Below is the **Mermaid flowchart** for running a Spark job in **Client Mode**. This execution mode places the Spark driver program on the submitting machine (e.g., an Airflow edge node).

### **Mermaid Flowchart for Client Mode Execution**

mermaid

Copy code

flowchart TD

subgraph Airflow\_Trigger

A1[Airflow DAG Trigger]

A2[SparkSubmitOperator]

A1 --> A2

end

A2 -->|Submit spark-submit| B1[Driver Program (on Airflow Edge Node)]

B1 -->|Connects to| C1[ResourceManager (YARN/Mesos/K8s)]

C1 -->|Allocate resources| D1[Cluster Manager]

D1 -->|Launch Executors| E1[Spark Executors (on Worker Nodes)]

E1 -->|Access HDFS| F1[HDFS or Data Source]

E1 -->|Write Output| F2[HDFS or Result Storage]

B1 -->|Logs Progress| G1[Airflow Logs]

%% Styling for clarity

classDef airflow fill:#FFD966,stroke:#333,stroke-width:2px;

classDef cluster fill:#B6D7A8,stroke:#333,stroke-width:2px;

classDef execution fill:#A4C2F4,stroke:#333,stroke-width:2px;

A1,A2:::airflow

B1:::airflow

C1,D1:::cluster

E1,F1,F2:::execution

### **Key Elements in Client Mode Flowchart**

1. **Airflow DAG Trigger**:
   * The Spark job is triggered by a task in an Airflow Directed Acyclic Graph (DAG).
   * SparkSubmitOperator is configured to submit the job in **Client Mode**.
2. **Driver Program**:
   * The driver runs on the machine where the spark-submit command is executed (Airflow edge node).
   * It connects to the Resource Manager to request resources for Spark Executors.
3. **Resource Manager**:
   * Manages resource allocation in the cluster (e.g., YARN or Kubernetes).
   * Allocates containers for Spark Executors.
4. **Cluster Manager**:
   * Coordinates the launching of executors across worker nodes.
   * Examples: YARN NodeManager, Kubernetes Pods, or Mesos tasks.
5. **Spark Executors**:
   * Executors run on worker nodes, performing the actual computation.
   * Communicate with the driver for task execution.
6. **HDFS or Data Source**:
   * Spark Executors read data from HDFS or other storage systems (e.g., S3 or Cassandra).
   * The results are stored back in HDFS or another specified location.
7. **Airflow Logs**:
   * The Airflow edge node logs the progress of the Spark job, including job status and error messages.

### **Example Code for Client Mode**

python

Copy code

from airflow import DAG

from airflow.providers.apache.spark.operators.spark\_submit import SparkSubmitOperator

from datetime import datetime

dag = DAG(

'spark\_job\_client\_mode',

schedule\_interval='@daily',

start\_date=datetime(2023, 1, 1),

catchup=False,

)

spark\_submit\_task = SparkSubmitOperator(

task\_id='spark\_submit\_client',

application='/path/to/your\_spark\_app.py',

conn\_id='spark\_default',

deploy\_mode='client', # Client mode specified here

name='spark\_client\_job',

execution\_timeout=600,

dag=dag,

)

### **Use Case for Client Mode**

1. **Development/Testing**:
   * Logs are immediately available on the edge node for debugging.
   * Driver failures can be quickly analyzed since the driver resides on the submitting machine.
2. **Low-Latency Jobs**:
   * For quick, interactive workloads, client mode eliminates the overhead of deploying the driver on the cluster.

### **Comparison with Cluster Mode**

| **Aspect** | **Client Mode** | **Cluster Mode** |
| --- | --- | --- |
| **Driver Location** | Runs on the submitting machine (e.g., Airflow edge node). | Runs on a worker node in the cluster. |
| **Use Case** | Development, testing, or debugging. | Production batch jobs. |
| **Fault Tolerance** | Lower; driver depends on submitting machine. | Higher; driver runs within the cluster. |
| **Latency** | Lower for setup, as the driver starts locally. | Higher due to driver setup on the cluster. |

This flowchart and explanation demonstrate how the **Client Mode** setup works for Spark jobs triggered by Airflow. Let me know if you need additional clarifications or modifications!

##### **You said:**

Amazing job, please act spark/airflow expert and create the same flowchart for cluster moder

###### **ChatGPT said:**

**GPT Icon**

Thank you! Below is the **Mermaid flowchart** for running a Spark job in **Cluster Mode**, where the Spark driver is executed on one of the cluster nodes rather than the submitting machine.

### **Mermaid Flowchart for Cluster Mode Execution**

mermaid

Copy code

flowchart TD

subgraph Airflow\_Trigger

A1[Airflow DAG Trigger]

A2[SparkSubmitOperator]

A1 --> A2

end

A2 -->|Submit spark-submit| B1[ResourceManager (YARN/Mesos/K8s)]

B1 -->|Allocate resources| C1[Cluster Manager]

C1 -->|Launch Driver| D1[Driver Program (on Cluster Node)]

D1 -->|Request Resources| C2[Cluster Manager]

C2 -->|Launch Executors| E1[Spark Executors (on Worker Nodes)]

E1 -->|Access HDFS| F1[HDFS or Data Source]

E1 -->|Write Output| F2[HDFS or Result Storage]

D1 -->|Logs Progress| G1[Airflow Logs]

%% Styling for clarity

classDef airflow fill:#FFD966,stroke:#333,stroke-width:2px;

classDef cluster fill:#B6D7A8,stroke:#333,stroke-width:2px;

classDef execution fill:#A4C2F4,stroke:#333,stroke-width:2px;

A1,A2:::airflow

B1,C1,D1:::cluster

E1,F1,F2:::execution

### **Key Elements in Cluster Mode Flowchart**

1. **Airflow DAG Trigger**:
   * Airflow initiates the Spark job via a DAG using the SparkSubmitOperator.
2. **ResourceManager**:
   * The spark-submit command interacts with the Resource Manager (e.g., YARN or Kubernetes) to allocate resources.
3. **Cluster Manager**:
   * The Cluster Manager coordinates the setup of the Spark Driver and Executors.
   * It ensures resource distribution across the cluster.
4. **Driver Program**:
   * In cluster mode, the Spark Driver runs on a designated node within the cluster instead of the submitting machine.
   * The Driver orchestrates task execution and communicates with Spark Executors.
5. **Spark Executors**:
   * Executors run on worker nodes to perform parallel computations.
   * They interact with HDFS or another data storage system to process and write data.
6. **HDFS or Data Source**:
   * Executors fetch data from HDFS or other data sources like Amazon S3, GCS, or NoSQL databases.
7. **Airflow Logs**:
   * The Airflow server logs the progress and status of the job, allowing for monitoring and debugging.

### **Example Code for Cluster Mode**

python

Copy code

from airflow import DAG

from airflow.providers.apache.spark.operators.spark\_submit import SparkSubmitOperator

from datetime import datetime

dag = DAG(

'spark\_job\_cluster\_mode',

schedule\_interval='@daily',

start\_date=datetime(2023, 1, 1),

catchup=False,

)

spark\_submit\_task = SparkSubmitOperator(

task\_id='spark\_submit\_cluster',

application='/path/to/your\_spark\_app.py',

conn\_id='spark\_default',

deploy\_mode='cluster', # Cluster mode specified here

name='spark\_cluster\_job',

execution\_timeout=600,

dag=dag,

)

### **Use Case for Cluster Mode**

1. **Production Workloads**:
   * **Batch Processing**: Long-running jobs benefit from the fault tolerance of cluster mode.
   * **Scalability**: The driver operates in the cluster, reducing dependency on the submitting machine.
2. **High Availability**:
   * In cluster mode, the Driver node is managed within the cluster and can handle node failures better than client mode.

### **Comparison: Cluster Mode vs. Client Mode**

| **Aspect** | **Cluster Mode** | **Client Mode** |
| --- | --- | --- |
| **Driver Location** | Runs on a cluster node. | Runs on the submitting machine (e.g., Airflow edge). |
| **Fault Tolerance** | Higher; Driver operates within the cluster. | Lower; Driver depends on the submitting machine. |
| **Latency** | Higher for setup, as Driver is deployed in the cluster. | Lower, since the Driver runs locally. |
| **Use Case** | Production-grade workloads. | Development, debugging, and quick iterative testing. |

This flowchart and explanation outline how **Cluster Mode** operates for Spark jobs triggered by Airflow. Let me know if you'd like further details or comparisons! 😊