Understanding Airflow Task Dependencies: Linear Chains and Fan-In/Fan-Out

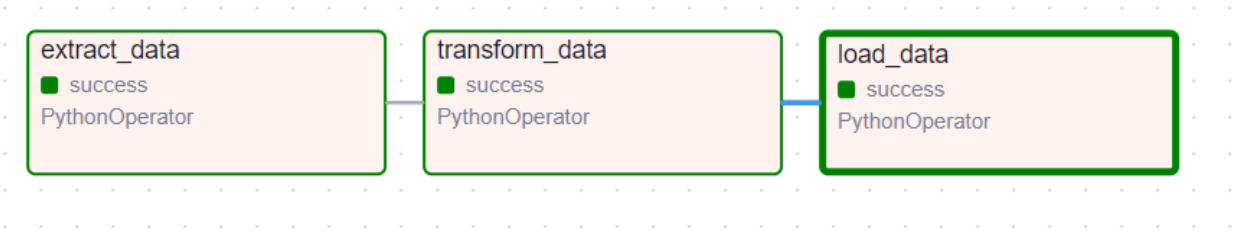
1. Linear Chain of Tasks

A linear chain is the simplest form of task dependency in Airflow. Each task in the chain depends on the completion of the previous one. This setup ensures that tasks are executed sequentially.

Example:

Consider an ETL (Extract, Transform, Load) pipeline with the following tasks:

- **Extract Sales Data**: Extract data from a source system.
- **Aggregate Data**: Perform aggregations and transformations on the extracted data.
- **Load into Platform**: Load the aggregated data into a target platform.



In this setup:

- The Extract Sales Data task starts first.
- The Transform Data task starts only after the Extract Sales Data task finishes.
- The Load into Platform task starts only after the Aggregate Data task finishes.

Usage: Linear chains are ideal for processes that need to be executed in a strict sequence, such as ETL workflows where each step depends on the completion of the previous one.

DAG Definition:

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

# Define functions for each task
def extract_data(**kwargs):
    print("Extracting data...")

def transform_data(**kwargs):
    print("Transforming data...")

def load_data(**kwargs):
    print("Loading data...")

# Define default arguments
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 7, 19),
    'retries': 1,
}

# Define the DAG
dag = DAG(
    'linear_chain_example',
    default_args=default_args,
    description='A simple linear chain DAG',
    schedule_interval='@daily',
)

# Define tasks
extract_task = PythonOperator(
    task_id='extract_data',
    python_callable=extract_data,
    dag=dag,
)

transform_task = PythonOperator(
    task_id='transform_data',
    python_callable=transform_data,
    dag=dag,
)

load_task = PythonOperator(
    task_id='load_data',
    python_callable=load_data,
    dag=dag,
)

# Set task dependencies
extract_task >> transform_task >> load_task
```

2. Fan-In and Fan-Out

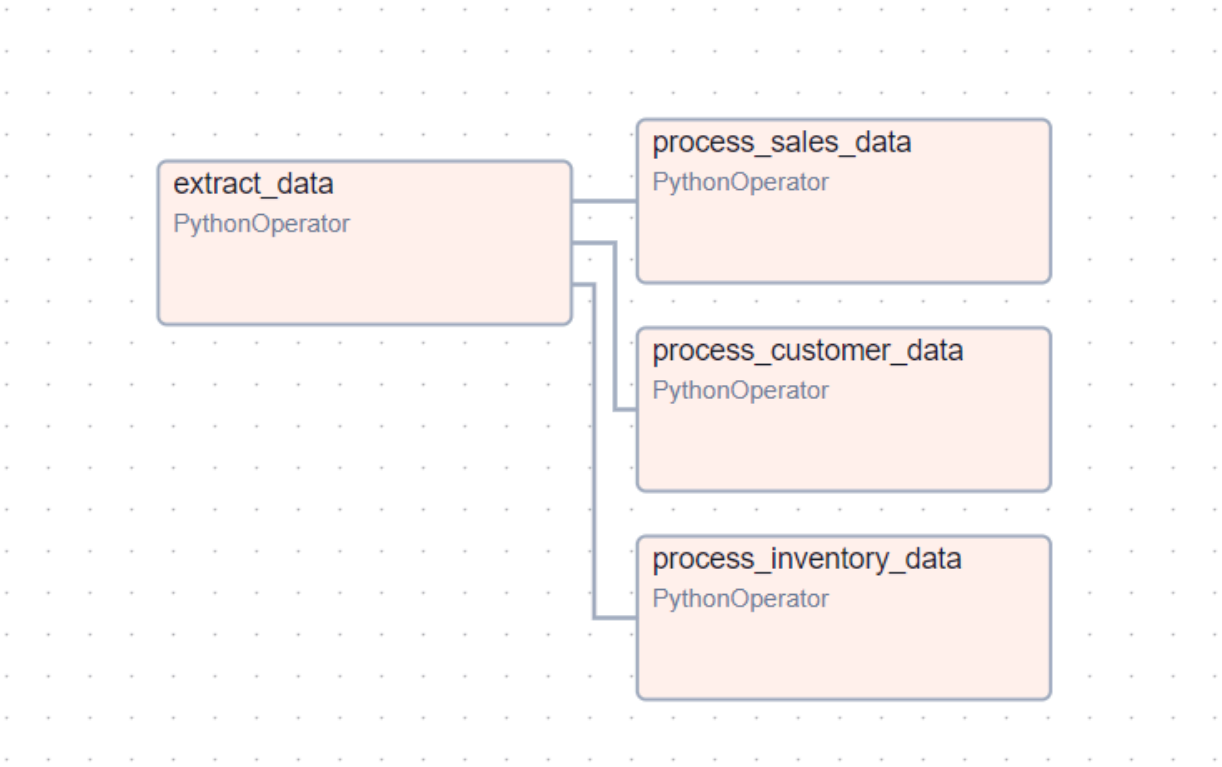
Fan-Out: This pattern is used when a single task needs to branch out into multiple tasks. Each task runs in parallel after the initial task completes.

Example:

Imagine you need to process multiple types of data after a common extraction task:

- **Extract Data**: Extract data from various sources.
- **Process Sales Data**: Process the sales data.
- **Process Customer Data**: Process the customer data.
- **Process Inventory Data**: Process the inventory data.

Visualization:



In this setup:

- The **Extract Data** task completes.
- The **Process Sales Data**, **Process Customer Data**, and **Process Inventory Data** tasks start simultaneously.

Usage: Fan-Out is useful when you need to perform parallel operations after a common task, such as processing different datasets concurrently.

DAG Definition:

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

# Define functions for each task
def extract_data(**kwargs):
    print("Extracting data from various sources...")

def process_sales_data(**kwargs):
    print("Processing sales data...")

def process_customer_data(**kwargs):
    print("Processing customer data...")

def process_inventory_data(**kwargs):
    print("Processing inventory data...")

# Define default arguments
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 7, 19),
    'retries': 1,
}

# Define the DAG
dag = DAG(
    'fan_out_example',
    default_args=default_args,
    description='A DAG demonstrating fan-out for different data processing tasks',
    schedule_interval='@daily',
)

# Define tasks
extract_task = PythonOperator(
    task_id='extract_data',
    python_callable=extract_data,
    dag=dag,
)

process_sales_task = PythonOperator(
    task_id='process_sales_data',
    python_callable=process_sales_data,
    dag=dag,
)

process_customer_task = PythonOperator(
    task_id='process_customer_data',
    python_callable=process_customer_data,
    dag=dag,
)

process_inventory_task = PythonOperator(
    task_id='process_inventory_data',
    python_callable=process_inventory_data,
    dag=dag,
)

# Set task dependencies
extract_task >> [process_sales_task, process_customer_task, process_inventory_task]
```

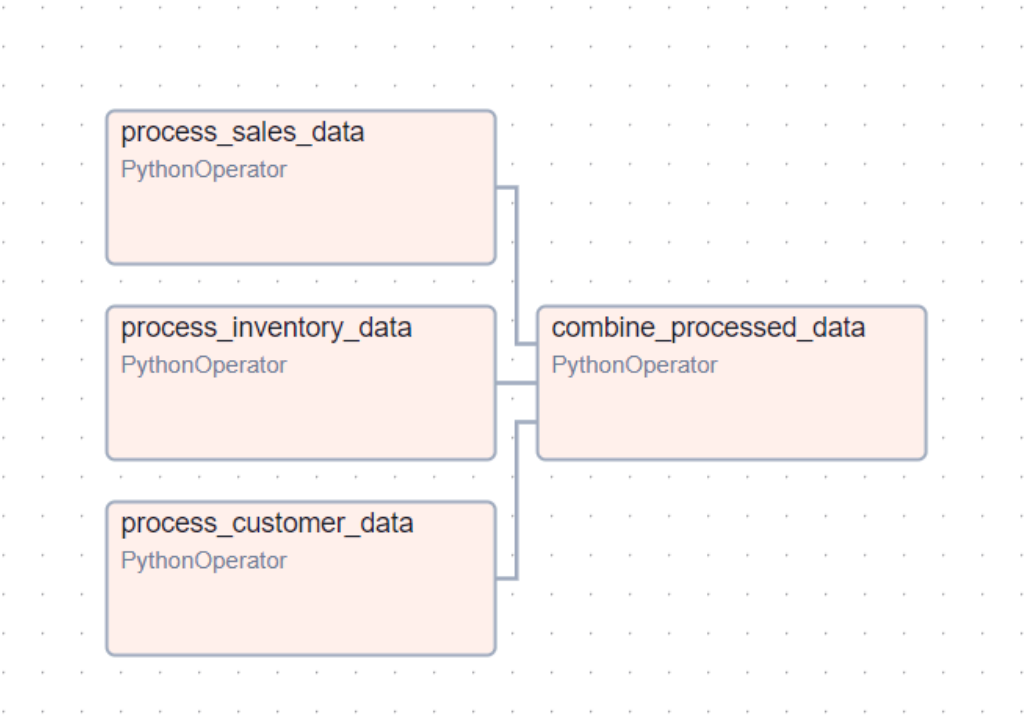
Fan-In: This pattern is used when multiple tasks converge into a single task. All upstream tasks must be completed before the downstream task starts.

Example:

After processing different types of data, you need to combine the results:

- **Process Sales Data**: Process sales data.
- **Process Customer Data**: Process customer data.
- **Process Inventory Data**: Process inventory data.
- **Combine Processed Data**: Combine the results from the three data processing tasks.

Visualization:



In this setup:

- **Process Sales Data**, **Process Customer Data**, and **Process Inventory Data** run in parallel.
- **Combine Processed Data** starts only after all three processing tasks are completed.

Usage: Fan-In is useful for aggregating results from multiple parallel tasks before proceeding to the next step, such as combining processed data from different sources

DAG Definition:

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

# Define functions for each task
def process_sales_data(**kwargs):
    print("Processing sales data...")

def process_customer_data(**kwargs):
    print("Processing customer data...")

def process_inventory_data(**kwargs):
    print("Processing inventory data...")

def combine_processed_data(**kwargs):
    print("Combining processed data...")

# Define default arguments
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 7, 19),
    'retries': 1,
}

# Define the DAG
dag = DAG(
    'fan_in_example',
    default_args=default_args,
    description='A DAG demonstrating fan-in',
    schedule_interval='@daily',
)

# Define tasks
process_sales_task = PythonOperator(
    task_id='process_sales_data',
    python_callable=process_sales_data,
    dag=dag,
)

process_customer_task = PythonOperator(
    task_id='process_customer_data',
    python_callable=process_customer_data,
    dag=dag,
)

process_inventory_task = PythonOperator(
    task_id='process_inventory_data',
    python_callable=process_inventory_data,
    dag=dag,
)

combine_processed_data_task = PythonOperator(
    task_id='combine_processed_data',
    python_callable=combine_processed_data,
    dag=dag,
)

# Set task dependencies
[process_sales_task, process_customer_task, process_inventory_task] >>
combine_processed_data_task
```