**Event-Driven Data Ingestion with AWS Lambda (S3 to RDS)**

In the [previous post](https://www.mydatahack.com/event-driven-data-ingestion-in-aws-s3-to-s3/), we discussed how to move data from the source S3 bucket to the target whenever a new file is created in the source bucket by using [AWS Lambda function](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html). In this post, I will show you how to use Lambda to execute data ingestion from S3 to RDS whenever a new file is created in the source bucket. AWS Lambda supports a few different programming languages. We will use Python 3.6 here.

To access RDS with the lambda function, your lambda function need to access the VPC where RDS reside by giving the right permission to the function. You also need to tell the function which VPC to access and which security group within the VPC to use. It is actually simpler than it sounds and I will cover all of these here. AWS has a documentation on how to configure lambda function to access RDS [here](https://docs.aws.amazon.com/lambda/latest/dg/vpc-rds.html), which I referred to write this post.

**Scenario**

Let’s think about a near real-time ingestion scenario. You have an application that does data dump in S3 every 15 minutes in JSON. Your requirement is to grab the data from S3, transform it and write it to Postgres RDS every time a new file comes to the bucket.

**Ingestion Details**

As an example, let’s use the JSON example data used here ([How Postgres JSON Query Handles Missing Key](https://www.mydatahack.com/how-postgres-json-query-handles-missing-key/)). This data was also used in the previous Lambda post ([Event-Driven Data Ingestion with AWS Lambda (S3 to S3)](https://www.mydatahack.com/event-driven-data-ingestion-in-aws-s3-to-s3/)). Essentially, we will change the target from S3 to Postgres RDS. As an ingestion method, we will load the data as JSON into Postgres. We discussed this ingestion method here ([New JSON Data Ingestion Strategy by Using the Power of Postgres](https://www.mydatahack.com/new-json-data-ingestion-strategy-by-using-the-power-of-postgres/)).

**Prerequisites**

* Mac or Linux machine with Python 3.6 development environment.
* Virtual environment for this lambda function with psycopg2-binary installed
* Postgres RDS with target schema and table (e.g. usermanaged.transaction)
* Source S3 Bucket (e.g. lambda.test.source)

If you have a Windows machine, I struggled to get psycopg2 working in lambda when I push the function from there. I recommend to use a Linux virtual box to develop lambda function if you are using Windows. Lambda function runs on Linux EC2 and developing code in the similar OS environment is better for a script language like Python.

To set up Python development environment in Linux, we have instructions.

* [How to Install Python 3 and Create Virtual Environment in Centos, Redhat and Amazon Linux](https://www.mydatahack.com/how-to-install-python-3-and-create-virtual-environment-in-centos-redhat-and-amazon-linux/)
* [How to Install Python 3 and Create Virtual Environment In Ubuntu](https://www.mydatahack.com/how-to-install-python-3-and-create-virtual-environment-in-ubuntu/)

To launch Postgres RDS, see the post below. You need to create a table with one column with the jsonb data type.

### [How To Launch Postgres RDS With AWS Command Line Interface (CLI)](https://www.mydatahack.com/how-to-launch-postgres-rds-with-aws-command-line-interface-cli/)

You need to create a table with one column with the jsonb data type. See the post below for the data set and table creation.

* [How Postgres JSON Query Handles Missing Key](https://www.mydatahack.com/how-postgres-json-query-handles-missing-key/)

**Code**

We will use the config file for database connection details. For the target schema and table name, we will set environment variables when we deploy the function.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

The main function is handler(). The file is saved as MoveS3ToPg.py, which will be the lambda function name.

The bucket name and key are retrieved from the event. S3 event is a JSON file that contains bucket name and object key.

The code retrieves the target file and transform it to a csv file. Then, it uploads to Postgres with copy command. For the detailed explanation on this ingestion pattern, refer to [New JSON Data Ingestion Strategy by Using the Power of Postgres](https://www.mydatahack.com/new-json-data-ingestion-strategy-by-using-the-power-of-postgres/).

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

**S3 Put Event JSON Example**

Whenever the file is created in the source bucket, it will send the JSON file to the lambda function. This is used to test the function manually. You need to change a few parameters (like bucket or object name) to suit your set up.

[cc lang="text" tab\_size="4" lines="-1"]

[/cc]

**Summary Steps**

1. Create VPC Endpoint for Amazon S3
2. Create a custom policy for the function (e.g. s3\_to\_pg\_lambda)
3. Attached the policy to the role used for the function (e.g. s3\_to\_pg\_lambda)
4. Create a function and config file.
5. Package the code with the required libraries and the config file
6. Deploy the function
7. Test the function by manually invoking it
8. Add permission for the function to access S3 bucket
9. Configure event in the source bcuket
10. Test the function by S3 PUT request.

**Steps**

1. **Create VPC Endpoint for Amazon S3**

To access S3 from Lambda function executed within a VPC, I needed to [configure VPC Endpoint for S3](https://aws.amazon.com/blogs/aws/new-vpc-endpoint-for-amazon-s3/). How to access S3 from a VPC depends on your setup. Your VPC might already allows access to S3 without creating an endpoint (e.g. using Internet Gateway or NAT). I am using the environment created here ([How to Create Your Own Personal Data Science Computing Environment In AWS](https://www.mydatahack.com/how-to-create-your-personal-data-science-computing-environment-in-aws/)). In this set up, the best approach is to create the endpoint.

If the lambda function times out before downloading the file from S3, you have the access configuration issue.

1. **Create a custom policy for the function**

Lambda function needs to get data from S3 and access to RDS within a VPC. For RDS access, you need EC2 actions to create ENIs (used to execute the function within the specified VPC) and CloudWatch Logs action to write logs. For this, you can use the pre-made AWSLambdaVPCAccessExecutionRole. The policy looks like below. I simply added a few S3 action permissions to AWSLambdaVPCAccessExecutionRole. Use this to create a custom policy.

[cc lang="text" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

1. **Create a new lambda execution role and attach the policy**

Once the policy is created, you need to create a new lambda execution role and attach the policy to it.

1. **Create a function and config file**

Refer to the config file and code above. The name of the main python file should be the name of the function.

1. **Package the code with the required libraries and the config file**

Go to the site-packages folder of the virtual environment where the lambda function was developed. Zip the entire content. Add both config and main function files.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

1. **Deploy the function**

Make sure to have the vpc-config parameter with all the subnets of the RDS subnet group. The security group cannot be the same as RDS. This is because AWS resources cannot connect within the same security group. Use the default security group for the VPC. It usually works.

If you have multiple environment variables as in this example, use double-quote to wrap the variables.

The handler should be set as {function-name}.{main-function-name-in-the-code}. The role is the custom role created in step 3.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

If you need to update the function, use the update-function-code command below.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

If you need to update the config (such as vpc-config or environment variables), use the update-function-configuration command.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

1. **Test the function by manually invoking it**

Use the lambda invoke command. The payload is the S3 event Json file you created manually from the example above.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

1. **Add permission for the function to access S3 bucket**

Run lambda add-permission command. Without the permission, you cannot configure lambda event in the source bucket.

1. **Configure event in the source bucket.**

Whenever the put request happens, you are sending the event to the lambda function.

1. **Test the function by S3 PUT request.**

You can use the aws cli to upload the file to the target bucket and check if the lambda function executes correctly.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic"]

[/cc]

**Error Handling**

When the permissions, execution role, VPC access to S3, security group or subnets are not configured correctly, your function will timeout. This means the function will max out the default execution time limit and stop execution.

If your function starts timing out after successful executions, you may need to re-deploy the function. Then, it will be fixed.