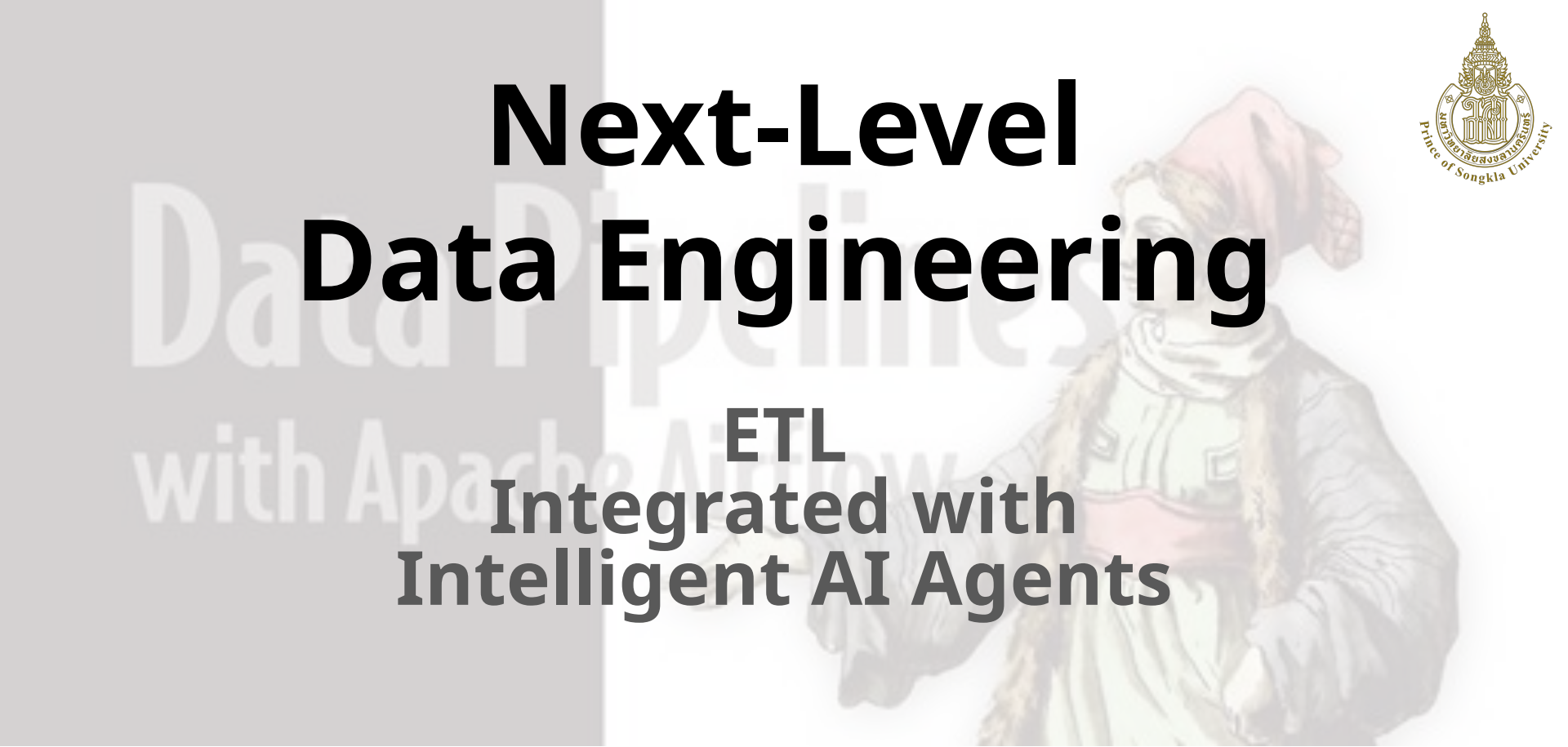
Next-Level Data Engineering

ETL Integrated with Intelligent AI Agents



ดร.อนันท์ ชกสุริวงค์

WHAT DOES A DATA ENGINEER DO, EXACTLY?

- Build data pipelines to collect data and move it to storage
- Prepare the data as part of an ETL or ELT process
- Stitch the data together with scripting languages
- Work with the DBA to construct data stores
- Ensure the data is ready for use
- Use frameworks and microservices to serve data

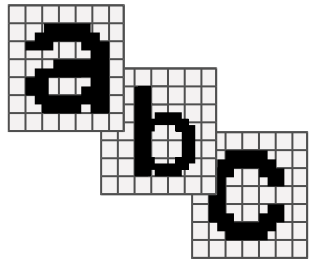
Database Administrator (DBA) A DBA is a professional responsible for the management, maintenance, security, and performance of databases

Tips

WHAT IS ETL ?

Outline

ETL is a process to extract data from a starting data source, transform the data in some fashion, then load it into another data store.



WHY IS ETL IMPORTANT?

CONTEXT

Context

ETL helps businesses gain deep historical context with data

CONSOLIDATION

Consolidation

It provides a consolidated view of data, for easier analysis and reporting

PRODUCTIVITY

Productivity

It improves productivity with repeatable processes that don't require heavy hand-coding

ACCURACY

Accuracy

It improves data accuracy and audit capabilities that most businesses require for compliance with regulations and standards

HOW DOES THE ETL PROCESS WORK?

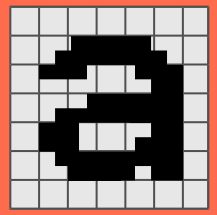
TRADITIONAL ETL

- That data is generally extracted into a staging area, storage that sits between the data source and the data target.
- In that staging area, the ETL tool transforms data, cleansing, joining, and otherwise optimizing it for analysis.

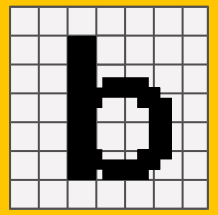
MODERN ETL (ELT)

- The cloud has the combination of speed, scalability, and practicality required for handling enormous amounts of structured and semi-structured data from literally dozens or hundreds of sources.

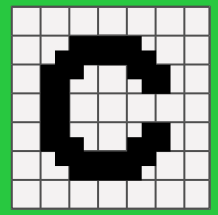
COMMON ETL CHALLENGES



SCALING



TRANSFORMING DATA ACCURATELY



HANDLING DIVERSE DATA SOURCES

WHAT ARE THE TYPES OF ETL TOOLS?

There are four different types of ETL tools available today.

Batch

BATCH PROCESSING ETL TOOLS

Cloud

CLOUD-NATIVE ETL TOOLS

Open source

OPEN SOURCE ETL TOOLS

Real-time

REAL-TIME ETL TOOLS

HOW IS ETL COMMONLY USED?

Outline

- DATA WAREHOUSING
- CLOUD MIGRATION
- MACHINE LEARNING AND AI
- MARKETING DATA INTEGRATION

WHAT IS THE FUTURE OF ETL?

Go | Past | Present | Future

Monday
May
2015

EXPONENTIAL DATA GROWTH : The Internet of Things will continue to expand and play an increasing role in business and our lives.

Monday
May
2025

MORE MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE : Preparing data for machine learning and artificial intelligence.

Monday
May
2035

THE DEMOCRATIZATION OF DATA : The more entire organizations can self-serve to gain actionable insights, the greater their competitive edge will be.

HOW TO BUILD AN ETL STRATEGY

UTILIZE **MOVE** **INVEST**

UTILIZE : Utilize all of the data it collects, structured and semi-structured, operational, and transactional, to gain the maximum amount of insight for innovation and decision making.

MOVE : Move to a platform where it can effectively collect and analyze all of this data. Right now, a cloud data warehouse is the most practical solution from the perspective of speed, scale, and cost.

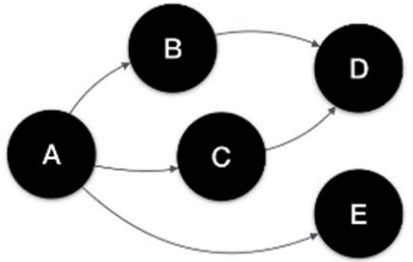
INVEST : Invest in an ETL tool that can help it extract data and transform it for cloud data analytics as quickly as possible to reduce time to insight.

Introduction to Apache **Airflow**

Outline

Apache Airflow is an **orchestration** tool that helps you to programmatically create and handle task execution into a single workflow. It then handles **monitoring** its progress and takes care of **scheduling** future workflows depending on the schedule defined.

Workflows are created using python scripts, which define how your tasks are executed. They are usually defined as Directed Acyclic Graphs(DAG).



What is Apache Airflow ?

Authoring

Workflows is Airflow are written as Directed Acyclic Graphs (DAGs) in Python

Scheduling

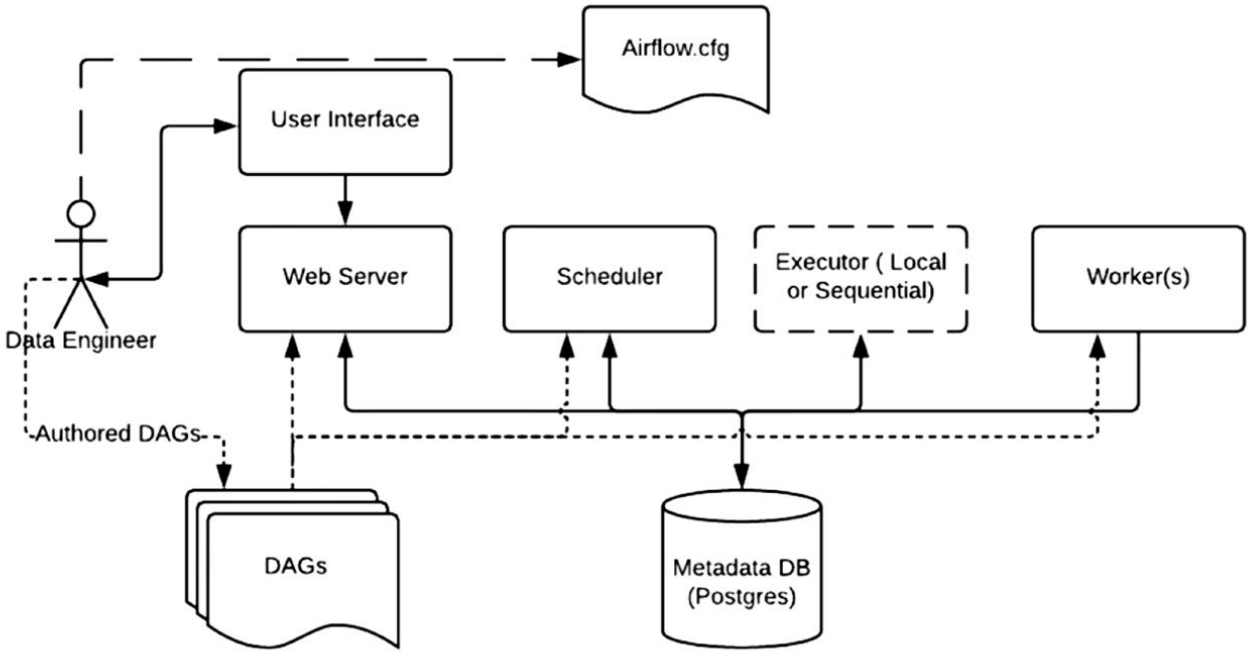
Users can specify when a workflow start, end after what interval it should run again

Monitoring

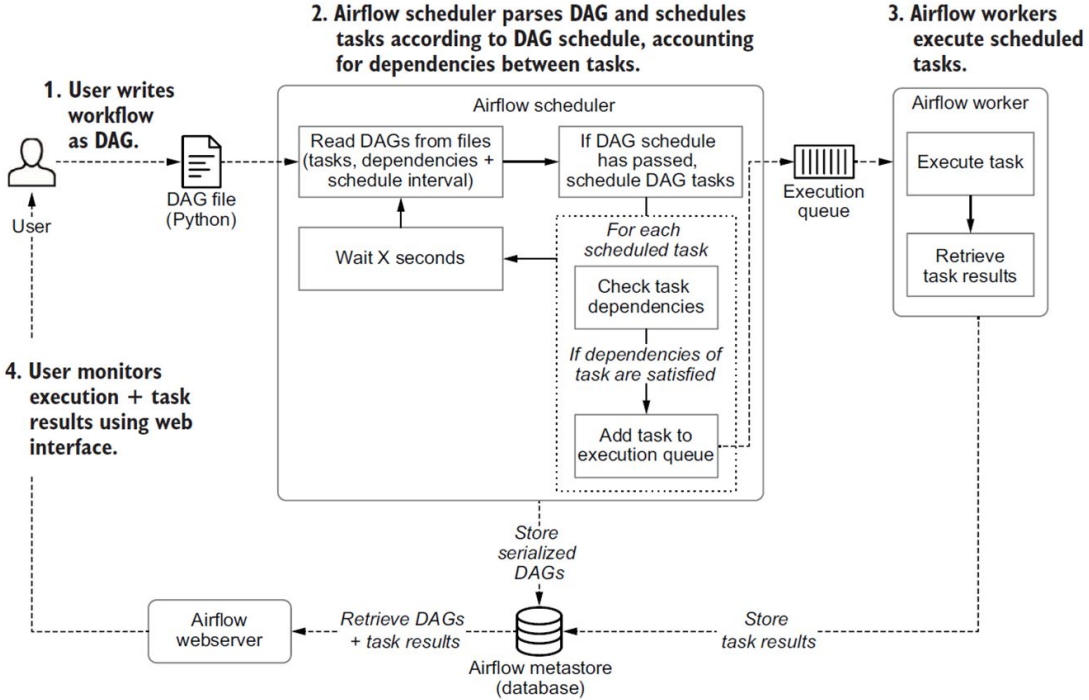
Airflow provides an interactive web interface to monitor your workflow

Aireflow is an open source platform for programmatic ally authoring, scheduling, and monitoring workflows.

Architecture Overview



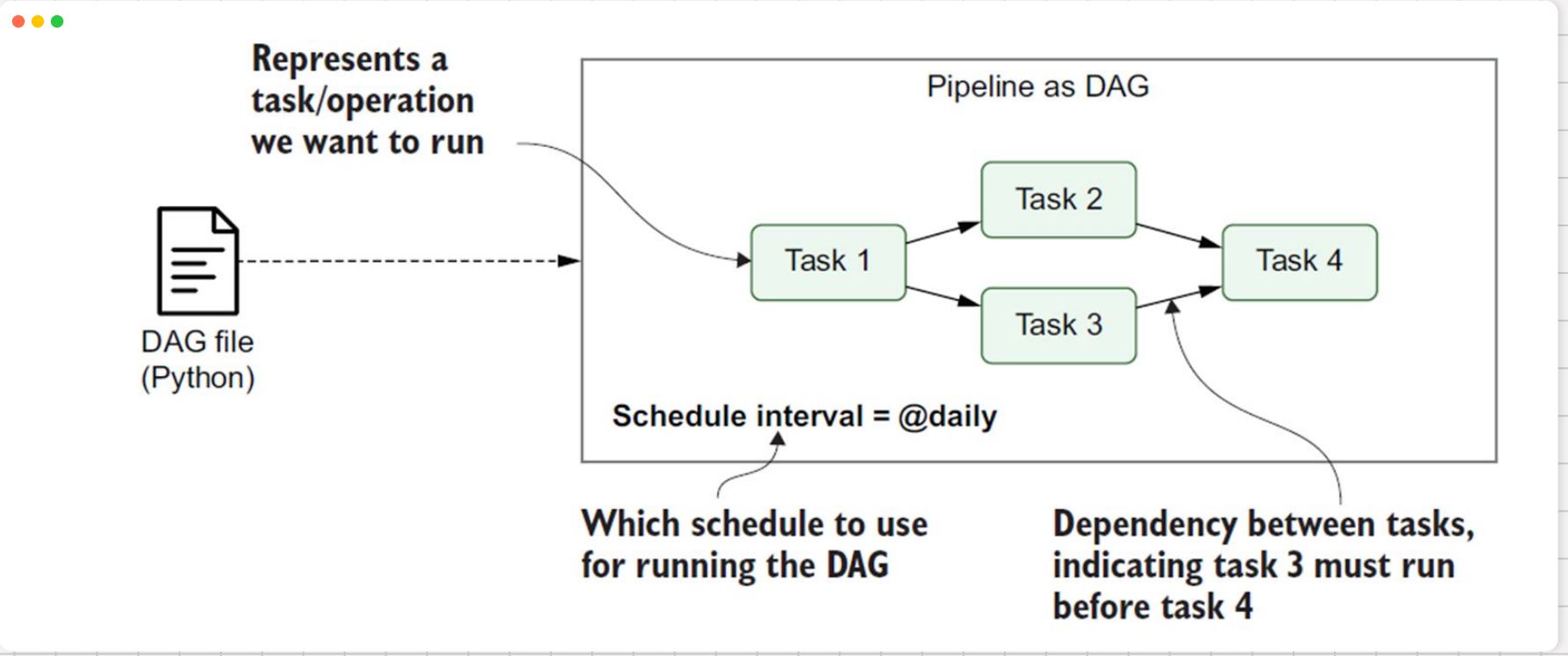
Process in developing and executing pipelines



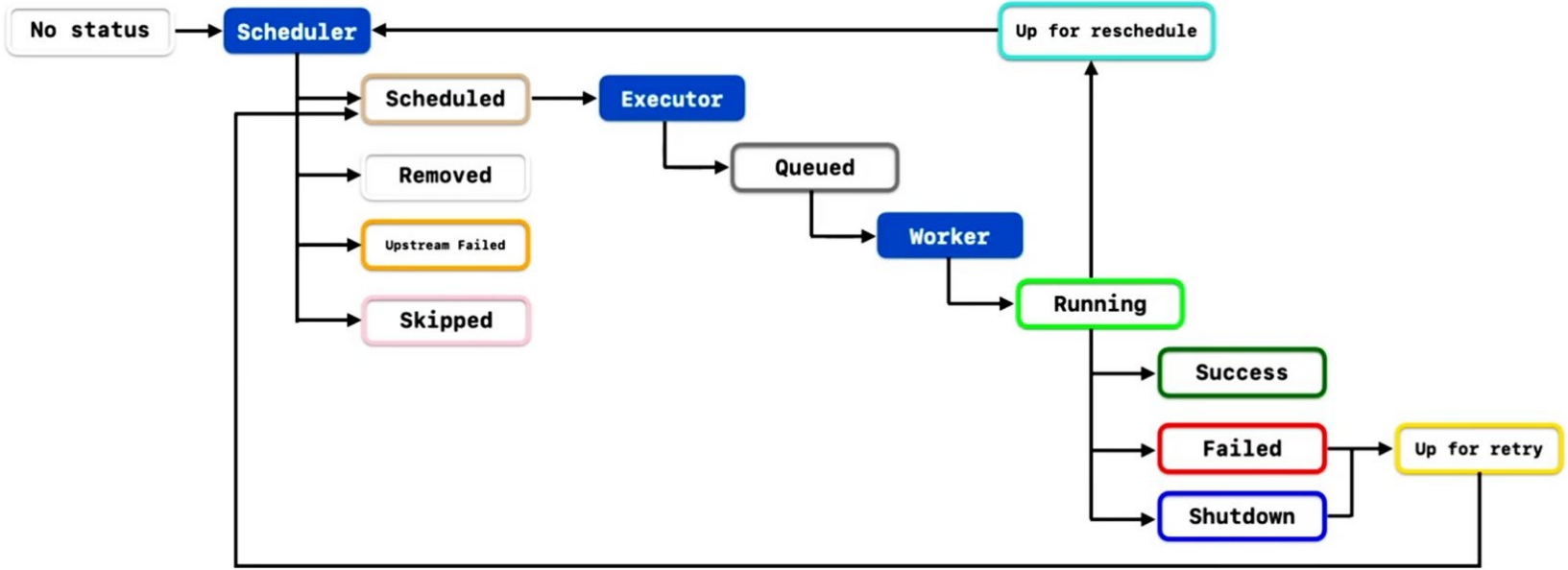
Core concepts of Apache Airflow

- **Core concepts - dags, tasks, operators, workflows**
- **Architecture - web server, scheduler, brokers, workers**
- **Advance concepts - Xcoms, pools, SLA**
- **Practice - Hands on experience by solving problems with airflow**

Airflow DAG

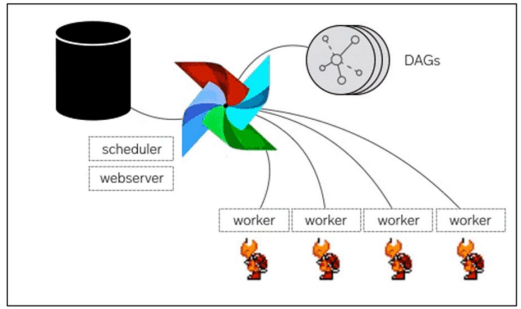

Represents a task/operation we want to run
Pipeline as DAG
Task 1
Task 2
Task 3
Task 4
DAG file (Python)
Schedule interval = @daily
Which schedule to use for running the DAG
Dependency between tasks, indicating task 3 must run before task 4

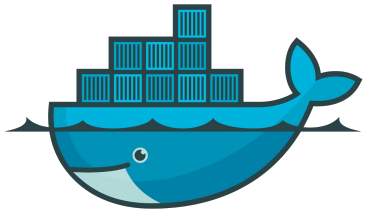
Airflow Task Lifecycle

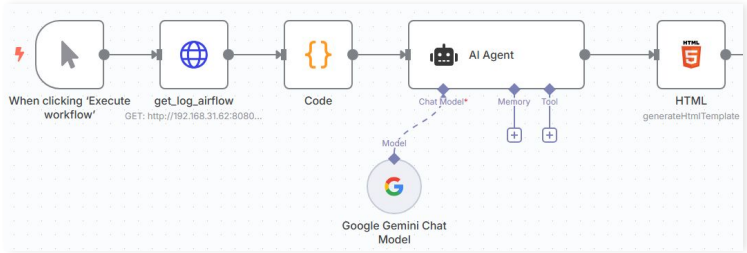


Check up !

- ❑ VScode/Cursor
- ❑ Docker
- ❑ Airflow
- ❑ n8n
- ❑ LLM
- ❑ AI Agents





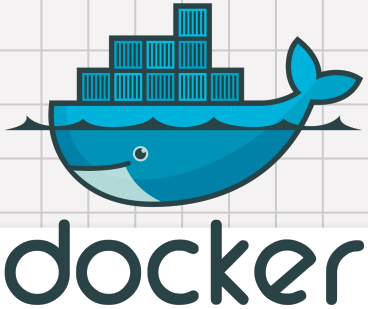


docker

Check up ! : Cursor 

- ✓ Cursor
- ❑ Docker
- ❑ Airflow
- ❑ n8n
- ❑ LLM
- ❑ AI Agents

Check up ! : Docker



- ✓ Cursor
- ✓ Docker
- ❑ Airflow
- ❑ n8n
- ❑ LLM
- ❑ AI Agents

Check up ! : Airflow

- ✓ Cursor
- ✓ Docker
- ✓ Airflow
- ❑ n8n
- ❑ LLM
- ❑ AI Agents

1. ดาวโหลดไฟล์ docker-compose.yaml ตามลิงค์
2. บันทึกที่ Desktop\ETL\airflow
3. ไปที่ Docker Terminal
4. ไปยัง folder บน Desktop\ETL\airflow
5. พิมพ์คำสั่ง mkdir dags logs plugins
6. สั่ง docker-compose up airflow-init
7. รอจนเสร็จสิ้น สังเกตข้อความ exited with code 0
8. เริ่ม airflow สั่ง docker-compose up -d
9. หยุด airflow สั่ง docker-compose down
10. ถอนติดตั้ง docker-compose down –volumes –rmi all

Check up ! : Airflow

- ✓ Cursor
- ✓ Docker
- ✓ Airflow
- ❑ n8n
- ❑ LLM
- ❑ AI Agents

1. เปิดใช้งาน airflow
2. เปิด browser ไปยังลิงค์ localhost:8080
3. Login เข้าใช้ user/password : airflow
4. ดู ip ของเครื่องคำสั่ง ipconfig ที่ command prompt

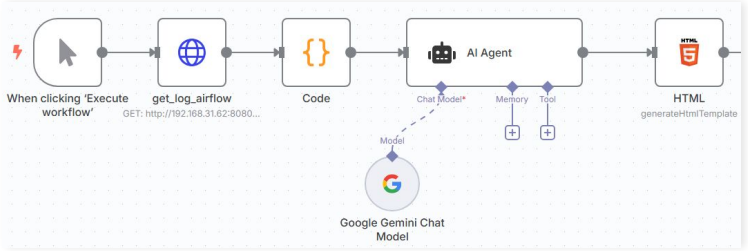
Check up ! : n8n

- ✓ Cursor
- ✓ Docker
- ✓ Airflow
- ✓ n8n
- ❑ LLM
- ❑ AI Agents

1. ไปที่ Docker terminal
2. cd ไปที่ Desktop\ETL\
3. สั่ง mkdir n8n
4. เริ่มติดตั้ง n8n
5. สั่ง docker run -it --rm --name n8n -p 5678:5678 -v [n8n_path]:/home/node/.n8n docker.n8n.io/n8nio/n8n
6. รอดำเนินการจนแล้วเสร็จ
7. เข้าใช้งานที่ browser ลิงค์ localhost:5678
8. กำหนด login แนะนำให้ใช้ gmail
9. ดำเนินการจนสำเร็จ ยินดีด้วย :)
10. หากต้องการใช้งานผ่าน internet ได้ติดตั้ง ngrok

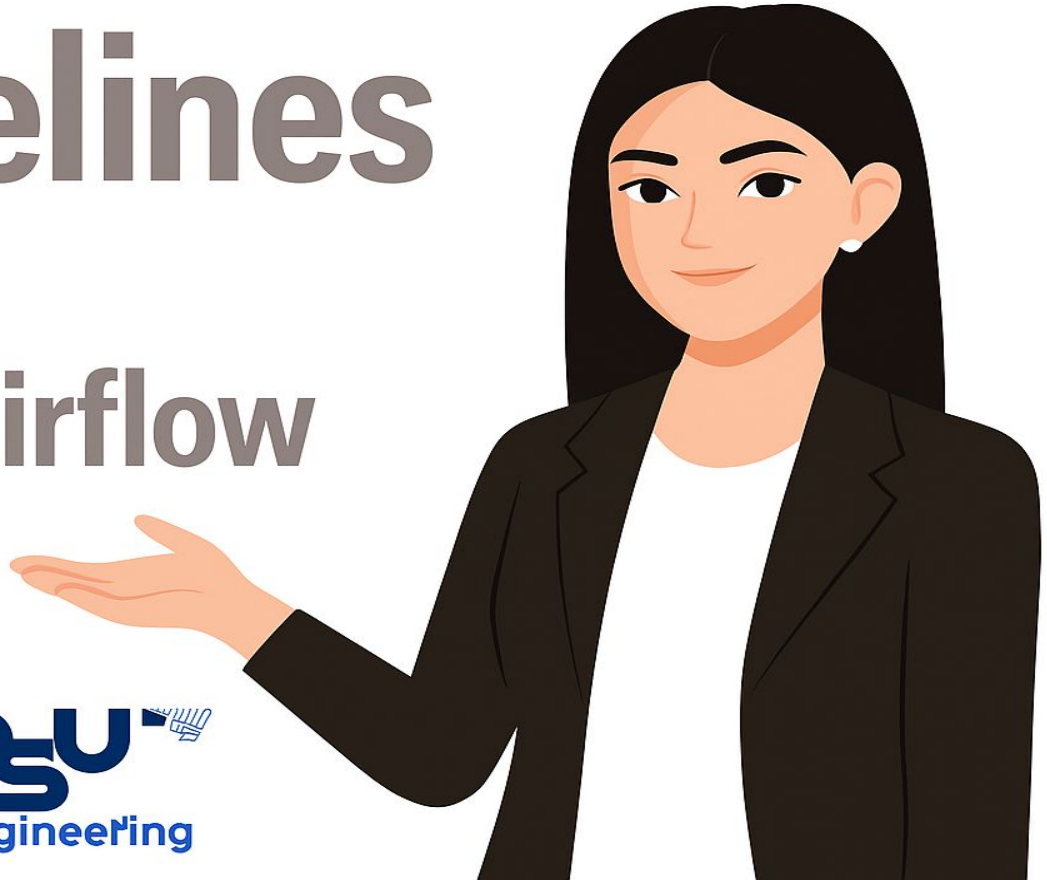
Check up ! : LLM&AI Agents

- ✓ Cursor
- ✓ Docker
- ✓ Airflow
- ✓ n8n
- ✓ LLM
- ✓ AI Agents



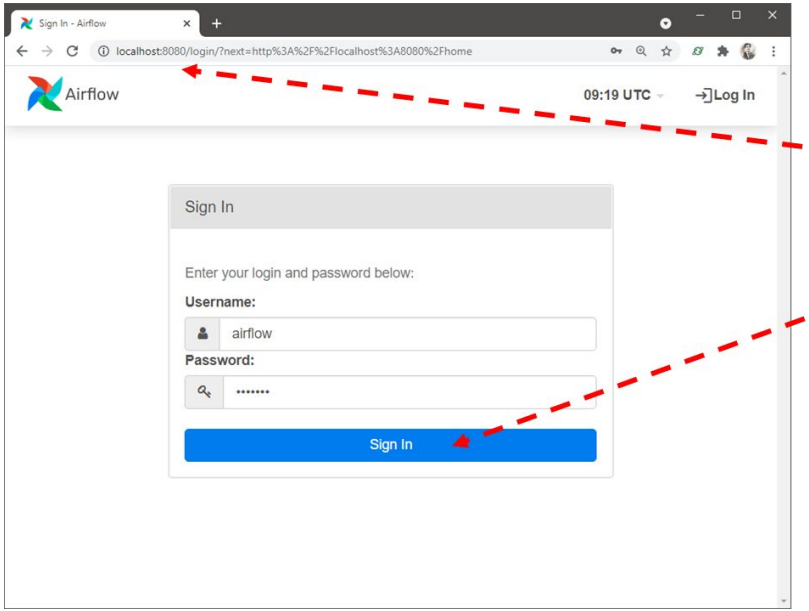
1. เข้าใช้ n8n
2. ขอ api_key สำหรับเปิดใช้ LLM
3. Gemini ไปที่ https://aistudio.google.com/
4. DeepSeek ที่ https://platform.deepseek.com/
5. ChatGPT ที่ https://platform.openai.com/
6. สร้าง AI Agent

Data Pipelines (ETL) with Apache Airflow



PSU Engineering

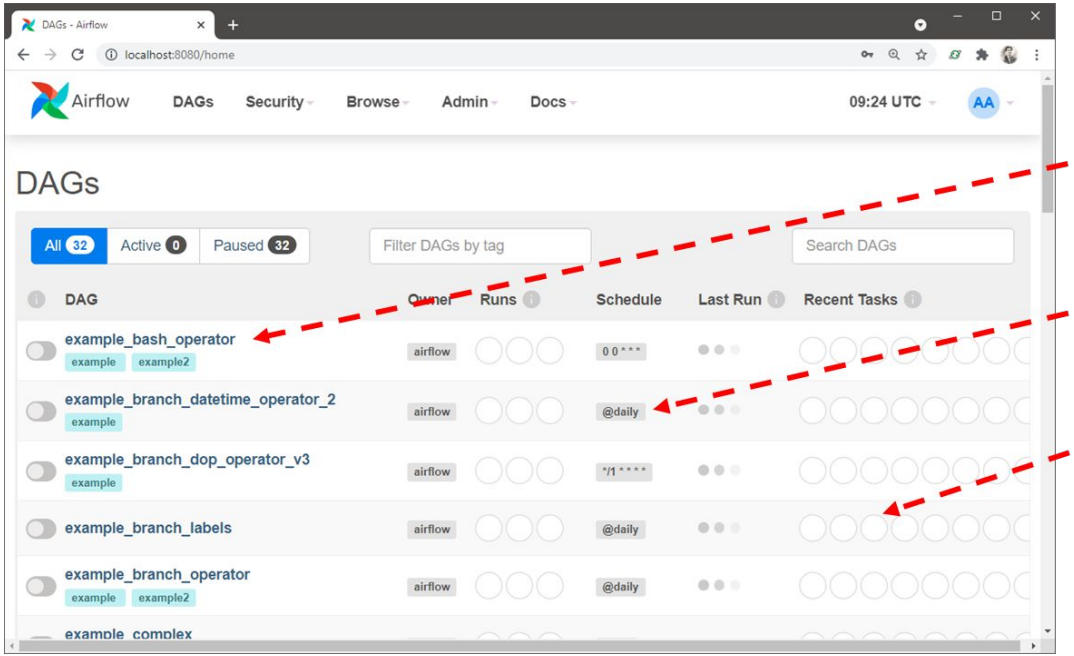
Login Airflow Web UI



(1) เปิด web browser ไปยัง url
localhost:8080

(2) login ด้วย user name / password
Username : airflow
Password : airflow

Airflow Web UI



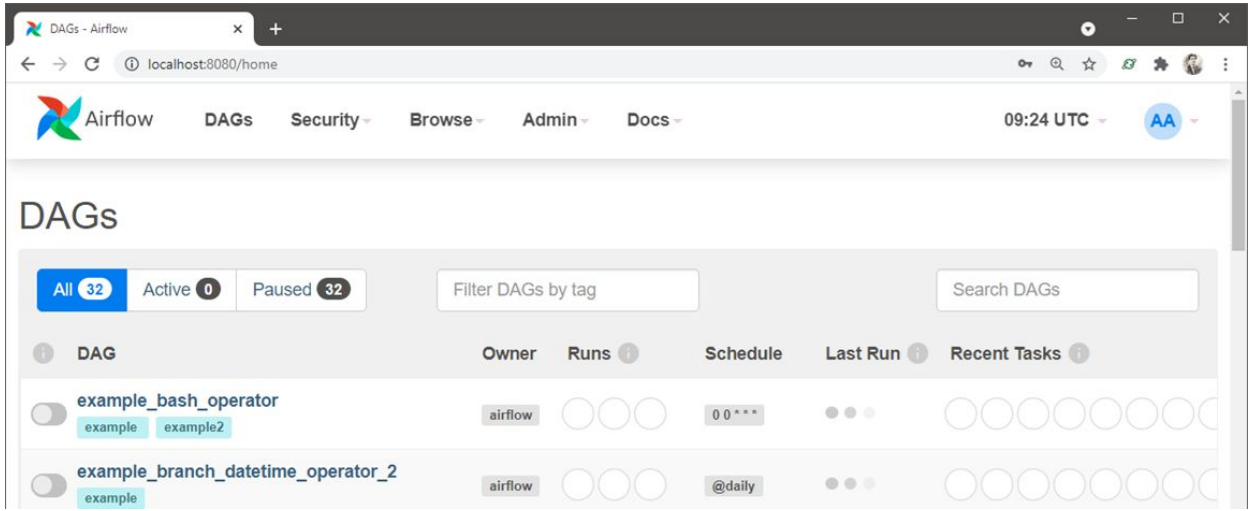
(1) ชื่อของ workflow

(2) กำหนดการ (schedule)

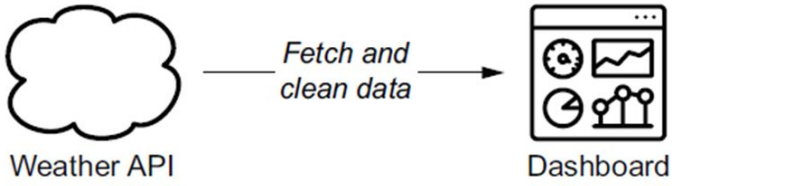
(2) สถานะของ workflow

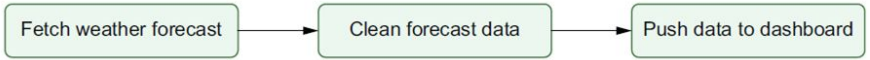
Check point #1

จับหน้าจอแสดงการเข้าใช้งาน Airflow Web UI ดังตัวอย่าง บันทึกภาพเก็บไว้ส่ง Exercise



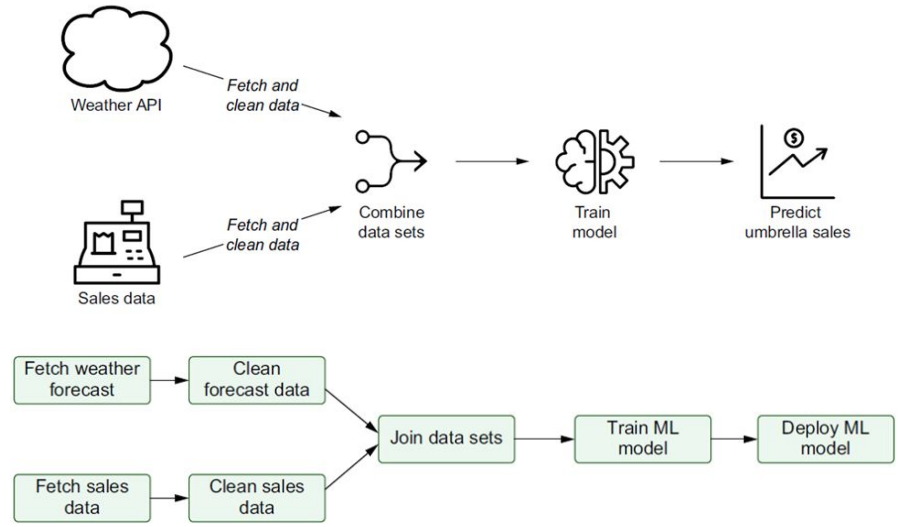
Introducing data pipelines





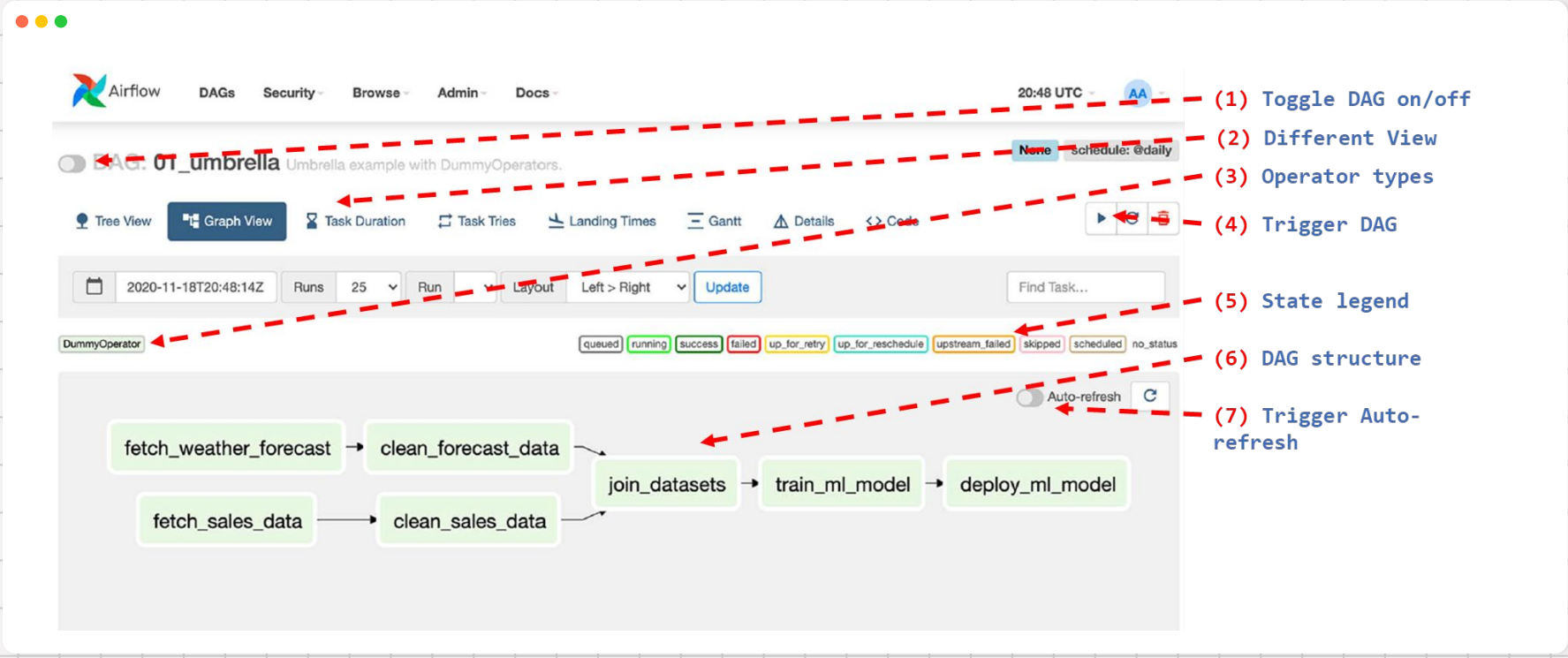
1. Fetch weather forecast data from a weather API.
2. Clean or otherwise transform the fetched data (e.g., converting temperatures from Fahrenheit to Celsius or vice versa), so that the data suits our purpose.
3. Push the transformed data to the weather dashboard.

Pipeline graphs vs. sequential scripts

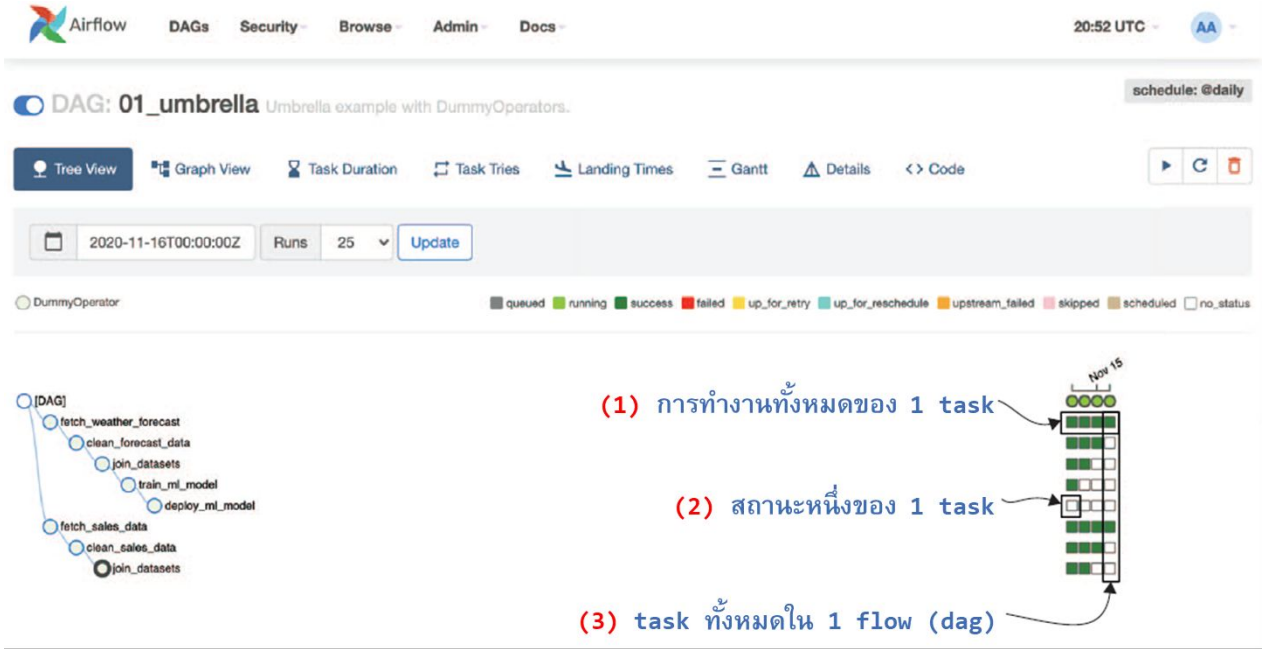


1. Prepare the sales data by doing the following:
   – Fetching the sales data from the source system
   – Cleaning/transforming the sales data to fit requirements
2. Prepare the weather data by doing the following:
   – Fetching the weather forecast data from an API
   – Cleaning/transforming the weather data to fit requirements
3. Combine the sales and weather data sets to create the combined data set that can be used as input for creating a predictive ML model.
4. Train the ML model using the combined data set.
5. Deploy the ML model so that it can be used by the business.

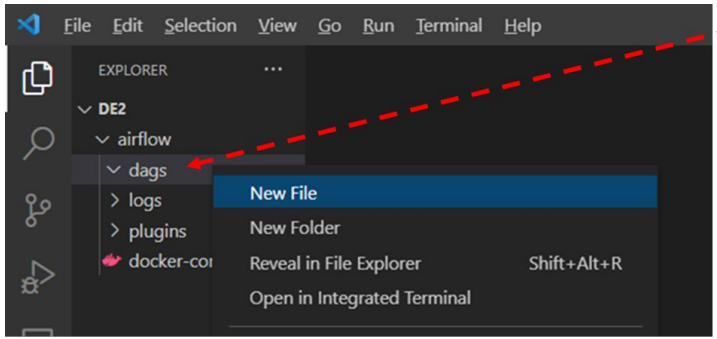
The graph view in Airflow's web interface



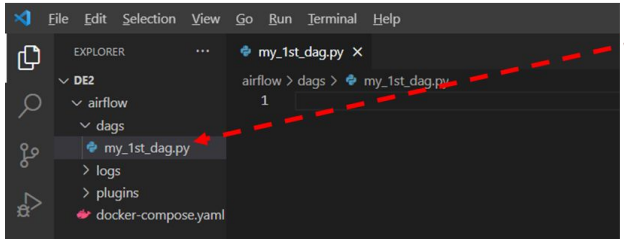
The graph view in Airflow's web interface



Let’s build your first DAG



(1) คลิกขวาที่ folder ชื่อ dags เลือก new file



(2) ตั้งชื่อว่า my_1st_dag.py

Build Your First DAG

Importing Packages

```
from airflow import DAG
from airflow.operators.dummy_operator import
DummyOperator
from airflow.utils import timezone
```

Setting Up default_args

```
default_args = {
    'owner': 'airflow',
}
```

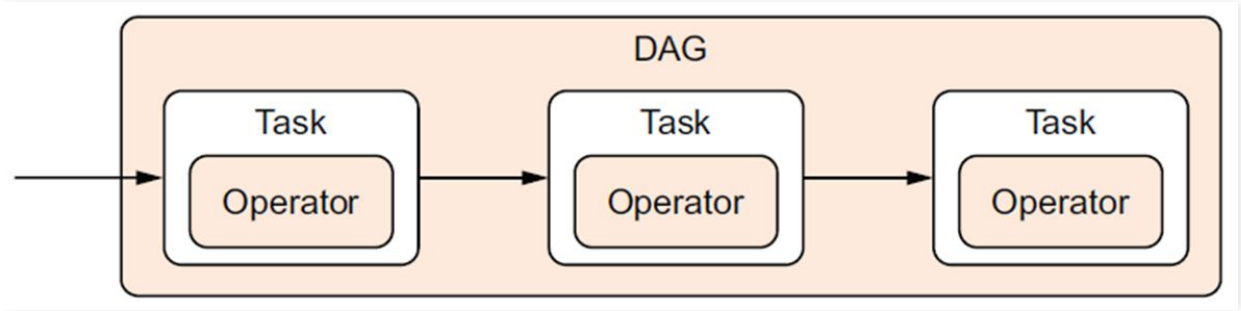
Build Your First DAG

Defining DAG, Tasks, and Operators

```
with DAG(
    'my_1st_dag',
    schedule_interval='*/5 * * * *',
    default_args=default_args,
    start_date=timezone.datetime(2020, 7, 10),
    tags = ['ETL','Hello World'],
    catchup=False,
) as dag :
  t1 = DummyOperator(
    task_id='my_1st_dummy_task',
    dag=dag,)
  t2 = DummyOperator(
    task_id='my_2nd_dummy_task',
    dag=dag,)
t1 >> t2
```

Tasks vs. operators

DAGs and operators are used by Airflow users. Tasks are internal components to manage operator state and display state changes (e.g., started/finished) to the user.



Basic operators

A workflow is made up of tasks and each task is an operator. Now, what is an operator? An operator a python class that does some work for you. These classes are provided by airflow itself. Some basic operators are

- BashOperator - used to run bash commands.
- PythonOperator - used to run a python function you define.
- BranchOperator - used to create a branch in the workflow.
- DummyOperator - used to represent a dummy task.

BashOperator

```
from airflow.operators.bash_operator import
BashOperator
echo_hello = BashOperator(
    task_id='echo_hello',
    bash_command='echo hello',
    dag=dag
)
```

PythonOperator

```
from airflow.operators.python_operator import
PythonOperator
def hello():
    return 'Hello, Python'
say_hello = PythonOperator(
    task_id='say_hello',
    python_callable=hello,
    dag=dag
)
```

Logging

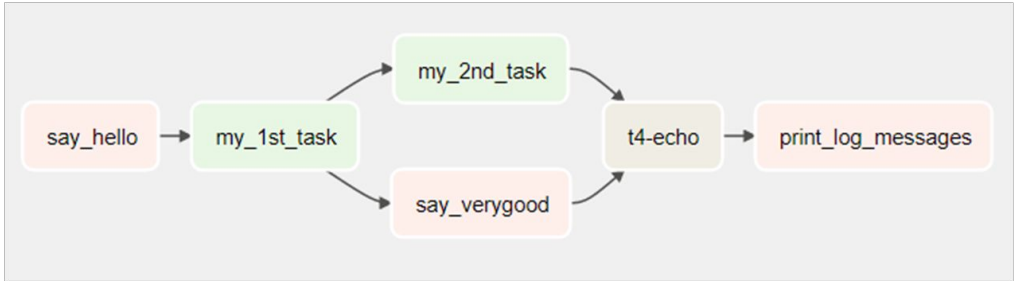
```
import logging

def print_log_messages():
    logging.debug('This is a debug message')
    logging.info('This is an info message')
    logging.warning('This is a warning message')
    logging.error('This is an error message')
    logging.critical('This is a critical message')

    return 'Whatever is returned also gets printed in the logs'

print_log = PythonOperator(
    task_id='print_log_messages',
    python_callable=print_log_messages,
    dag=dag,
)
```

Defining dependencies between tasks

>> : dependency (one-to-one)

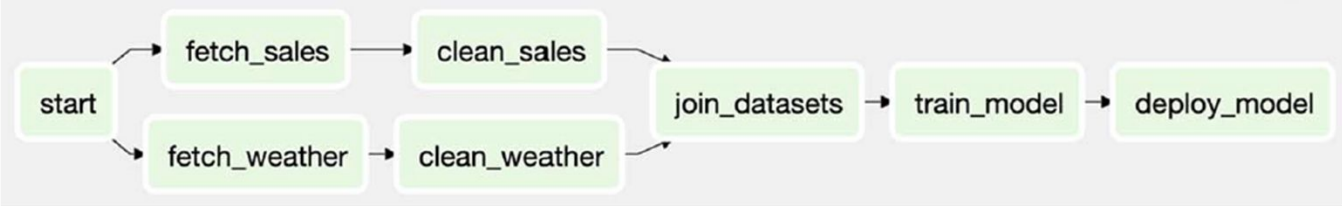
>> [, , ...] : fan-out (one-to-multiple) dependency

[, , ...] >> : fan-in (multiple-to-one) dependencies




```
say_hello >> my_1st_task
my_1st_task >> [my_2nd_task, say_verygood]
[my_2nd_task, say_verygood] >> t4-echo
t4-echo >> print_log_messages
```

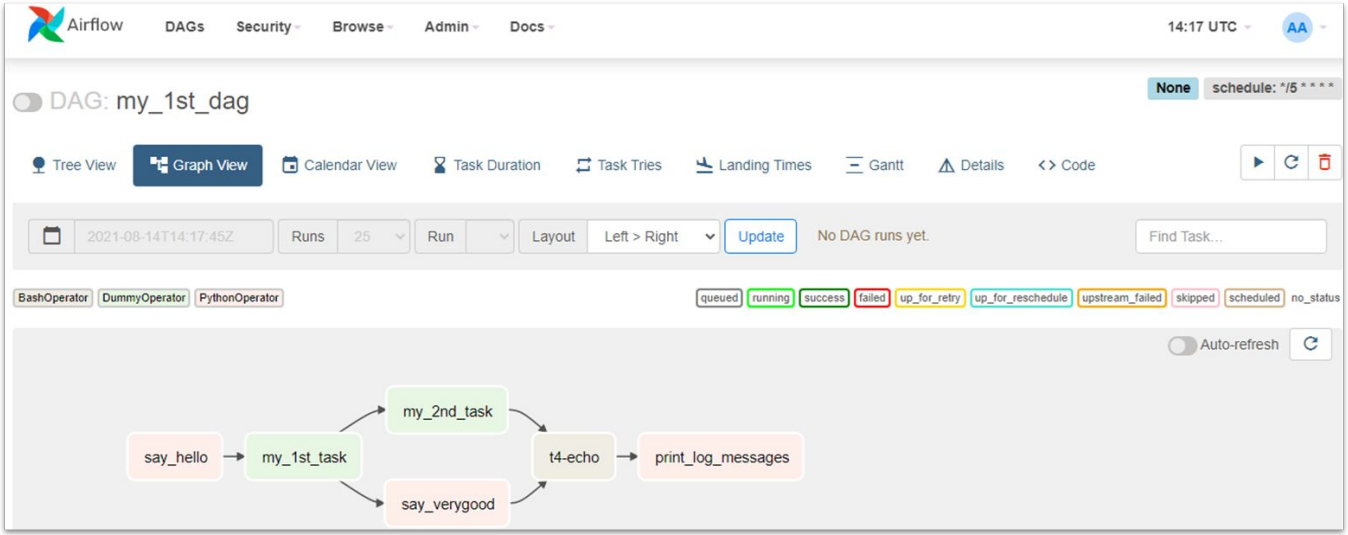
Branching within the DAG (more)




```
start >> [fetch_weather, fetch_sales]
fetch_weather >> clean_weather
fetch_sales >> clean_sales
[clean_weather, clean_sales] >> join_datasets
join_datasets >> train_model >> deploy_model
```

Check point #2

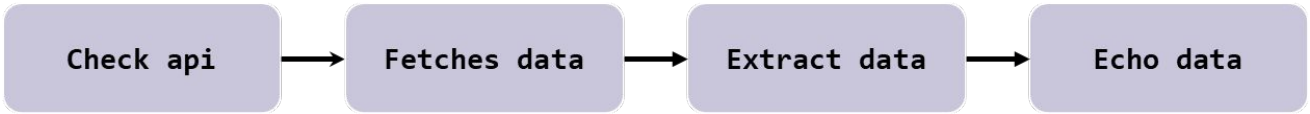
ปรับปรุงไฟล์ my_1st_dag.py ที่ได้สร้างขึ้น ทดลองใช้งาน Operators ตัวอื่น ๆ แล้ว ดู Graph View ใน Airflow UI จับหน้าจอ บันทึกภาพเก็บไว้ส่ง Exercise



Anatomy of an Airflow.

As an example, Let's create a workflow that does the following

- Check if the URL is available
- Fetches some data from an URL
- Extract certain fields from it.
- Print the extracted fields using the bash echo command.



Creating your DAG Definition

First, we will create a skeleton of the workflow, i.e, in this case, is the DAG definition.

```
from airflow.models import DAG

default_args = {
    'start_date': datetime(2020, 1, 1)
}
with DAG('user_processing',
         schedule_interval='@daily',
         default_args=default_args,
         tags=['ETL','ETL demo 1'],
         catchup=False) as dag:
```

Creating our first task

Now, to check if our URL is reachable, we are going to use a sensor operator. In this case, it's an HttpSensor operator.

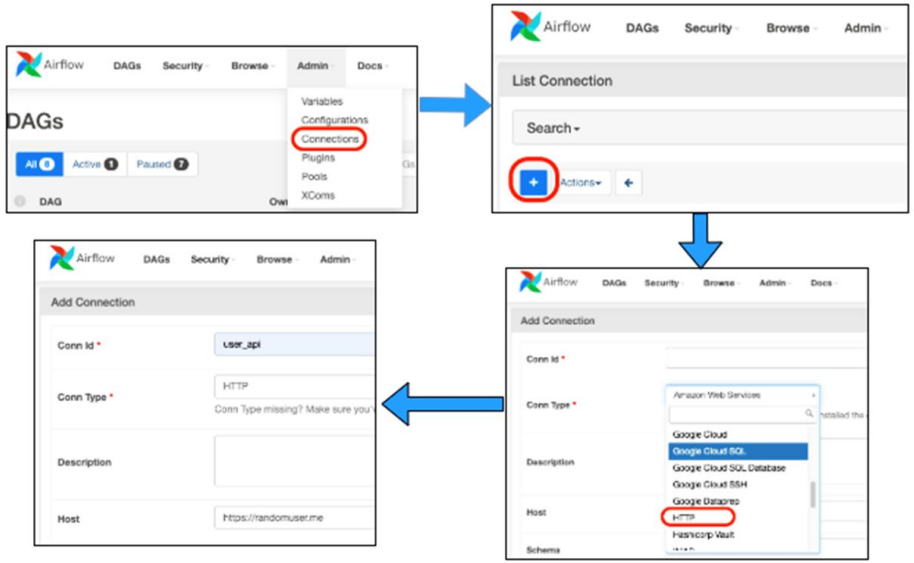
```
with DAG('user_content_processing',
         schedule_interval='@daily',
         default_args=default_args) as dag:

    is_api_available = HttpSensor(
        task_id='is_api_available',
        http_conn_id='user_api',
        endpoint='api/'
    )
```

Creating Http Sensor

URL https://randomuser.me

In our case, we would be using the HTTP connection option.

Here we would provide the URL we want to trigger and setting the connection id to user_api which will return a JSON response of some random user information.



Finally, with this configured

The task is ready to make a call to the URL with the endpoint provided.

Execute next task

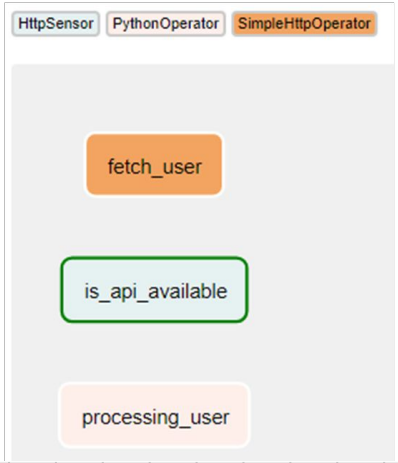
Now, once the first task execution succeeds, we are going to make use of an HttpOperator called SimpleHttpOperator.

```
is_api_available = HttpSensor(
    ...
)

fetch_user = SimpleHttpOperator(
    task_id='fetch_user',
    http_conn_id='user_api',
    endpoint='api/',
    method='GET'
)
```

Xcoms

Xcoms is a way you can share data between your tasks. It basically stores key-value pairs of the information you want to store that can be accessed by other tasks in a DAG.

```
fetch_user = SimpleHttpOperator(
    ...
)

processing_user = PythonOperator(
    task_id='processing_user',
    python_callable=_processing_user
)
```



Let’s look at the function

```
def _processing_user(ti):
    users_txt = ti.xcom_pull(task_ids=["fetch_user"])[0]
    users = json.loads(users_txt)

    if not len(users) or 'results' not in users:
        raise ValueError("User is empty")
    user = users['results'][0]
    user_map = {
        'firstname': user['name']['first'],
        'lastname': user['name']['last']
    }
    processed_user = json_normalize(user_map)
    Variable.set("user", processed_user)
```

Variables

Variables are a way to store and extract some values that you would like to use across any DAG

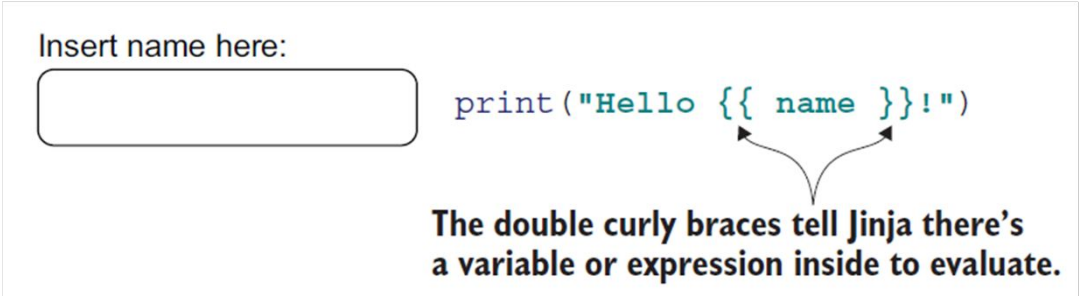
Now that we have stored the values inside Variables, Let print them to the logs using the echo command. To do this we are going to use the BashOperator

```
processing_user = PythonOperator(
    ...
)
print_user = BashOperator(
    task_id='log_user',
    bash_command='echo "{{ var.value.user }}"'
)
```

Templating

Airflow uses Jinja Templating. As you have seen in the code above, Curly braces which allow to materialize some values. In this case, the user key from Variables (using var).

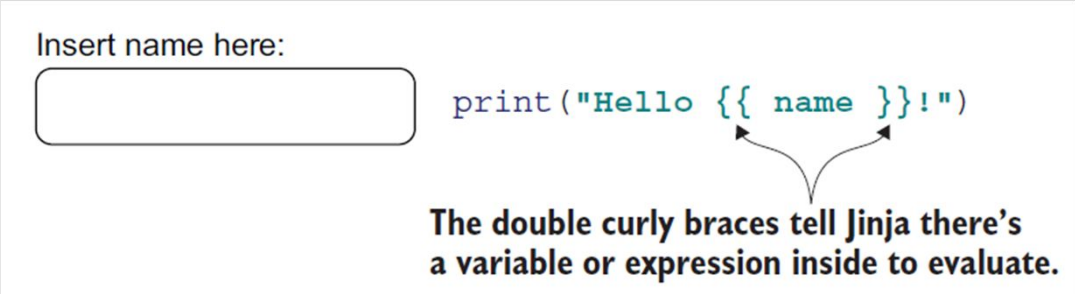
```
{{ var.value.user }}
```



Ordering Your Tasks

Airflow uses Jinja Templating. As you have seen in the code above, Curly braces which allow to materialize some values. In this case, the user key from Variables (using var).

```
{{ var.value.user }}
```



Ordering Your Tasks

Now, With all of this, we have created all the tasks instances.

```
{{ var.value.user }}
```

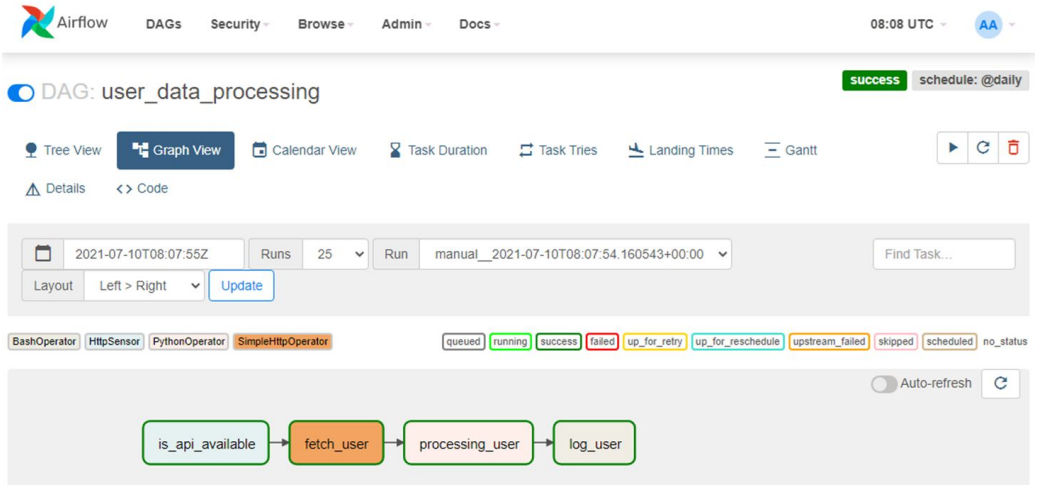
This means Task 1 will run before Task 2. In our case, we will order this as

```
is_api_available >> fetch_user >> processing_user >> print_user
```

Check point #3

We just created a simple Airflow DAG workflow from scratch covering some of the important concepts of Airflow.

Now, enable the DAG and hit run. It will start executing.



Next : AI Agents

What is an AI agentic workflow?

An AI agentic workflow is a mix of AI agents with traditional workflow automation. Unlike standard workflows that follow predefined steps, AI agentic workflows employ intelligent agents to make decisions, adapt to new situations, and autonomously achieve goals.

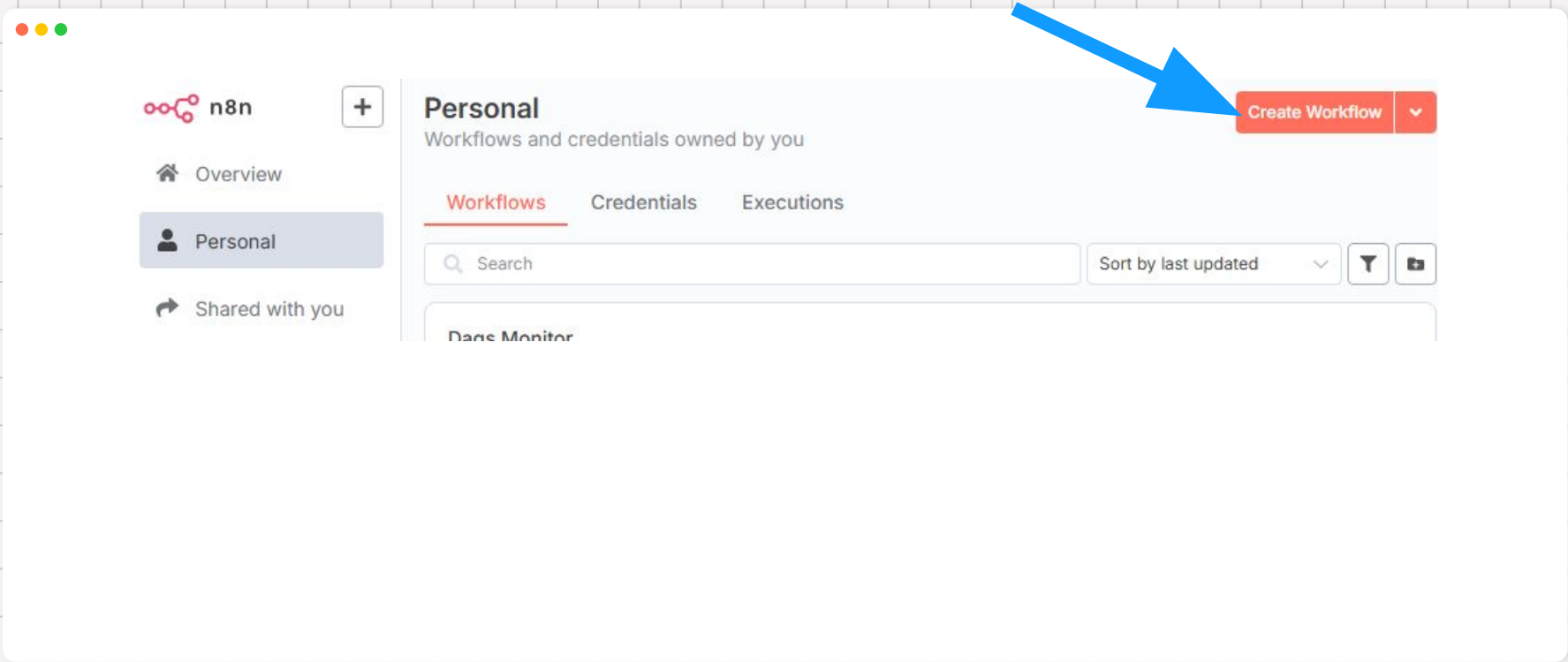
Key characteristics of AI agentic workflow

AI agentic workflows leverage large language models (LLMs) as their "brain power", allowing them to understand complex instructions, reason about tasks, and generate appropriate responses or actions.

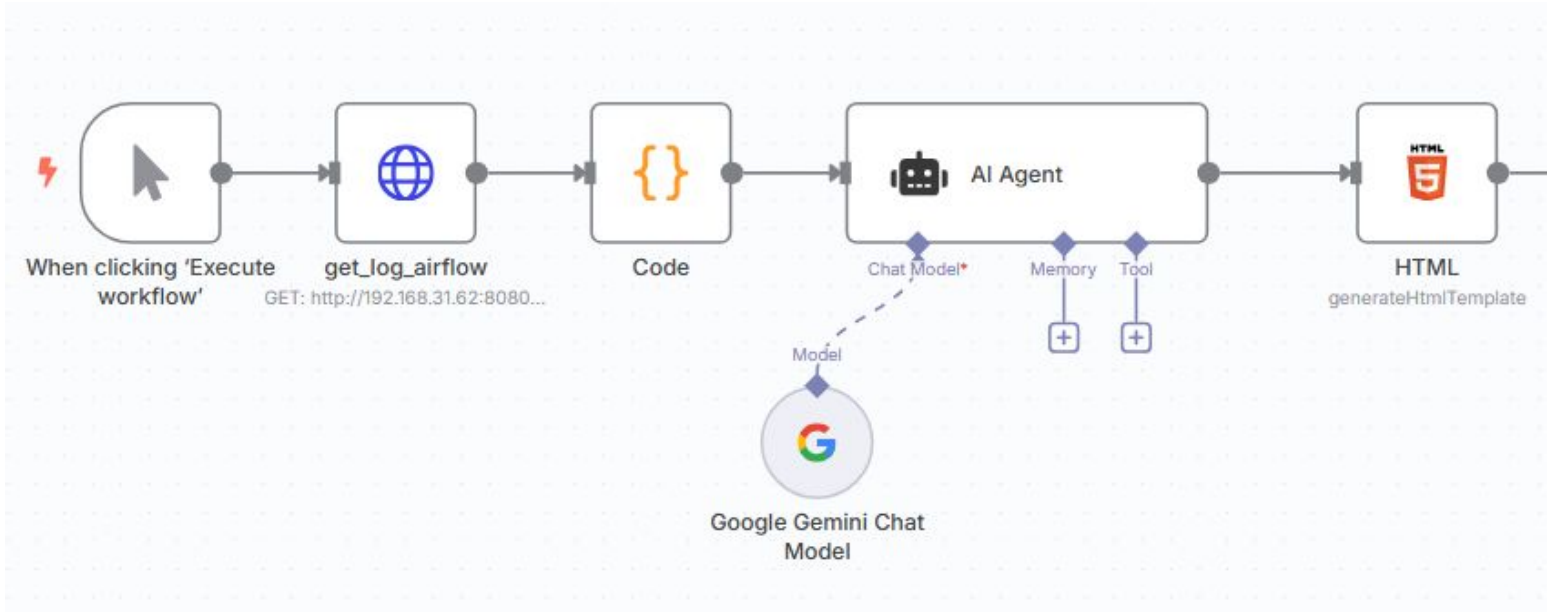
Key characteristics of AI agentic workflow

- **Autonomy**: Agents can operate independently, making decisions without constant human input.
- **Adaptability**: They adjust their actions in response to changes in the environment or new information.
- **Goal-oriented**: Agents work toward specific objectives rather than simply a set of rules.
- **Learning capability**: Many AI agents can improve their performance over time.
- **Scalability**: as agents learn, they can solve increasingly complex tasks without major reprogramming.

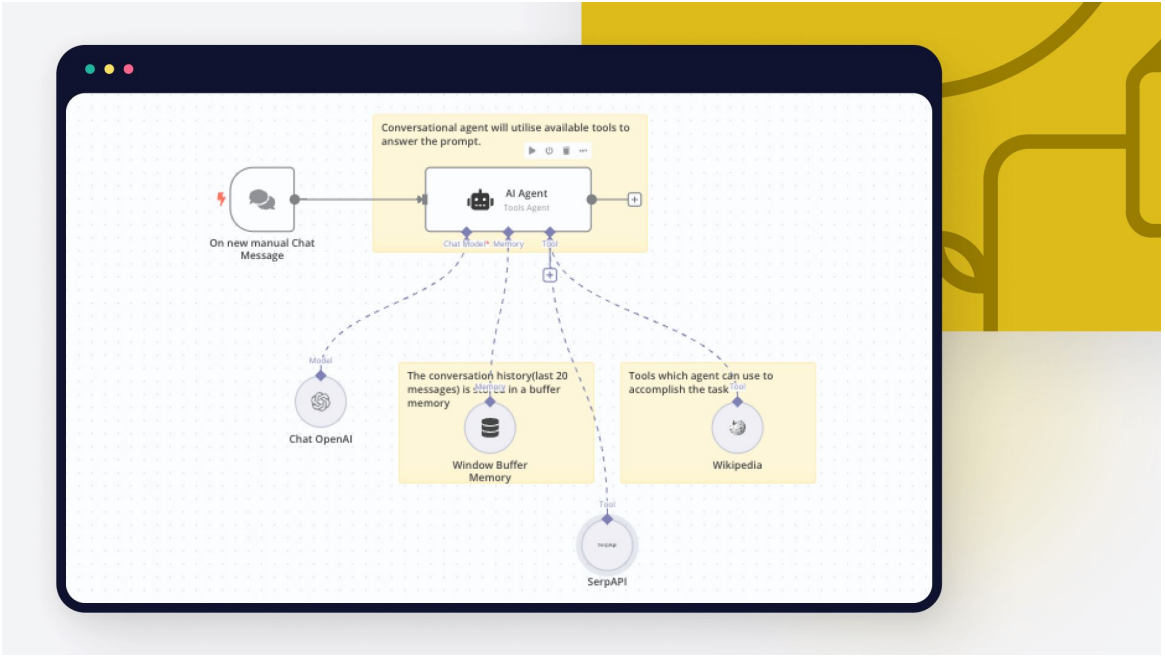
Let’s build your first AI Agent



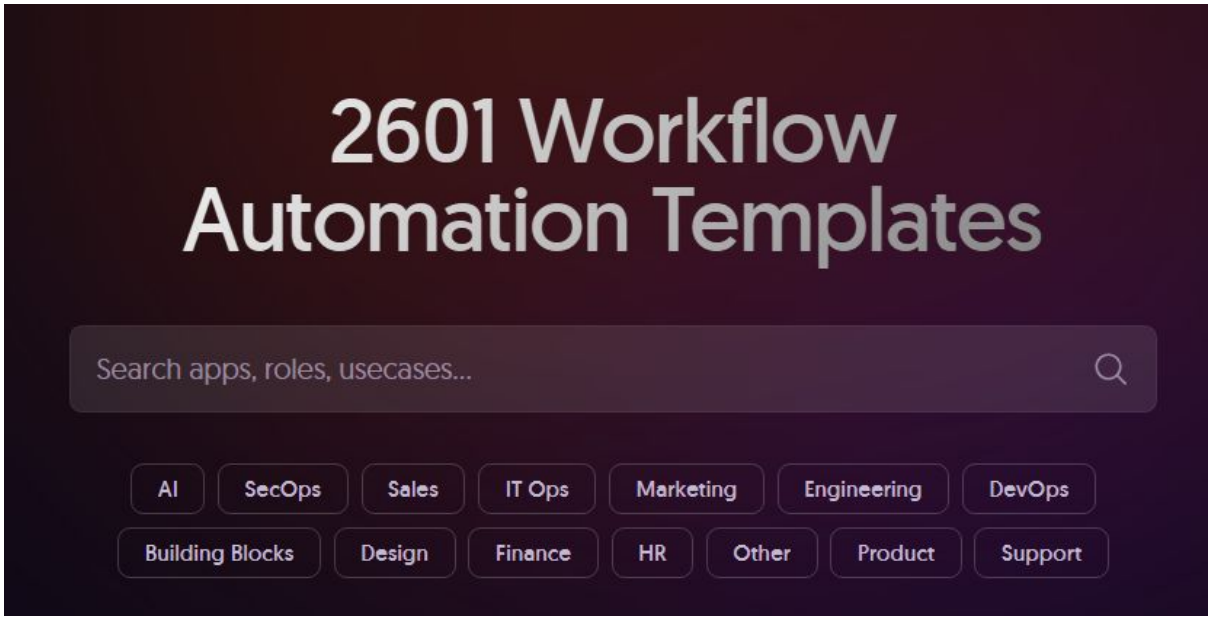
Let’s build your first AI Agent



Your Practical Guide to LLM Agents in 2025



Your turn , Check point #4



PRINCE OF SONGKLA UNIVERSITY

ขอบคุณทุกท่าน