To convert csv to parquet we need python libraries that can be achieved by various snaps:

Script snap

Remote python script

Unix Snap- <https://community.snaplogic.com/t5/designing-and-running-pipelines/how-to-use-3rd-party-python-libraries-in-python-script/m-p/19542><https://docs-snaplogic.atlassian.net/wiki/spaces/SD/pages/2561606032/Unix+Execute>

Pyspark snap

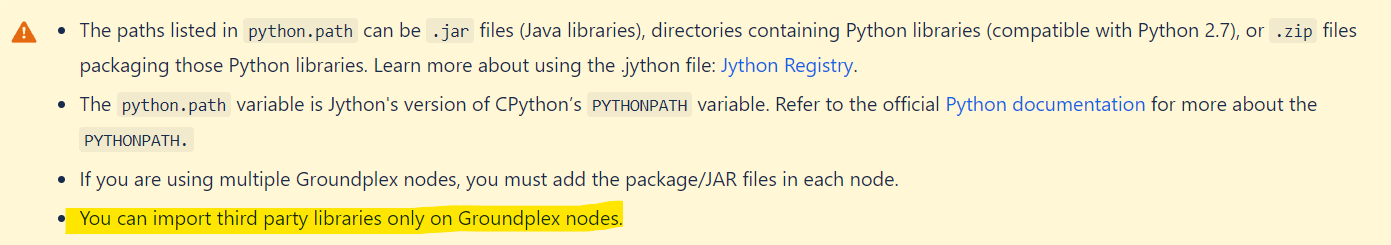
Parquet writer

SCRIPT SNAP IN SNAPLOGIC

This script snap allows ruby, java script & python language to execute.

PYTHON

Basically, it will accept all the python code with built import to run but while executing it requires certain methods to execute the code properly



There is a structure of code to execute properly

The **ScriptHook** interface that can be used by the script:

from com.snaplogic.scripting.language import ScriptHook

from com.snaplogic.scripting.language.ScriptHook import \*

class TransformScript(ScriptHook):

def \_\_init\_\_(self,input, output, error, log):

self.input = input

self.output = output

self.error = error

self.log = log

# The "execute ()" method is called once when the pipeline is started

# and allowed to process its inputs or just send data to its outputs.

def execute(self):

#Write code to execute

def cleanup(self):

self.log.info ("Cleaning up")

# The Script Snap will look for a ScriptHook object in the "hook"

# variable. The snap will then call the hook's "execute" method.

hook = TransformScript(input, output, error, log)

By this way we can make it execute successfully   
Inside the execute method can write any code and make it run successfully

Basic Ideallgy is to connect out pipeline result as an input and work according to the script written

Self.input = input that sends the input data from the pipeline

**attributes and methods of ScriptHook**

**"Error"**

**"Input"**

**"Output"**

**"\_\_class\_\_"**

**"\_\_copy\_\_"**

**"\_\_deepcopy\_\_"**

**"\_\_delattr\_\_"**

**"\_\_doc\_\_"**

**"\_\_ensure\_finalizer\_\_"**

**"\_\_eq\_\_"**

**"\_\_format\_\_"**

**"\_\_getattribute\_\_"**

**"\_\_hash\_\_"**

**"\_\_init\_\_"**

**"\_\_ne\_\_"**

**"\_\_new\_\_"**

**"\_\_reduce\_\_"**

**"\_\_reduce\_ex\_\_"**

**"\_\_repr\_\_"**

**"\_\_setattr\_\_"**

**"\_\_str\_\_"**

**"\_\_subclasshook\_\_"**

**"\_\_unicode\_\_"**

**"class"**

**"cleanup"**

**"equals"**

**"execute"**

**"getClass"**

**"hashCode"**

**"notify"**

**"notifyAll"**

**"toString"**

**"wait"**

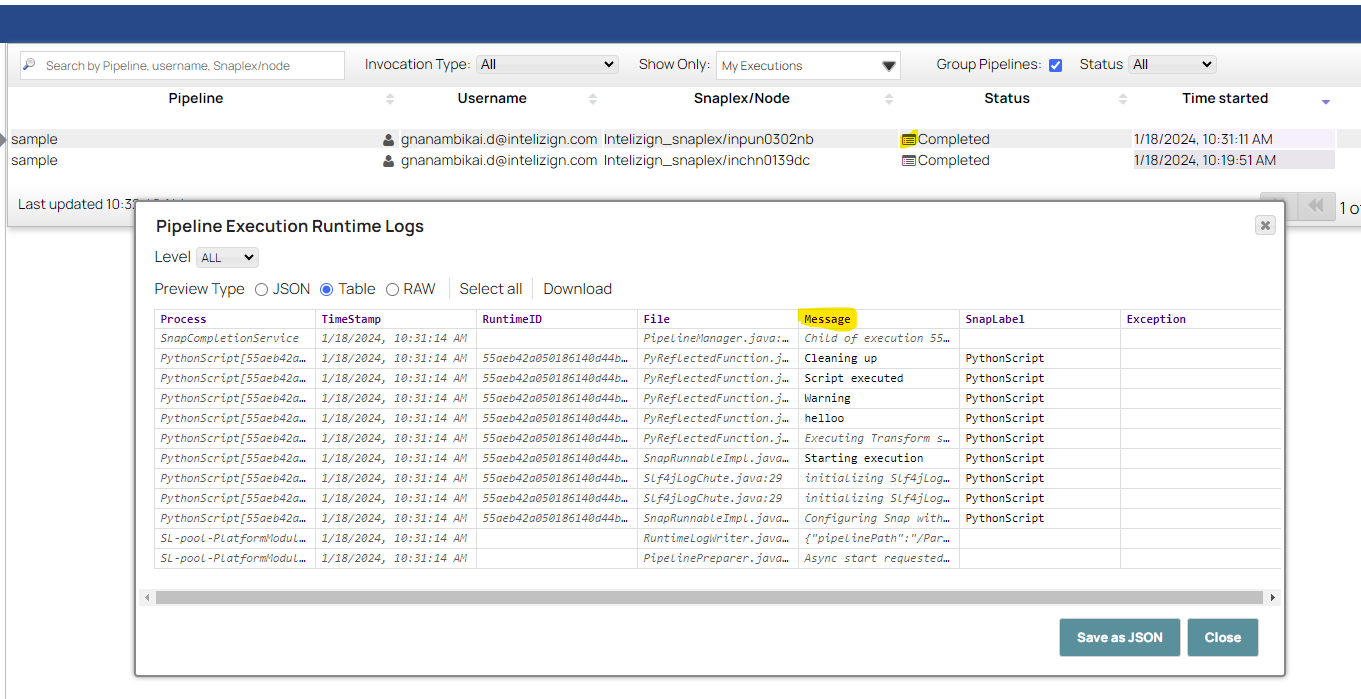
ScriptHook is an interface provided by SnapLogic to enable scripts to interact with the data flow in a pipeline. It provides the script access to the input, output, error, and log objects.

The key things to know about scriptHook are:

* It is a global variable that must be declared in the script. The name is case-sensitive.
* It allows the script to access the following objects:
  + input - provides access to read input documents
  + output - allows writing output documents
  + error - allows writing error documents
  + log - provides access to logging
* The script defines a class that implements ScriptHook. An instance of this class is assigned to the global hook variable.
* The Script snap will call the execute() method on the hook instance during pipeline execution. This is where the main script logic goes.
* The cleanup() method can be used for any cleanup tasks after execution.

So in summary, scriptHook enables interaction with the pipeline data flow and logging from within a script snap. It is a key interface that scripts should implement to function properly in SnapLogic.

Logs can be viewed after executing the pipeline in dashboard



Here In script snap, we use **ONLY JAVA JARS** & write in terms of language we select (python code Syntax) and we can achieve the result.

--------------------------------------------------------------------------------------------------------------------

**REMOTE PYTHON SCRIPT**

**Use the built-in script editor in the Snap configuration. This allows you to write and edit the Python code directly in the Snap:**

* In the Snap configuration, select the "Script" tab.
* Select the "Edit Script" button to open the script editor.
* Write your Python code here. You can access the input, output, error and log objects to process data.

**Pass an external script file into the Snap:**

* Use a File Reader Snap to read a Python script file from disk or SFTP.
* Pass this file into the second input view of the Remote Python Script Snap.
* The code from the external file will be executed instead of the default script.

**Import modules and packages:**

* You can import Python modules and packages in your script.
* For third party packages, add them to the python.path in the .jython config file.
* Then import and use them in your script as needed.

**Access Java libraries:**

* You can also call Java code from the Python script.
* Add the Java JAR files to the python.path in the .jython config file.
* Then import and use the Java classes in your script

**THIRD-party librarirs**

Here are the steps to use third party python libraries in the Remote Python Script Snap:

1. Install the libraries on the machine where the Remote Python Executor (RPE) is running. You can do this by SSHing into the machine and running pip install <library> for each library you want to use.
2. Import the libraries in your python script inside the Snap. For example:

import pandas as pd  
import numpy as np

1. Add the following code snippet at the beginning of your script to ensure the libraries are available:

from snaplogic.common.sl\_tools import SLTool  
  
SLTool.ensure\_libraries(['pandas', 'numpy'])

This will check if the libraries are installed and raise an error if not.

1. Save the script and publish to the pipeline.
2. In the Accounts page, select the RPE account used by the Snap and click "Test Connection" to validate the libraries are accessible.

The key steps are installing the libraries on the RPE machine, importing them in your script, and using SLTool.ensure\_libraries() to verify availability. This will allow you to leverage third party libraries when executing python scripts from the Snap.

**-------------------------------------------------------------------------------------------------------------------------------------** #Imports

#Global Variables

#This function will be executed once before consuming the data.

def snaplogic\_init():

return None

#This function will be executed on each document from the upstream snap.

def snaplogic\_process(row):

output = {}

output["original"] = row

return output

#This function will be executed after consuming all documents from the upstream snap.

def snaplogic\_final():

return None

The def snaplogic\_init() function in the Remote Python Script Snap is used to initialize any global variables or perform any setup required before the Snap starts processing data.

The **def snaplogic\_process(row)** function is one of the key functions used in the Remote Python Script Snap to process input data. Here is what it does:

* It is called for each incoming row of data to the Snap.
* The row parameter contains the input document passed to the Snap.
* This function allows you to implement the data transformation or processing logic that will be applied to each input row.
* Any processing done on the input row can mutate the row object directly.
* The return value of this function is ignored - you don't need to return the row.
* After snaplogic\_process() returns, the mutated row object will be passed downstream in the Pipeline.

So in summary, this function allows you to define the per-row processing logic for the Snap. Each input row is passed to this function, you can perform any actions on that row, and then the changed row is automatically passed along to downstream Snaps. This allows you to focus on just the data transformation logic without worrying about the Snap input and output handling.

The return type should be a dictionary or list of dictionary