



01

## Integration Style – Event Centric

Kafka Usage

02

## Integration Style – Data Centric

Data Hub Usage

03

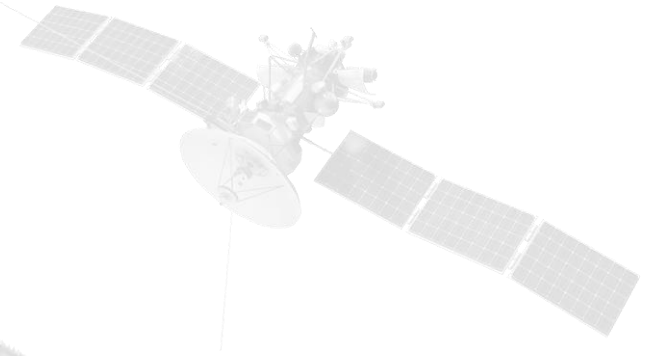
## Integration Style – Application Centric

API Usage

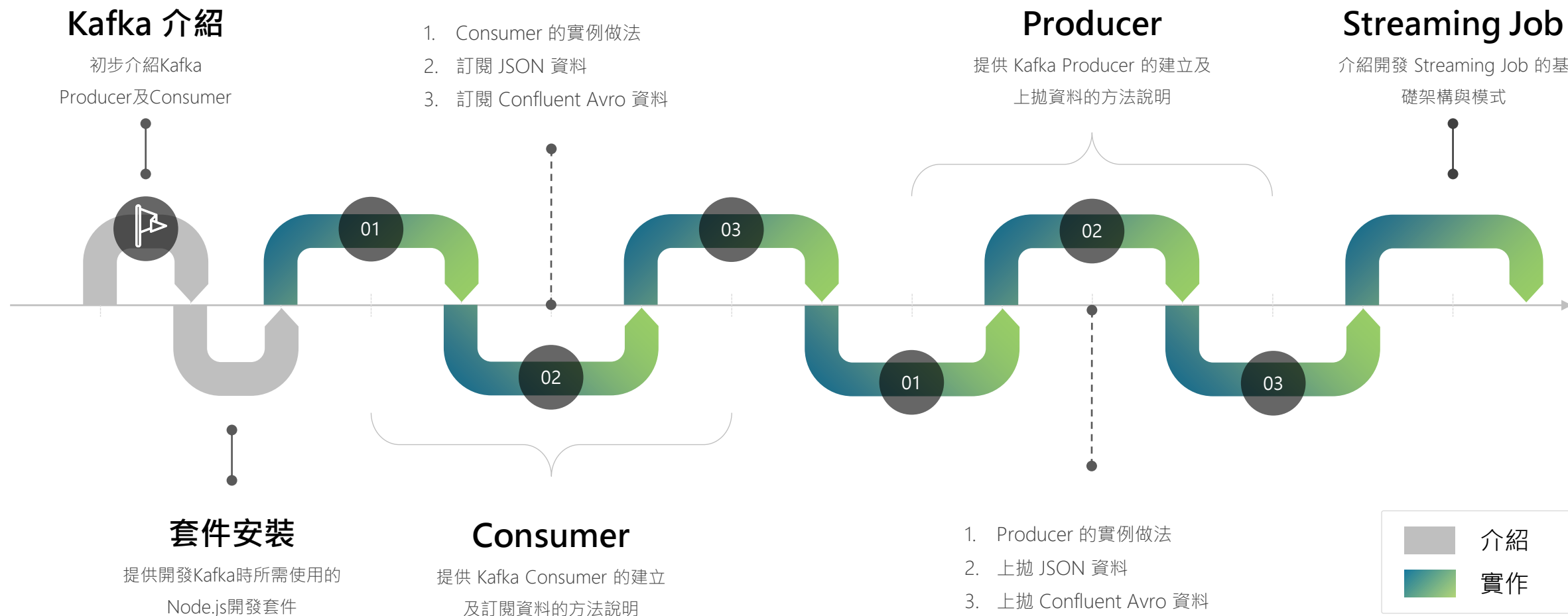
# DATA INTEGRATION

Integration Style – Event Centric

Kafka Usage



# Course Roadmap

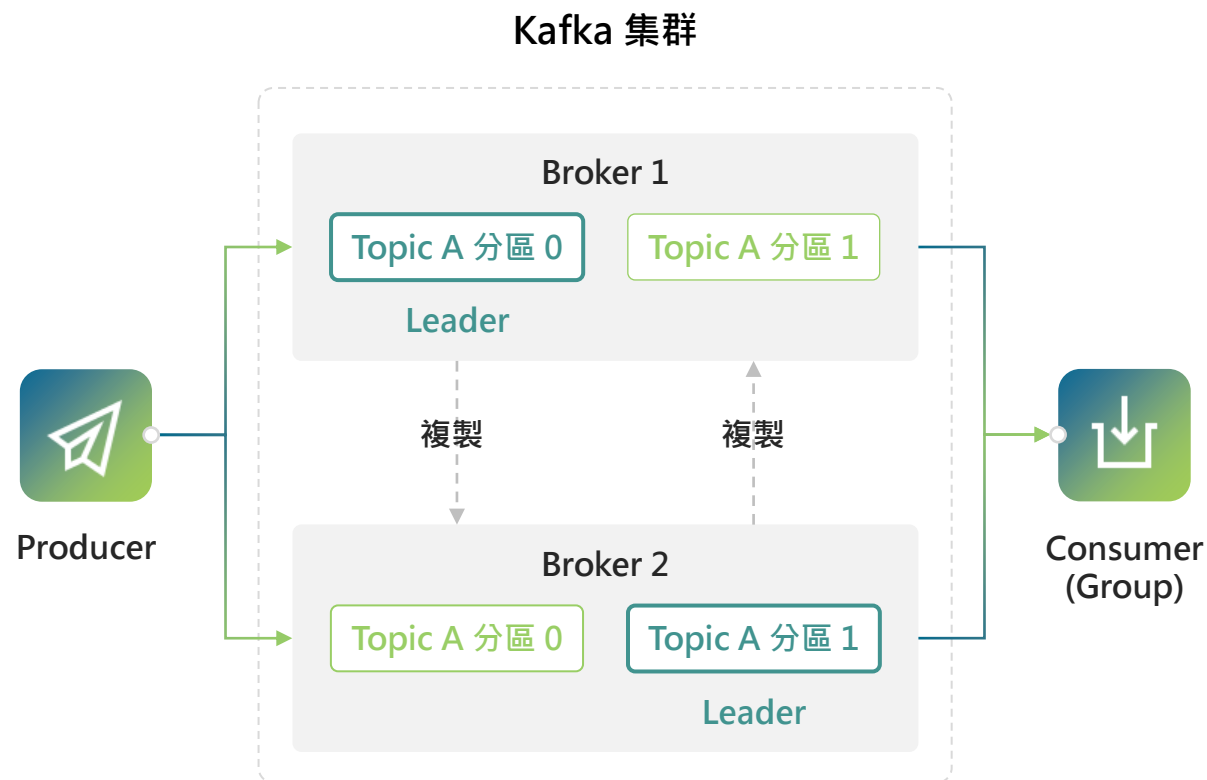


# Kafka 介紹

## Consumer & Producer

**Producer** 生產者，即訊息的釋出者，其會將某 topic 的訊息釋放到相應的 partition 中。生產者在預設情況下把訊息均衡地分佈到主題的所有分割槽上，而並不關心特定訊息會被寫到哪個分割槽。

**Consumer** 消費者，即訊息的使用者，一個消費者可以消費多個 topic 的訊息，對於某一個 topic 的訊息，其只會消費同一個 partition 中的訊息



## Install Node.js Kafka Package

此次範例將使用 Typescript 作為 Node.js 開發的主要程式語言，並使用套件 kafka-node 以及 wisrtoni40-confluent-schema 進行 Kafka 數據串接。

Kafka-node 安裝



```
npm install kafka-node
```

wisrtoni40-confluent-schema 安裝



```
npm install wisrtoni40-confluent-schema
```

# Kafka Consumer

## 引入 Consumer

```
import { ConsumerGroup } from 'kafka-node';
```

```
import { v4 as uuidv4 } from 'uuid';
```

```
const kafkaHost = 'localhost:9092,localhost:9093,localhost:9094';
```

```
const topic = 'your.topic';
```

```
const consumer = new ConsumerGroup({ 實例 Kafka Consumer
```

```
  kafkaHost, Kafka Hosts · 可用「,」添加多台 Host
```

```
  groupId: uuidv4(), Kafka Consumer Group ID
```

```
  fromOffset: 'latest', latest: 訂閱最新數據 ; earliest: 訂閱最早數據
```

```
  sasl: { SASL 驗證連線設置
```

```
    mechanism: 'plain',
```

```
    username: 'username',
```

```
    password: 'password',
```

```
  },
```

```
}, topic) 要訂閱的 Topic
```

## Consumer 配置

# Kafka Consumer

訂閱的資料使用一般的 JSON

JSON 解析策略

```
import { JsonSubResolveStrategy } from 'wisrtoni40-confluent-schema';
```

```
const resolver = new JsonSubResolveStrategy();
```

實例 Kafka JSON 解析策略

Consumer 訂閱資料

```
consumer.on('message', async (msg) => {
```

Consumer 訂閱資料

```
  const result = await resolver.resolve(msg.value as string);
```

解析 Consumer 訂閱的資料並轉換為 JSON

```
  console.log(result);
```

```
});
```

# Kafka Consumer

訂閱的資料使用 Confluent Avro Schema

Confluent 解析策略

```
import { ConfluentAvroStrategy, ConfluentMultiRegistry, ConfluentSubResolveStrategy } from 'wisrtoni40-confluent-schema';
```

```
const registryHost = 'http://localhost:8084,http://localhost:8085,http://localhost:8086';
```

 Confluent Schema Registry · 可用「,」添加多台 Host

```
const schemaRegistry = new ConfluentMultiRegistry(registryHost);
```

```
const avro = new ConfluentAvroStrategy();
```

```
const resolver = new ConfluentSubResolveStrategy(schemaRegistry, avro);
```

 實例 Kafka Confluent 解析策略

Consumer 訂閱資料

```
consumer.on('message', async (msg) => {
```

 Consumer 訂閱資料

```
  const result = await resolver.resolve(msg.value as string);
```

 解析 Consumer 訂閱的資料並轉換為 JSON

```
  console.log(result);
```

```
});
```



# Kafka Producer

## 引入 Producer 及 Client

```
import { HighLevelProducer, KafkaClient } from 'kafka-node';  
  
import { v4 as uuidv4 } from 'uuid';
```

## Kafka Client 配置

```
const kafkaHost = 'localhost:9092';
```

```
const kafkaClient = new KafkaClient({ 實例 Kafka Client
```

```
  kafkaHost, Kafka Hosts · 可用「,」添加多台 Host
```

```
  clientId: uuidv4(), Kafka Group ID
```

```
  sasl: { SASL 驗證連線設置
```

```
    mechanism: 'plain',
```

```
    username: 'username',
```

```
    password: 'password',
```

```
  },
```

```
});
```

## Producer 配置

```
const producer = new HighLevelProducer(kaf 透過 Kafka Client · 實例 Producer
```

# Kafka Producer

發送的資料使用一般的 JSON

JSON 解析策略

```
import { JsonPubResolveStrategy } from 'wisrtoni40-confluent-schema';
```

```
const resolver = new JsonPubResolveStrategy();
```

 實例 Kafka JSON 解析策略

```
(async () => {
```

```
  const data = { ... };
```

```
  const messages = await resolver.resolve(data);
```

 上拋前，將資料解析成 JSON String

```
  const topic = 'your.topic';
```

 資料的目標 Topic

```
  const key = 'topic.key';
```

 上拋資料的 Key

```
  const payload = { topic, messages, key };
```

 將資料整理成所需的上拋格式

```
  producer.send(payload, (error, result) => {
```

 透過 Producer 將資料上拋，並監聽上拋結果

```
    // TODO
```

```
  });
```

```
})();
```

Producer 發送資料

# Kafka Producer

發送的資料使用 Confluent Avro Schema

JSON 解析策略

```
import { ConfluentMultiRegistry, ConfluentAvroStrategy, ConfluentPubResolveStrategy } from 'wisrtoni40-confluent-schema';
```

```
const topic = 'your.topic';
```

資料的目標 Topic

```
const registryHost = 'http://localhost:8084,http://localhost:8085';
```

Confluent Schema Registry · 可用「,」添加多台 Host

```
const schemaRegistry = new ConfluentMultiRegistry(registryHost);
```

```
const avro = new ConfluentAvroStrategy();
```

```
const resolver = new ConfluentPubResolveStrategy(schemaRegistry, avro, topic);
```

實例 Kafka Confluent 解析策略

```
(async () => {
```

```
  const data = { ... };
```

```
  const messages = await resolver.resolve(data);
```

上拋前 · 將資料解析成 Confluent Avro

```
  const key = 'topic.key';
```

上拋資料的 Key

```
  const payload = { topic, messages, key };
```

將資料整理成所需的上拋格式

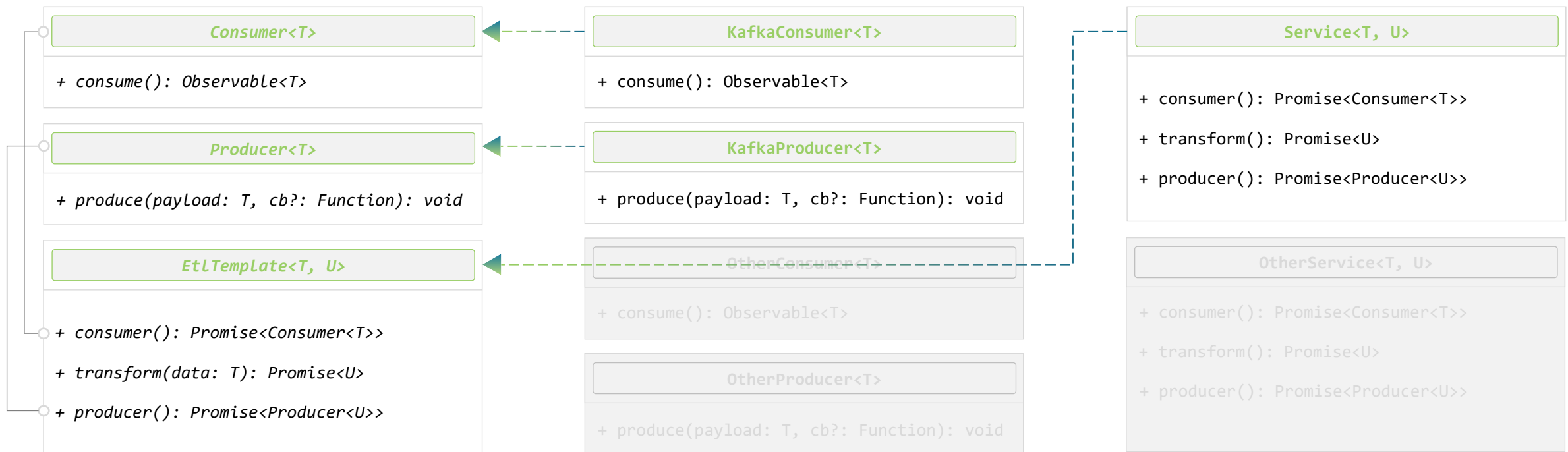
```
  producer.send(payload, (error, result) => {
```

透過 Producer 將資料上拋 · 並監聽上拋結果

```
  })();
```

Producer 發送資料

# Streaming Job Design

**core****shared****Application**

# END

Thank you for listening



簡報者：Steve CY Lin



STEVE\_CY\_LIN@WISTRON.COM



01

## Integration Style – Event Centric

Kafka Usage

02

## Integration Style – Data Centric

Data Hub Usage

03

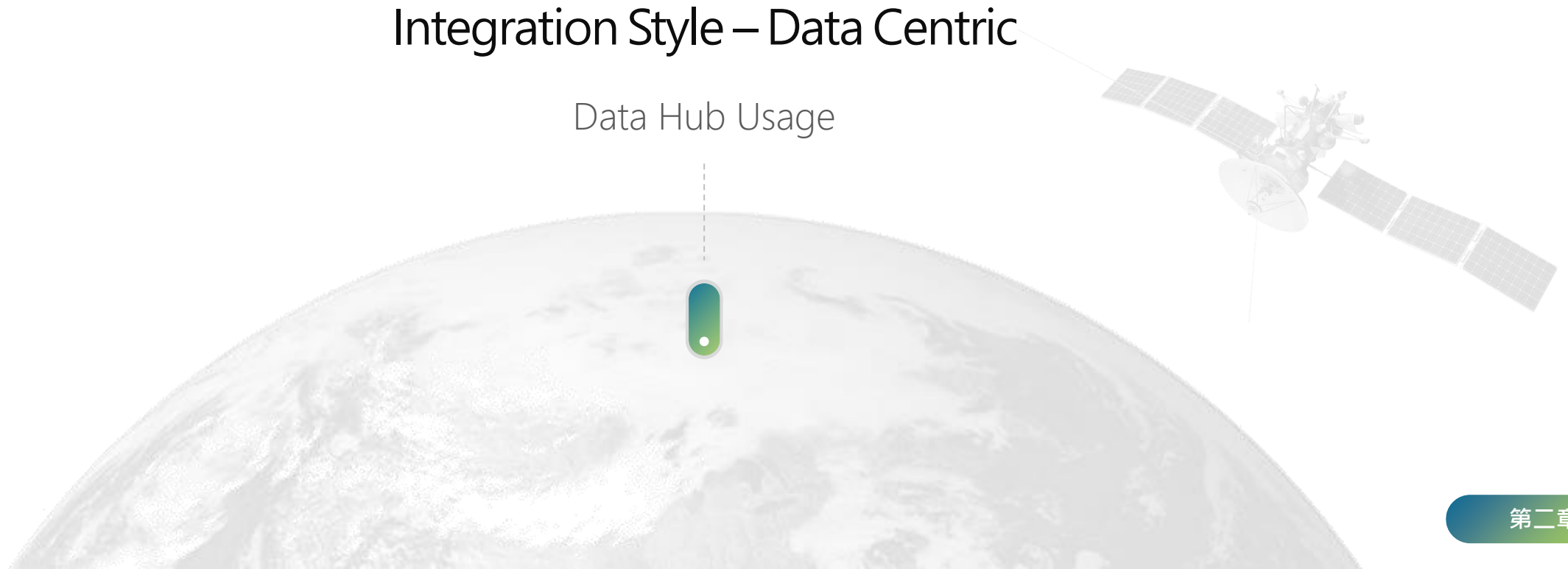
## Integration Style – Application Centric

API Usage

# DATA INTEGRATION

Integration Style – Data Centric

Data Hub Usage

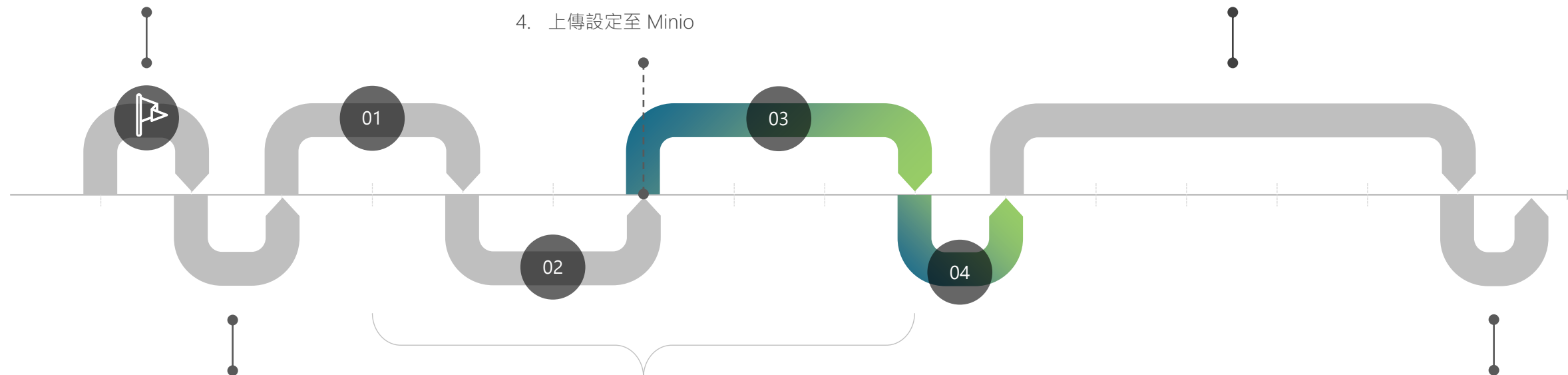


# Course Roadmap

## Data Hub 介紹

初步介紹 Data Governance

整體架構



1. 查詢及申請訂閱 Topic
2. 定義地端數據庫 Schema
3. 設定 Airflow
4. 上傳設定至 Minio

## Airflow 介紹

介紹在 Topic 設定完畢後，如何透過 Airflow 檢視資料流

介紹

實作

## 開發流程

介紹 Data Hub 的開發流程以及申請方法

## Topic 申請/設定

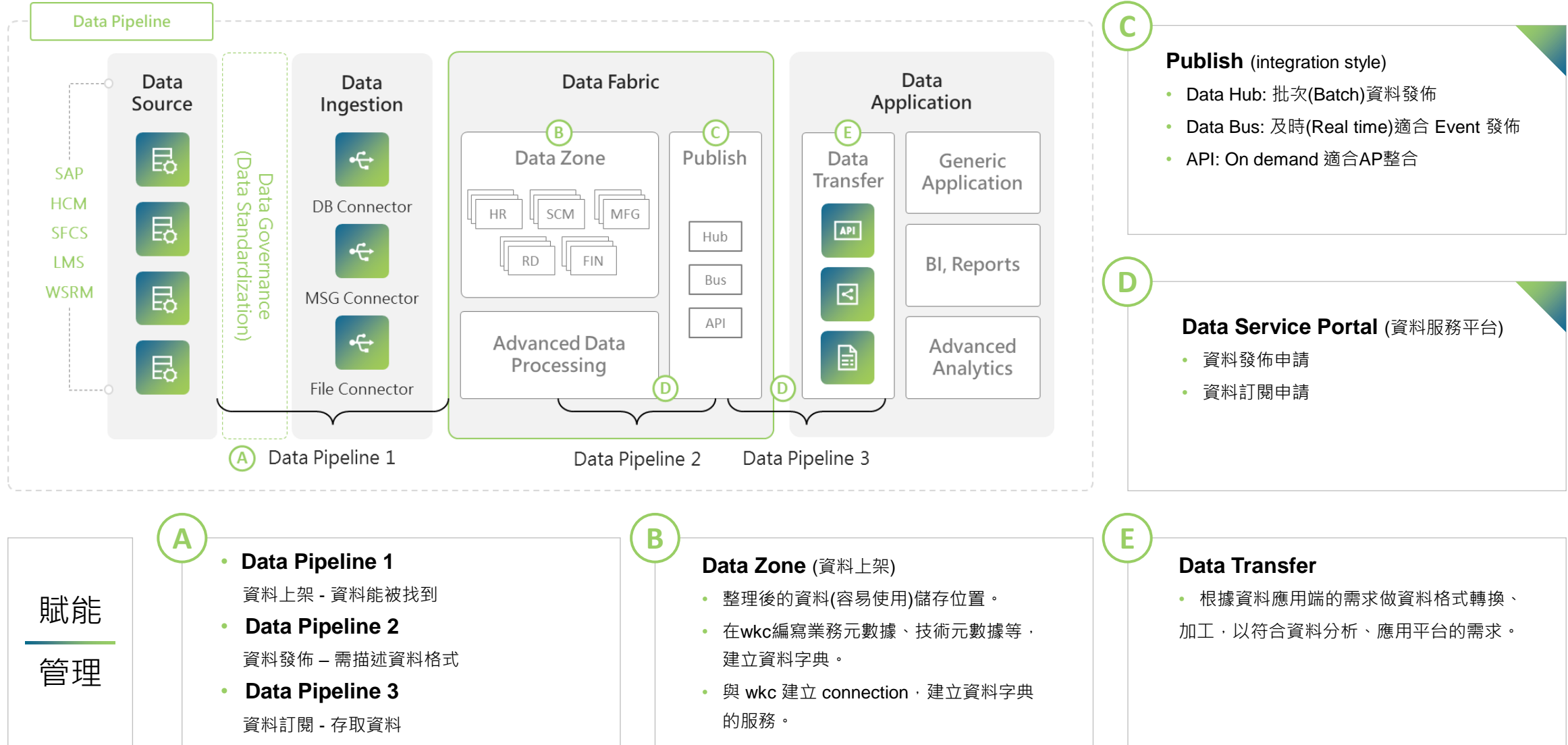
介紹如何查詢所需的 Topic，並在申請後設定 Airflow DAG

## 問題諮詢

提供 Data Service Portal 及 Airflow 的諮詢對象



## DATA INTEGRATION



定義地端數據庫表格

Data Hub 連線資訊

上傳連線設定至 Minio

Step 1.

Step 2.

Step 3.

Step 4.

Step 5.

Step 6.

查詢及申請訂閱 Topic

數據庫連線資訊

撰寫 Python 腳本

## Step 1. 查詢及申請訂閱 Topic



### 查詢 Topic

路線：Data Service Portal → 資料訂閱 → Topic 資料集資訊

目的：

1. 填寫申請表需要了解預訂閱的 Topic 隸屬於哪個資料管家
2. Topic 是否有資料集、對應 Schema 等訊息



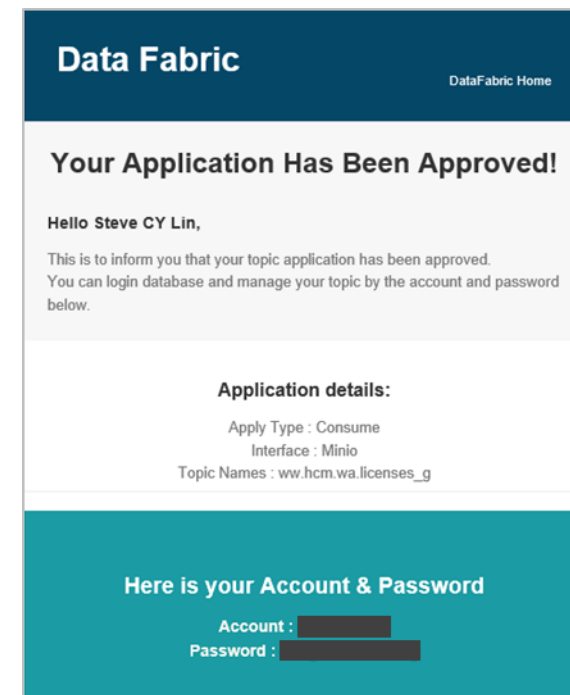
### 申請訂閱 Topic

路線：路線: Data Service Portal → 資料訂閱 → Consumer表單

目的：取得後續訂閱 Topic 的連線帳密

參考：詳細申請步驟請參考[附件](#)

申請完畢



## Step 2. 定義地端數據庫表格

進入 Data Service Portal 查詢訂閱 Topic 的 Schema

路線：Data Service Portal → 資料訂閱 → 我的訂閱列表 → 表單詳情資訊

Schema 瀏覽

| ID | 名稱      | 型別     | 精度/長度 | 刻度 | 預設值  | 描述      | 是否可空值                               |
|----|---------|--------|-------|----|------|---------|-------------------------------------|
| 1  | batchid | string | 17    |    |      | batchid | <input type="checkbox"/>            |
| 2  | bg      | string | 4     |    | NULL | bg      | <input checked="" type="checkbox"/> |
| 3  | site    | string | 6     |    | NULL | site    | <input checked="" type="checkbox"/> |

根據 Schema 定義數據庫 Table

注意：Schema 不會註明 **pkid 的欄位**，需自行添加，型別為 int8

Table Name:

Tablespace:

Partition by:

Comment:

Object ID:

Owner: [dtp](#)

☐ Partitions

Extra Options:

Columns

Constraints

Foreign Keys

Indexes

Dependencies

References

Partitions

Triggers

Rules

Statistics

Permissions

DDL

Virtual

| Column Name        | #  | Data type | Identity | Collation | Not Null |
|--------------------|----|-----------|----------|-----------|----------|
| 123 pkid           | 1  | int8      |          |           | [v]      |
| abc batchid        | 2  | text      |          | default   | [v]      |
| abc bu             | 3  | text      |          | default   | [v]      |
| abc bg             | 4  | text      |          | default   | [v]      |
| abc site           | 5  | text      |          | default   | [v]      |
| abc plant          | 6  | text      |          | default   | [v]      |
| abc location       | 7  | text      |          | default   | [v]      |
| abc emplid         | 8  | text      |          | default   | [v]      |
| abc name           | 9  | text      |          | default   | [v]      |
| abc name_a         | 10 | text      |          | default   | [v]      |
| hire_dt            | 11 | timestamp |          |           | [v]      |
| abc sal_location_a | 12 | text      |          | default   | [v]      |
| abc company        | 13 | text      |          | default   | [v]      |
| abc deptid         | 14 | text      |          | default   | [v]      |

## Step 3. 數據庫連線資訊



# Step 4. Data Hub 連線資訊

連線 ID

Minio S3

連線描述

Data Hub 連線設定

```
{
  "conn_id": "wihi40_dtp_con_dev_airflow_delivery",
  "conn_type": "s3",
  "description": "DTP hcm.wa.employeeinfo_g1 airflow delivery",
  "host": "",
  "schema": "",
  "login": <your_login_id>,
  "password": <your_password>,
  "port": "",
  "extra": {}
}
```

須與檔案名稱一致

Airflow 與 Minio 綁定使用，請 Story Type 是使用 S3

無須填寫

無須填寫

無須填寫

無須填寫

## Step 5. 撰寫 Python 腳本 - 1

引入 Airflow DAG 套件

```
from airflow import DAG  
  
from airflow.utils.dates import days_ago  
  
from datahub.operators.datahub_to_rdb import DatahubToRDBOperator  
  
from datetime import datetime, timedelta
```

預設參數配置

```
default_args = {  
    'owner': '工號/英文名', owner 是 DAG的開發者  
    # 'retries': 3, 失敗重新拋送最大次數  
    # 'retry_delay': timedelta(minutes=2), 失敗重新拋送間隔時間  
    'email_on_failure': True, 失敗是否開啟郵件通知  
    'email': ['steve_cy_lin@wistron.com'] 失敗郵件通知對象  
}
```

## Step 5. 撰寫 Python 腳本 - 2

DAG 配置

```
dag = DAG(  
    dag_id='wihi40_dtp_con_dev_employee_info', DAG ID  
    schedule_interval='00 02 * * *', DAG 執行週期  
    start_date=days_ago(1),  
    catchup=False,  
    default_args=default_args,  
    access_control={  
        'wihi40': {'can_read', 'can_edit'} DAG 歸屬那個團隊[tenant id]與權限  
    },  
    tags=['dtp employee info hub to rdb']) DAG 標籤
```




## Step 5. 撰寫 Python 腳本 - 3


Producer 配置

```
con_hubtordb_task = DatahubToRDBOperator(  
    task_id='con_employee_info', Task ID  
    datahub_conn_id='wihi40_dtp_con_dev_airflow_delivery', Data Hub 帳密的 Connection ID  
    datahub_topic='ww.hcm.wa.employeeinfo_g1_wmi', 要訂閱的topic  
    datahub_topic_version=1, 指定特定的 Schema 版本  
    rdb_type='postgres', Target 資料庫類型  
    rdb_conn_id='wihi40_dtp_con_dev_postgres_delivery', Target 數據庫的 Connection ID  
    table_name='public.datahub_employee_info', Target 資料表  
    # datahub_batch_id = '20220113000131934', 指定要拿取的資料  
    dag=dag)  
  
# task dependency  
con_hubtordb_task
```

## Step 6. 上傳連線設定至 Minio




將連線設定的 JSON 檔放入 ./connections 資料夾中





將 Python 腳本放入根目錄資料夾中

上傳




上傳

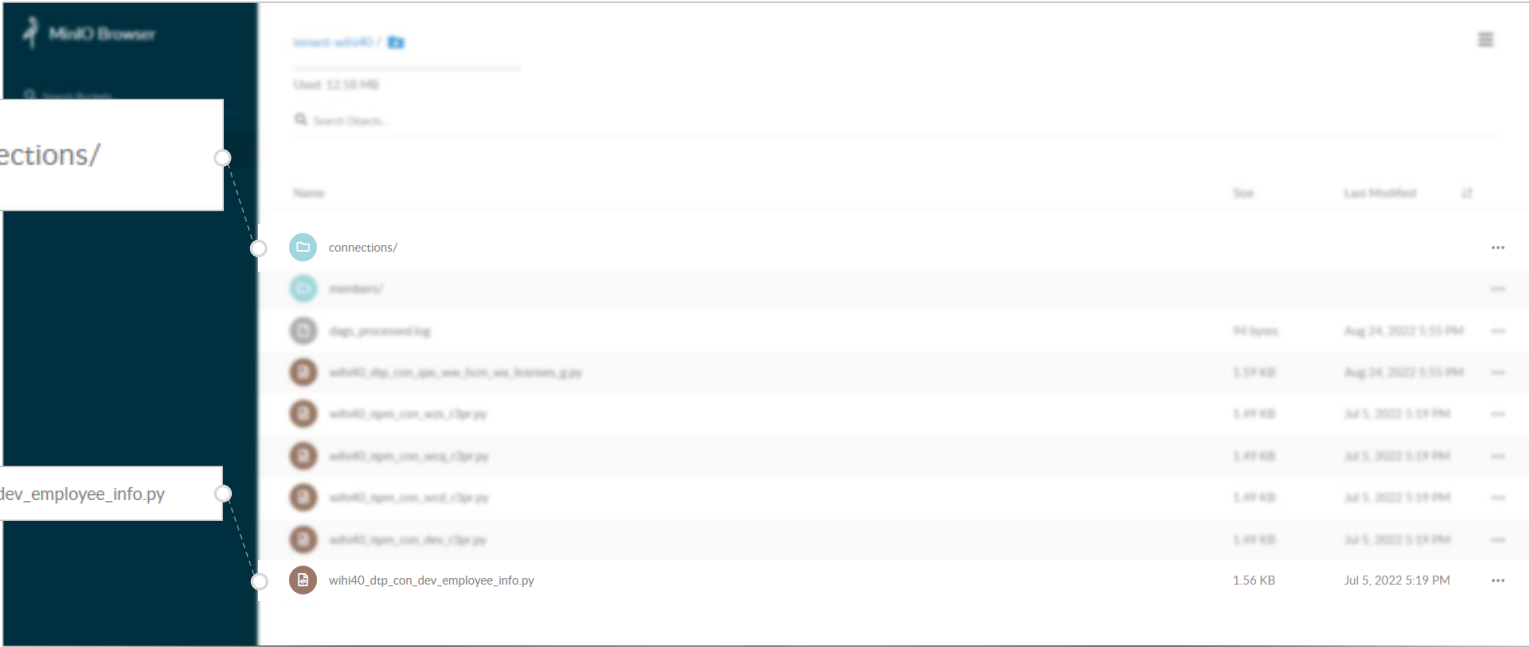




connections/



wihi40\_dtp\_con\_dev\_employee\_info.py



Minio Browser

tenant: wihi40 /

Used: 12.58 MB

Search Objects...

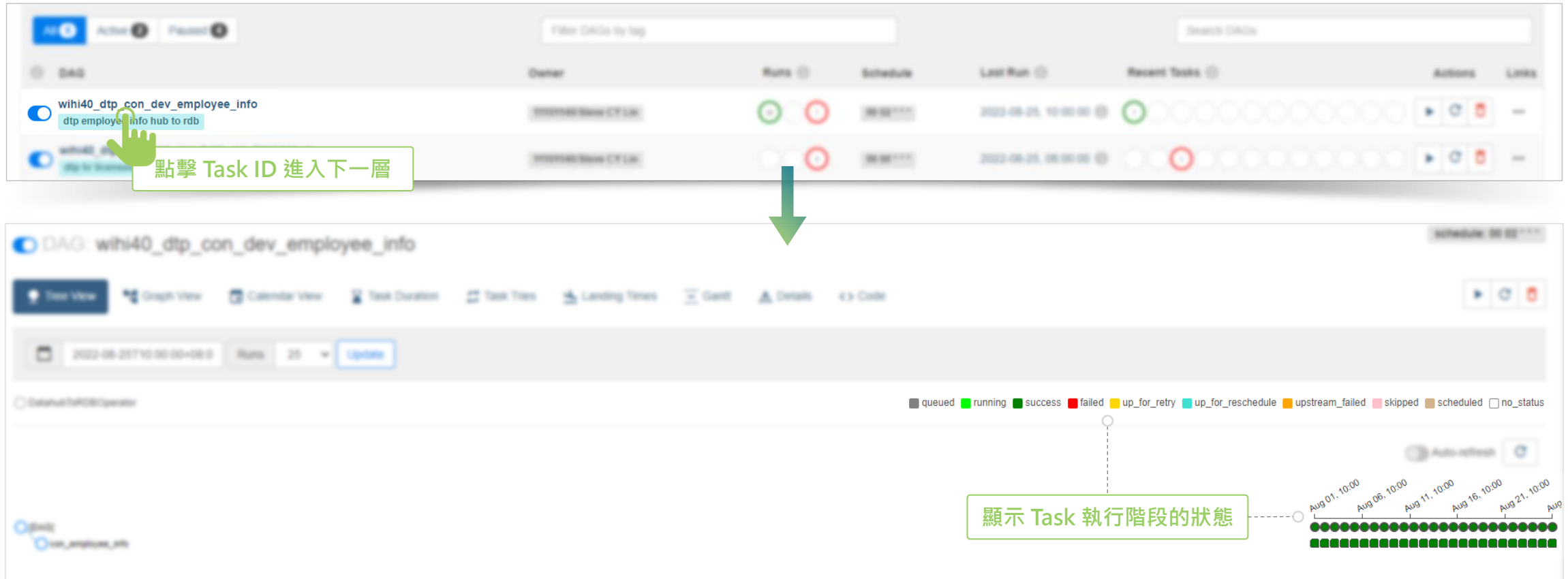
| Name                                | Size     | Last Modified        |     |
|-------------------------------------|----------|----------------------|-----|
| connections/                        |          |                      | ... |
| members/                            |          |                      | ... |
| dlaps_processed.log                 | 94 bytes | Aug 24, 2022 5:55 PM | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.59 KB  | Aug 24, 2022 5:55 PM | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.49 KB  | Jul 5, 2022 5:19 PM  | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.49 KB  | Jul 5, 2022 5:19 PM  | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.49 KB  | Jul 5, 2022 5:19 PM  | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.49 KB  | Jul 5, 2022 5:19 PM  | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.49 KB  | Jul 5, 2022 5:19 PM  | ... |
| wihi40_dtp_con_dev_employee_info.py | 1.56 KB  | Jul 5, 2022 5:19 PM  | ... |

# Airflow DAG Operation

The screenshot displays the Airflow web interface with a table of DAGs. The table columns include DAG, Owner, Runs, Schedule, Last Run, Recent Tasks, Actions, and Links. The first DAG, 'wihi40\_dtp\_con\_dev\_employee\_info', is highlighted with a green box and a hand icon pointing to the 'Open Task' button. The 'Schedule' column for this DAG shows '00 02 \*\*\*', which is also highlighted with a green box and a hand icon. The 'Actions' column for this DAG shows a green box with a hand icon pointing to the 'Manual Task Execution' button. The 'Recent Tasks' column for this DAG shows a green box with a hand icon pointing to the 'Task Scheduling Time' field.

| DAG  | Owner               | Runs | Schedule  | Last Run             | Recent Tasks | Actions | Links |
|--|---------------------|------|-----------|----------------------|--------------|---------|-------|
| wihi40_dtp_con_dev_employee_info<br>dtp employee info hub to rdb | WISTRON Data CT Ltd |      | 00 02 *** | 2023-08-25, 10:00:00 |              |         | ---   |
| ...  | ...                 | ...  | ...       | ...                  | ...          | ...     | ---   |

# Airflow DAG Operation

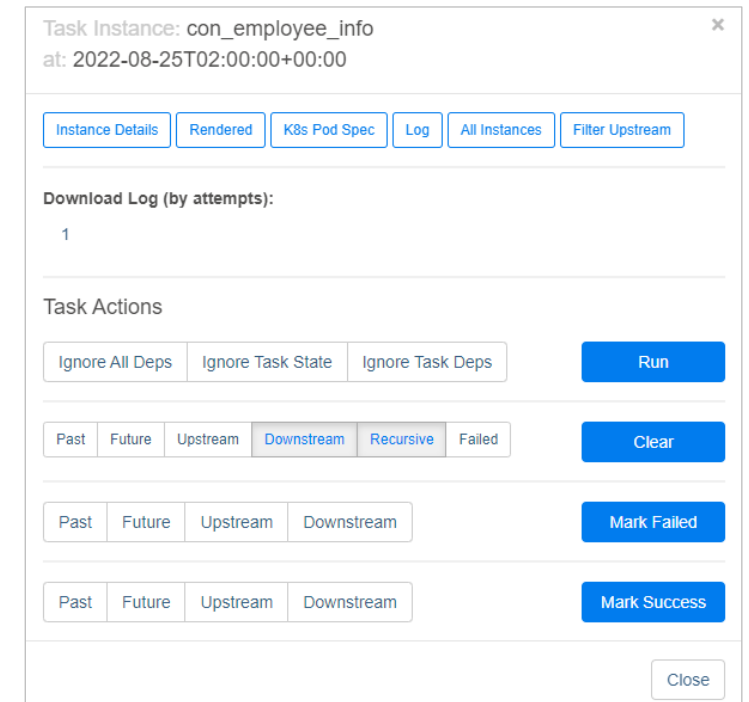


The image shows two screenshots of the Apache Airflow web interface. The top screenshot displays a list of DAGs. A green box highlights the DAG named 'wihi40\_dtp\_con\_dev\_employee\_info' with the task 'dtp employee info hub to rdb'. A green arrow points from this task to the bottom screenshot, which shows the detailed view of the DAG. In the bottom screenshot, a green box highlights the 'Task ID' field, and another green box highlights the 'Task execution status' legend, which includes states like 'queued', 'running', 'success', 'failed', 'up\_for\_retry', 'up\_for\_reschedule', 'upstream\_failed', 'skipped', 'scheduled', and 'no\_status'. A timeline at the bottom right shows the execution history of the task, with green dots indicating successful runs.

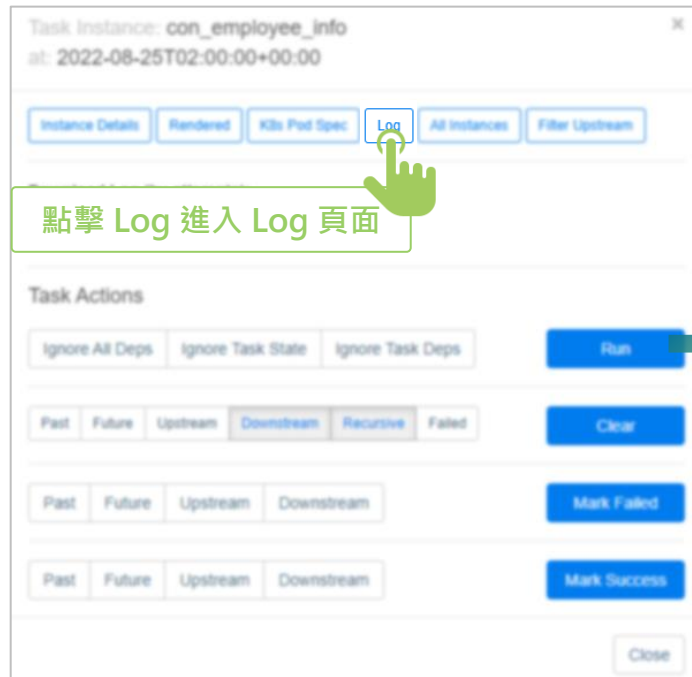
點擊 Task ID 進入下一層

顯示 Task 執行階段的狀態

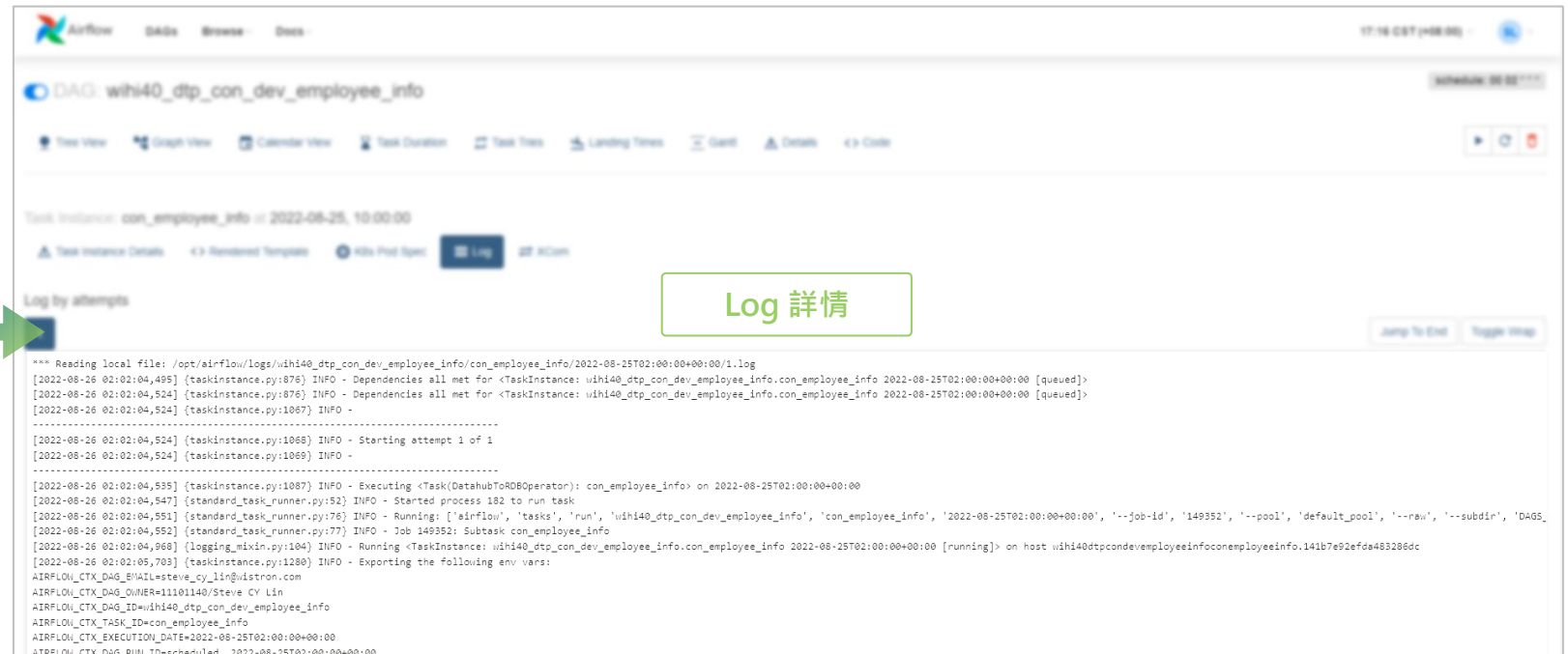
## Airflow DAG Operation



# Airflow DAG Operation



點擊 Log 進入 Log 頁面



Log 詳情

## Data Hub 平台發佈訂閱相關諮詢人



### Airflow 諮詢

Wits.AnnaWu@wistron.com



### Data service portal 諮詢

Wits.DoryWu@wistron.com

### 如有疑問，可發信詢問

建議在信件中提供 2 個 json(connection)、python(DAG)、Log 的紀錄，以上檔案建議壓縮成一個檔案

# END

Thank you for listening



簡報者：Steve CY Lin



STEVE\_CY\_LIN@WISTRON.COM





01

## Integration Style – Event Centric

Kafka Usage

02

## Integration Style – Data Centric

Data Hub Usage

03

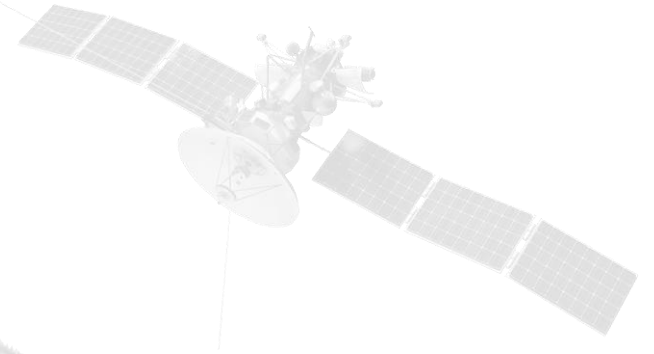
## Integration Style – Application Centric

API Usage

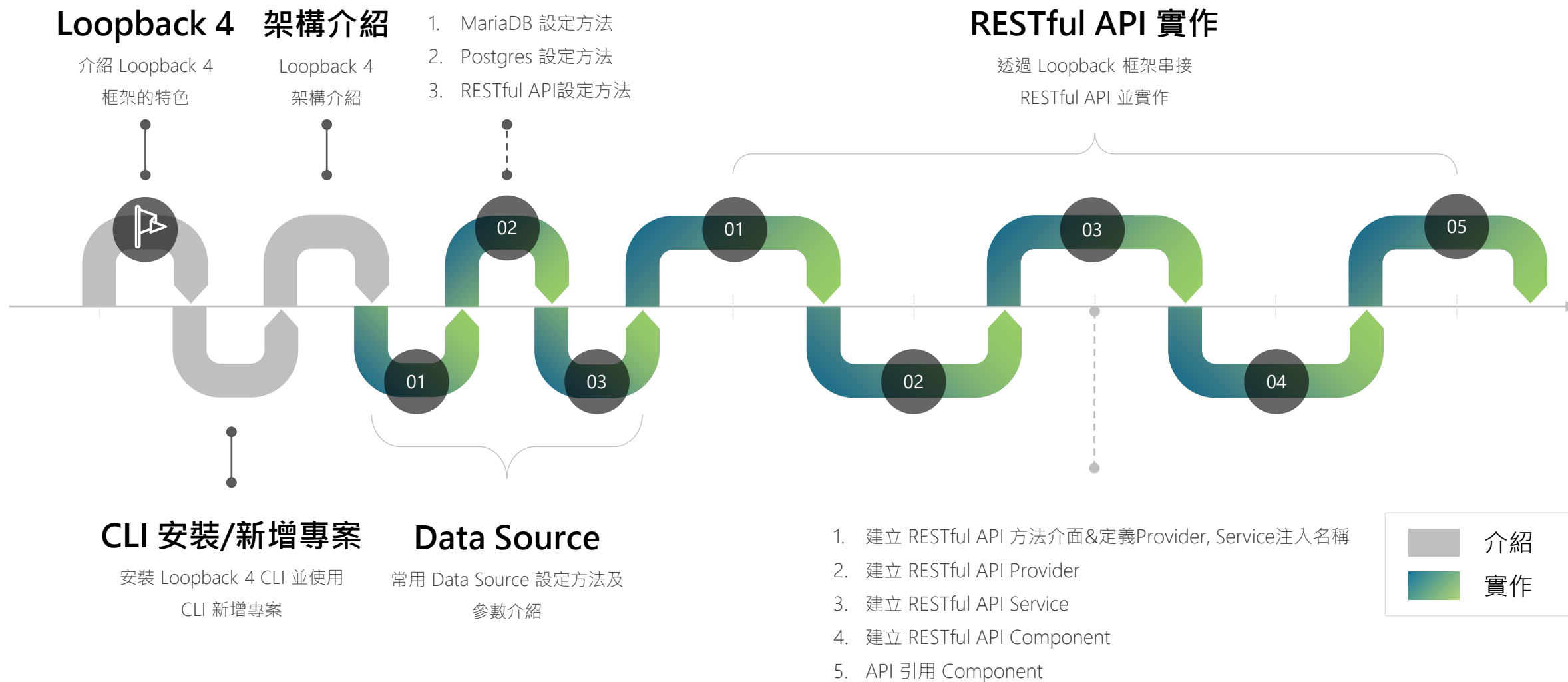
# DATA INTEGRATION

Integration Style – Application Centric

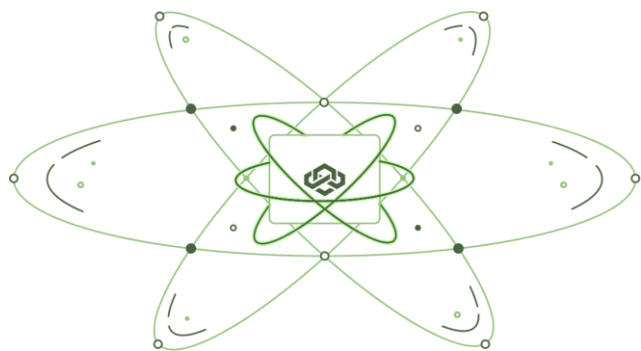
API Usage



# Course Roadmap

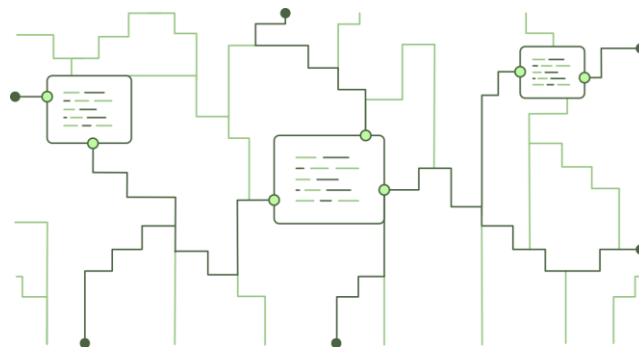


## Loopback 4 介紹



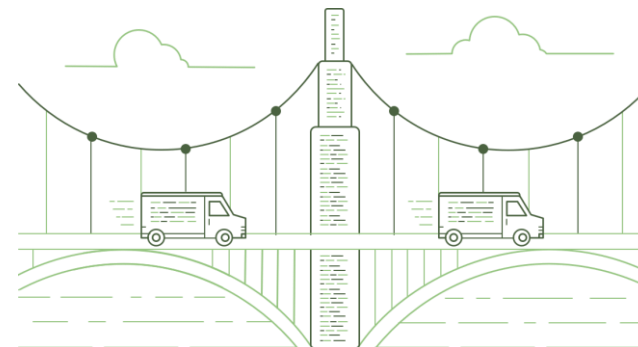
### 全新的核心

Loopback 4 改使用 TypeScript/ES2017 來進行 API 的開發



### OpenAPI 規範

內建提供支援 Open API 規範的修飾器以及 Swagger Explorer 操作介面



### 高度擴展性

具有依賴注入，並支持主流的數據源，例如 MongoDB、MySQL 等和 REST API

# Create Loopback 4 Project

## Loopback 4 CLI 安裝



```
npm install -g @loopback/cli
```

## 創建 Loopback 專案

```
lb4 app
```

```
? 專案名稱 loopback4-sandbox
```

```
? 專案說明 loopback4 sandbox
```

```
? 專案根目錄 loopback4-sandbox
```

```
? 應用程式類別名稱 Loopback4SandboxApplication
```

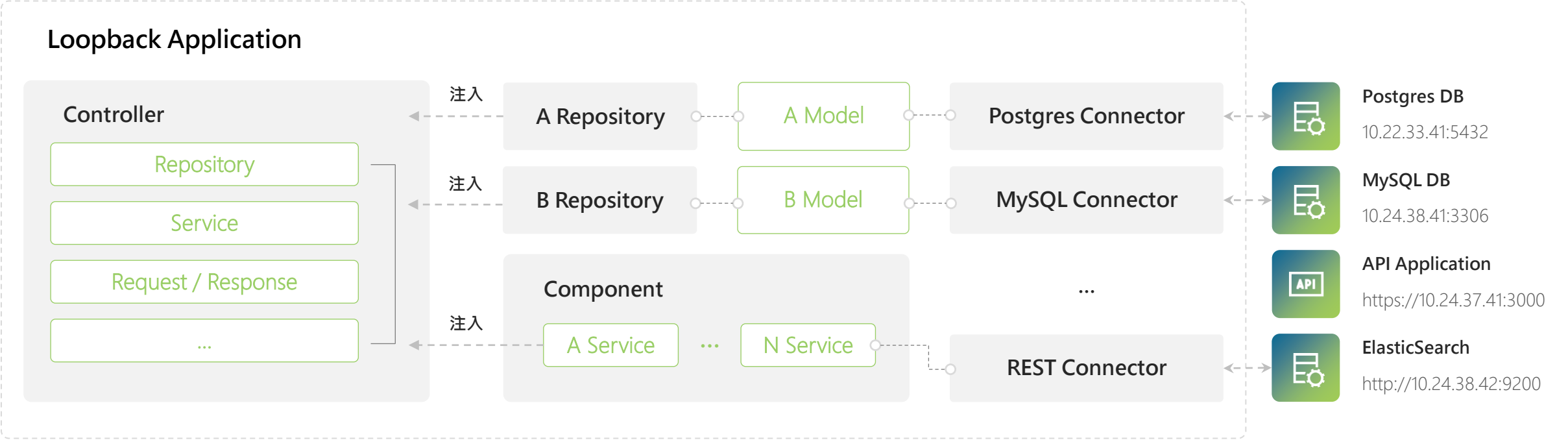
```
? 選取要在專案中啟用的特性
```

```
>(*) 啟用 eslint: 新增 linter, 且其中含有預先配置的 lint 規則
```

```
(*) 啟用 prettier: 安裝 prettier, 以根據規則將程式碼格式化
```

```
(*) 啟用 mocha: 安裝 mocha 以執行測試
```

```
(*) ...
```



Step 1. Data Source

Step 2. Model

Step 3. Repository

Step 4. Controller

 lb4 datasource

 lb4 model

 lb4 repository

 lb4 controller

# Data Source Configuration

使用 MariaDB 作為 Data Source

數據庫連線配置

```
{  
  "name": "mariadb",  Loopback Data Source 連線名稱  
  "connector": "mysql",  數據庫連接器 · MariaDB 及 MySQL 皆使用 mysql  
  "url": "",  數據庫 URL 連線方式  
  "host": <database_host>,  數據庫 Host  
  "port": <database_port>,  數據庫端口  
  "user": <database_username>,  數據庫帳號  
  "password": <database_password>,  數據庫密碼  
  "database": <database>  連線的數據庫名稱  
}
```

# Data Source Configuration

使用 Postgres 作為 Data Source

數據庫連線配置

```
{  
  "name": "postgres", Loopback Data Source 連線名稱  
  "connector": "postgres", 數據庫連接器 · MariaDB 及 MySQL 皆使用 mysql  
  "url": "", 數據庫 URL 連線方式  
  "host": <database_host>, 數據庫 Host  
  "port": <database_port>, 數據庫端口  
  "user": <database_username>, 數據庫帳號  
  "password": <database_password>, 數據庫密碼  
  "database": <database> 連線的數據庫名稱  
}
```



# Data Source Configuration

使用 RESTful API 作為 Data Source，以 Elasticsearch 為例

API 連接器

API 配置

```
{
  "connector": "rest", API 連接器使用 rest
  "debug": true,
  "operations": [
    {
      "template": { API 設定範本
        "method": "POST", HTTP Method
        "url": "http://127.0.0.1:9200/{index:string}/_search", API URL 及路徑配置，路徑可使用 {param:type} 進行參數設定
        "headers": { API Header 設定，Header 屬性可使用 {param:type} 進行參數設定
          "accept": "application/json",
          "content-type": "application/json"
        },
        "body": "{query:object}" API Request Body，可使用 {param:type} 進行參數設定
      },
      "functions": { 將 RESTful API 以 Loopback 的方法進行定義
        "search": ["index", "query"] 提供方法名稱，並將需要使用的參數添加至此，以便進行後續抽象該方法的來源
      }
    }
  ]
}
```

# Implements RESTful API Service

實作 RESTful API 服務



File Name elasticsearch-api.ts

```
export interface ElasticSearchApi {  
  search<T>(index: string, query: object): Promise<T[]>;  
}
```

File Name elasticsearch-api.constant.ts

```
import { BindingKey } from '@loopback/core';  
import { ElasticSearchService } from './elasticsearch.service';  
  
export namespace ElasticSearchConstant {  
  export const ELASTIC_PROVIDER_INJECT =  
    'providers.elasticsearch.api';  
  
  export const ELASTIC_SERVICE_BINDING =  
    BindingKey.create<ElasticSearchService>(  
      'services.elasticsearch.api'  
    );  
}
```

# Implements RESTful API Service

實作 RESTful API 服務



File Name elasticsearch.provider.ts

```
import { inject, Provider } from '@loopback/core';
import { getService } from '@loopback/service-proxy';
import { ElasticsearchDataSource } from '../../datasources';
import { ElasticSearchApi } from './elasticsearch-api';

export class ElasticSearchProvider implements Provider<ElasticSearchApi> {
  constructor(
    @inject('datasources.elasticsearch')
    protected dataSource: ElasticsearchDataSource = new ElasticsearchDataSource()
  ) {}

  public value(): Promise<ElasticSearchApi> {
    return getService(this.dataSource);
  }
}
```

# Implements RESTful API Service

實作 RESTful API 服務



File Name elasticsearch.service.ts

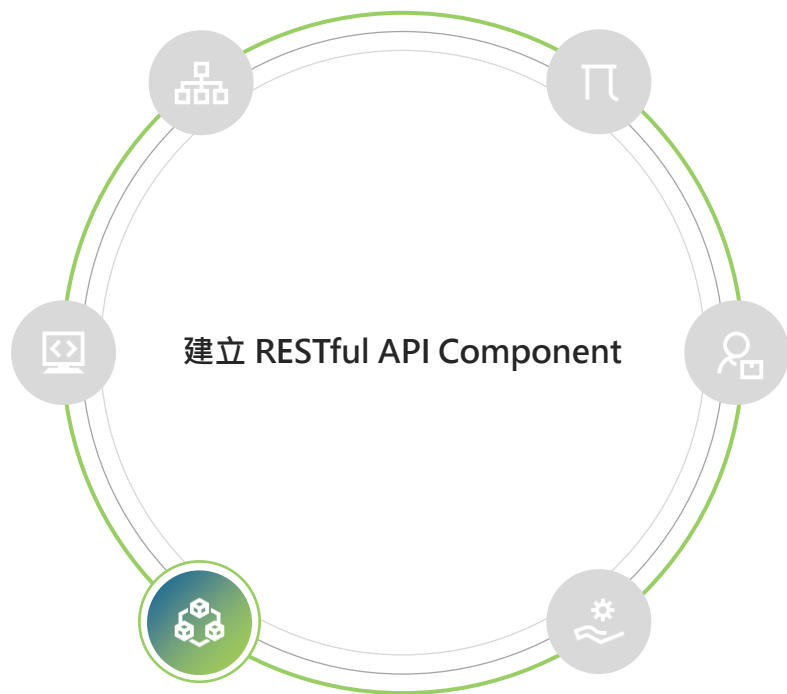
```
import { inject } from '@loopback/core';
import { ElasticSearchApi } from './elasticsearch-api';
import { ElasticSearchConstant } from './elasticsearch-api.constant';

export class ElasticSearchService implements ElasticSearchApi {
  constructor(
    @inject(ElasticSearchConstant.ELASTIC_PROVIDER_INJECT)
    public readonly elasticSearchApi: ElasticSearchApi
  ) {}

  public async search<T>(index: string, query: object): Promise<T[]> {
    return await this.elasticSearchApi.search(index, query);
  }
}
```

# Implements RESTful API Service

實作 RESTful API 服務



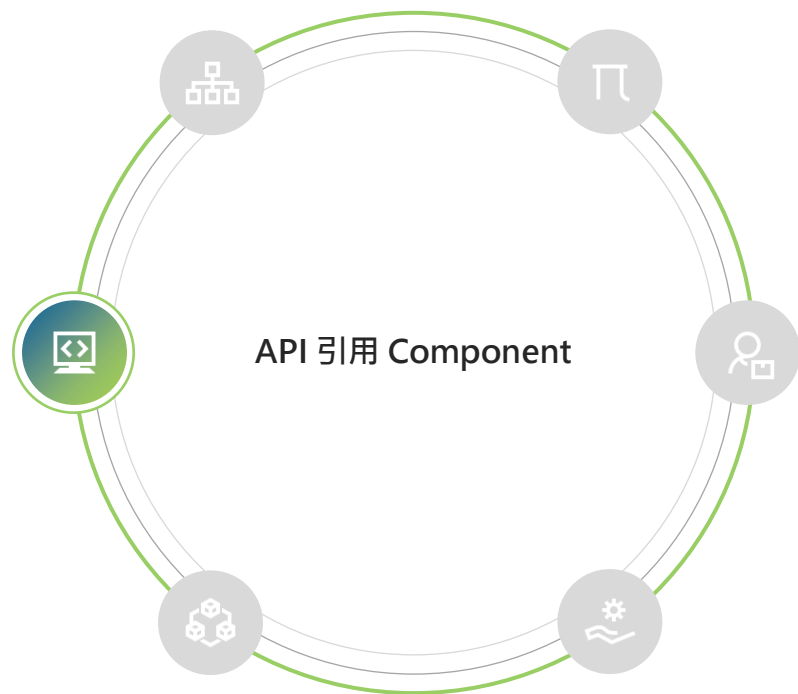
File Name elasticsearch.component.ts

```
import { Binding, Component } from '@loopback/core';
import {
  ElasticSearchProvider,
  ElasticSearchService,
  ElasticSearchConstant,
} from './services';

export class ElasticSearchComponent implements Component {
  public bindings: Binding[] = [
    Binding.bind(ElasticSearchConstant.ELASTIC_PROVIDER_INJECT).toProvider(
      ElasticSearchProvider
    ),
    Binding.bind(ElasticSearchConstant.ELASTIC_SERVICE_BINDING).toClass(
      ElasticSearchService
    ),
  ];
}
```

# Implements RESTful API Service

實作 RESTful API 服務



File Name application.ts

```
...  
import { ElasticsearchComponent } from './components';  
  
export class ApiApplication extends BootMixin(  
  ServiceMixin(RepositoryMixin(RestApplication))  
) {  
  constructor(options: ApplicationConfig = {}) {  
    super(options);  
    ...  
  
    this.component(ElasticSearchComponent);  
  }  
}
```

# END

Thank you for listening



簡報者：Steve CY Lin



STEVE\_CY\_LIN@WISTRON.COM