AWS S3, Matillion Python and AWS Lambda for Python

Contents

[AWS S3, Matillion Python and AWS Lambda for Python 3](#_Toc89163547)

[Matillion 3](#_Toc89163548)

[Grid Variables in Python 3](#_Toc89163549)

[To get Grid Variable data and load it into a Python array: 3](#_Toc89163550)

[And to place Python array data into a Grid Variable: 3](#_Toc89163551)

[Grid-Based Components 4](#_Toc89163552)

[Matillion Python Scripts 4](#_Toc89163553)

[Matillion Python Scripts component is not designed for large scale data processing scripts involving PySpark or other such compute-intensive packages 5](#_Toc89163554)

[Example: moves all of the objects within an S3 bucket into another S3 bucket 5](#_Toc89163555)

[Example: database query which retrieves a single (aggregate) row of data, and stores the result into a Variable 6](#_Toc89163556)

[AWS Lambda for Python 6](#_Toc89163557)

[Building Lambda functions with Python 7](#_Toc89163558)

[Example lambda-function.py 7](#_Toc89163559)

[AWS S3 and Python Bota3 8](#_Toc89163560)

[AWS S3 review 8](#_Toc89163561)

[Uploading objects using presigned URLs 9](#_Toc89163562)

[S3 Bota3 Python Functions 9](#_Toc89163563)

[Matillion and Lambda 9](#_Toc89163564)

[Read Settings 10](#_Toc89163565)

[MAIN JOB 10](#_Toc89163566)

[Python Script for creating and initializing Python Variables 10](#_Toc89163567)

[Also see Matillion Cookbook 10](#_Toc89163568)

[Archive CSV for Iterators 12](#_Toc89163569)

[Example: Achieve CSV for Dimensions 12](#_Toc89163570)

[Matillion Grid Commands 13](#_Toc89163571)

[**Example: Grid variable updating in python script** 14](#_Toc89163572)

# AWS S3, Matillion Python and AWS Lambda for Python

# Matillion

## Grid Variables in Python

Warning: Variables become first class variables in Python and Bash scripts and great care should be taken to avoid naming them in a manner that clashes with key words in either language. We recommend using a prefix (for example, v\_) to ensure no such conflicts occur.

It is possible to access and update Grid Variables in Python Scripts.

### To get Grid Variable data and load it into a Python array:

**context.getGridVariable('<GridName>')**

### And to place Python array data into a Grid Variable:

**context.updateGridVariable('<GridName>', <values>)**

Below is an example Python3 script that takes data from one Grid Variable 'people' into an array and prints it before updating a different Grid Variable 'names' using that array.

**# get grid variables from people and please in an array\_A in python**

array\_A = context.getGridVariable('people')

for data in array:

print(data)

**# update the grid from array\_A**

context.updateGridVariable('names', array\_A)

In your example, if you created a Grid Variable called people and the default values were ['matt', 'frank', 'glen'], you then run the job the job or just the component that executes context.updateGridVariable('people', [['Kyle']]), your new value will be just Kyle until the job ends and then it will revert back to the default ['matt', 'frank', 'glen'].

Keep in mind when you run context.updateGridVariable('people', [['Kyle']]) you are replacing the default values with just Kyle not adding to what is already existing. Therefore your Grid Variable for that run will only contain Kyle. If you want to add to it then you will need to get the Grid Variable values and then add on to those values and then update the grid variable with the whole collection which would include the Kyle you added. Hopefully this helps clarify this a bit more.

In a Matillion Python Script component, grid variables are represented as a list of lists. So it sounds like you might be trying to update a grid variable using a scalar value (string or int) instead of a list?

## Grid-Based Components

* [**Table Metadata To Grid:**](https://documentation.matillion.com/docs/2930331) Take metadata from a table and load it into a grid
* [**Remove From Grid:**](https://documentation.matillion.com/docs/2930142) Remove rows from a preexisting grid.
* [**Append To Grid:**](https://documentation.matillion.com/docs/2930141) Add rows to a preexisting grid.
* [**Query Result To Grid:**](https://documentation.matillion.com/docs/2930140) Query a table and load resulting data into a grid.

# Matillion Python Scripts

|  |  |
| --- | --- |
| * Python scripts within this component are executed by the underlying virtual machine (VM) hosting Matillion ETL, and use the memory and compute of this VM. | * This component is not designed for large scale data processing scripts involving PySpark or other such compute-intensive packages. |
| * To ensure that instance credentials access is managed correctly at all times, we always advise that customers limit scopes (permissions) where applicable. * Python script is executed in-process by an interpreter of the user's choice (Jython, Python2 or Python3). Any output written via print statements will appear as the task completion message, and so output should be brief. | * While it is valid to handle exceptions within the script using try/except, any uncaught exceptions will cause the component to be marked as failed and its failure link to be followed. * You may import any modules from the Python Standard Library. Matillion ETL does not uninstall any customer-installed Python libraries. Matillion ETL runs as a Tomcat user and care must be taken to ensure this user has sufficient access to resources. |
| **Matillion Variables and Python**   * When run, the Python Script component creates a set of new variables of the same name, type, and default value as those listed in the Environment Variables list. Thus, Environment Variables can be used within the script (the syntax ${variable} is not required, you may simply use variable. * Since the Python script already contains Python counterparts of the Environment Variables, users should be careful to not use those same names for their own variables, especially when of a different type. * Please Note: The Python script variables will disappear after the Python script ends. If you need to push values back to Environment Variables to use in other components later in the job, use the special 'context' object, like so:   context.updateVariable("variable", "new value")  Both arguments are strings that should parse as the target variable type. | **Matillion Grid Variables and Python**   * Similar to Variables, Grid Variables can also be accessed through the Python Script component. Details on using Grid Variables in this manner can be found in the Grid Variables documentation. |

## Matillion Python Scripts component is not designed for large scale data processing scripts involving PySpark or other such compute-intensive packages

## [Example: moves all of the objects within an S3 bucket into another S3 bucket](https://documentation.matillion.com/docs/2234735)

import boto3

s3\_resource = boto3.resource('s3')

new\_bucket\_name = "targetBucketName"

bucket\_to\_copy = "sourceBucketName"

s3bucket = s3\_resource.Bucket(bucket\_to\_copy)

for obj in s3bucket.objects.all():

files = obj.key

copy\_source = {'Bucket': bucket\_to\_copy,'Key': files}

s3\_resource.meta.client.copy(copy\_source, new\_bucket\_name, files)

print(files)

The Python script imports the “boto” module and uses it to move the files. In fact, the script copies the objects to the other bucket, and then removes the source object. A similar script could instead rename the objects and leave them within the same bucket. A list of available variables is given on the left of the window, and used in code written on the right. The script can be executed by clicking Run as though the component had been run on the Matillion UI. The output of the code is shown beneath after running.

## Example: database query which retrieves a single (aggregate) row of data, and stores the result into a Variable

cursor = context.cursor()

cursor.execute("select count(\*) from flights")

result = cursor.fetchone()

print result

context.updateVariable("total\_count", str(result[0]))

# AWS Lambda for Python

|  |  |
| --- | --- |
| You can run Python code in AWS Lambda. Lambda provides runtimes for Python that run your code to process events.  Your code runs in an environment that includes the SDK for Python (Boto3), with credentials from an AWS Identity and Access Management (IAM) role that you manage. |  |
|  |  |

<https://aws-lambda-for-python-developers.readthedocs.io/en/latest/home/>

Create the Lambda function

Use a function blueprint to create the Lambda function. A blueprint provides a sample function that demonstrates how to use Lambda with other AWS services. Also, a blueprint includes sample code and function configuration presets for a certain runtime. For this tutorial, you can choose the blueprint for the Node.js or Python runtime.

To create a Lambda function from a blueprint in the console

1. Open the Functions page of the Lambda console.
2. Choose Create function.
3. On the Create function page, choose Use a blueprint.
4. Under Blueprints, enter s3 in the search box.
5. In the search results, do one of the following:
6. For a Node.js function, choose s3-get-object.
7. For a Python function, choose s3-get-object-python.
8. Choose Configure.
9. Under Basic information, do the following:
10. For Function name, enter my-s3-function.
11. For Execution role, choose Create a new role from AWS policy templates.
12. For Role name, enter my-s3-function-role.
13. Under S3 trigger, choose the S3 bucket that you created previously.
14. When you configure an S3 trigger using the Lambda console, the console modifies your function's resource-based policy to allow Amazon S3 to invoke the function.
15. Choose Create function.

# Building Lambda functions with Python

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html>

## Example lambda-function.py

import json

import urllib.parse

import boto3

print('Loading function')

s3 = boto3.client('s3')

def lambda\_handler(event, context):

#print("Received event: " + json.dumps(event, indent=2))

# Get the object from the event and show its content type

bucket = event['Records'][0]['s3']['bucket']['name']

key = urllib.parse.unquote\_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

try:

response = s3.get\_object(Bucket=bucket, Key=key)

print("CONTENT TYPE: " + response['ContentType'])

return response['ContentType']

except Exception as e:

print(e)

print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.'.format(key, bucket))

raise e

# [AWS S3 and Python Bota3](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/index.html)

## AWS S3 review

|  |  |
| --- | --- |
| **Buckets**   * Buckets are containers for data stored in S3 * To upload your data (photos, videos, documents, etc.) to Amazon S3, you must first create an S3 bucket in one of the AWS Regions. You can then upload any number of objects to the bucket. * Amazon S3 has a flat structure instead of a hierarchy like you would see in a file system. However, for the sake of organizational simplicity, the Amazon S3 console supports the folder concept as a means of grouping objects. * It does this by using a shared name prefix for objects (that is, objects have names that begin with a common string). Object names are also referred to as key names. | * Amazon S3 supports various options for you to configure your bucket * configure your bucket for website hosting, * add a configuration to manage the lifecycle of objects in the bucket * configure the bucket to log all access to the bucket. * Amazon S3 supports subresources for you to store and manage the bucket configuration information. * You can use the Amazon S3 API to create and manage these subresources. * However, you can also use the console or the AWS SDKs. |
| **Prefixes**   * prefixes to organize the data that you store in Amazon S3 buckets hieratically. * A prefix value is similar to a directory name that enables you to group similar objects together in a bucket. * When you programmatically upload objects, you can use prefixes to organize your data. * Because these names don't usually contain punctuation, you might use slash (/) as the delimiter. | Keys   * In Amazon S3, keys can be listed by prefix. You can choose a common prefix for the names of related keys and mark these keys with a special character that delimits hierarchy. You can then use the list operation to select and browse keys hierarchically. This is similar to how files are stored in directories within a file system. * Keys are selected for listing by bucket and prefix. For example, consider a bucket named "dictionary" that contains a key for every English word. You might make a call to list all the keys in that bucket that start with the letter "q". List results are always returned in UTF-8 binary order. |

# [Uploading objects using presigned URLs](https://docs.aws.amazon.com/AmazonS3/latest/userguide/PresignedUrlUploadObject.html)

# [S3 Bota3 Python Functions](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html)

* + [Client](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#client)
  + [Paginators](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#paginators)
  + [Waiters](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#waiters)
  + [Service Resource](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#service-resource)
  + [Bucket](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucket)
  + [BucketAcl](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketacl)
  + [BucketCors](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketcors)
  + [BucketLifecycle](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketlifecycle)
  + [BucketLifecycleConfiguration](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketlifecycleconfiguration)
  + [BucketLogging](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketlogging)
  + [BucketNotification](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketnotification)
  + [BucketPolicy](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketpolicy)
  + [BucketRequestPayment](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketrequestpayment)
  + [BucketTagging](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#buckettagging)
  + [BucketVersioning](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketversioning)
  + [BucketWebsite](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#bucketwebsite)
  + [MultipartUpload](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#multipartupload)
  + [MultipartUploadPart](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#multipartuploadpart)
  + [Object](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#object)
  + [ObjectAcl](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#objectacl)
  + [ObjectSummary](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#objectsummary)
  + [ObjectVersion](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#objectversion)
  + [Examples](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3.html#examples)
* [S3Control](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3control.html)
  + [Client](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3control.html#client)
  + [Paginators](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3control.html#paginators)
* [S3Outposts](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3outposts.html)
  + [Client](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3outposts.html#client)
  + [Paginators](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3outposts.html#paginators)

# Matillion and Lambda

# Read Settings

## MAIN JOB

### Python Script for creating and initializing Python Variables

 Matillion ETL runs as a Tomcat user and care must be taken to ensure this user has sufficient access to resources.

The script is executed in-process by an interpreter of the user's choice (Jython, Python2 or Python3). Any output written via print statements will appear as the task completion message, and so output should be brief.

### Also see Matillion Cookbook

**#create python array file\_settings and add grid variable [0] to array**

########################################################

# Get file\_settings grid variables and put into a python array

########################################################

print('file\_settings array from grid:')

print()

file\_settings = context.getGridVariable('file\_settings')[0]

#print(file\_settings)

print()

####################################################

# create and initialize file\_settings variables for python local session

####################################################

print(‘create and initialize variables for python local session:')

print()

context.updateVariable('transpose', file\_settings[0])

context.updateVariable('file\_type', file\_settings[1])

context.updateVariable('sheet', file\_settings[2])

context.updateVariable('skip\_rows', file\_settings[3])

context.updateVariable('drop\_rows', file\_settings[4])

context.updateVariable('skip\_columns', file\_settings[5])

context.updateVariable('drop\_columns', file\_settings[6])

####################################################

# Get s3\_settings grid variables and put into a python array

####################################################

print('s3\_settings array from grid:')

s3\_settings = context.getGridVariable('s3\_settings')

print(s3\_settings)

print()

print(‘create and initialize variables for python local session:')

print()

####################################################

# create and initialize s3\_settings variables for python local session

####################################################

context.updateVariable('bucket', s3\_settings[0])

context.updateVariable('prefix', s3\_settings[1])

########################################################

# Get snowflake\_settings grid variables and put into a python array

########################################################

snowflake\_settings = context.getGridVariable('snowflake\_settings')[0]

print(snowflake\_settings)

print()

print(‘create and initialize variables for python local session:')

print()

####################################################

# create and initialize snowflake\_settings variables for python local session

####################################################

context.updateVariable('database', snowflake\_settings[0])

context.updateVariable('schema', snowflake\_settings[1])

context.updateVariable('table', snowflake\_settings[2])

########################################################

# Get snowflake\_settings grid variables and put into a python array

########################################################

print("Settings Imported:")

print("Bucket: [{}] Prefix: [{}] Sheet Name: [{}]".format(bucket, prefix,sheet))

print("Rows: SKIP [{}] DROP [{}]\nColumns: SKIP [{}] DROP [{}]".format(skip\_rows, drop\_rows,skip\_columns,drop\_columns))

print("Database: [{}] Schema: [{}] Table: [{}]".format(database, schema,table))

print("Transpose: [{}]".format(transpose))

# Archive CSV for Iterators

iterator\_base\_folder - Base Folder

iterator\_sub\_folder - Subfolder

iterator\_filename - Filename

import boto3

client = boto3.client('s3')

source\_key = iterator\_base\_folder

if iterator\_sub\_folder is not None:

source\_key = source\_key + iterator\_sub\_folder

source\_key = source\_key + iterator\_filename

print("Source Key: {}".format(source\_key))

processed\_key = "{}csv/{}".format(processed\_dir,iterator\_filename)

print("Processed Key: {}".format(processed\_key))

copy\_source = {

'Bucket': bucket,

'Key': source\_key

}

client.copy(copy\_source, Bucket= bucket, Key=processed\_key)

client.delete\_object(Bucket = bucket, Key = source\_key)

# Example: Achieve CSV for Dimensions

Key:

CCAAS/DIMENSIONS/tenant\_1035-agent\_details-D/tenant\_1035-agent\_details-D.csv

S3 URI

s3://star2star-ingestion-mpg/CCAAS/DIMENSIONS/tenant\_1035-agent\_details-D/tenant\_1035-agent\_details-D.csv

Object URL

<https://star2star-ingestion-mpg.s3.us-west-2.amazonaws.com/CCAAS/DIMENSIONS/tenant_1035-agent_details-D/tenant_1035-agent_details-D.csv>

import boto3

client = boto3.client('s3')

source\_key = bucket

if prefix is not None:

source\_key = source\_key + prefix

source\_key = source\_key + iterator\_filename

print("Source Key: {}".format(source\_key))

processed\_key = "{}csv/{}".format(processed\_dir,iterator\_filename)

print("Processed Key: {}".format(processed\_key))

copy\_source = {

'Bucket': bucket,

'Key': source\_key

}

# client.copy (copy\_source, Bucket= bucket, Key=processed\_key)

# client.delete\_object(Bucket = bucket, Key = source\_key)

# Matillion Grid Commands

<https://metlcommunity.matillion.com/s/question/0D54G00007iBj40SAC/grid-variables-python>

<https://documentation.matillion.com/docs/2917841>

array = context.getGridVariable('people')

for data in array:

 print(data)

context.updateGridVariable('names', array)

1. At design-time, you first create the Grid Variable using the UI, the values you add are just default values. Those default values exist when you first start the job.
2. At run-time, if you do something like updateGridVariable, whatever you updated the Grid Variable with is what it will have until job finishes. As soon as the job finishes, the Grid Variable is exactly what it was before the job started. Meaning it reverts back to it's defaults. This is the difference between run time and design time.

The idea is that for each job run, the values in the Grid Variable could dynamically change based on that particular job run. As an example, you could make an API call and save the results to a Grid Variable that had 1 or more columns. From there you could process each result individually. The kicker here is that with each job run the API call may return a difference result set each time.

In your example, if you created a Grid Variable called people and the default values were ['matt', 'frank', 'glen'], you then run the job the job or just the component that executes context.updateGridVariable('people', [['Kyle']]), your new value will be just Kyle until the job ends and then it will revert back to the default ['matt', 'frank', 'glen'].

Keep in mind when you run context.updateGridVariable('people', [['Kyle']]) you are replacing the default values with just Kyle not adding to what is already existing. Therefore your Grid Variable for that run will only contain Kyle. If you want to add to it then you will need to get the Grid Variable values and then add on to those values and then update the grid variable with the whole collection which would include the Kyle you added. Hopefully this helps clarify this a bit more.

In a Matillion Python Script component, grid variables are represented as a list of lists. So it sounds like you might be trying to update a grid variable using a scalar value (string or int) instead of a list?

The handling of grid variables in matillion python components have a value per column and a list per row and all this is in one list.

# [**Example: Grid variable updating in python script**](https://metlcommunity.matillion.com/s/question/0D54G00007HZjnDSAT/grid-variable-updating-in-python-script)

Something like this:

X = ['qwer','tyui','uiop','sdfgh']

gv = []

for data in X:

[#reset](javascript:void(0);) the 'list'

 g = []

 print data

[#add](javascript:void(0);) the correct amount of grid variable 'columns'

 g.append(data)

 print g

[#adding](javascript:void(0);) to final grid variable list

 gv.append(g)

 print gv

[#loop](javascript:void(0);) is done, updating grid variable

context.updateGridVariable('gc',gv)

[#checking](javascript:void(0);) if grid variable was updated as expected.

print context.getGridVariable('gc')