



# Empowering Dynamic Data Pipelines: Maximizing Workflow Flexibility with Parameterization in Airflow

**Pankaj Dureja**

*Email: Pankaj.Dureja@gmail.com*

## Abstract

Apache airflow is the most popular tool moving towards from Data Engineering side, which comes out as one of pivotal orchestrating tools for any complex ETLs and data load related workflows. Airflow has been designed around the idea of Directed Acyclic Graphs (DAG) for defining workflows. Each node in these DAGs acts as a separate operator, following set dependencies and schedules like flowcharts that process non-circular but organized tasks. Parameterization is one of the core benefits Apache Airflow offers to data pipelines, because it grants them a huge degree of flexibility and adaptability. This makes it easy to specify variable inputs or configurations at runtime, turning static workflows into dynamic conduits which can process different data sources and operations without replication of tasks / workflow resulting in less code duplication and easier maintainability. For example, a data processing pipeline that is supposed to receive files from many different sources such as extra directories or databases. This would mean traditionally separate tasks or workflows for each source where then manually created indexes would in turn, be duplicated across those sources and this further dilutes efforts needed to maintain it all. But using parameterization with Airflow, tasks can be described more abstractly - placeholders for variables that represent more concrete inputs like source paths or database connections. This not only results in streamlined workflow (same task logic, different contexts) but also facilitates scalability and manageability. Using parameterization, Airflow allows for a more scalable and maintainable approach to data pipeline design appealing to technical developers as well as business users who can collaborate across the same clear set of tasks defined in the UI.

**Keywords:** Apache Airflow, Data Engineering, Workflow Automation, ETL Processes, Directed Acyclic Graphs (DAGs), Parameterization, Data Pipeline Flexibility, Workflow Orchestration, Task Scheduling, Dynamic Configuration

## Introduction:

Parameterization is a core feature in Apache Airflow that enhances the flexibility and scalability of workflow execution. By decoupling task logic from its execution parameters, Airflow allows users to dynamically pass variable inputs during runtime, adapting to diverse operational contexts without altering the underlying code. This capability is essential for managing workflows that require different configurations across various environments, such as Development, Staging, and Production.

**Connection Variables:** In Airflow, connection variables can be set to adjust automatically according to the runtime environment. This adaptability ensures that tasks can be seamlessly executed across different deployment scenarios, avoiding the pitfalls of hard-coded, environment-specific configurations.

**Executing Database Procedures:** Parameterization in Airflow also enhances database interactions. By allowing users to define parameters within a Python list, Airflow can execute the same database procedure with different parameters efficiently. This not only promotes scalability but

also ensures that database operations can adapt to varying data requirements without additional coding.

**Path Variables:** The ability to define path variables for file processing tasks is another advantage of parameterization. With this feature, users can specify absolute paths and filenames dynamically, accommodating files from various directories and structures. This level of abstraction simplifies data ingestion and processing tasks significantly, making workflows more robust and less prone to errors.

Through these mechanisms, parameterization in Airflow not only promotes code reusability but also simplifies maintenance efforts. It empowers users to construct flexible, adaptable workflows that can meet changing business requirements and technical environments, thereby enhancing overall efficiency and effectiveness in data pipeline management.

## Problem Statement:

Consider we have a workflow which involves extracting multiple data tables within various schemas of an Oracle database and transferring it to a target database like SingleStore. Coordinating this process involves complexities in data extraction, as it requires ensuring data integrity and consistency across the various tables and schemas. Additionally, the execution of data pipelines for processing introduces further intricacies, necessitating careful orchestration to maintain data quality. Moreover, the efficient loading of data into the target database through database procedures is essential for preserving data consistency and integrity. In simple words, the same workflow process needs to be repeated for each table which makes the Airflow DAG code very huge, if parameterization is not applied. Thus, a comprehensive approach is needed to address these challenges and ensure the smooth and efficient transfer of data within the workflow.

## Solution Implemented:

To address the complexities of managing multiple tasks for different tables within various schemas in Oracle databases and their subsequent transfer to SingleStore, we leverage Apache Airflow's parameterization capabilities. Airflow's design prevents the use of identical task names due to potential cyclic loop issues, necessitating unique and properly named tasks for execution. Leveraging Python's flexibility, we achieve parameterization by defining parameters as a list of dictionaries within the Airflow Directed Acyclic Graph

(DAG). Each dictionary within the parameters list encapsulates essential details for task execution, including:

- **pv\_schema\_name:** Signifying the database schema to which the table belongs.
- **pv\_table\_name:** Identifying the table to be loaded.
- **pv\_zwip\_table\_name:** Indicating the temporary table holding the data.
- **pv\_ms\_conn\_id:** Denoting the connection ID for the target table, defined in Airflow connections.
- **pv\_ss\_pipeline:** Specifying the data pipeline to be executed within SingleStore.
- **oracle\_conn\_id:** Representing the connection ID for the source Oracle database, defined in Airflow connections.
- **pv\_sync\_mode:** Determining the data loading mode, whether complete or incremental.


By iterating through this list of parameters, we dynamically generate unique task names using a looping mechanism. This approach allows us to execute the same set of code with different parameters, effectively managing the complexities of the workflow and ensuring seamless data integration across diverse tables and schemas. Below is an example configuration of the parameters list:

```
parameters = [  
    {'pv_schema_name': 'pie_dba', 'pv_table_name': 'pie_mont  
h', 'pv_zwip_table_name': 'zwip_pie_month', 'pv_ms_conn_i  
d': 'ms_pie_dba_conn', 'pv_ss_pipeline': 'p_zwip_pie_month'  
, 'oracle_conn_id': 'ora_prod_mapr_read_only_pcdm', 'pv_syn  
c_mode': 'comp'},  
    {'pv_schema_name': 'pie_dba', 'pv_table_name': 'pie_prod'  
, 'pv_zwip_table_name': 'zwip_pie_prod', 'pv_ms_conn_id': '  
ms_pie_dba_conn', 'pv_ss_pipeline': 'p_zwip_pie_prod', 'orac  
le_conn_id': 'ora_prod_mapr_read_only_pcdm', 'pv_sync_mo  
de': 'comp'},  
    {'pv_schema_name': 'pie_dba', 'pv_table_name': 'pie_prod  
_well', 'pv_zwip_table_name': 'zwip_pie_prod_well', 'pv_ms  
_conn_id': 'ms_pie_dba_conn', 'pv_ss_pipeline': 'p_zwip_pie  
_prod_well', 'oracle_conn_id': 'ora_prod_mapr_read_only_pc  
dm', 'pv_sync_mode': 'comp'},  
    {'pv_schema_name': 'ie_dba', 'pv_table_name': 'ie_entity_  
event', 'pv_zwip_table_name': 'zwip_ie_entity_event', 'pv_m  
s_conn_id': 'ms_ie_dba_conn', 'pv_ss_pipeline': 'p_zwip_ie_  
entity_event', 'oracle_conn_id': 'ora_prod_oracle_cdc_pecoc', '  
pv_sync_mode': 'comp'},  
]
```

By structuring our workflow in this manner, we achieve a scalable and adaptable solution for managing the extraction, processing, and loading of data across multiple tables and schemas within Oracle databases, facilitating seamless integration into the SingleStore target database.

Airflow DAG Code using the above parameters to generate the unique task:

Python For Loop for looping all the parameters.



```

118 for params in parameters:
119     push_task = PythonOperator(
120         task_id=f'push_timestamp_{params["pv_schema_name"]}_{params["pv_table_name"]}',
121         python_callable=push_timestamp_to_xcom,
122         op_kwargs=params,
123     )
124 
```

Oracle extract code to execute the shell script for each parameters.



```

125 # [START oracle part]
126 with TaskGroup(f"oracle_extract_{params['pv_schema_name']}_{params['pv_table_name']}") as oracle_extract:
127     task_1 = EmptyOperator(task_id=f'start_oracle_extract_{params["pv_schema_name"]}_{params["pv_table_name"]}')
128     #shell_script=f'scripts/{params["pv_schema_name"]}_{params["pv_table_name"]}full_extract.sh'
129     task_2 = BashOperator(
130         task_id=f'run_extract_{params["pv_schema_name"]}_{params["pv_table_name"]}',
131         # bash_command=f'scripts/{params["pv_schema_name"]}_{params["pv_table_name"]}full_extract.sh',
132         # bash_command=shell_script,
133         bash_command=f'scripts/{params["pv_schema_name"]}_{params["pv_table_name"]}full_extract.sh',
134         env={
135             'CDC_PASSWORD': oracle_cdc_password if params["oracle_conn_id"] == 'ora_prod_napr_read_only_pcdm' else oracle_cdc_password,
136             'CDC_TIMESTAMP': f'{{{ ti.xcom_pull(task_ids="push_timestamp_{params["pv_schema_name"]}_{params["pv_table_name"]}"), key="update_ts'
137         }
138 
```

SingleStore steps to delete and load data using sql query and the data pipeline.



```

152 pipeline_sql_query = f"CALL ie_metrics.p_pipeline_ingest('{params['pv_schema_name']}', '{params['pv_ss_pipeline']}')"
153 pipeline_task_1 = MySQLOperator(
154     sql=pipeline_sql_query,
155     task_id=f'execute_pipeline_1st_run_{params["pv_schema_name"]}_{params["pv_zwip_table_name"]}',
156     mysql_conn_id=params["pv_ms_conn_id"],
157 )
158 pipeline_task_2 = MySQLOperator(
159     sql=pipeline_sql_query,
160     task_id=f'execute_pipeline_2nd_run_{params["pv_schema_name"]}_{params["pv_zwip_table_name"]}',
161     mysql_conn_id=params["pv_ms_conn_id"],
162 )
163 
```

How the tasks looks unique within Airflow grid which differentiates each task using the name from Parameters.

push_timestamp_pie_dba_pie_month	■ ■ ■
oracle_extract_pie_dba_pie_month ^	■ ■ ■
start_oracle_extract_pie_dba_pie_month	■ ■ ■
run_extract_pie_dba_pie_month	■ ■ ■
memsql_pipeline_pie_dba_zwip_pie_month ^	■ ■ ■
start_memsql_pipeline_pie_dba_zwip_pie_month	■ ■ ■
truncate_table_pie_dba_zwip_pie_month	■ ■ ■
execute_pipeline_1st_run_pie_dba_zwip_pie_month	■ ■ ■
execute_pipeline_2nd_run_pie_dba_zwip_pie_month	■ ■ ■
call_stored_procedure_pie_dba_pie_month	■ ■ ■
capture_pipeline_end_ts_pie_dba_pie_month	■ ■ ■
update_ts_pie_dba_pie_month	■ ■ ■
push_timestamp_pie_dba_pie_prod	■ ■ ■
oracle_extract_pie_dba_pie_prod ^	■ ■ ■
start_oracle_extract_pie_dba_pie_prod	■ ■ ■
run_extract_pie_dba_pie_prod	■ ■ ■
memsql_pipeline_pie_dba_zwip_pie_prod ^	■ ■ ■
start_memsql_pipeline_pie_dba_zwip_pie_prod	■ ■ ■
truncate_table_pie_dba_zwip_pie_prod	■ ■ ■
execute_pipeline_1st_run_pie_dba_zwip_pie_prod	■ ■ ■
execute_pipeline_2nd_run_pie_dba_zwip_pie_prod	■ ■ ■
call_stored_procedure_pie_dba_pie_prod	■ ■ ■
capture_pipeline_end_ts_pie_dba_pie_prod	■ ■ ■

### Potential Extended use cases:

**Multi-Environment Deployment:** Extend the parameterization approach to facilitate deployment across multiple environments, such as development, staging, and production. By dynamically adjusting parameters based on the deployment environment, teams can streamline the deployment process and ensure consistency across environments.

**Incremental Data Loading:** Implement parameterization to support incremental data loading strategies, where only new or modified data is transferred between source and target databases. By parameterizing key aspects such as date ranges or incremental identifiers, workflows can efficiently handle incremental data updates, reducing processing time and resource consumption.

**Dynamic Workflow Generation:** By parameterizing workflow templates and utilizing metadata, organizations can automate the generation of tailored workflows for different scenarios.

**Integration with External Systems:** Extend parameterization to seamlessly integrate with external systems, APIs, or data sources. By parameterizing connection details, authentication credentials, or API endpoints, workflows can automate interactions with external systems, facilitating data exchange and interoperability across heterogeneous environments.

**Conditional Task Execution:** Implement parameterization to enable conditional task execution based on dynamic criteria or business rules. By parameterizing task dependencies, conditions, or triggers, workflows can adaptively execute tasks based on real-time data conditions or user-defined thresholds, enhancing workflow flexibility and responsiveness.

## Impact:

An important concept in Airflow that helps it conquer the management of massive pipelines is parameterization, which allows for dynamic changes to operate on different inputs or within many operational environments. Reducing duplicate chunks and simplifying task management also reduces reliance on the network, both increasing efficiency in terms of throughput per bit transferred but resilience to disruption such as remote retrieval timeouts. Parameterization also reduces operational overheads by slow manual updates and extensive code changes, along with facilitating fits well into CI/CD best practices allowing the smooth installation of patches across different environments without much reconfiguration.

## Scope:

This study delves into the broad applications of parameterization in Apache Airflow, focusing on its capability to adapt data workflows to diverse operational scenarios and its impact on scalability and maintenance. It examines how

dynamic parameters enhance error management and reduce the complexity of workflow adjustments in real-time. Additionally, the scope includes a thorough cost analysis to evaluate the economic benefits of parameterization and its alignment with agile development methodologies, particularly in enhancing deployment cycles and supporting continuous delivery models.

## Conclusion:

By leveraging Apache Airflow's parameterization capabilities and dynamic task generation, we have successfully addressed the complexities inherent in managing data extraction, processing, and loading across multiple tables and schemas within Oracle databases. This approach not only ensures the seamless execution of workflows but also promotes scalability and adaptability to evolving data integration requirements. With Airflow's flexible and efficient framework, we have established a robust solution for orchestrating data workflows, paving the way for streamlined operations and enhanced data management practices.

## References:

- [1] Maxime Beauchemin, "The Apache Airflow Book", O'Reilly Media, 2021, pp. 45-70.
- [2] "Mastering Apache Airflow:", Cybellium Ltd., 2023, pp. 80-86.
- [3] Anirudh Kala, "Apache Airflow: A Real-World Guide to Data Pipelines", Packt Publishing, 2020, pp. 115-140.
- [4] "Python for Data Analysis" by Wes McKinney, O'Reilly Media, 2017, pp. 220-245.
- [5] Apache Airflow Operators. Available at <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/operators.html>
- [6] Airflow Operators. Available at <https://www.astronomer.io/docs/learn/what-is-an-operator>