twMVC #51

《以平台工程重新思考系統設計 - 以 Batch System 為例》



歡迎參加我們的每週四固定聚會

以平台工程重新思考系統設計 - 以 Batch System 為例

Jamis Liao



I AM Jamis

- **產業經歷**
  - SI、 線上售票、電商開店平台、電子支付、共享停車平台、博弈、人資系統、區塊鏈
- **角色經歷**
  - 工程師、共同創辦人、Team Leader、總監
- **專長技能**
  - 什麼都做 (aka 雜魚)
- **twMVC 核心成員**



挑戦

Batch System 的挑戰

1 反應力不足

傳統系統難以適應快速變化的業務需求。

2 可擴展性受限

擴展能力有限，不容易應對大規模任務處理。

3 維護成本高

代碼複雜度高，維護和更新困難。

4 適用範圍單一

單一語言開發的 Batch System 無法適用不同程式語言。



Batch System 常見的需求

Batch System 的需求



不同的觸發方式

多樣性的觸發方式，例如：週期性觸發、監控外部資源變化時觸發、透過 WebAPI 觸發。



任務之間的順序與交互

不同任務之間的順序、依賴關係設定



專注在業務流程

只專注在業務流程，其他基礎建設不需要處理，例如：Logs、Metrics、部署方式、基礎資源的容量



容易查詢相關資訊

容易查詢每一個任務的狀態，並且很簡單的取消已排定的任務

以軟體視角來設計的話



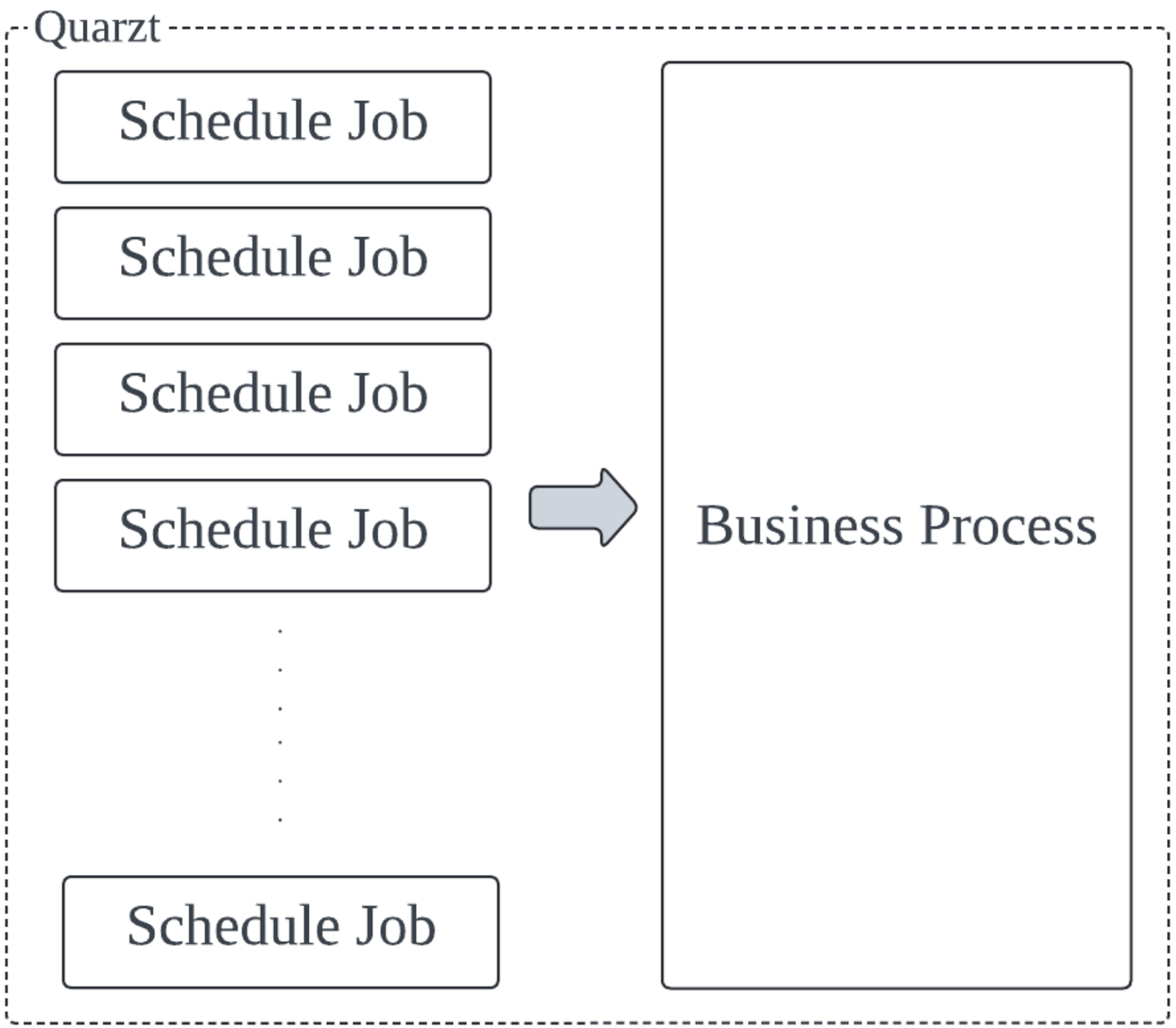
使用 Schedule 類型的框架來達成自動執行，例如：Quarzt

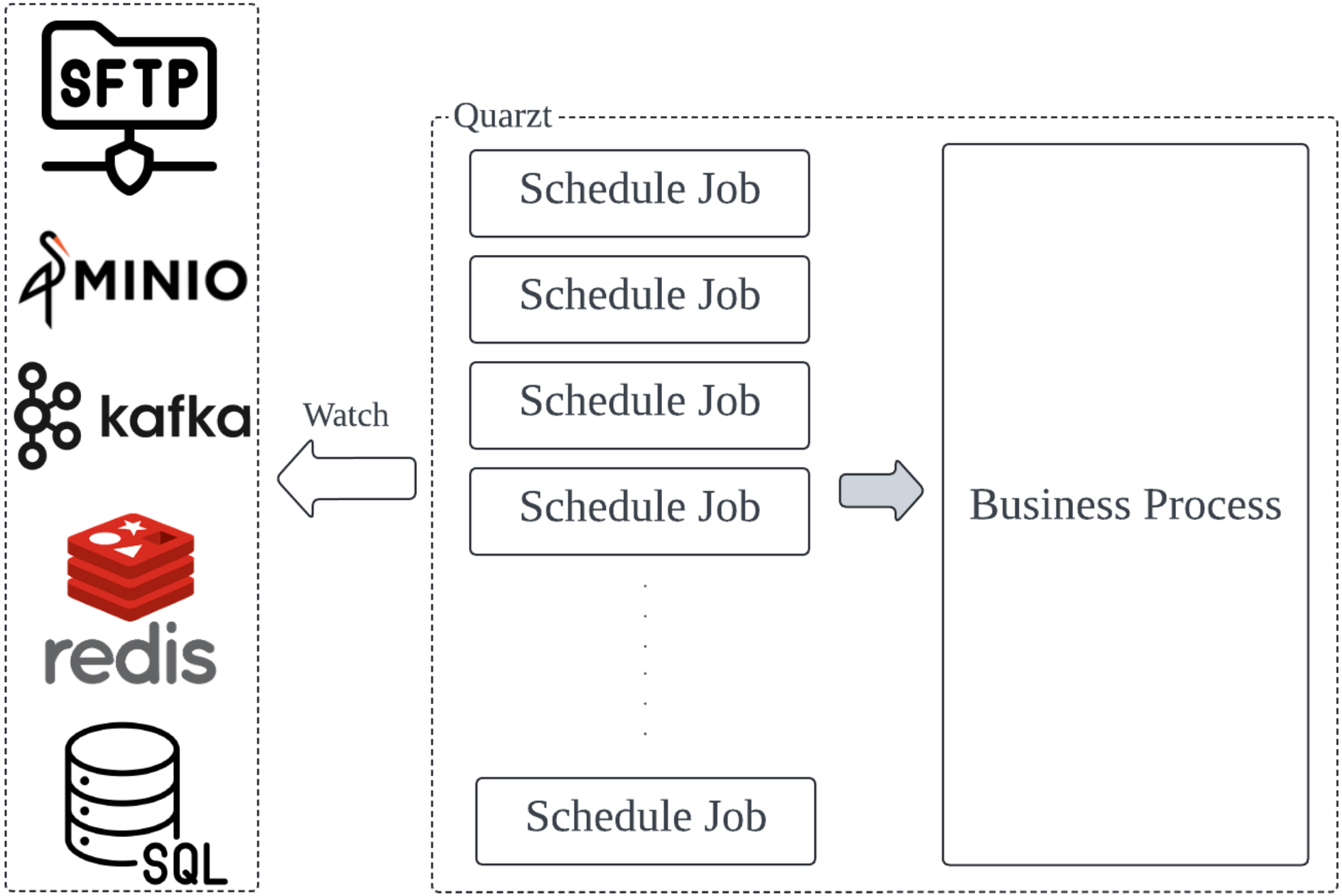


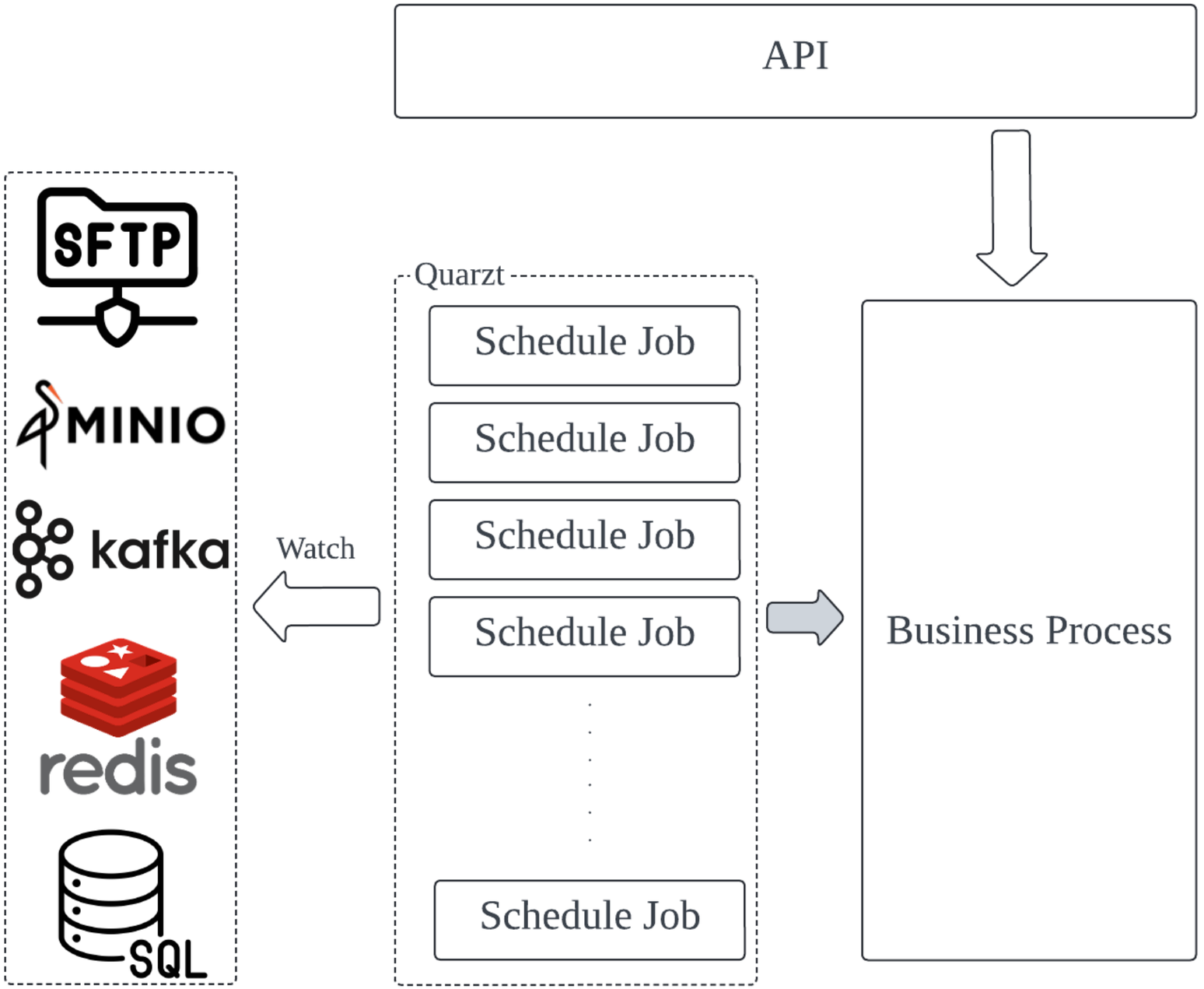
外部資源監控與流程只能透過程式碼完成



透過外部資料來取消任務


Quarzt
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Business Process


SFTP
MINIO
kafka
redis
SQL
Watch
Quarzt
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Business Process


API
SFTP
MINIO
kafka
redis
SQL
Watch
Quarzt
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Schedule Job
Business Process

如何透過平台工程來解決問題

平台的價值在於：賦能



提高生產力

平台可以為團隊提供工具和資源，讓他們更高效地完成任務。



簡化操作

平台提供易於使用的界面，讓使用者更容易地使用和管理工具。

讓我們換個角度來思考！

純軟體的視角

傳統的批次處理系統設計通常過於侷限用軟體代碼來完成所有需求，而忽略了以整個系統的全面性來思考。

平台工程的視角

平台工程讓我們從更全局的角度思考系統，以賦能的方式讓實作業務流程的任務擁有不同的能力。



軟體視角與平台工程的融合

1

軟體層

程序層實現批次處理邏輯。

2

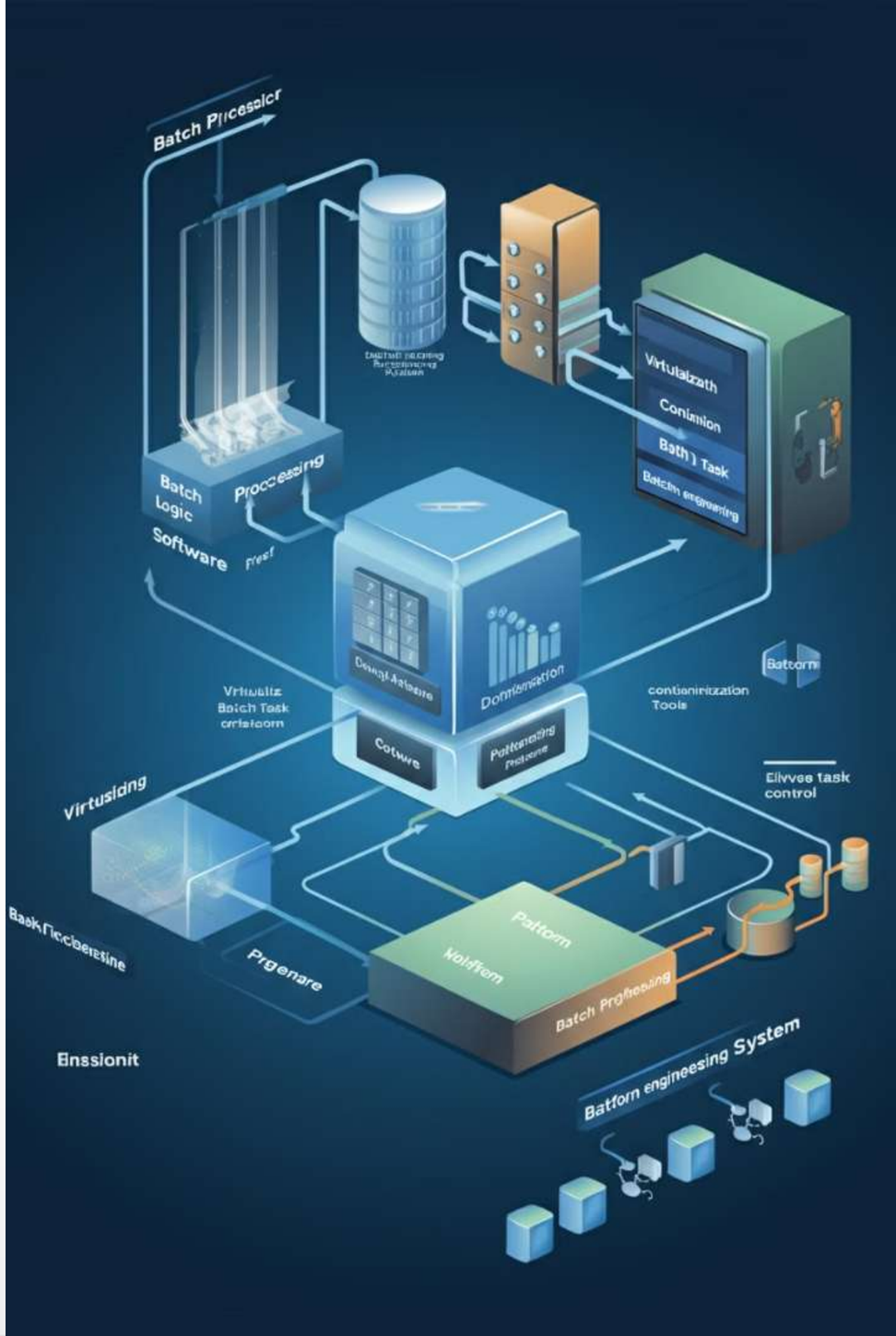
虛擬化

引入容器化和編排工具。

3

平台工程

將批次任務處理的控制提升到平台層面，賦予任務有不同的能力。



透過平台賦於任務



多樣性的觸發執行

平台賦予任務多樣的觸發方式，例如時間觸發、事件觸發、API 觸發等，讓任務執行更靈活。



彈性的任務流程控制與交互

平台提供靈活的流程控制機制，靈活控制任務之間的依賴，支持跨任務的交互與資料傳遞，實現複雜的流程管理。



容易的部署方式

平台提供快速的部署方式，例如定義好 YAML 文件後透過 gitops 部署，快速的將任務部署到平台上。

讓開發者專注於業務領域

專注於業務流程與邏輯

團隊或開發者可以安心專注於業務流程與邏輯，不必煩惱技術細節。

平台負責技術細節

平台負責管理基礎架構、部署、監控，讓開發者專心創造價值。

名詞定義

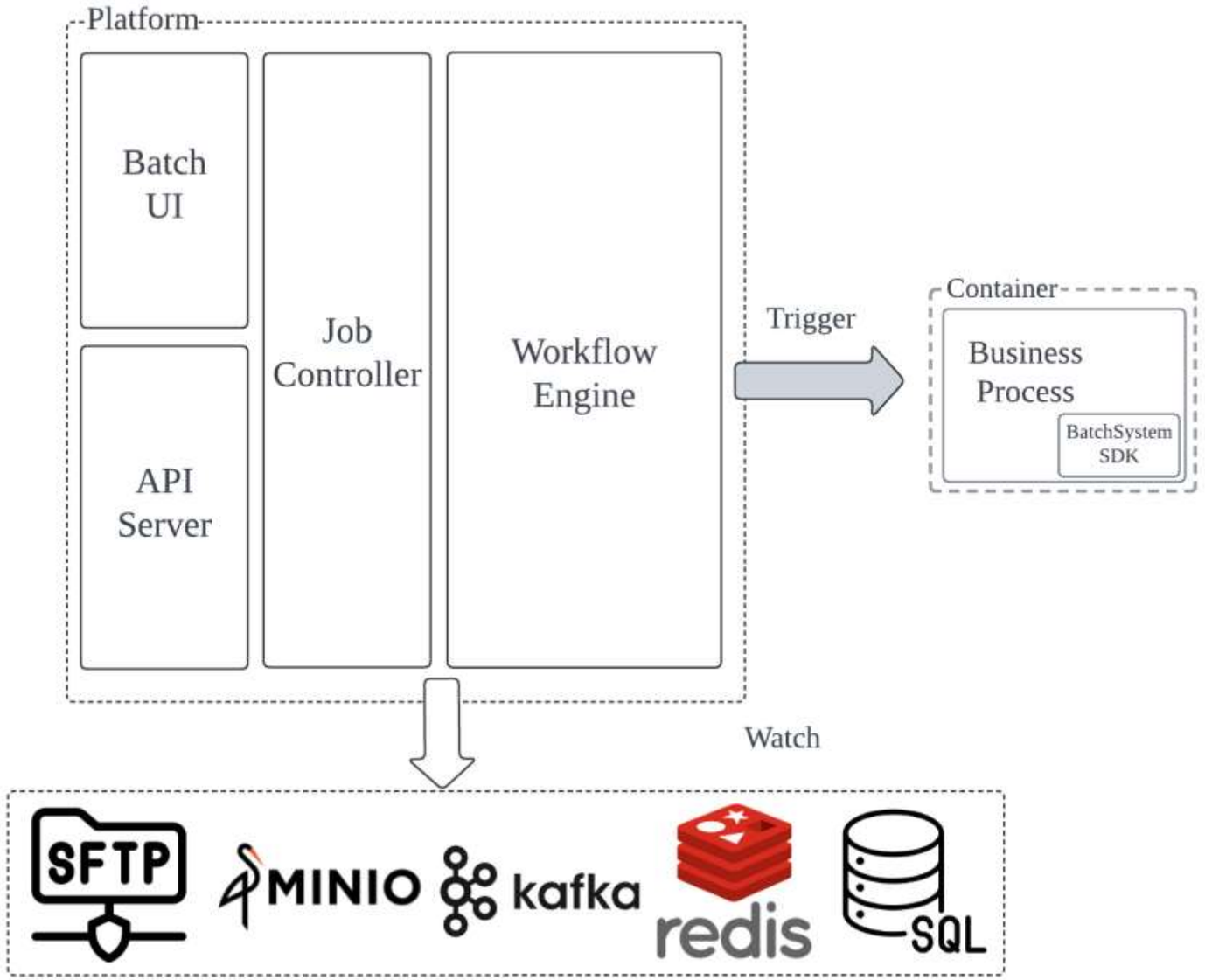
Workflow

等同於一個任務或是 Job，Workflow 可以定義每一個任務 (Job) 的每一個步驟

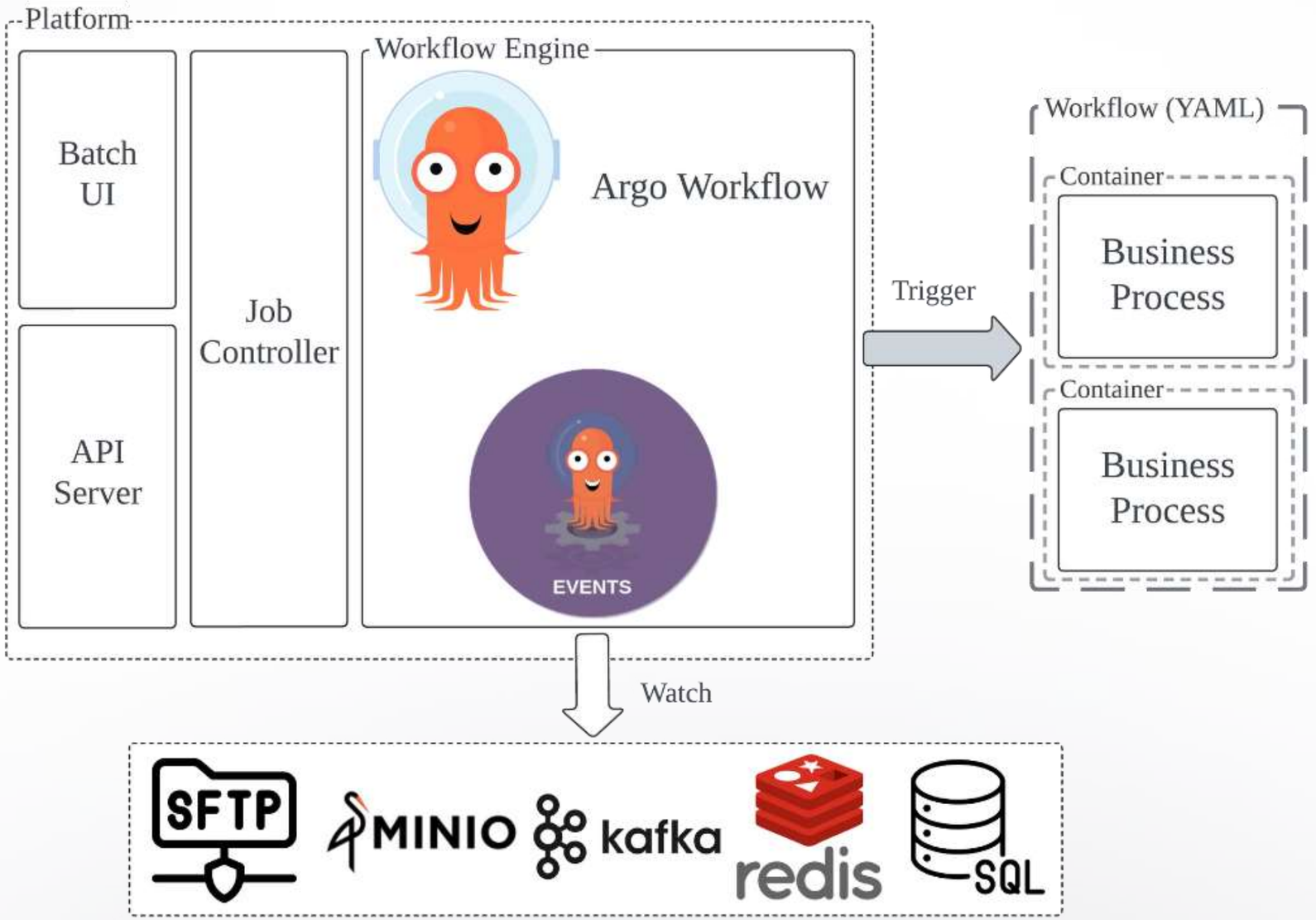
```yaml
kind: Workflow
metadata:
  generateName: multi-steps
spec:
  entrypoint: hello-hello-hello
  templates:
    - name: hello-hello-hello
      steps:
        - - name: hello1
            template: whalesay
            arguments:
              parameters:
                - name: message
                  value: "hello1"
        - - name: hello2a
            template: whalesay
            arguments:
              parameters:
                - name: message
                  value: "hello2a"
          - name: hello2b
            template: whalesay
```

以平台的視角來設計的 Batch System

Batch System 架構圖



加入 Argo


Platform
Batch UI
API Server
Job Controller
Workflow Engine
Argo Workflow
EVENTS
Trigger
Workflow (YAML)
Container
Business Process
Container
Business Process
Watch
SFTP
MINIO
kafka
redis
SQL

Argo Workflow簡介

工作流編排

Argo Workflow是Kubernetes原生的工作流引擎。

任務管理

支持複雜批處理任務的定義和執行。

可視化界面

提供直觀的工作流監控和管理界面。



Argo Workflow的核心概念

工作流定義

使用YAML文件定義工作流結構和步驟。

模板復用

支持模板定義和復用提高開發效率。

並行執行

支持任務的並行處理,提高執行效率。

Argo Workflow 常見的模板類型



Container 模板

Container 模板用於執行容器。



Script 模板

Script 模板允許執行腳本。



Resource 模板

Resource 模板可用於創建或刪除 Kubernetes 資源。



HTTP 模板

HTTP 模板可以容易的呼叫外部的 API

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  labels:
    workflows.argoproj.io/archive-str
  annotations:
    workflows.argoproj.io/description
      This is a simple hello world ex
  namespace: argo
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["hello world"]
```

Container 模板

封裝任務

把需要執行的任務打包成容器，方便重複使用和管理

隔離環境

每個模板都執行在各自的容器裡，不同任務的環境互不干擾

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: script-workflow-
spec:
  entrypoint: script-example
  templates:
  - name: script-example
    steps:
    - - name: run-script
        template: run-bash-script

  - name: run-bash-script
    script:
      image: alpine:3.7
      command: [sh]
      source: |
        echo "Hello, this is a script
```

Script 模板

快速執行腳本

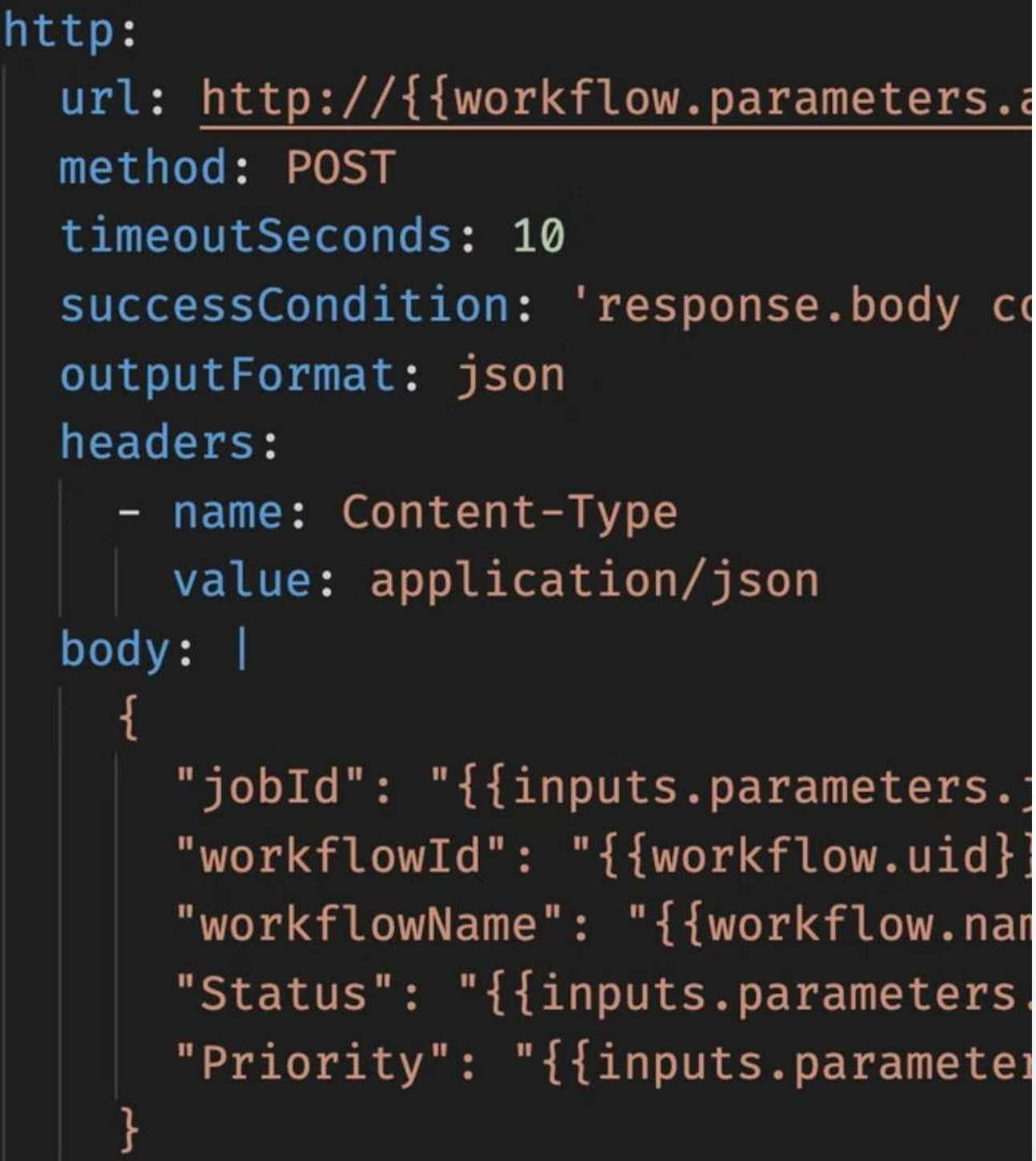
允許你在工作流中直接寫腳本來執行一些簡單的命令或邏輯處理。

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: k8s-jobs-
spec:
  entrypoint: pi-tmpl
  templates:
  - name: pi-tmpl
    resource:                   # indicates that this is a
      action: create            # can be any kubectl actio
      successCondition: status.succeeded > 0
      failureCondition: status.failed > 3
      manifest: |               #put your kubernetes spec
        apiVersion: batch/v1
        kind: Job
        metadata:
          generateName: pi-job-
        spec:
          template:
            metadata:
              name: pi
            spec:
              containers:
              - name: pi
                image: perl
                command: ["perl",  "-Mbignum=bpi", "-wle",
              restartPolicy: Never
          backoffLimit: 4
```

Resource 模板

管理 Kubernetes 資源

直接在工作流中操作 Kubernetes 物件



Http 模板

簡潔的 API 呼叫機制

Argo Workflow 提供了簡潔的 API 呼叫機制。

輕鬆地與外部服務交互

讓您可以在工作流中輕鬆地與外部服務交互。

Argo Workflow 兩種流程管理方式



Step

步驟式的線性流程，任務按步驟順序依次執行。每一步的任務可以是串行的，也可以是並行的，但其執行順序由步驟結構來決定。



DAG（有向無環圖）

用來指定每個任務之間的依賴關係。這種方法可以讓複雜的工作流程更容易維護，並且可以實現任務之間的最大平行運行。

```yaml
kind: Workflow
metadata:
  generateName: multi-steps
spec:
  entrypoint: hello-hello-hello
  templates:
    - name: hello-hello-hello
      steps:
        - - name: hello1
            template: whalesay
            arguments:
              parameters:
                - name: message
                  value: "hello1"
        - - name: hello2a
            template: whalesay
            arguments:
              parameters:
                - name: message
                  value: "hello2a"
          - name: hello2b
            template: whalesay
```

Step

簡單的執行流程

任務接續執行，適合順序明確的工作流。

彈性的控制

允許在每個步驟內定義多個並行任務，但步驟之間的關聯仍然是按順序進行。

```yaml
kind: Workflow
metadata:
  generateName: dag-diamond-
spec:
  entrypoint: diamond
  templates:
  - name: diamond
    dag:
      tasks:
      - name: A
        template: echo
        arguments:
          parameters: [{name: message, value: A}]
      - name: B
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: B}]
      - name: C
        dependencies: [A]
        template: echo
        arguments:
          parameters: [{name: message, value: C}]
      - name: D
        dependencies: [B, C]
```

DAG

彈性且並行的任務調度

適合有複雜依賴關係的工作流，允許多個任務並行運行，只要它們的依賴條件滿足即可。

更適合處理任務依賴性

當工作流中的任務需要根據其他任務的結果來執行時，DAG 模式是最理想的選擇。

透過流程管理



任務模組化

將複雜的工作流程分解成更小的、可重用的任務，方便維護和調整。



更好的重用性

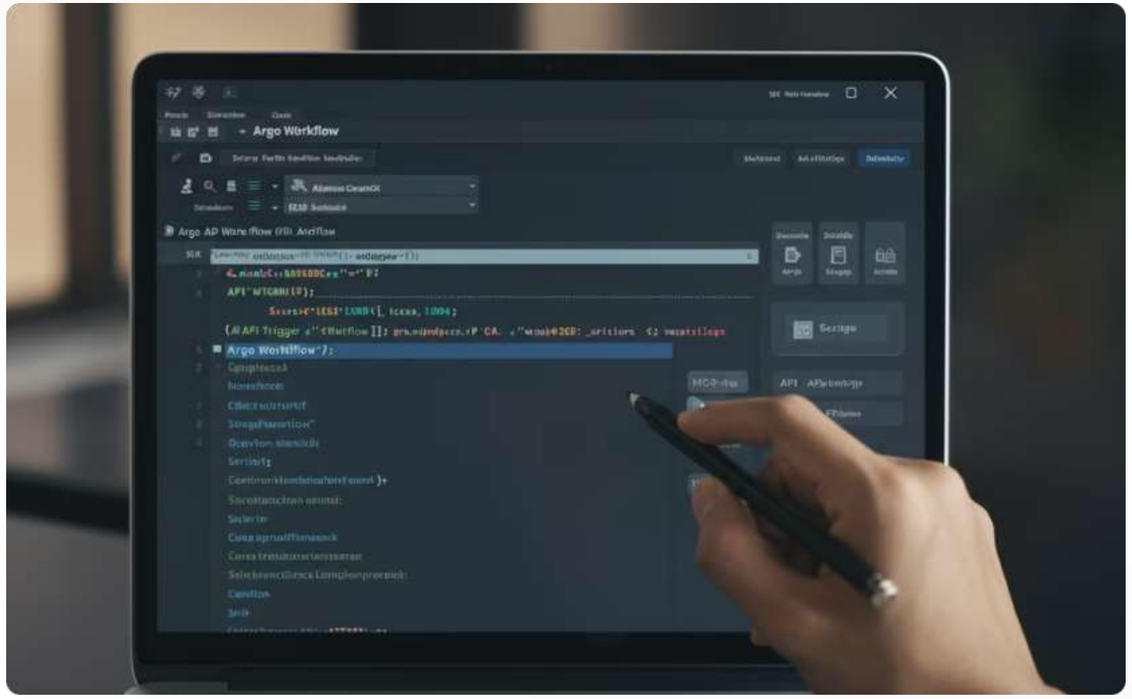
任務模組化，讓工作流程更易於重用，提高效率和靈活性。



彈性的任務組合、錯誤處理

根據不同的需求，可以輕鬆地組合和調整任務，並有當錯誤發生能夠更有效的處理。

Argo Workflow 也提供不同的任務觸發方式



API

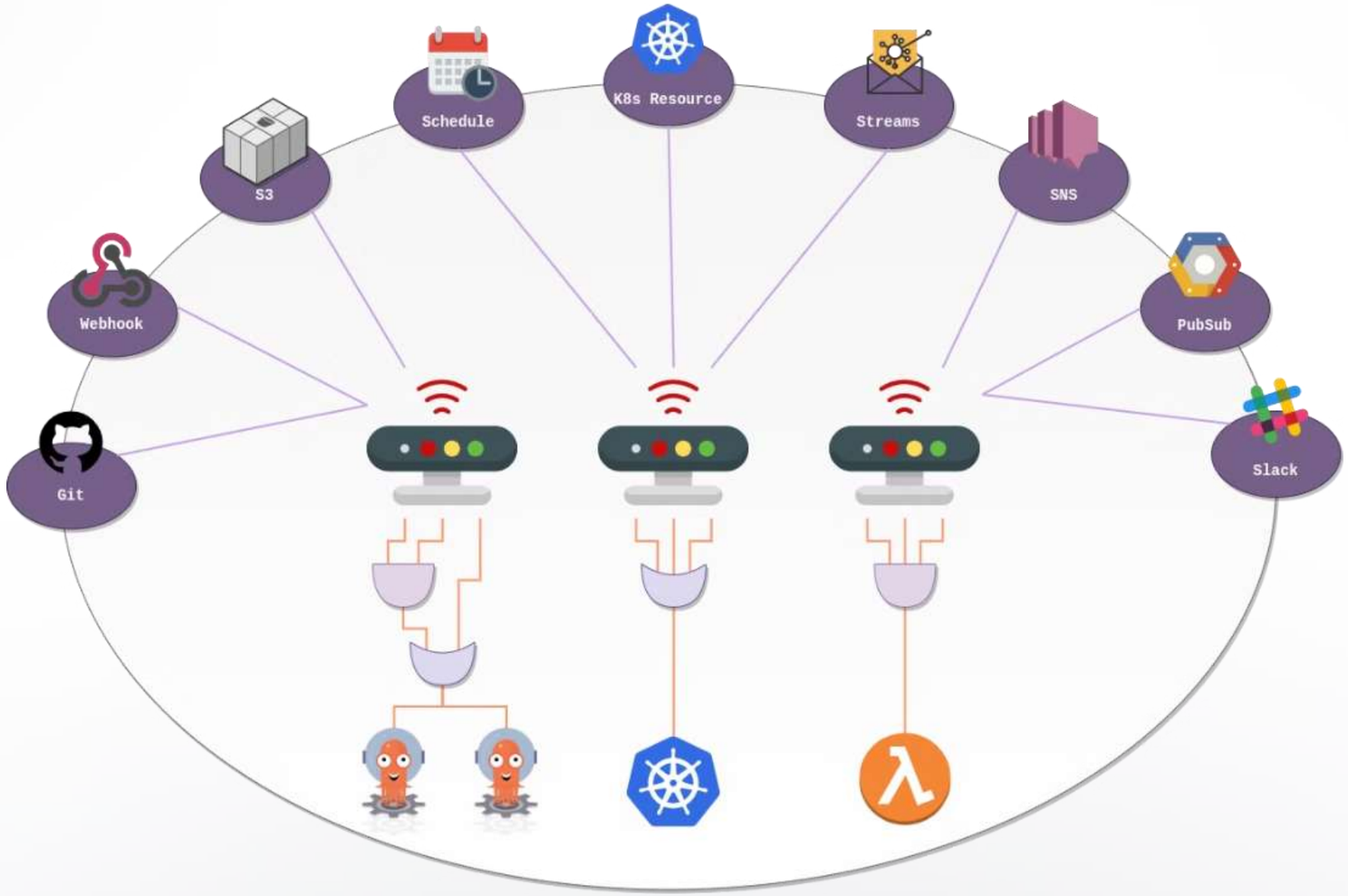
Argo Workflow 可透過 API 呼叫啟動工作流程，例如，當外部系統偵測到特定事件時，可以發送 API 請求觸發對應的工作流程執行。



Cron

您可以使用 Cron 表達式來設定工作流程的排程，例如，定期執行資料分析任務或每日發送報告。

Argo Event - 提供多樣性的資源監控



該是  (k8s) 出場的時候了

Kubernetes的優勢

容器化

實現應用程序的標準化部署和運行。

自動化

以 Resource 為中心的自動化流程

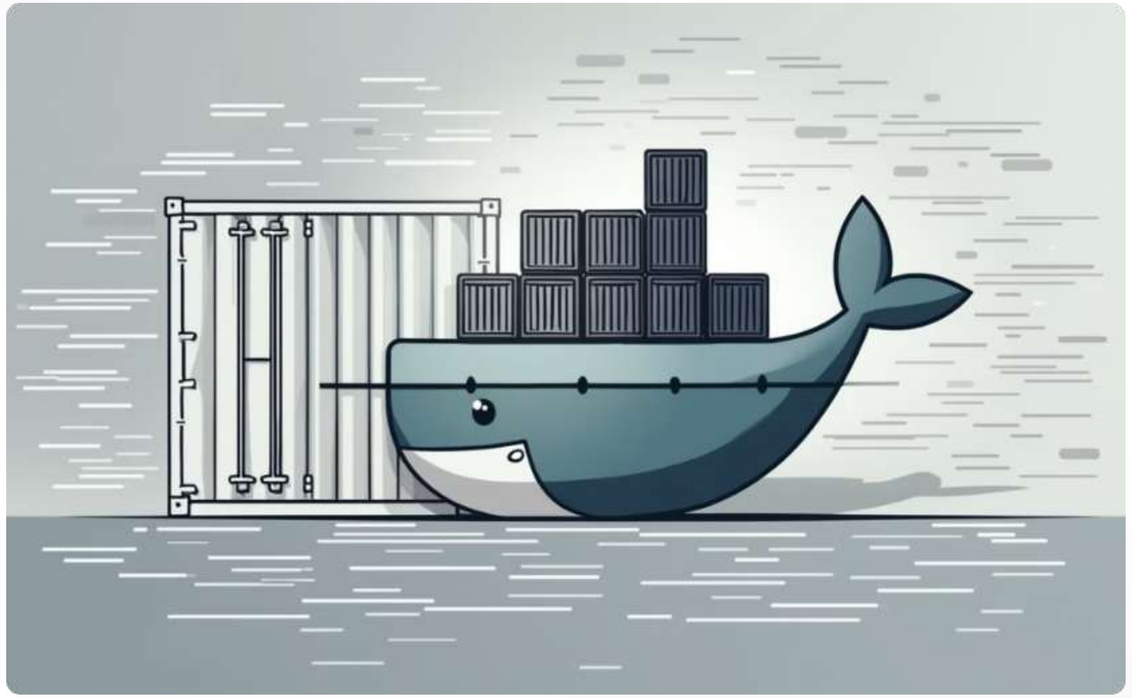
雲原生

支持多雲環境，提高系統靈活性。



kubernetes

Kubernetes 賦予平台的優勢



容器化

容器化後讓平台跳脫了語言的束縛，使平台能夠運行各種語言開發的應用。



自動化

平台可以根據 Resource 的類型、異動來推動系統資料的變更，實現自動化的資源管理，提高效率和可靠性。

系統設定管理與平台結合

透過 GitOps 與 k8s Operator

透過 GitOps 與 k8s Operator 來達成系統內的任務版本控管以及任務的新增、異動

將 GitOps 加入流程

1 **任務定義**

在Git倉庫中定義任務。

2 **版本控制**

利用Git進行任務版本管理。

3 **自動部署**

通過 ArgoCD 自動更新集群中的任務定義。

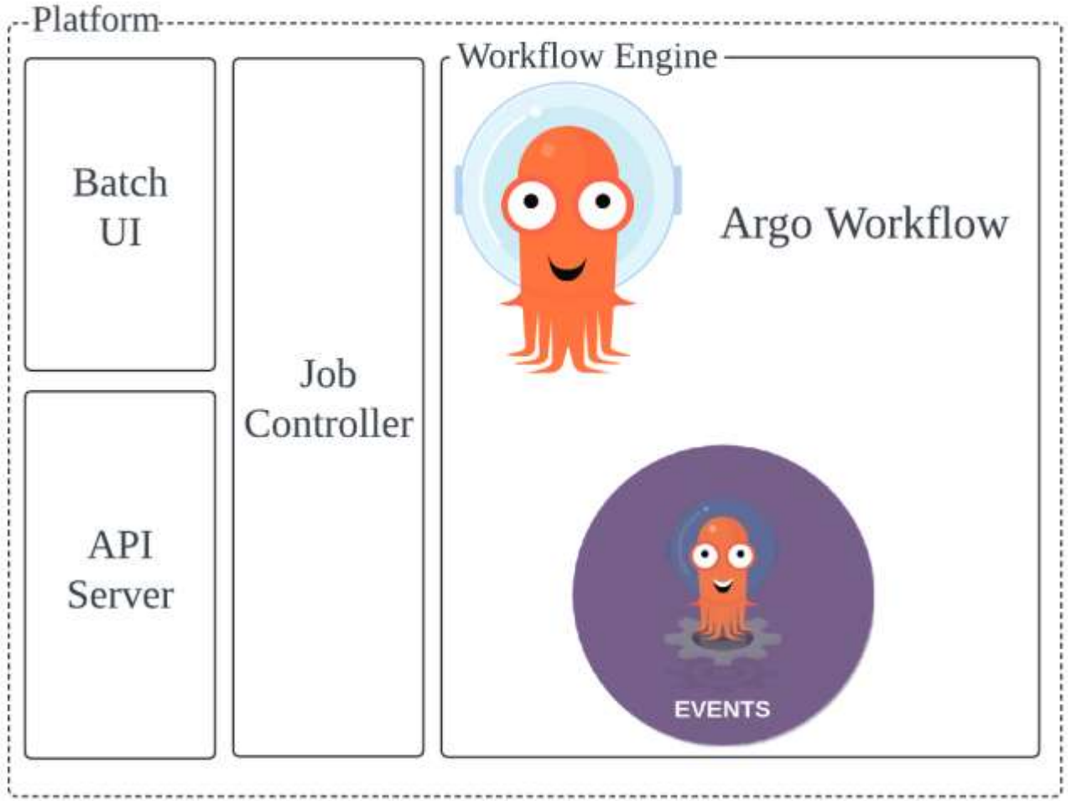


https://reurl.cc/NlDKxk


{DEV}
YAML
Push
GitLab
argo
WorkflowEngine
argo workflows
EVENTS

DEMO

Platform
怎麼知道有新的
Schedule Job?



Kubernetes CRD和CR的應用

自定義資源定義(CRD)

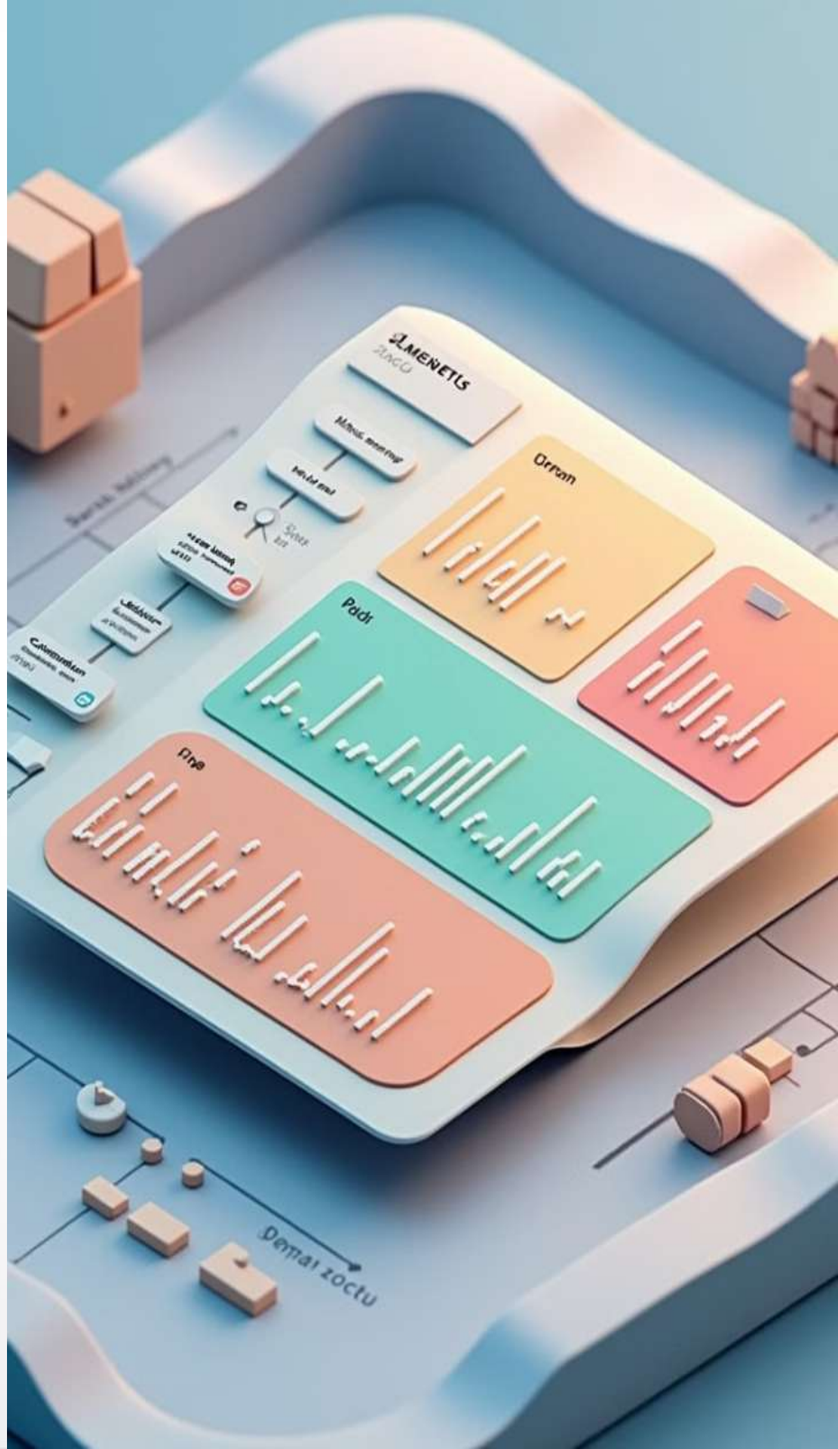
定義新的資源類型,擴展Kubernetes API。

自定義資源(CR)

CRD的實例，用於描述具體的批處理任務。

與自定義的 Operator 整合

使用CR觸發和管理 Schedule Job。



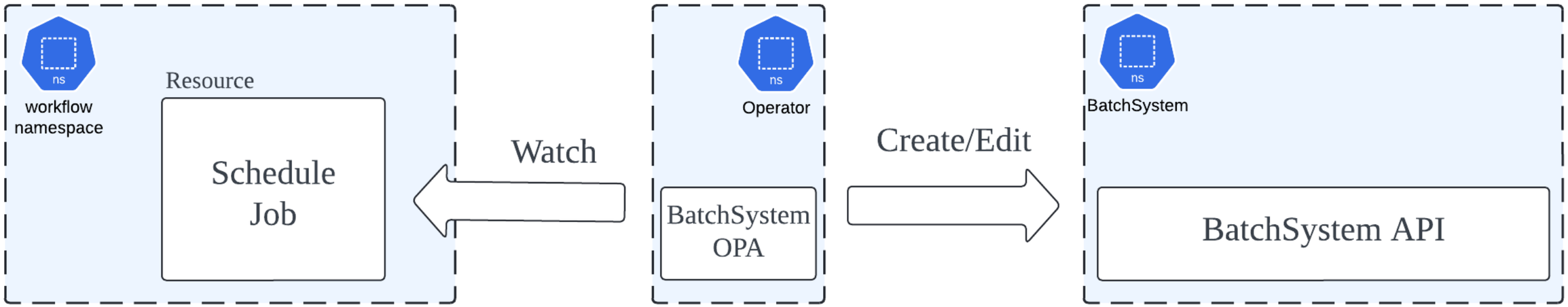
搭配 K8S Operator

監控特定 CRD

通過監控 Namespace 下特定的 CRD 來獲取 Batch System 的狀態資料

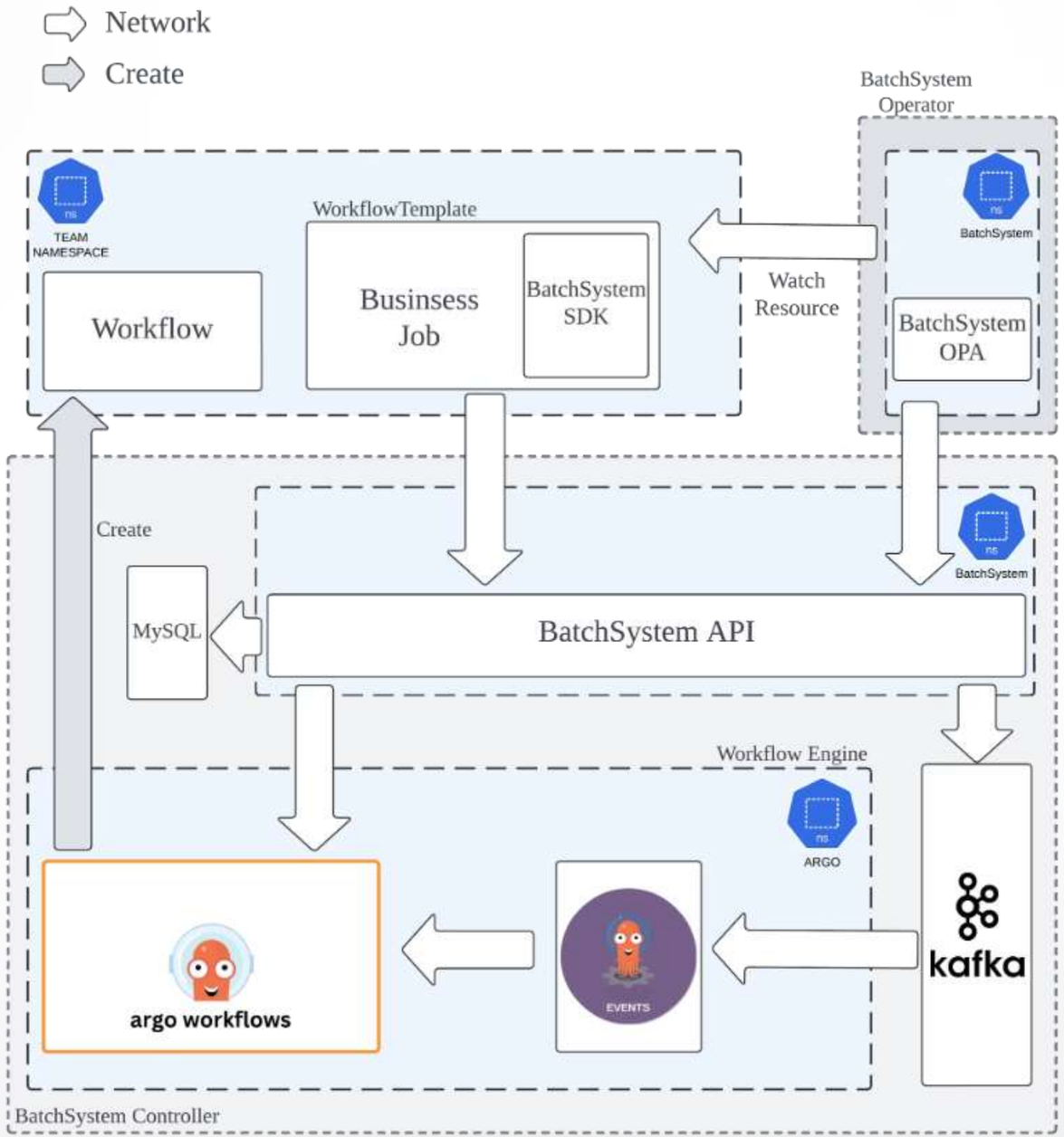
更合理的流程

將 Schedule Job 在系統的建立統一入口，並且自動化



DEMO

Batch System Final.....



平台化的優勢

1 **任務服務化**

實現 Job as a Service

2 **事件驅動**

更容易地達成事件驅動，而不是傳統的時間驅動

3 **資源利用**

優化資源分配,提高集群利用率。

4 **統一監控告警**

Batch System 集中，容易監控與告警



Please think about....