

KUBESUMMIT 2026

地端 K8s 叢集的自主運維實踐

結合自動自愈、風險分級核准與受控 LLM 轉發的巡檢 Agent

—— 陳家丞 (Michael) 玉山銀行 智能金融處 • 主任工程師

陳家丞

Michael Chen • Staff Engineer

玉山銀行 • 智能金融處

 Chia-Chen Chen

負責 Kubernetes 與生態系導入評估，混合雲架構設計，推動雲原生永續工具的發展

 稱職貓
奴

 重度抹茶愛好
者

歷年講題 • PAST TALKS

DevOpsDays Taipei '26

不是我的系統，但與我有關！團隊從「各自為政」到「群體決策」的協作實踐

KubeSummit '25

打造永續 AI：從 Kepler 看見能耗，用 GreenOps 改善未來

KCD Taipei '25

Kepler in Action：探索 Kubernetes 工作負載的功耗

Kubernetes Summit '24

Code-Driven Development & Monitoring of ML Services on K8s

Kubernetes Summit '23

Offline K8s 升級怎麼辦？跨大版升級甘苦談一次說給你聽！



ABOUT THE SPEAKER

AGENDA

今天想帶大家走過的五件事

- 01 為什麼要做——值班痛點、封閉環境的先天限制與指令式工具的極限。
- 02 我怎麼做——雙軌治理：結構化自愈、風險分級、人工核准與資安防線。
- 03 帶來什麼——實際案例：值班日常從人工排查變成看報告做決策。
- 04 跟既有工具比——k8sgpt / kagent 的定位差異、優勢與取捨。
- 05 經驗與建議——受限環境下，可以直接帶走的設計與建議。

地端維運的日常

L1（第一線值班）：24 小時輪值接告警、做初步排查與處置的角色——本方案要自動化的，就是這一層的例行工作。

Terminating / Unknown

殘留資源清不完

卡死的 Pod 需要人工逐一 force delete，放著不管就持續累積、吃掉叢集資源。

kubectl logs / describe

告警後手動翻日誌

每次告警都要重跑同一套排查流程：狀態、事件、各容器日誌，再拼湊根因。

3-4 min alert delay

告警本身就慢

Prometheus scrape + 評估 + 分組等待，事故發生數分鐘後值班人員才收到通知。

想引進開源工具的我

VS.



資安規範

安全第一 · 合規至上 · 風險可控

開源萬歲！
效率 + 創新 + 生產力
先來試試看！

這個好用！
快上車！



Prometheus



Grafana



OpenTelemetry



Kepler



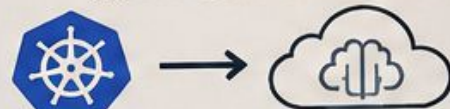
Argo CD

K8s
管理者

我的願景清單 ✨

- ✓ 自動化部署
- ✓ 可觀測性提升
- ✓ AI 助手加持
- ✓ 開發效率 UP UP!

接上外部 LLM
讓 AI 幫我解決問題！



- 自動分析 Logs
- 異常根因分析
- 建議修復方案

不行！
沒那麼簡單！

禁止直接串接外部 LLM

機敏資料必須遮罩 / 脫敏

資料不得外洩

僅限白名單 API / 代理通道

使用需經過審核與 Logging

開源工具引進檢查清單

- ✓ 工具來源可信嗎？
- ✓ 社群活躍嗎？
- ✓ 是否有資安漏洞？
- ✓ 是否符合內部規範？
- ✓ 維護與支援計畫？

企業資安防火牆 多層防護 · 滴水不漏

- 網路隔離
- 存取控制
- 資料遮罩
- 日誌審計
- 合規稽核

好吧...
安全很重要QQ

結論：想用可以，但要走對流程、守住底線，才能安心飛！

✓ 工具評估

✓ 資安審查

✓ 資料保護

✓ 受控使用

✓ 持續監控

K8s
管理者

三個限制，決定了後面所有設計

應用連不了外網

封閉金融內網，僅少數白名單管道可受控呼叫外部服務



LLM 只能走受控介接層

要求直連外部 SaaS 的工具直接出局

進叢集要層層跳板

封閉內網須經 VMware Horizon 等跳板才連得到機器，步驟繁瑣



先推 Teams，讓第一線秒知狀況

不用先連進去就能掌握問題；核准也走簽章連結

任何刪除動作都要能追溯、能停用

金融環境對自動化變更的基本要求



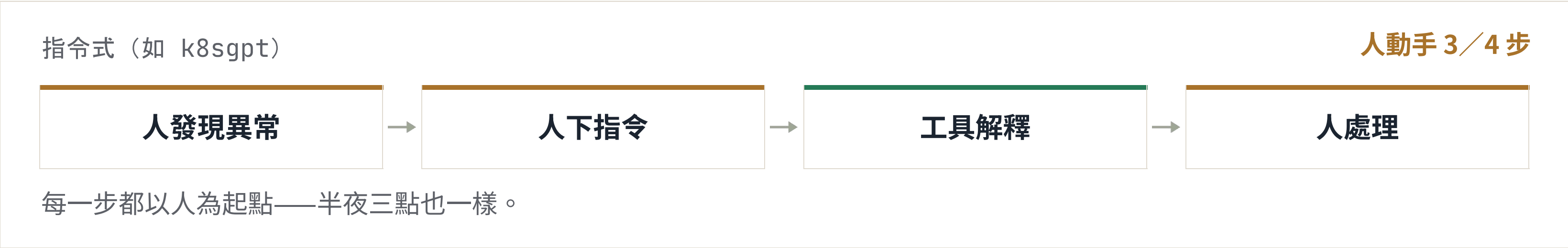
風險分級＋人工核准＋留痕

不是附加功能，是架構的一部分

很多開源方案在這裡不是不好用，是進不來——所以我們自己做。

指令式工具的極限

■ 人動手 ■ 機器自動



真正的差別在誰來觸發，以及處置有沒有一路做完。

Agent 整體架構

01 · TRIGGER

即時監看
K8s Watch，狀態變更
秒級推送

定期掃描
每 15 分鐘安全網補漏

02 · PIPELINE

告警處理管線

- ① 蒐證：Pod 狀態 + 日誌 + Events
- ② 機敏資訊遮罩去識別化
- ③ namespace 風險分級
- ④ 決定處置：自動修復／送核准

03 · ANALYSIS

AI 根因診斷 可隨時停用
日誌經白名單管道 →
Control API → 外部大模型

狀態摘要
直接取自 K8s API，不靠
LLM，一定會產生

04 · NOTIFY

Teams 通知卡片

- 狀態摘要 + AI 診斷
- 自愈執行紀錄
- 核准請求連結

DEPLOYMENT TOPOLOGY

Deployment ×1 **核心服務**：即時監看 + 定期掃描 + 控制面日誌稽核

內含 **Approval UI** 核准網頁——高風險動作的人工核准入口，後面詳述

DaemonSet

節點代理：每台控制節點的 audit／etcd 日誌稽核，不綁叢集權限

— — 虛線：LLM 路徑，可整條停用

雙軌治理架構

問題先分類，再決定「自己修」還是「找 LLM 問」。

1 結構化異常自愈

Pod 殘留、卡死、反覆重啟——規則可判定的問題，在 RBAC 安全邊界內 **直接自動處置**，不呼叫 LLM。

2 非結構化問題診斷

模糊的應用程式或系統錯誤——自動彙整上下文、內網去識別化後，經白名單管道 **轉發外部大模型** 做根因分析。

—— 兩軌之外，還有一層 **namespace 風險分級**——決定 Agent 能不能自己動手。

自愈的安全邊界

動手之前先劃線：唯讀為主的最小權限，唯一的寫入動作是 pods delete，用四道防線圍住它。

```
# ClusterRole : Agent 服務帳號的全部權限
rules:
- resources: ["pods"]
  verbs: [ "get", "list", "watch", "delete" ]
- resources: ["pods/log"]
  verbs: [ "get" ] # 唯讀
- resources: ["events"]
  verbs: [ "get", "list" ] # 唯讀
```

沒有 create、patch、exec——這就是全部權限。

"delete" 唯一的寫入權限，四道防線圍住它：

- 01 **保護清單放 Vault**：保護清單不落在可被篡改的 ConfigMap
- 02 **高風險改走人工核准**：核准制 namespace 的刪除動作，須有人驗明身分核准才執行
- 03 **事後可稽核**：所有刪除進 K8s Audit Log，Agent 自己也稽核它
- 04 **node-local 角色零權限**：DaemonSet 不綁任何 ClusterRole，只讀本機檔案

雙路徑偵測設計

比較項	Alertmanager Webhook	K8s Watch（本方案）
觸發延遲	3-4 分鐘（scrape+eval+group_wait）	秒級，狀態變更即時推送
外部依賴	需修改 Prometheus / Alertmanager 設定	無，完全自主控制
API Server 負擔	額外 scrape 流量	單一條 Watch 長連線

Watch補即時、APScheduler 補連線中斷的漏網事件——兩條路徑共用 30 分鐘去重快取（存於 Redis，Pod 重啟不遺失），不會重複騷擾值班人員。

軌道一 | 結構化異常自愈

三種寫入型動作，都不需要 LLM 參與判斷。卡片上的 GATE 是動作閾值；下方三個時間屬於偵測與通知層。

FORCE DELETE Terminating 卡死清理 以--grace-period=0 強制刪除卡死的 Pod，成功／失敗都推 Teams 留痕。 GATE · Terminating 卡住 > 10 min	DELETE → RECREATE CrashLoop 自主修復 CrashLoopBackOff／OOMKilled 達雙重閾值才刪除讓控制器重建；Init Container／sidecar 一併偵測。 GATE · restart ≥ 5 且連續 3 次掃描	STALE CLEANUP 殭屍 Pod 自動清理 清理殘留的 Evicted／ContainerStatusUnknown Pod，排除恢復中的暫時狀態。 GATE · 連續 2 次掃描仍殘留
WATCH · 秒級 狀態變更即時偵測與通知，不執行寫入動作。	SCAN · 每 15 min 安全網掃描；三種寫入動作都只在這一層判定與執行。	COOLDOWN · 30 min 同一個 Pod 的同一個問題，30 分鐘內只通知一次、避免洗版；修復動作照常執行。

這些動作是否自主執行，由下一頁的namespace 風險分級決定。

Namespace 三級風險分級

同一個動作，依所在 namespace 的風險等級，決定「禁止／需核准／可自主」。

banned

完全禁止

命中保護清單：一律不執行任何動作，只發通知，交由人工 kubectl 處理。

approval

核准制

命中核准清單：不自主執行，建立待核准請求並推 Teams 核准連結，由人工在內建的 Approval UI 核准網頁驗證授權後才動手。

autonomous

自主執行（預設）

兩者皆未命中：維持原行為，偵測到即自動處置，全程僅以 Teams 留痕通知。

保護清單放 Vault 而不放 ConfigMap——「**哪些 namespace 碰不得**」本身就是機敏設定，不能是叢集內誰都能改的一行 YAML。

人工核准工作流程

高風險動作不自主執行，改成一筆可稽核、可過期、可撤回的核准請求。



STATE MACHINE



狀態存於 Redis；兩人同時按核准，只有先送出的生效，後到的會被告知已處理。

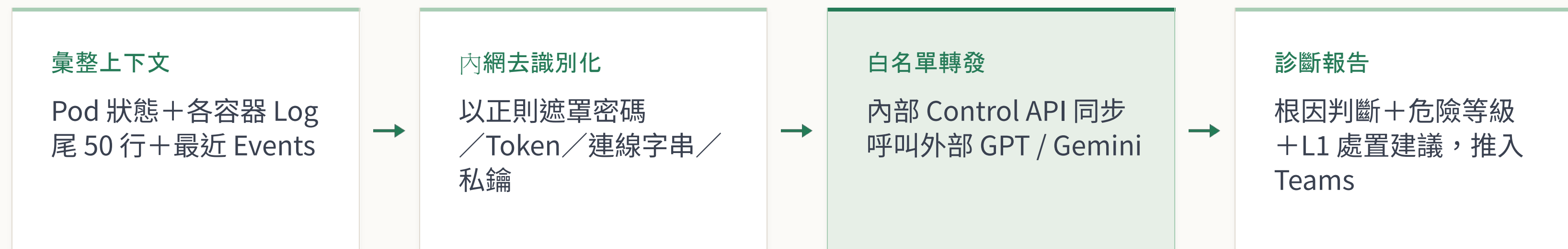
SAFEGUARDS

連錯即鎖、故障即鎖：密碼連錯 5 次鎖 15 分鐘；Redis 掛掉一律當已鎖定，寧可誤擋不放行

核准當下再驗一次：按下核准那一刻重新驗證連結簽章，被改過或偽造就拒絕

只開一條路徑：整個核准頁只暴露 /approvals，可攻擊面極小

軌道二 | 非結構化問題診斷



日誌只進 LLM 管道 —— 原文不會出現在 Teams 卡片；卡片上的 Describe 摘要直接來自 K8s API。

LLM 可隨時關閉 —— 一個環境變數即可停用，偵測、自愈、Describe 摘要全部照常，先驗證再放行 AI。

分析結果快取 —— 同一元件同一訊息在有效期內重複出現，沿用先前分析，不重複燒 LLM 成本。

Control API 資安護城河

資料出內網前的四道關卡——Agent 本身自始至終沒有外網路徑。

第 1 關 · 程式端遮罩

送出前先遮罩去識別化，
並縮限單次日誌行數壓低
曝險面

第 2 關 · Vault 機敏管理

API
URL／JWT／Webhook
／簽章密鑰全由 Vault 提
供

第 3 關 · Control API

JWT 驗身，DLP 去識別
化為主要防線，統一出口
留痕

第 4 關 · 網路白名單

僅 Control API 具備核准
的對外管道；Egress 政
策鎖定其餘流量

`trace_id` 貫穿偵測 → 核准 → 執行，全程輸出 JSON 結
構化稽核 log，事後可完整追溯。

Events 內容跳脫：卡片上的 Event reason／message 一
律先做字元跳脫，避免叢集內訊息注入 Adaptive Card。

只要每一筆出去的資料都**可控、可稽核**，外部模型也能通過合規要求。

Teams 即時通報

L1 告警分析

Watch 偵測 CrashLoop / OOMKilled，附 Describe 摘要+AI 診斷

自愈執行結果

Force Delete／CrashLoop 重建留痕；banned namespace 改發人工介入警告

核准請求

Action.OpenUrl 核准按鈕+純文字備援連結，附簽章與到期時間

Pod 刪除通知

附操作者反查（audit log 跨 Pod 查詢）；Succeeded 清理不擾民



透過 工作流程 來代表

下午 02:52

[STAGING] Control-Plane Log 異常: kube-apiserver

異常清單：

Text		
[W] (1次) 06:46:10\.	912031	
[W] (1次) 06:46:10\.	912039	
[W] (1次) 06:48:09\.	351019	
[W] (1次) 06:49:54\.	078520	
1 logging\.		go:59\] \[core\] \[Chann
1 logging\.		go:59\] \[core\] \[Chann
1 logging\.		go:59\] \[core\] \[Chann
1 logging\.		go:59\] \[core\] \[Chann

AI 診斷建議：
以下判斷以「control-plane 連線 et cetd」角度整理：

主要問題

kube-apiserver (kube-apiserver) 對 etcd:2379 的 gRPC 連線偶發失敗／被中止，錯誤型態包含：

- authentication handshake failed: context canceled

▲ 實際通知卡片：Control-Plane Log 異常+AI 診斷

Agent 上線前 vs. 上線後的值班日常

同樣是 K8s 管理者，心情差很多😓

上線前：被動救火，靠人力 + 靠問 AI

ALERT
Pod CrashLoopBackOff
namespace: prod
pod: user-service-5d6f7c9b6c-abc12
Time: 23:47

又來了...
半夜告警😓

1 進 K8s 找 Pod
\$ kubectl get pods -n prod
...
Running CrashLoopBackOff

2 翻 Log 找線索
\$ kubectl logs -f ...
Error: connect to DB failed
timeout
...

3 問 AI 才知道原因
我：這段 log 是什麼問題？
AI：資料庫連線逾時，
可能是 DB 負載過高或
網路問題，建議檢查...

4 再想解法 + 執行修復
☑ 確認 DB 狀態
☑ 調整連線參數
☑ 重啟 Pod
☑ 祈禱不要再掛...

為什麼又是我😓

K8s 文件
厚到懷疑人生

心情寫照：
找不到問題在哪，時間一直過...
睡眠 -1，頭髮 -10，壓力 +10086

上線後：Agent 自主監控，自動分析，主動通知

我來顧 K8s，
你安心睡覺😎

K8s Cluster

- Pod Running
- Pod Running
- Pod CrashLoopBackOff
- Pod Running
- ...

Agent 在 K8s 裡做的事

- ☑ 監控 Pod 狀態 (K8s API)
- ☑ 偵測異常
- ☑ 自動收集 Log / Events
- ☑ 詢問 LLM 分析原因
- ☑ 提供根因分析 + 建議解決方案
- ☑ 透過 Teams 票卡通知管理者頻道

Teams 票卡通知

K8s Agent BOT 00:23

[P1] Pod 異常偵測
Namespace: prod
Pod: user-service-5d6f7c9b6c-abc12
Status: CrashLoopBackOff

Describe 摘要 (來自 K8s API)

- State: Waiting (CrashLoopBackOff)
- Last State: Terminated (Exit Code: 1)
- Ready: False Restart Count: 7
- Event: Back-off restarting failed container

根因分析 (LLM)
資料庫連線逾時導致應用啟動失敗
可能是 DB 負載過高或網路不穩。

建議解決方案

1. 檢查 DB 連線狀態與負載
2. 確認 NetworkPolicy / Service 可達性
3. 調整 DB 連線參數 (timeout / pool)
4. 必要時重啟 Pod 並觀察

相關連結: Grafana | Logs | Runbook

終於可以
準時下班👏
(還能睡好覺)

心情寫照：
Agent 幫我顧、幫我查、幫我分析、幫我通知
效率 +100，壓力 -100，頭髮 +10😎

GOOD NIGHT

同樣的值班，不一樣的命運。讓 Agent 當你的 K8s 神隊友！🚀

實際帶來的改變

值班日常從「人工排查」變成「看報告做決策」。

BEFORE · 告警排查

收到 Prometheus 告警或通知信後，得登入系統翻找 log、逐步確認問題在哪。



AFTER · 即時通知

通知直接呈現**LLM 分析結果**——建議要做什麼、或已經做了什麼，一眼掌握，不必再進系統翻 log。

BEFORE · 孤魂 Pod

刪除不乾淨、殘留在叢集的 Pod，要人工逐一找出來 force delete。



AFTER · 自動清理

Agent**自動偵測並清理孤魂 Pod**，結果推 Teams 留痕，不再累積吃掉叢集資源。

實際案例

透過 工作流程 來代表

下午 07:34

[STAGING] 🚨 L1 告警分析: PodUnhealthy:CrashLoopBackOff

Pod: kube\-system/kube\-controller\-manager\- Worker-n1

目前狀態 (Describe 摘要):

Phase: Running

Conditions:

PodReadyToStartContainers=True

Initialized=True

Ready=False

ContainersReady=False

PodScheduled=True

Containers:

kube-controller-manager: Waiting/CrashLoopBackOff (restarts=156)

[Events - 最近 8 筆]:

[2026-08-26 11:29:09+00:00] Pulled: Container image " Harbor-image.com.tw /library/kube-controller-manager:v1.30.6" already present on machine

[2026-08-26 11:29:13+00:00] Created: Created container kube-controller-manager

[2026-08-26 11:29:13+00:00] Started: Started container kube-controller-manager

[2026-08-26 11:34:07+00:00] BackOff: Back-off restarting failed container kube-controller-manager in pod kube-controller-manager- Worker-n1 _kube-system(388ab1ed6999a7a9f9a94249822525e8)

AI 診斷建議:

以下依你提供的告警與 Log，整理成 根本原因分析 與 L1 建議修復步驟 (繁體中文):

一、根本原因分析

1. kube-controller-manager 不是因為啟動失敗，而是因為「無法續租 (renew lease)」導致 leader election 失敗後退出

Log 明確顯示:

Failed to update lock

failed to renew lease kube-system/kube-controller-manager

leaderelection lost

這代表 Controller Manager 原本已啟動成功，但在更新領導權租約時失敗，最後主動退出，造成 CrashLoopBackOff。

2. 主要異常點在 kube-apiserver 回應超時

關鍵訊息:

Put "https://127.0.0.1:6443/.../leases/kube-controller-manager?...": Client.Timeout exceeded

context deadline exceeded

CASE 1 CrashLoopBackOff 秒級告警+AI 根因診斷

透過 工作流程 來代表

下午 08:45

新增

[STAGING] ⚡ Force Delete 執行: default/mock-terminating-pod

Pod default/mock\-terminating\-pod 已卡在 Terminating 狀態 1.7 分鐘，已自動執行 Force Delete (gracePeriodSeconds=0)。

時間: 2026-08-26 20:45:32 (UTC+8)



在對話中回覆

CASE 2 Terminating 卡死 Pod 的自動清理留痕

20 / CASES

實際案例 | 人工核准流程

CASE 3 核准制 namespace 偵測到卡住的 Terminating Pod——不自主執行，改走人工核准

 透過 工作流程 來代表

下午 08:49

新增

[STAGING]  需人工核准: polaris/mock-terminating-pod

動作: force_delete_stuck_terminating
原因: 卡在 Terminating 狀態 1.2 分鐘
點擊下方按鈕開啟核准頁面 (需 VDI 內網連線 + AD 帳密驗證)。
若按鈕點擊後無法連線 (例如 Teams 用戶端所在裝置本身不在 VDI/內網),
請改為複製以下連結, 貼到 VDI 內的瀏覽器開啟:

https://approval-web.com.tw /approvals/971ababd19ad4468b036d4e6d9f26921?

時間: 2026-08-26 20:49:14 (UTC+8)

開啟核准頁面



在對話中回覆

核准請求

動作類型: 強制刪除卡住的 Terminating Pod

Namespace: polaris

Pod: mock-terminating-pod

原因: 卡在 Terminating 狀態 1.2 分鐘

建立時間: 2026-08-26T12:49:14.493180+00:00

AD 帳號

密碼

核准 拒絕

▲ Teams 核准請求通知+簽章連結

▲ 核准網頁：動作明細+AD 驗證後核准／拒絕

與 kagent / k8sgpt 的定位比較

	k8sgpt	kagent	本方案
定位	診斷＋解釋 CLI	通用 Agent 框架	垂直場景 L1 值班代理
觸發方式	人工執行／排程掃描	依 Agent 設計而定	Watch 秒級＋排程雙路徑
自動處置	唯讀分析，設計上不修復	可綁定工具，需自行開發	Force Delete＋CrashLoop 修復內建
人工核准	無（唯讀，無此需求）	框架層級通用核准關卡	內建人工核准閘門
Tier2 落地	可行；--explain 需內部模型	Web UI 需另闢存取路徑	原生相容：免外網、互動面極小

多數差異是「設計定位不同」，而非對方做不到——三者互補大於競爭。**本方案的優勢**：原生相容封閉環境、內建人工閘門；**取捨**：場景較窄，聚焦 Pod 生命週期與控制面稽核。

三個帶回去的重點

- 01 自主的關鍵不在模型多聰明，在誰來觸發——主動巡檢、自動處置，從偵測到留痕一路做完。
- 02 內部 Control API 是資安護城河——在合規前提下用上外部大模型。
- 03 自主也要留一道人工閘門——RBAC 最小權限＋三級風險分級＋高風險動作人工核准與留痕稽核。

THANK YOU

謝謝聆聽

歡迎會後一起交流 🖐️