airflow-hdinsight

Release 0.0.1.3

Jul 06, 2020

Contents:

A set of airflow hooks, operators and sensors to allow airflow DAGs to operate with the Azure HDInsight platform, for cluster creation and monitoring as well as job submission and monitoring. Also included are some enhanced Azure Blob and Data Lake sensors.

This project is both an amalgamation and enhancement of existing open source airflow extensions, plus new extensions to solve the problem.

CHAPTER 1

Installation

```
pip install airflow-hdinsight
```

CHAPTER 2

Extensions

`airflowhdi`

Type	Name	What it does
Hook	AzureHDInsightHook	Uses the HDInsightManagementClient from the HDInsight SDK for Python to expose several operations on an HDInsight cluster - get cluster state, create, delete.
Operator	AzureHDInsightCreateClusterOperator	Use the AzureHDInsightHook to create a cluster
Operator	AzureHDInsightDeleteClusterOperator	Use the AzureHDInsightHook to delete a cluster
Operator	ConnectedAzureHDInsightCreateClusterOperator	Extends the AzureHDInsightCreateClusterOperator to allow fetching of the security credentials and cluster creation spec from an airflow connection
Operator	AzureHDInsightSshOperator	Uses the AzureHDInsightHook and SSHHook to run an SSH command on the master node of the given HDInsight cluster
Sensor	AzureHDInsightClusterSensor	A sensor to monitor the provisioning state or running state (can switch between either mode) of a given HDInsight cluster. Uses the AzureHDInsightHook.
Sensor	WasbWildcardPrefixSensor	An enhancement to the WasbPrefixSensor to support sensing on a wildcard prefix
Sensor	AzureDataLakeStorageGen1WebHdfsSensor	Uses airflow's AzureDataLakeHook to sense a glob path (which implicitly supports wildcards) on ADLS Gen 1. ADLS Gen 2 is not yet supported in airflow.

`airflowlivy`

Type	Name	What it does
Hook	Livy-Batch-Hook	Uses the Apache Livy Batch API to submit spark jobs to a livy server, get batch state, verify batch state by quering either the spark history server or yarn resource manager, spill the logs of the spark job post completion, etc.
Operator	Livy-Batch-Operator	Uses the LivyBatchHook to submit a spark job to a livy server
Sensor	Livy-Batch-Sensor	Uses the LivyBatchHook to sense termination and verify completion, spill logs of a spark job submitted earlier to a livy server

Origins of the HDinsight operator work

The HDInsight operator work is loosely inspired from alikemalocalan/airflow-hdinsight-operators, however that has a huge number of defects, as to why it was never accepted to be merged into airflow in the first place. This project solves all of those issues and more, and is frankly a full rewrite.

Origins of the livy work

The livy batch operator is based on the work by panovvv's project airfllow-livy-operators. It does some necessary changes:

- Seperates the operator into a hook (LivyBatchHook), an operator (LivyBatchOperator) and a sensor (LivyBatchSensor)
- Adds additional verification and log spilling to the sensor (the original sensor does not)
- Removes additional verification and log spilling from the operator - hence alllowing a async pattern akin to the EMR add step operator and step sensor.
- Creates livy, spark and YARN airflow connections dynamically from an Azure HDInsight connection
- Returns the batch ID from the operator so that a sensor can use it after being passed through XCom
- Changes logging to LoggingMixin calls
- Allows templatization of fields

State of airflow livy operators in the wild..

As it stands today (June of 2020), there are multiple airflow livy operator projects out there:

- panovvv/airflow-livy-operators: the project which this project bases its work on
- the official livy provider in airflow 2.0, with a backport available for airflow 1.1.x: alas the official provider has very limited functionality - it does not spill the job's logs, and it does not do additional verification for job completion using spark history server or yarn resource manager, amongst other limitations
- rssanders3/airflow-spark-operator-plugin: this is the oldest livy operator, which only supports livy sessions and not batches. there's a copy of this in alikemalocalan/airflow-hdinsight-operators.

2.1 airflowhdi package

2.1.1 Subpackages

2.1.1.1 airflowhdi.hooks package

class airflowhdi.hooks.**AzureHDInsightHook**(*azure_conn_id='azure_default'*)
Bases: airflow.hooks.base_hook.BaseHook

Uses the HDInsightManagementClient from the HDInsight SDK for Python to expose several operations on an HDInsight cluster: get cluster state, create, delete.

Example HDInsight connection

Parameters **azure_conn_id** (*string*) – connection ID of the Azure HDInsight cluster. See example above.

create_cluster(*cluster_create_properties: azure.mgmt.hdinsight.models._models_py3.ClusterCreateProperties*, *cluster_name*)
Creates an HDInsight cluster

This operation simply starts the deployment, which happens asynchronously in azure. You can call *get_cluster_state()* for polling on its provisioning.

Note: This operation is idempotent. If the cluster already exists, this call will simple ignore that fact. So this can be used like a "create if not exists" call.

Parameters

- **cluster_create_properties** (*ClusterCreateProperties*) – the ClusterCreateProperties representing the HDI cluster spec. You can explore some sample specs here. This python object follows the same structure as the HDInsight arm template.

  Example ClusterCreateProperties

- **cluster_name** (*string*) – The full cluster name. This is the unique deployment identifier of an HDI cluster in Azure, and will be used for fetching its state or submitting jobs to it HDI cluster names have the following restrictions.

delete_cluster(*cluster_name*)
Delete and HDInsight cluster

Parameters **cluster_name** (*string*) – the name of the cluster to delete

get_cluster_state(*cluster_name*) → azure.mgmt.hdinsight.models._models_py3.ClusterGetProperties
Gets the cluster state.

get_conn() → azure.mgmt.hdinsight._hd_insight_management_client.HDInsightManagementClient
Return a HDInsight Management client from the Azure Python SDK for HDInsight

This hook requires a service principal in order to work. You can create a service principal from the az CLI like so:

```
az ad sp create-for-rbac --name localtest-sp-rbac --skip-assignment \
  --sdk-auth > local-sp.json
```

See also:

- Create an Azure AD app & service principal in the portal - Microsoft identity platform
- Azure HDInsight SDK for Python
- How to authenticate Python applications with Azure services

2.1.1.2 airflowhdi.operators package

class airflowhdi.operators.**AzureHDInsightSshOperator**(*cluster_name*, *azure_conn_id='azure_hdinsight_default'*, **args*, ***kwargs*)

Bases: airflow.contrib.operators.ssh_operator.SSHOperator

Uses the AzureHDInsightHook and SSHHook to run an SSH command on the master node of the given HDInsight cluster

The SSH username and password are fetched from the azure connection passed. The SSH endpoint and port of the cluster is also fetched using the *airflowhdi.hooks.AzureHDInsightHook.get_cluster_state()* method.

Parameters

- **cluster_name** (*str*) – Unique cluster name of the HDInsight cluster
- **azure_conn_id** – connection ID of the Azure HDInsight cluster.
- **command** (*str*) – command to execute on remote host. (templated)
- **timeout** (*int*) – timeout (in seconds) for executing the command.
- **environment** (*dict*) – a dict of shell environment variables. Note that the server will reject them silently if *AcceptEnv* is not set in SSH config.
- **do_xcom_push** (*bool*) – return the stdout which also get set in xcom by airflow platform
- **get_pty** (*bool*) – request a pseudo-terminal from the server. Set to `True` to have the remote process killed upon task timeout. The default is `False` but note that *get_pty* is forced to `True` when the *command* starts with `sudo`.

execute(*context*)

Raises **AirflowException** – when the SSH endpoint of the HDI cluster cannot be found

class airflowhdi.operators.**AzureHDInsightCreateClusterOperator**(*cluster_name*, *cluster_params: azure.mgmt.hdinsight.models._models_py azure_conn_id='azure_hdinsight_defaul *args*, ***kwargs*)

Bases: airflow.models.baseoperator.BaseOperator

See also:

See the documentation of *airflowhdi.hooks.AzureHDInsightHook* for explanation on the parameters of this operator

Parameters

- **azure_conn_id** (*string*) – connection ID of the Azure HDInsight cluster.
- **cluster_name** (*str*) – Unique cluster name of the HDInsight cluster

- **cluster_params** (*ClusterCreateProperties*) – the azure.mgmt.hdinsight.models.ClusterCreateProperties representing the HDI cluster spec. You can explore some sample specs here. This python object follows the same structure as the HDInsight arm template.

  Example ClusterCreateProperties

execute(*context*)

This is the main method to derive when creating an operator. Context is the same dictionary used as when rendering jinja templates.

Refer to get_template_context for more context.

template_fields = ['cluster_params']

class airflowhdi.operators.**ConnectedAzureHDInsightCreateClusterOperator**(*azure_conn_id=None*, *hdi_conn_id=None*, **args*, ***kwargs*)

Bases: airflowhdi.operators.azure_hdinsight_create_cluster_operator.AzureHDInsightCreateClusterOperator

An extension of the *AzureHDInsightCreateClusterOperator* which allows getting credentials and other common properties for azure.mgmt.hdinsight.models.ClusterCreateProperties from a connection

Parameters

- **azure_conn_id** (*string*) – connection ID of the Azure HDInsight cluster.
- **hdi_conn_id** (*str*) – connection ID of the connection that contains a azure.mgmt.hdinsight.models.ClusterCreateProperties object in its extra field
- **cluster_params** (*ClusterCreateProperties*) – cluster creation spec
- **cluster_name** (*str*) – Unique cluster name of the HDInsight cluster

execute(*context*)

This is the main method to derive when creating an operator. Context is the same dictionary used as when rendering jinja templates.

Refer to get_template_context for more context.

param_field_types = [<enum 'OSType'>, <enum 'Tier'>, <class 'azure.mgmt.hdinsight.mode

class airflowhdi.operators.**AzureHDInsightDeleteClusterOperator**(*cluster_name*, *azure_conn_id='azure_hdinsight_defaul*, **args*, ***kwargs*)

Bases: airflow.models.baseoperator.BaseOperator

See also:

See the documentation of *airflowhdi.hooks.AzureHDInsightHook* for explanation on the parameters of this operator

execute(*context*)

This is the main method to derive when creating an operator. Context is the same dictionary used as when rendering jinja templates.

Refer to get_template_context for more context.

2.1.1.3 airflowhdi.sensors package

class airflowhdi.sensors.**AzureHDInsightClusterSensor**(*cluster_name*, *azure_conn_id='azure_hdinsight_default'*, *provisioning_only=False*, *poke_interval=20*, *mode='poke'*, **args*, ***kwargs*)

Bases: airflow.sensors.base_sensor_operator.BaseSensorOperator

Asks for the state of the HDInsight cluster until it achieves the desired state: provisioning or running If it fails the sensor errors, failing the task.

See also:

See the documentation of *airflowhdi.hooks.AzureHDInsightHook* for explanation on the parameters of this operator

Parameters

- **cluster_name** (*str*) – name of the cluster to check the state of
- **azure_conn_id** (*str*) – azure connection to get config from
- **provisioning_only** (*bool*) – poke up till provisioning only if *True*, else poke till the cluster hasn't achieved a terminal state

FAILED_STATE = [<HDInsightClusterProvisioningState.failed: 'Failed'>, <HDInsightCluste

NON_TERMINAL_STATES = [<HDInsightClusterProvisioningState.in_progress: 'InProgress'>,

PROV_ONLY_TERMINAL_STATES = [<HDInsightClusterProvisioningState.in_progress: 'InProgr

poke(*context*)

Function that the sensors defined while deriving this class should override.

template_fields = ['cluster_name']

class airflowhdi.sensors.**AzureDataLakeStorageGen1WebHdfsSensor**(*glob_path*, *azure_data_lake_conn_id='azure_data_*, **args*, ***kwargs*)

Bases: airflow.operators.sensors.BaseSensorOperator

Waits for blobs matching a wildcard prefix to arrive on Azure Data Lake Storage.

Parameters

- **glob_path** – glob path, allows wildcards
- **azure_data_lake_conn_id** – connection reference to ADLS

poke(*context*)

Function that the sensors defined while deriving this class should override.

template_fields = ('glob_path',)

ui_color = '#901dd2'

class airflowhdi.sensors.**WasbWildcardPrefixSensor**(*container_name*, *wildcard_prefix*, *wasb_conn_id='wasb_default'*, *check_options=None*, **args*, ***kwargs*)

Bases: airflow.operators.sensors.BaseSensorOperator

Waits for blobs matching a wildcard prefix to arrive on Azure Blob Storage.

Parameters

- **container_name** (*str*) – Name of the container.
- **wildcard_prefix** (*str*) – Prefix of the blob. Allows wildcards.
- **wasb_conn_id** (*str*) – Reference to the wasb connection.
- **check_options** (*dict*) – Optional keyword arguments that *WasbHook.check_for_prefix()* takes.

See also:

See the documentation of `airflow.contrib.sensors.wasb_sensor.WasbBlobSensor`

poke(*context*)
Function that the sensors defined while deriving this class should override.

template_fields = ('container_name', 'wildcard_prefix')

2.1.2 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

U

W