Upgrade Check Script

Getting the Airflow Upgrade Check Package

Apache Airflow is published as `apache-airflow` package in PyPI. The Upgrade Check Script is part of a separate Python Package, since it is separate from the core Apache Airflow package and is only needed for a period of time and specifically only for upgrading from Airflow 1.10 releases to Airflow 2.0.

While there has been a lot of work put into making this upgrade as easy as possible, there are some changes which are compatible between Airflow 1.10 and Airflow 2.0. In order to make this as simple to navigate, we recommend that people first upgrade to the latest release in the 1.10 series (at the time of writing: 1.10.14) and then to download this package and run the script as detailed below.

Note

On November 2020, new version of PIP (20.3) has been released with a new, 2020 resolver. This resolver might work with Apache Airflow as of 20.3.3, but it might lead to errors in installation. It might depend on your choice of extras. In order to install Airflow you might need to either downgrade pip to version 20.2.4 `pip install --upgrade pip==20.2.4` or, in case you use Pip 20.3, you need to add option `--use-deprecated legacy-resolver` to your pip install command.

While `pip 20.3.3` solved most of the `teething` problems of 20.3, this note will remain here until we set `pip 20.3` as official version in our CI pipeline where we are testing the installation as well. Due to those constraints, only `pip` installation is currently officially supported.

While they are some successes with using other tools like poetry or pip-tools, they do not share the same workflow as `pip` - especially when it comes to constraint vs. requirements management. Installing via `Poetry` or `pip-tools` is not currently supported.

If you wish to install airflow using those tools you should use the constraint files and convert them to appropriate format and workflow that your tool requires.

```
pip install apache-airflow-upgrade-check
```

This will install the latest version of the Airflow Upgrade check package.

Running the Airflow Upgrade Check Package

```
airflow upgrade_check
```

This will print out to the screen a number of actions that should be taken before upgrading the Airflow release to 2.0.0 or higher.

The exit code of the command will be 0 (success) if no problems are found in running the command, or 1 if problems were encountered in running the check.

A sample output as a result of a successful run of the upgrade check is shown below.

```
============================================== STATUS ==============================================

Check for latest versions of apache-airflow and checker.................................SUCCESS
Remove airflow.AirflowMacroPlugin class.................................................SUCCESS
Chain between DAG and operator not allowed..............................................SUCCESS
Connection.conn_id is not unique........................................................SUCCESS
Connection.conn_type is not nullable....................................................SUCCESS
Fernet is enabled by default............................................................FAIL
GCP service account key deprecation.....................................................SUCCESS
Changes in import paths of hooks, operators, sensors and others.........................FAIL
Users must delete deprecated configs for KubernetesExecutor.............................FAIL
Legacy UI is deprecated by default.......................................................SUCCESS
Logging configuration has been moved to new section.....................................FAIL
Removal of Mesos Executor...............................................................SUCCESS
Users must set a kubernetes.pod_template_file value.....................................FAIL
SendGrid email uses old airflow.contrib module..........................................SUCCESS
Changes in import path of remote task handlers..........................................SUCCESS
Jinja Template Variables cannot be undefined.............................................FAIL
Found 7 problems.

========================================= RECOMMENDATIONS ==========================================

Fernet is enabled by default
----------------------------
The fernet mechanism is enabled by default to increase the security of the default installation.

Problems:

1.  fernet_key in airflow.cfg must be explicitly set empty as fernet mechanism is enabledby default. This means that the apache-airflow

Changes in import paths of hooks, operators, sensors and others
---------------------------------------------------------------
Many hooks, operators and other classes has been renamed and moved. Those changes were part of unifying names and imports paths as desc
The contrib folder has been replaced by providers directory and packages:
https://github.com/apache/airflow#backport-packages

Problems:

1.  Using ``airflow.operators.python_operator.PythonOperator`` will be replaced by ``airflow.operators.python.PythonOperator``. Affecte
```

The following sections describe what is being done and how to apply the recommendations shown above. Please note that the above results shown are only a partial set, where only the first two of the seven problems identified are shown in the section above. In reality, all the problems are shown on the screen.

Understanding what is being checked

The Upgrade Check checks the configuration data from airflow.cfg, the meta data from the Airflow database, as well as the DAGs which have been set up in the current Airflow environment.

Using the above results as an example, there are two specific problems which have been identified.

The first problem is identified in the configuration file airflow.cfg where the current configuration option for the fernet_key is no longer acceptable and needs to be changed. This is because as of Airflow 2.0, the fernet_key cannot be left empty, but needs to have a defined value. Examining the problematic airflow.cfg and searching for the fernet_key entries would show the following:

```
fernet_key =
```

The second problem was identified in one of the DAGs. In this case, this import statement for the `PythonOperator` needs to be changed, since the location is different in Airflow 2.0. Examining the DAG file would probably show the following:

```
from airflow.operators.python_operator import PythonOperator
```

We will discuss how to fix these and make them compatible with Airflow 2.0 in the next section.

Applying the Recommendations

In most cases, the Recommendations result section of the Upgrade check contains enough information to make the change.

For the first problem identified above with respect to the fernet_key, the solution is to enter a valid value in the Airflow Configuration file airflow.cfg for the fernet_key.

For the second problem, looking at the source of the DAG file and changing the import statement for the Python Operator to be as follows will make this DAG work in Airflow 2.0.

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

However, at the time of writing, this is incompatible in Airflow 1.10.14. So, this change can only be made while moving to Airflow 2.0.

Turning off checks

Advanced Airflow users or those with multiple Airflow deployments may want to customize the Upgrade Checks to their environment by turning off certain checks which are not applicable to them. An example of this is users with Airflow deployments not using the `KubernetesPodOperator` may want to turn off upgrade checks related to the `KubernetesPodOperator` .

This can be done by creating an “upgrade config file” in YAML as shown below:

```
ignored_rules:
    - PodTemplateFileRule
```

To use this configuration file (named `upgrade-configuration.yaml` for this example) while running the upgrade check script, use the following command syntax:

```
airflow upgrade_check --config=./upgrade-configuration.yaml
```

Adding custom checks

Advanced Airflow users or those with multiple Airflow deployments may also want to add additional upgrade checks for specific elements in their environment whether it is DAGs or configuration related.

These additional checks should be defined in a Python class and added as `custom_rules` in the “upgrade config file” as shown below:

```
custom_rules:
    - path.CustomCheckClass1
    - path.CustomCheckClass2
```

Now, invoke the upgrade check script using this configuration file as shown below (the config file is named `upgrade-configuration.yaml` in this example):

```
airflow upgrade_check --config=./upgrade-configuration.yaml
```

Previous Next

Want to be a part of Apache Airflow? Join community

License Donate Thanks
Security