how to trigger event notification for s3 in airflow

ChatGPT

As of my knowledge cutoff in September 2021, Apache Airflow does not provide built-in functionality to directly trigger AWS S3 event notifications. Instead, AWS S3 event notifications are typically configured using AWS services like AWS Lambda or AWS EventBridge (formerly known as Amazon CloudWatch Events). These services can be used to trigger external processes, such as an Apache Airflow DAG, when events (e.g., object creation, deletion) occur in an S3 bucket.

Here's a high-level overview of how you can set up S3 event notifications to trigger an Apache Airflow DAG:

1. **Create an AWS Lambda Function**:
   * Write an AWS Lambda function that will be triggered by the S3 event (e.g., object creation) you are interested in. You can use the AWS SDK for Python (Boto3) in your Lambda function to interact with Apache Airflow's REST API and trigger your DAG.
   * In the Lambda function, you can use the **requests** library or any other HTTP client to make an HTTP POST request to the Airflow REST API endpoint to trigger your DAG.
2. **Configure S3 Event Notifications**:
   * In the AWS S3 Management Console or using AWS CLI/API, configure the S3 bucket to send an event notification (e.g., "ObjectCreated") to the AWS Lambda function you created in the previous step. This will ensure that the Lambda function is triggered whenever an object is created in the specified S3 bucket.
3. **Set Up Airflow DAG**:
   * In your Apache Airflow DAG, create a task that corresponds to the processing you want to perform when an S3 event is triggered. For example, you can create a PythonOperator that runs a Python function to handle the event.
   * Ensure that the DAG is properly scheduled and has all the necessary connections and variables configured.
4. **Apache Airflow Webserver**:
   * Make sure your Apache Airflow webserver is running and accessible.

With this setup, whenever a new object is created in the S3 bucket, S3 will send an event notification to the AWS Lambda function. The Lambda function, in turn, will trigger your Airflow DAG by making an HTTP POST request to the Airflow REST API endpoint. The DAG will then be executed, and the corresponding task will perform the processing logic you defined.

Please note that using event-driven architecture with AWS services like AWS Lambda and Apache Airflow can have some complexity, and it's essential to follow best practices to handle any potential failures or scaling requirements in your setup. Additionally, there might have been updates or new features related to S3 event notifications and Apache Airflow since my knowledge cutoff date. Therefore, I recommend consulting the latest AWS documentation and Apache Airflow documentation for the most up-to-date information and best practices.

Airflow to Amazon Simple Storage Service (S3) integration provides several operators to create and interact with S3 buckets.

The **S3KeySensor** is useful when you want to coordinate the execution of tasks in your DAG based on the presence of specific files or objects in an S3 bucket. It allows you to design workflows that wait for external data to become available before proceeding with further processing.

from airflow import DAG

from airflow.operators.dummy\_operator import DummyOperator

from airflow.contrib.sensors.aws\_s3\_key\_sensor import S3KeySensor

from datetime import datetime, timedelta

default\_args = {

'owner': 'airflow',

'depends\_on\_past': False,

'start\_date': datetime(2023, 8, 1),

'retries': 1,

'retry\_delay': timedelta(minutes=5),

}

dag = DAG(

'wait\_for\_s3\_file',

default\_args=default\_args,

schedule\_interval=None, # You can manually trigger the DAG or use a schedule\_interval

catchup=False,

)

# Wait for the specific S3 file to be present in the bucket

wait\_for\_file\_sensor = S3KeySensor(

task\_id='wait\_for\_file',

bucket\_key='path/to/s3/file.txt', # The key of the file you want to wait for

bucket\_name='your-s3-bucket', # Replace with your S3 bucket name

wildcard\_match=False, # Set to True if you want to use wildcard matching

poke\_interval=60, # Interval in seconds to check for the file

timeout=600, # Sensor timeout in seconds

dag=dag,

)

# This is the task that will run after the file is detected by the sensor

process\_file\_task = DummyOperator(

task\_id='process\_file',

dag=dag,

)

# Set the task dependencies

wait\_for\_file\_sensor >> process\_file\_task