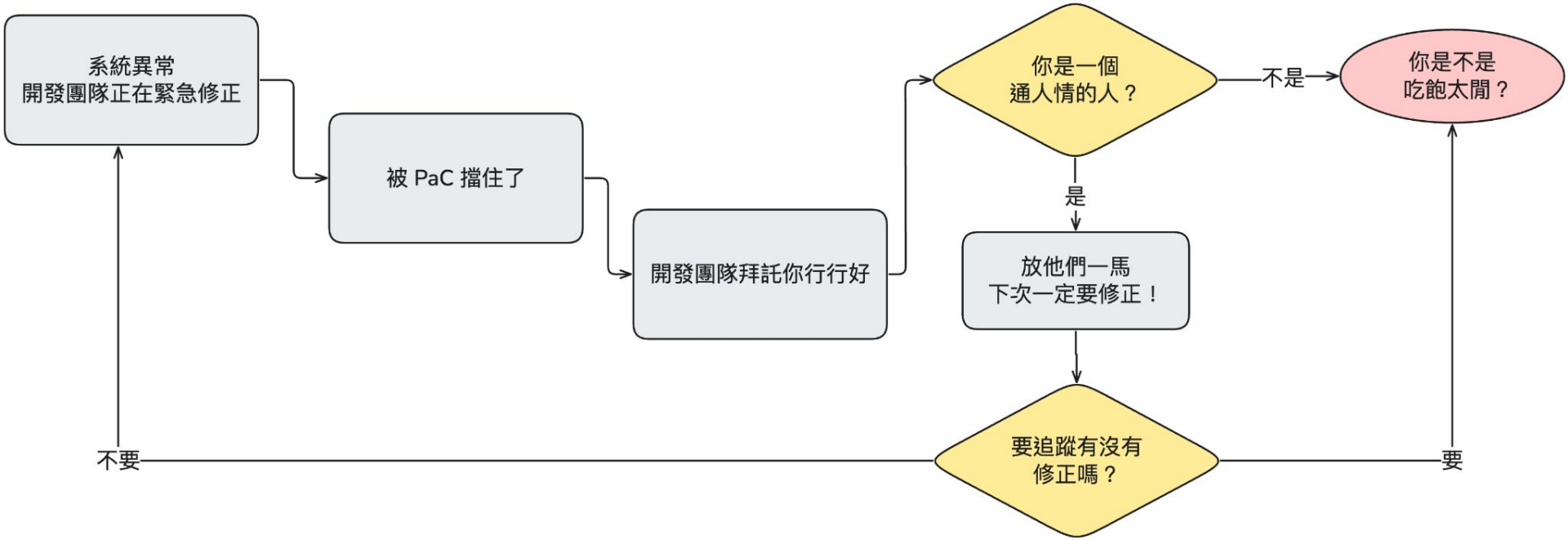
治理不該是突如其來，而是水到渠成

探索期	重點不是套用規則，而是建立「能被觀察的機制」。 目標：**打造出一個能收集、能驗證的 PaC 原型。**	
無感期	掃描運作中，但不阻擋也不提醒。 觀察：**這些規則真的能反映組織的真實行為嗎？**	
適應期	錯誤開始被提示，但部署流程不會中斷。 **引導開發者習慣錯誤提示，從排斥走向認識。**	
落實期	規範正式上線，流程開始 enforce。 **規則不再只是建議，而是流程的一部分。**	

這功能沒有不好，但我現在就是要救火啊！



Platform Team 也想當個好人

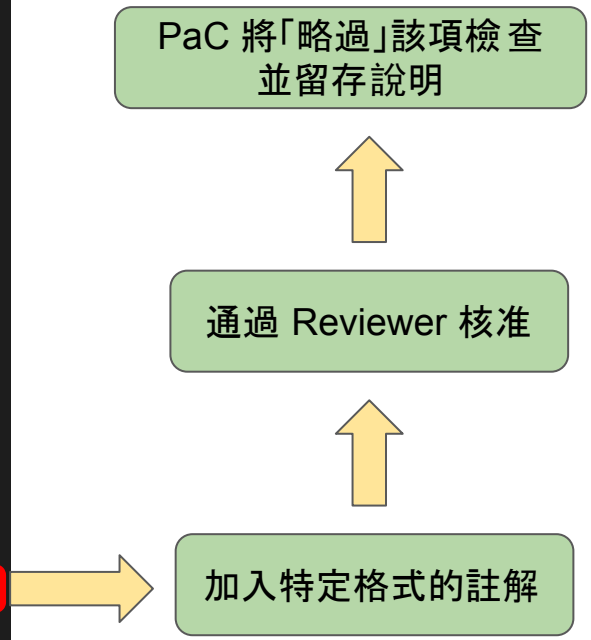


Suppress 是治理成功落地的關鍵機制

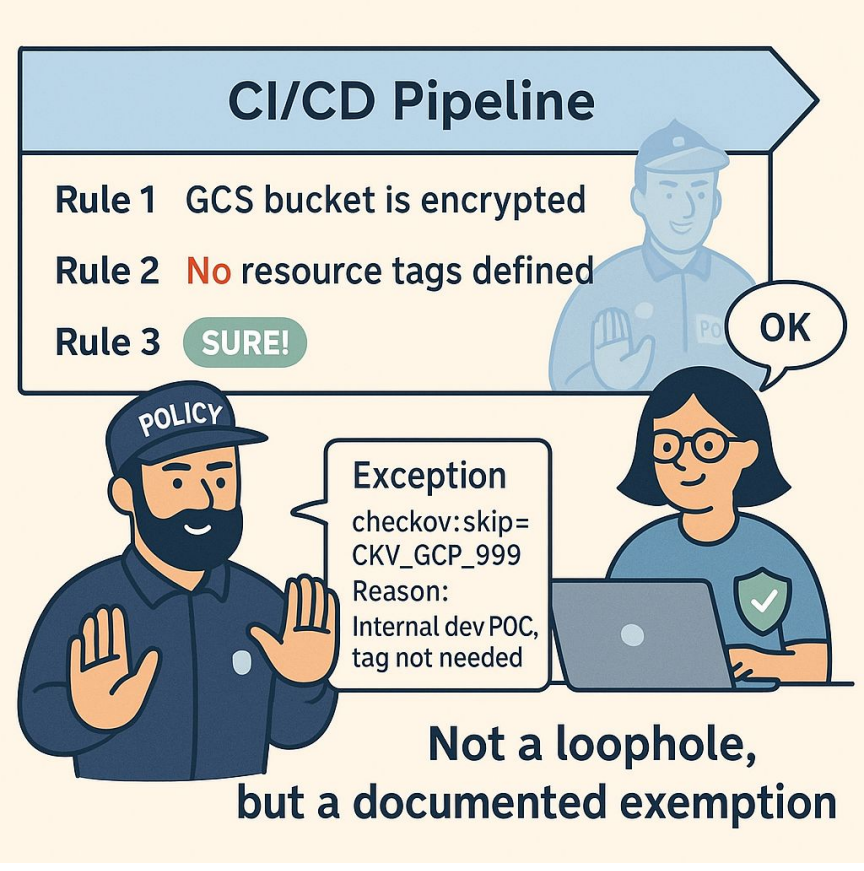
```
resource "google_compute_instance" "demo_instance" {
    name         = "demo-instance"
    machine_type = "n1-standard-1"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-9"
        }
    }

    network_interface {
        network = "default"
        access_config {
            nat_ip = "34.123.45.67"
        }
    }
    metadata = {
        # checkov:skip=CKV_GCP_32 Suppressing policy violation for block-project-ssh-keys
        block-project-ssh-keys = "false" // This violates Checkov policy CKV_GCP_32
    }
}
```



規則真的不能破例嗎？可以，但要留下紀錄



Suppress 為什麼重要？

沒有 Suppress	有 Suppress
報錯後直接擋下，沒有彈性	可由 Developer 註明原因 Reviewer 同意後，壓制警告
開發者可能繞過流程 不信任工具	建立透明紀錄 有紀錄、有責任
沒有人敢導入新規則	迭代，提出後仍有機會調整

哪裡被 Suppress，哪裡就有 Feedback 的機會

- 當**一個團隊**使用 Suppress 機制，是**例外管理**。
- 當**很多團隊**使用 Suppress 同一條規則，是**審視規則**的訊號。
- Suppress 讓 policy 成為「雙向對話」的開始，而不是上對下管理。

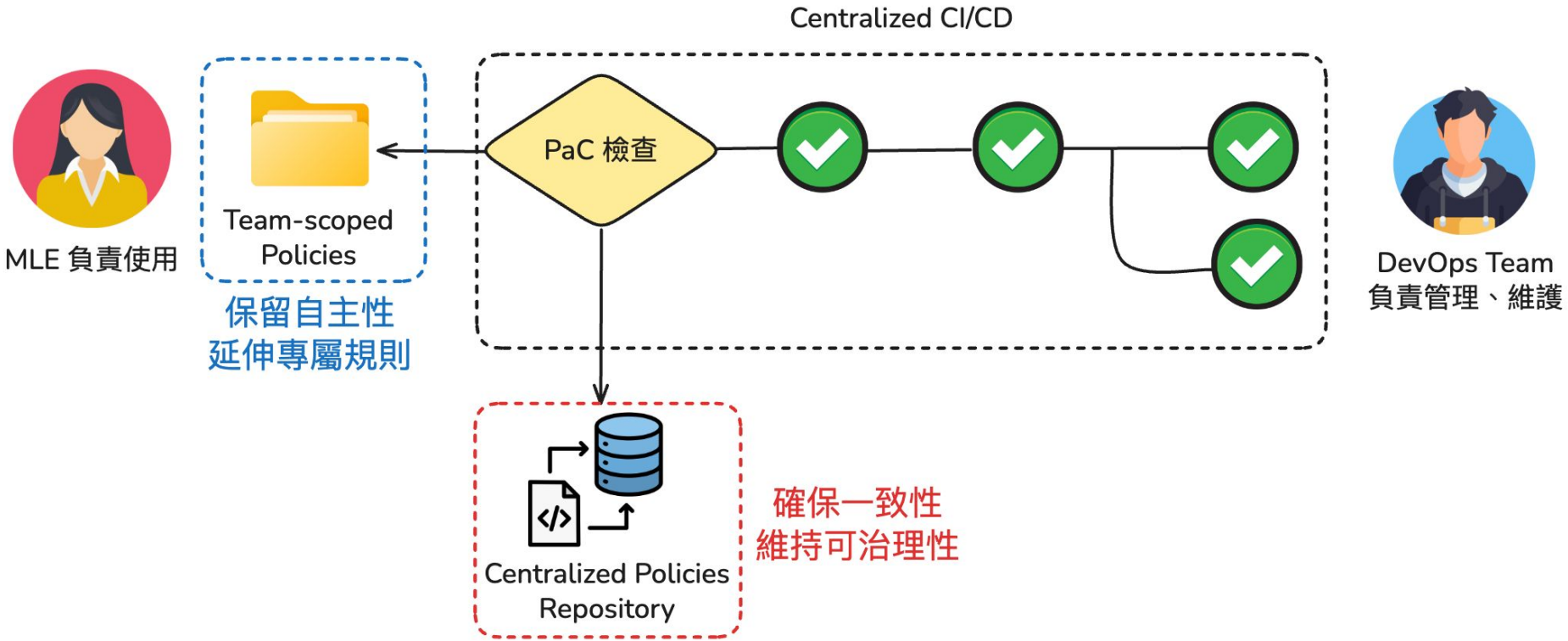
回頭來看，PaC 如何解決不同類的規範問題？

規範類型	違反會怎樣？	PaC 帶來的價值是什麼？	是否有解決？
硬限制	用不了、卡住	預先提示錯誤 減少 Trial & Error	**V**
軟規範	管理成本激增	強化組織治理 讓資源能被追蹤及歸屬	**V**
業務規則	文化難以落實	自我約束可被程式化 變成實際機制	**？**

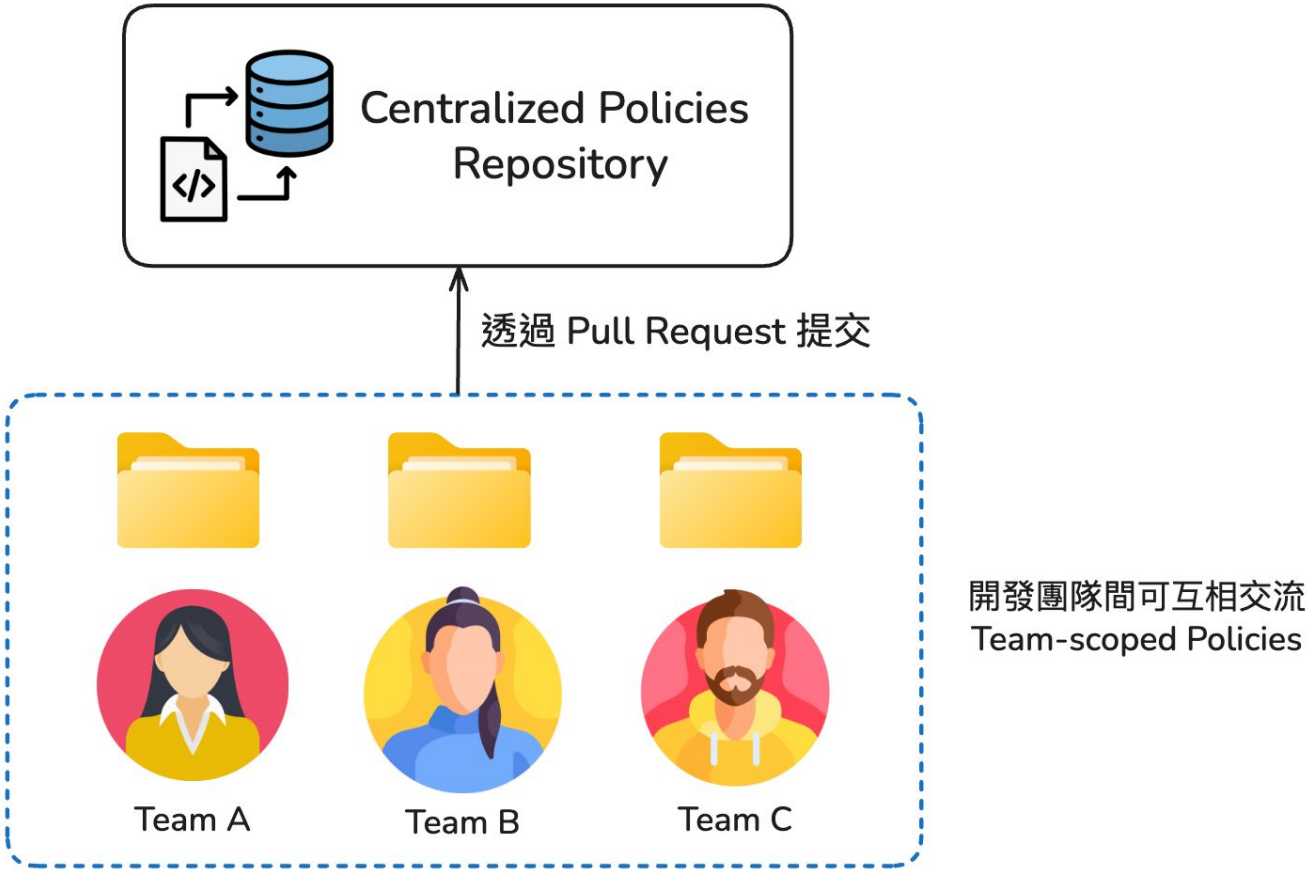
PaC 不是僵硬的管制工具

是允許 Local Extend 的機制

Platform Team 負責最小底線
Developer 負責進一步價值



從實務中升級：讓團隊經驗變成組織知識



眾多 PaC 工具，為什麼我們選擇 **Checkov**？

有很多 PaC 工具：

- OPA
- Terraform Sentinel
- KICS

Checkov 可以透過 **Python** , YAML, JSON 寫規則

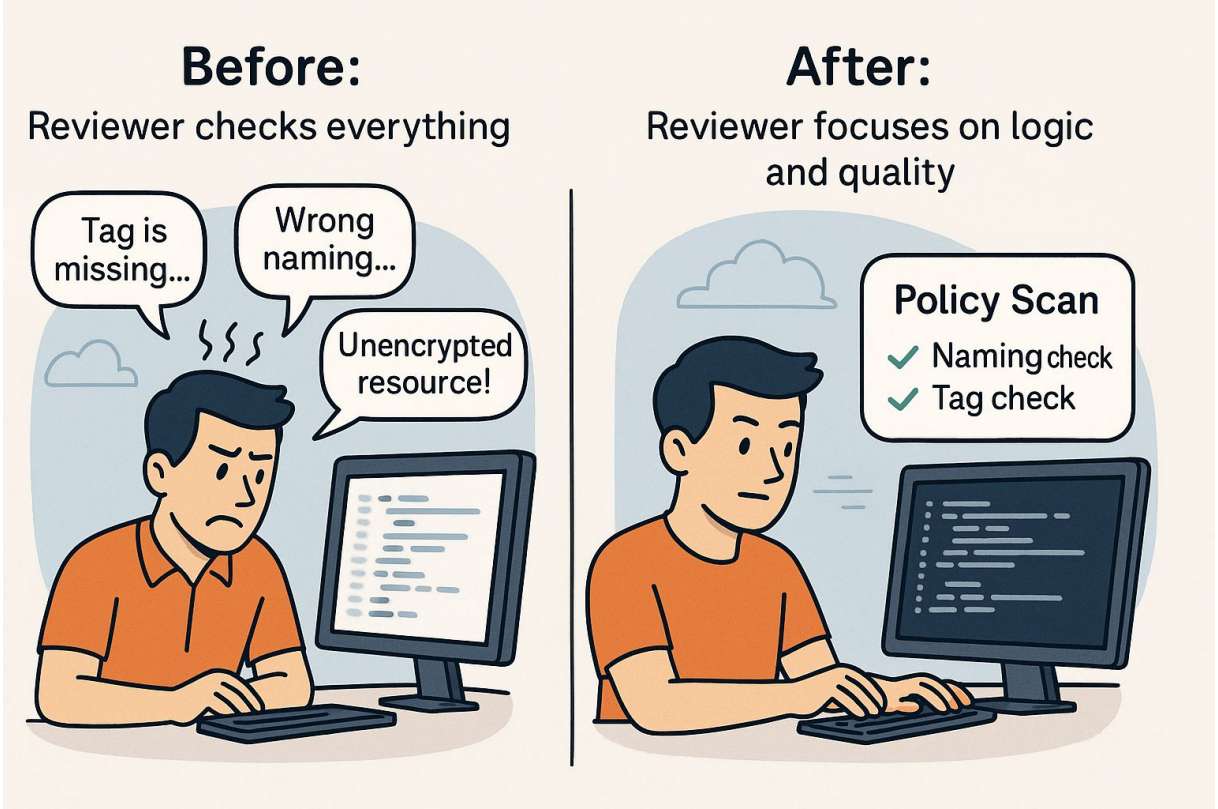
對於熟悉 **Python** 的 MLE 來說，降低了門檻，也更容易將 PaC 實踐到各團隊中。

不只是因為它好用，而是我們希望讓規範能夠被「**共同維護**」。

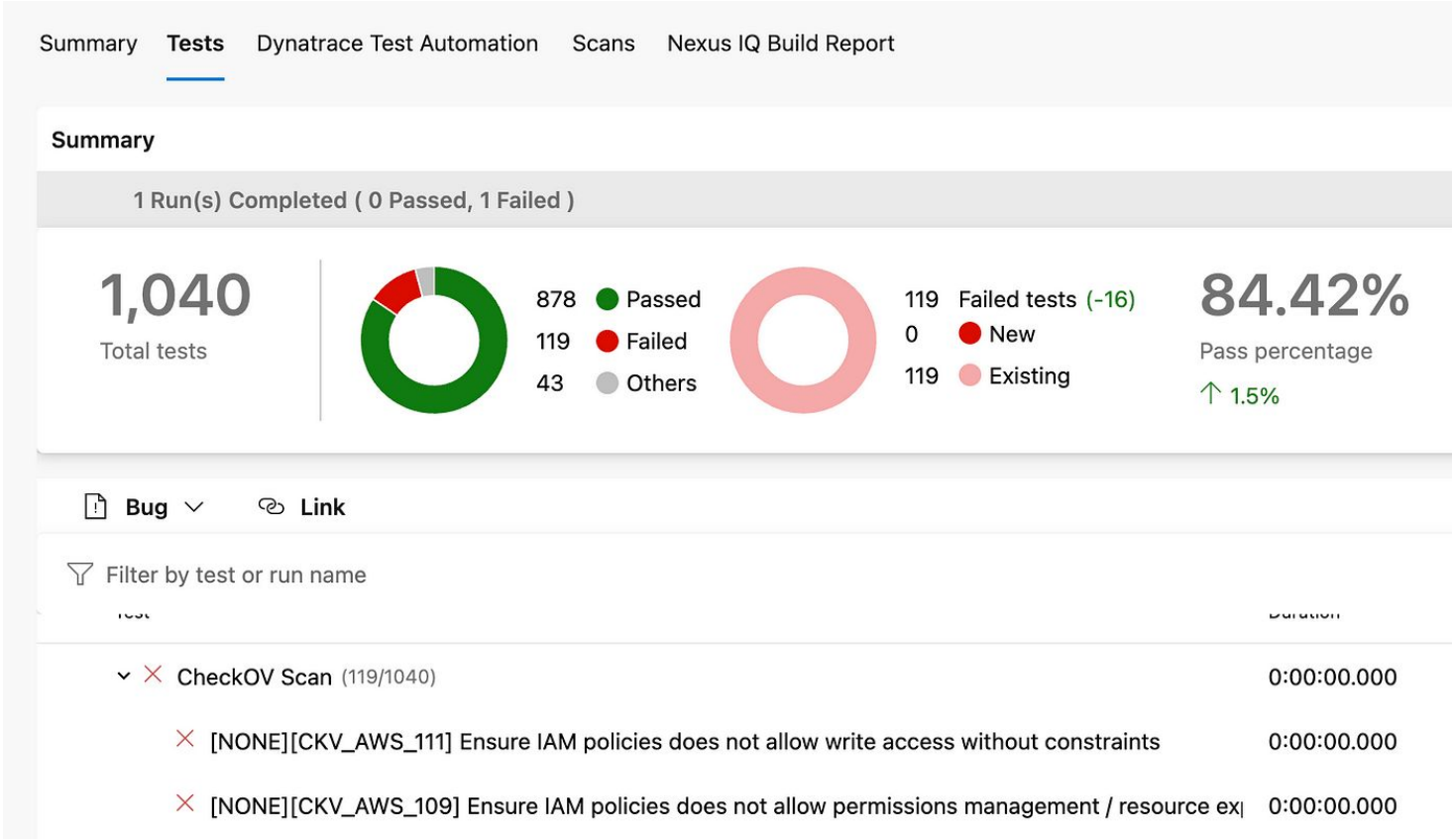
不是最強大、最高階的語言，而是大家都能懂，願意動手修改的語言。

Policy as Code 帶來的效益是什麼？

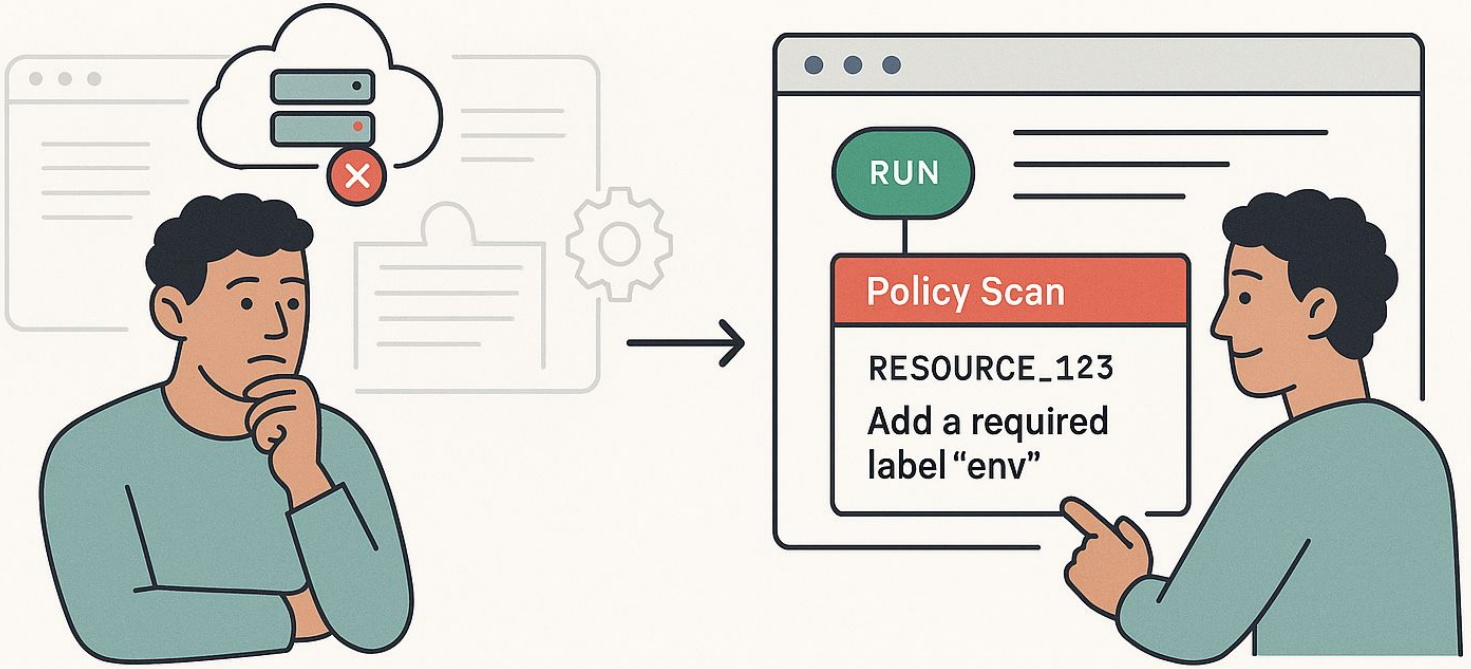
Reviewer 負擔下降



Developer Experience 提升



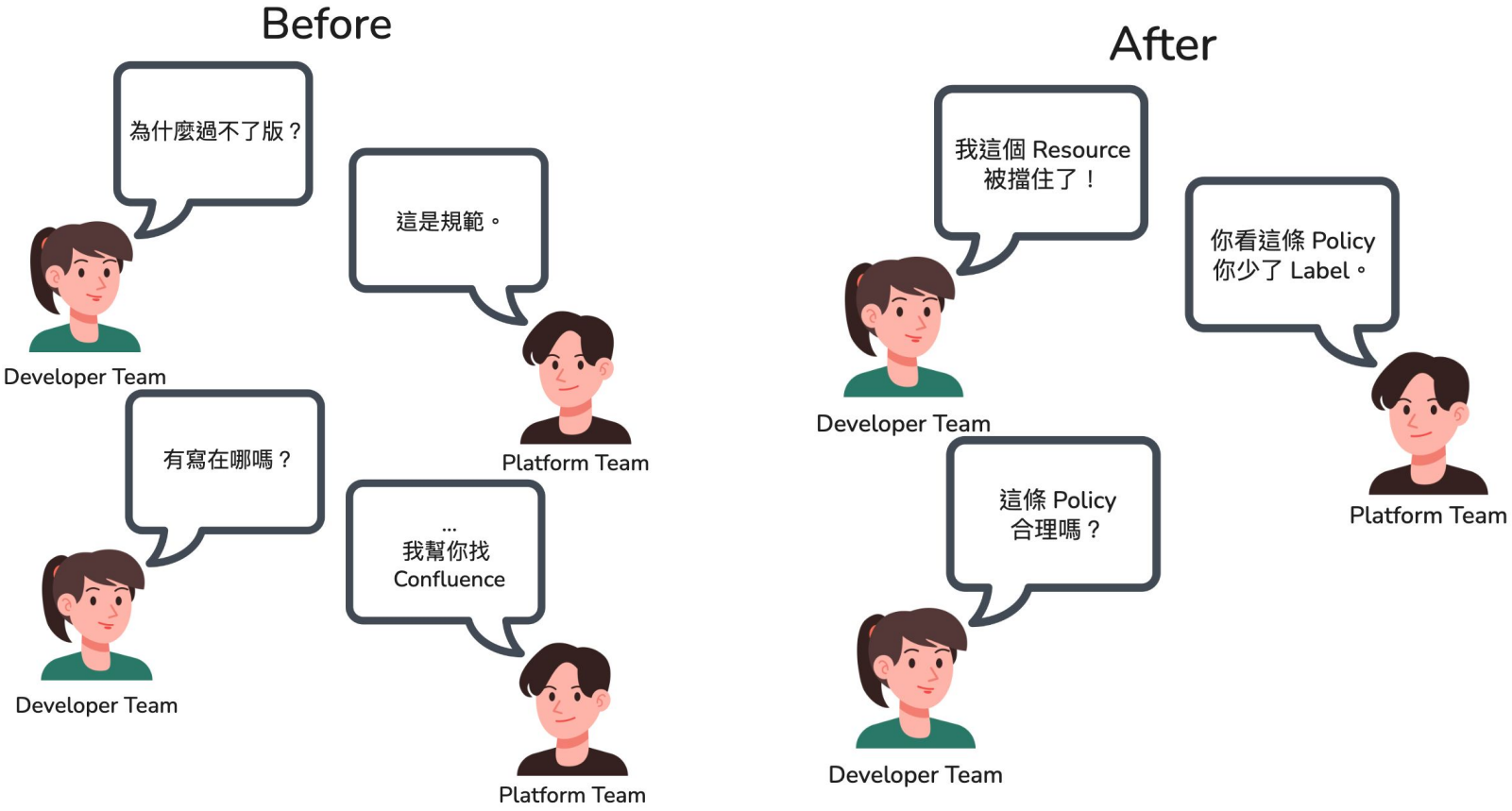
Improved Developer Experience



Before, finding issues was a guessing game

Now, issues are highlighted during CI/CD

溝通及維運成本下降


Before
為什麼過不了版？
Developer Team
這是規範。
Platform Team
有寫在哪嗎？
Developer Team
...
我幫你找
Confluence
Platform Team
After
我這個 Resource
被擋住了！
Developer Team
你看這條 Policy
你少了 Label。
Platform Team
這條 Policy
合理嗎？
Developer Team

比較

過去的規範管理方式	導入 PaC 後
規範四散在 Confluence	規範在版本控制下的程式碼
不知道是誰寫的？ 什麼時候訂的？	Git history 清楚追蹤來源及時間
Review 時要靠印象和記憶	CI/CD 自動提醒 Reviewer 關注核心邏輯
對新進成員來說難以理解	學習曲線較低

Takeaway

治理不是限制，而是讓好規則被實踐

- 好的規則，是**大家都認同卻總會忘記的那些事** 。
- 用 PaC 落實痛點，讓規則變得說得清、改得動、執行得下去。
- 可協作、可版本化、可回滾的機制，才是真正能走長遠的治理。

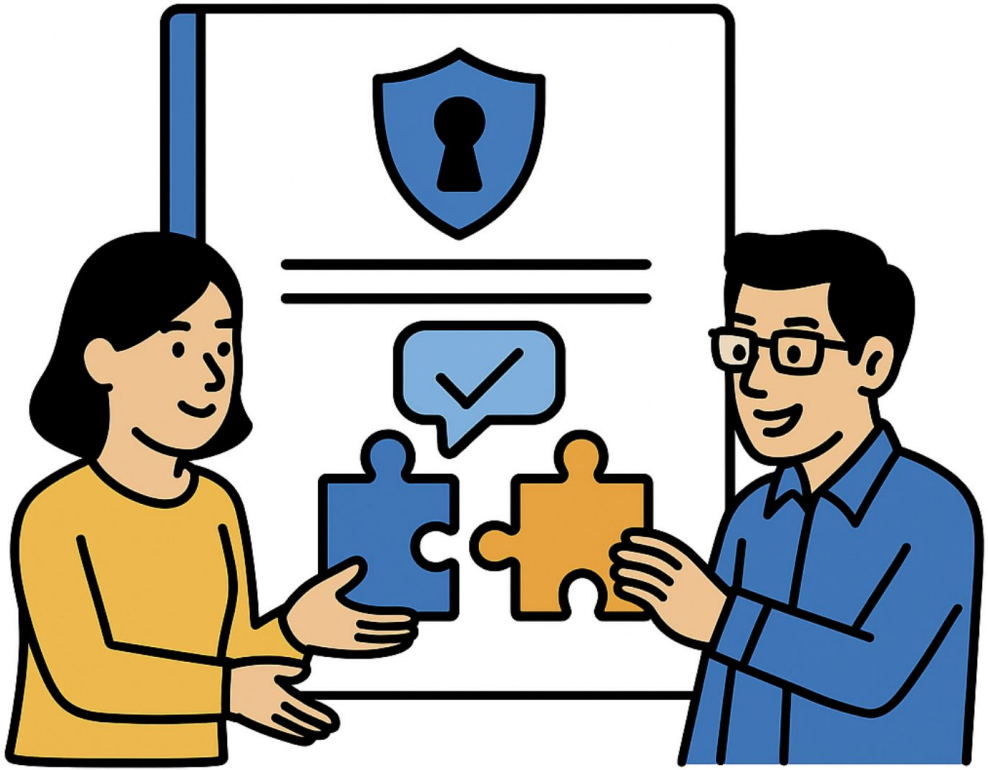
規則不屬於某個團隊，而是整個組織的共識

- PaC 讓規則成為可理解、可參與的討論空間。
- **小團隊**的經驗可以向上成為**組織標準**，
- 社群式的氛圍，讓治理真正變成大家的責任。

導入 PaC，不是設定規則，是設計變化的節奏

- 治理從理解現況開始，先觀察、逐步推進。
- 留下**彈性空間**與**信任遞進**的節奏，才是讓規則長出來的養分。
- Suppress 不是破口，而是制度成長的觀察點。

治理，從來不只是控制，而是理解與協作



Thanks!