



2026 KUBERNETES SUMMIT | 動手做工作坊

Private Model as a Service

在自己的叢集裡，經營一座模型服務

以 Red Hat OpenShift AI 建立可控管、可計量、可自助取用的私有模型服務

Ato Lin / Jeff Yan | Senior Solution Architect, Red Hat Taiwan

你今天會親手做的

六個模組 · 取金鑰、接 IDE、Agent、看報表

水面

今天要拆給你看的

- 模型目錄與部署 AI Hub · KServe · vLLM
- 存取治理 Subscription · 金鑰 · 限流
- 資源調度 Hardware profile · 佇列
- 安全、品質與計量 Guardrails · TrustyAI · 用量

Red Hat
Summit

Red Hat Summit Connect 2026

台北站

馬上掃描，立即報名！



11月3日星期二

9:30-17:00

台北寒舍艾美酒店

席位有限，立即報名參觀展覽與論壇
掌握產業最新技術動態



活動議程



今天怎麼進行

四個回合：先講你要做什麼 → 動手做完 → 再回頭拆背後的機制

 情境（講）

 動手（做）

 拆解（講）

開場

為什麼需要一座模型服務平台

10 min

回合 1

拿到一個模型

Module 1 · 2

8

15

12

min

回合 2

接進開發流程

Module 3 · 4 (4 為彈性)

5

25

8

min

回合 3

讓 Agent 動手

Module 5

6

20

8

min

回合 4

算帳與治理

Module 6

5

15

12

min

收尾

平台全貌、導入路徑、Q&A

12

min

Module 4 (Game Time) 是彈性項目 — 現場時間不夠會直接跳過，不影響後面的回合。

今天你會輪流扮演三種人

同一座平台，三種立場關心的事情完全不同

1 開發者 Developer

把公司自有的模型，接進每天在用的開發工具裡。

回合 1 取得模型端點與 API Key

回合 2 把模型接上 IDE、用 AI 協作寫程式

2 維運工程師 SRE / DevOps

用自然語言查叢集狀態，並把結果送到團隊頻道。

回合 3 以 Agent 操作 OpenShift 與 Slack

3 決策者 Technical Decision Maker

看見誰在用、用了多少、以及這座平台值不值得。

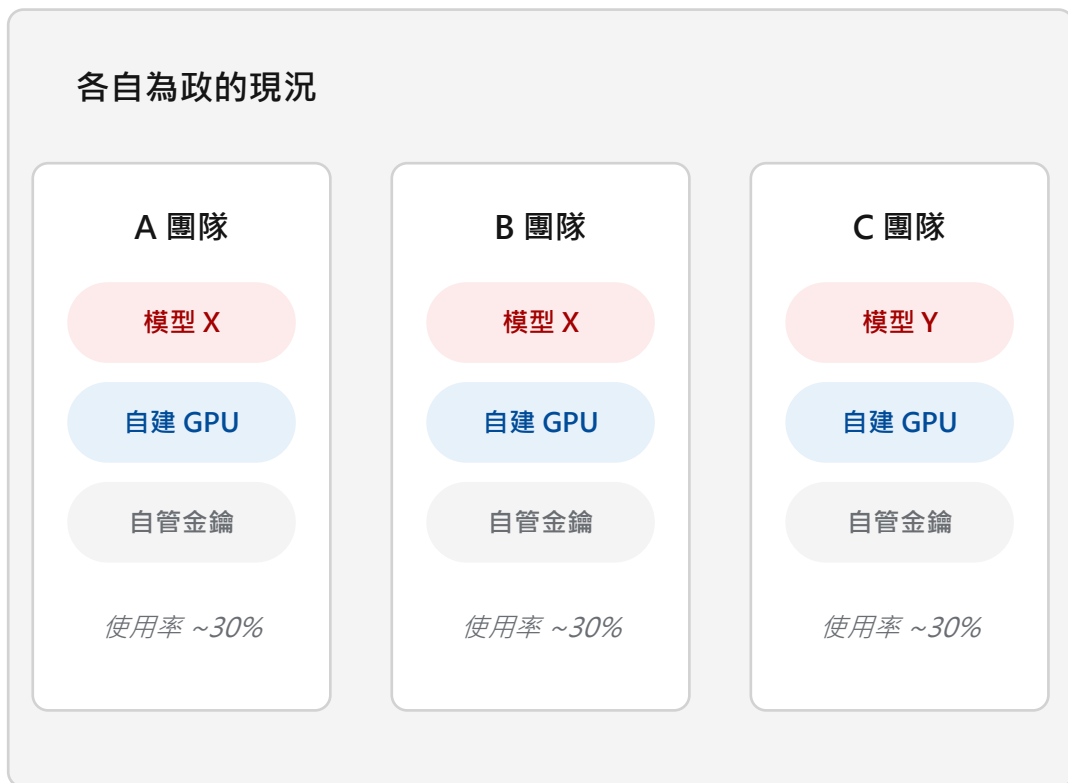
回合 4 檢視全叢集的用量與成本分佈

但這三個角色，全部都站在「使用者」這一側。

真正決定這座平台能不能上線的第四個角色 — 平台管理員 — 今天不在 Lab 裡。每個回合做完，我們就拆一段他的工作給你看。

當每個團隊都自己養一套模型

模型跑得起來，不代表這件事撐得住



問題不在「模型跑不起來」，而在沒有一套大家共用的規矩。

01

GPU 買了卻閒著

每個團隊各留一份餘裕，加起來就是一整櫃的閒置算力。

02

同一個模型部署三次

三份權重、三份維運、三份升級成本，能力卻完全一樣。

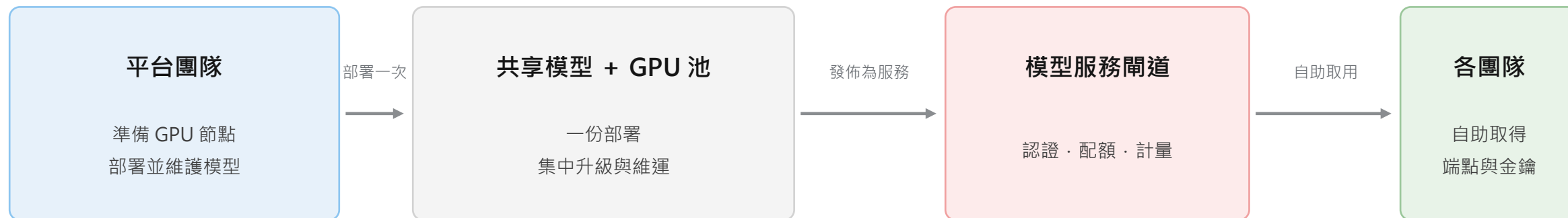
03

沒有人說得出用量

誰在用、用多少、花多少 — 沒有共同入口，就沒有數字。

改成「一次部署，全公司自助取用」

這就是今天實驗環境採用的架構



支撐這張圖的四件事



那為什麼不直接呼叫公有 API 就好？

兩條路都成立，差別在三件事上

資料落地

公有 LLM API

提示詞與檔案內容離開你的網路

成本結構

公有 LLM API

依 token 計價，用量一上升就往上跑

稽核能力

公有 LLM API

紀錄保存在服務供應商手上

自有模型服務（今天的環境）

全程留在自己的叢集內，未跨出邊界

自有模型服務（今天的環境）

硬體為既有投資，邊際成本趨近於零

自有模型服務（今天的環境）

指標與紀錄都落在自己的叢集

本場 Lab 不主張哪一條路比較好，而是示範第二條路實際做起來是什麼樣子。

你今天摸得到的，只是水面上那一層

Lab 示範的是結果；讓這個結果可重複、可控、算得出帳的，在水面下

水面上 — 你親手做的

1 認識環境

2 取得金鑰

3 接上 IDE

4 AI 協作開發

5 Agent 維運

6 用量與成本

水面

水面下 — 讓這件事可重複、可控、算得出帳的

1 模型目錄

AI Hub · 精選與量化模型

2 模型部署

KServe · 服務三要素

3 推論引擎

vLLM · MLServer

4 規模化

llm-d 分散式推論

5 存取治理

Subscription · 金鑰 · 限流

6 資源調度

Hardware profile · 佇列與配額

7 安全與品質

Guardrails · TrustyAI

8 計量與算帳

Token 計量 · 用量儀表板

每做完一個回合，我們就回到這張圖，把你剛剛用到、卻完全沒看到的那幾塊點亮。

回合

1

拿到一個模型

從零到手上有一組可以呼叫的端點與金鑰

情境

你要做什麼

8 min — 環境與目標

動手

Module 1 · 2

15 min — 取得端點與金鑰

拆解

背後發生了什麼

12 min — 平台側四件事

你等下要操作的環境長什麼樣

三個地方會用到：拿金鑰、用模型、看用量

你會直接操作的 Consumers

情境

動手

拆解

OpenShift AI Console

在這裡取得端點與金鑰

回合 1

雲端 IDE 與 AI 助理

把模型接進開發流程

回合 2 + 3

用量儀表板

看誰用了多少

回合 4

模型服務閘道

所有對模型的呼叫都從這裡進出 — 驗證金鑰、檢查配額、記錄用量

回合 1 拆解：這一層在做什麼

平台側控制面

模型從哪裡來、怎麼部署、誰能用、GPU 怎麼分、安全怎麼管、帳怎麼算 — 全部在這一層決定。

今天你不會操作到它，但每個回合結束我們都會拆一塊出來看。

水面下的那八塊

Red Hat OpenShift — 運算 / 網路 / 儲存 / GPU 節點

等下你要做的事

目標很單純：拿到一組可以被程式呼叫的模型端點

情境

動手

拆解

Module 1 · 2

登入環境，找到已經部署好的模型，替自己產生一把 API 金鑰。

- 1 登入 OpenShift Console，切換到分配給你的工作區
- 2 從右上角的應用程式選單進入 OpenShift AI
- 3 打開 Gen AI Studio，找到可用模型的清單
- 4 在「Models as a service」分頁選擇模型，產生一把 API Key

完成訊號

做完之後你會看到：

一組 https:// 開頭的端點網址

一串只顯示一次的 API 金鑰

提醒

- 金鑰只會顯示一次，先貼到記事本
- 「Models」分頁是空的很正常
- 卡住就舉手，工程師在場

先不解釋原理 — 等你把金鑰拿到手，我們再回頭拆它背後到底發生了什麼事。 時間：15 分鐘

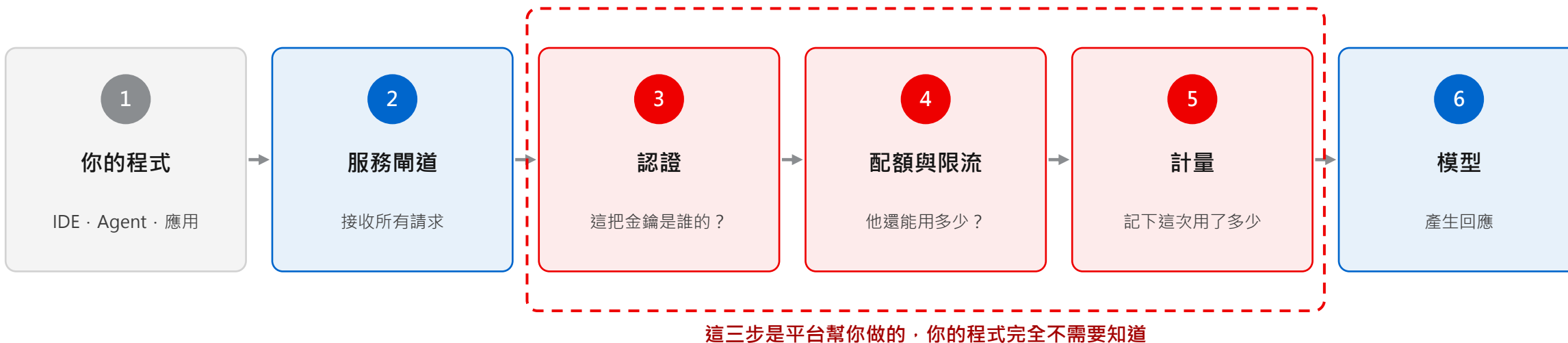
你剛才按的那一下，其實啟動了一整套 API 治理

從你的程式到模型，中間有三道關卡

情境

動手

拆解



全部建立在開放標準之上

這一層不是 Red Hat 自己發明的黑盒子，而是雲原生社群共用的三個專案：

Envoy

Kuadrant

Istio

為什麼不能只發一組帳密就好

沒有這一層，你無法回答：誰在呼叫、呼叫了多少、要不要擋、以及這個月該跟哪個部門收多少錢。

現在就有

OpenShift AI 內建能力

誰能用、能用多少、用多久？

把「發一把金鑰」變成一件可以稽核的事

情境

動手

拆解



現在就有

訂閱式的模型存取、可設定 token 配額與速率限制、使用者自助管理金鑰 — 都是 OpenShift AI 內建功能

回頭看你剛才做的那一步

你按一下就拿到金鑰 — 那是上面的第 2 步。第 1 步的方案是誰定的、配額訂多少、到期怎麼辦、外洩了去哪裡撤銷，Lab 全部沒有交代。而這正好就是你回公司要導入時，第一個會卡住的地方。

為什麼你的「Models」分頁是空的？

因為部署模型的人不是你 — 這正是這套做法的重點

情境

動手

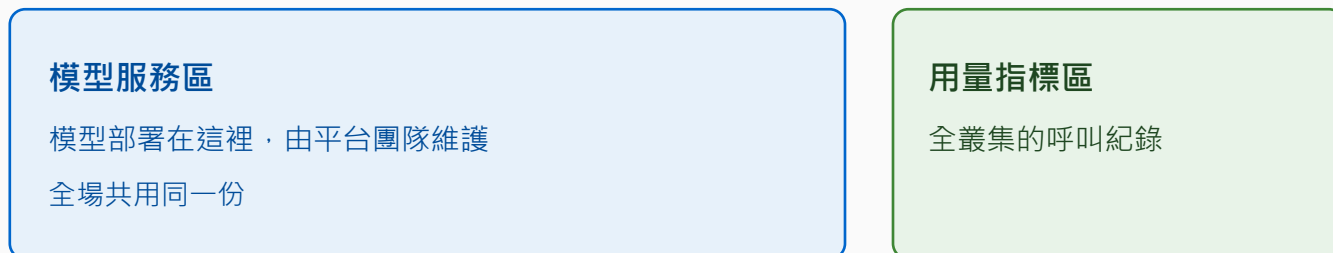
拆解

今天這座共享叢集

每位參加者各自一個工作區



全場共用的服務區



Q1 Models 分頁為什麼空的？

Q1

那一頁只列「你自己部署的」。模型不在你的工作區裡，所以是空的。

Q2 那我剛才是在哪裡拿的？

Q2

「Models as a service」分頁 — 列的是「被發佈給你使用的」模型。

Q3 這樣不是很受限嗎？

Q3

反過來想：你不必會部署模型，也不必管它跑在哪張卡上，就能用它寫程式。

「部署的人」與「使用的人」分開 — 這不是限制，而是讓同一份模型可以安全地給全公司用的前提。

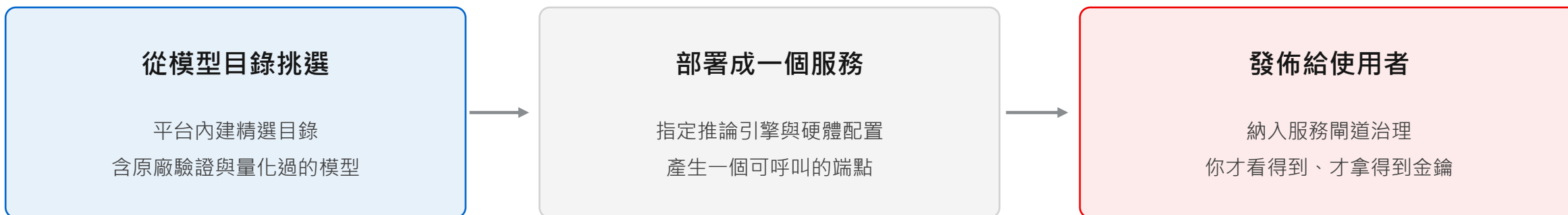
那個模型，是誰放上去的？

在你登入之前，平台團隊已經走完這三步

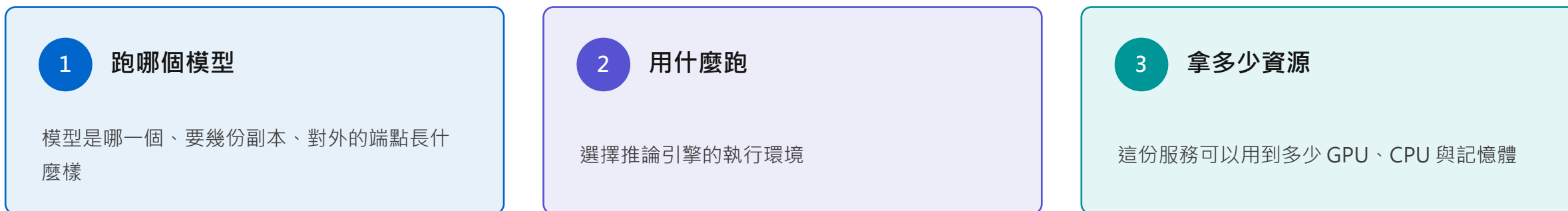
情境

動手

拆解



第 2 步的「部署」，實際上要配好三件事



現在就有

模型目錄、部署與發佈為服務，都是 OpenShift AI 內建的操作 — 不需要自己寫佈署腳本

你花三秒鐘拿到的那組端點，是這三步走完之後的最終產物。

回合 1 補上了三塊

你只按了幾個按鈕，但水面下已經用掉了三項平台能力

水面上 — 你親手做的



水面

水面下 — 讓這件事可重複、可控、算得出帳的



接下來回合 2：把這組端點接進真正的開發流程 — 然後我們再拆兩塊。