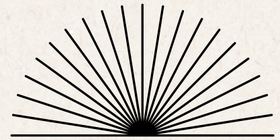
基於 KUBERNETES 的自動擴展影音轉檔系統

資訊116 許雅慈

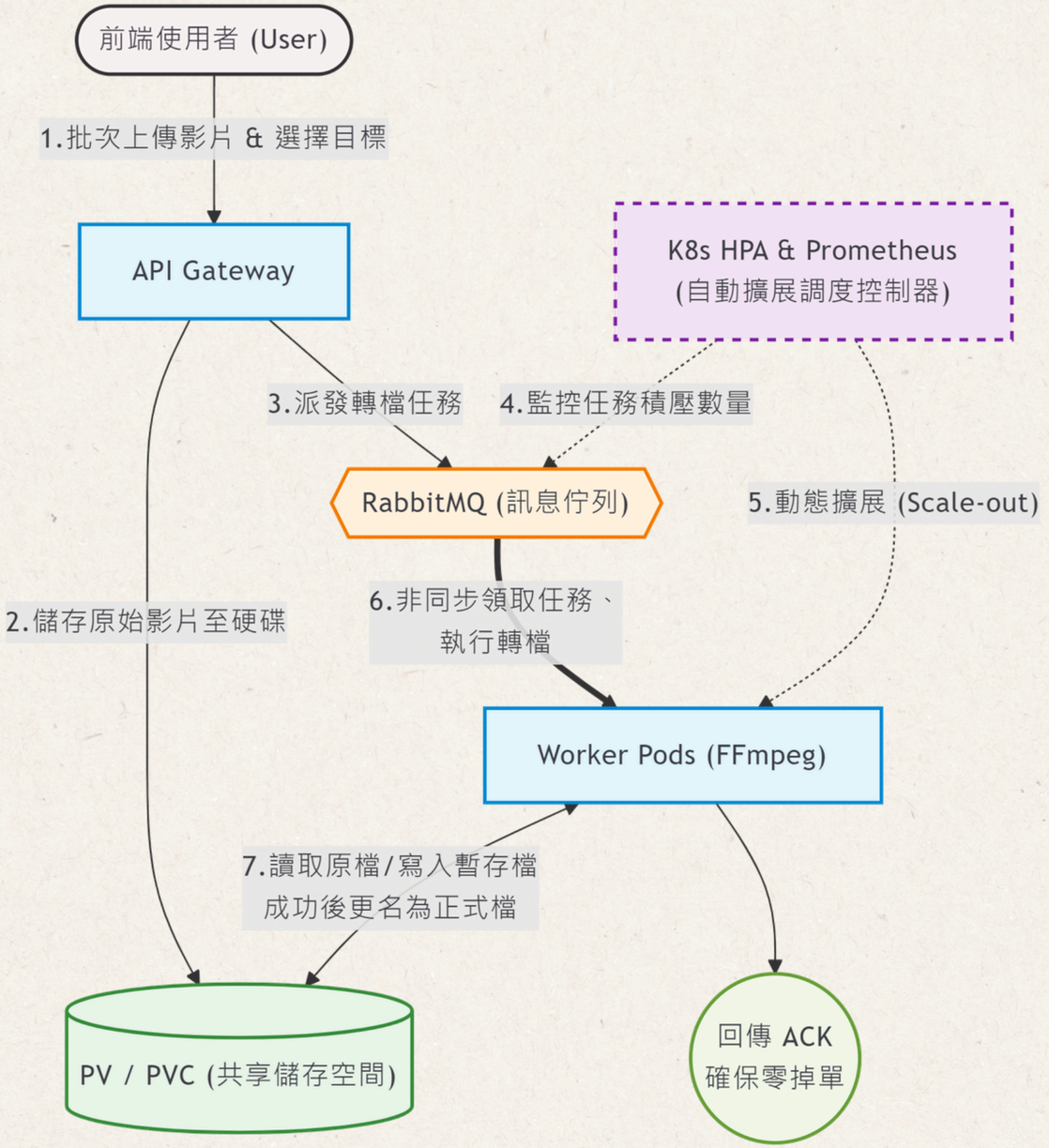


實作動機

傳統系統問題： 單機系統處理多工時容易造成效能瓶頸

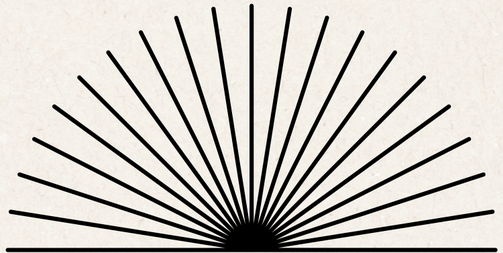
解決方案： 引入 Kubernetes (K8s) 容器編排技術，建立具備「自動擴展」與「負載平衡」能力的轉檔工廠 。

系統架構


前端使用者 (User)
1.批次上傳影片 & 選擇目標
API Gateway
K8s HPA & Prometheus
(自動擴展調度控制器)
3.派發轉檔任務
4.監控任務積壓數量
RabbitMQ (訊息佇列)
5.動態擴展 (Scale-out)
2.儲存原始影片至硬碟
6.非同步領取任務、
執行轉檔
Worker Pods (FFmpeg)
7.讀取原檔/寫入暫存檔
成功後更名為正式檔
PV / PVC (共享儲存空間)
回傳 ACK
確保零掉單

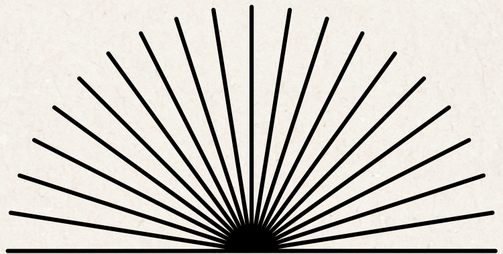
基礎設施與容器編排

- **Kubernetes (K8s)**：微服務叢集調度、Worker Pod的自動水平擴展
- **Docker**：封裝 Python 程式與 FFmpeg 轉檔引擎
- **Minikube**：在本地端（Windows + WSL2）模擬 K8s 叢集



後端框架與前端介面

- **Python 3.10**：結合 FastAPI 框架建立高併發、非同步的 API 服務
- **Tailwind CSS**：前端介面設計



核心轉檔與任務佇列

- **FFmpeg：**核心影音處理引擎、強制限制執行緒與 720p 縮放控制運算資源實現多格式轉檔
- **RabbitMQ：**將「網頁接收」與「轉檔運算」完全解耦、嚴格的 ACK 容錯機制

監控與自動擴展機制

- **Prometheus**：即時檢查與採集 RabbitMQ 佇列中的任務數量
- **HPA**：影片變多時自動增加 Worker Pod 進場工作

結論與未來展望

- 成功結合 Kubernetes HPA 與 RabbitMQ 訊息佇列，能依據任務負載動態擴展 Worker Pod，解決傳統單機問題
- 非同步處理與 ACK 機制達成任務零遺失，暫存更名法防堵使用者下載到半成品
- 後續規劃引入Redis快取資料庫與WebSocket技術，跨容器紀錄並即時推送轉檔進度百分比
- GPU容器化優化底層FFmpeg的編碼效率，針對不同影片規格設計優先權佇列