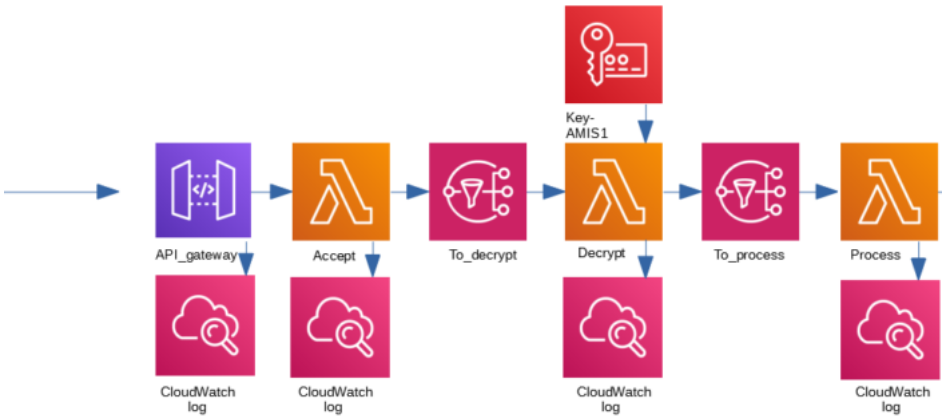


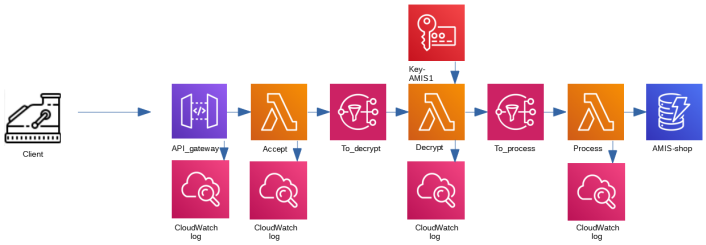
AWS shop example: Lambda

0

BY FREDERIQUE RETSEMA ON APRIL 29, 2020 AWS, CLOUD, SERVERLESS

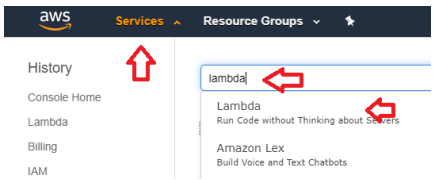
Introduction

In the previous blog [1], I wrote about an example shop application in AWS. Let me show the AWS architecture of this shop again:



In this blog, I will tell a little bit more about the Lambda functions in this shop example. Lambda functions are serverless functions: you don't need to configure a virtual machine in the cloud to use them. You are also not able to logon to the machine where your function runs. One of the advantages is that you will never have to patch these servers with updates: Amazon will do that for you.

You can write Lambda functions in several languages. Let's look at the list that is available on the moment you read this blog: logon to your AWS account, in the top menu select Servers, type and choose Lambda:



After that, click on the "Create function" button:

ABOUT AUTHOR

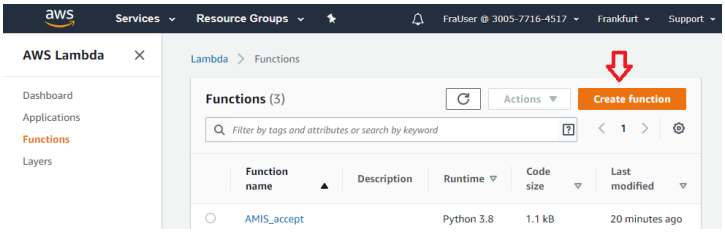
Frederique Retsema

Frederique Retsema is active in IT si 1993. Senior Consultant and develop on diverse areas including SQL and Java. She likes work with automation tools like Bamboo, Jenkins, Ansible, Terraform and CloudFormation.

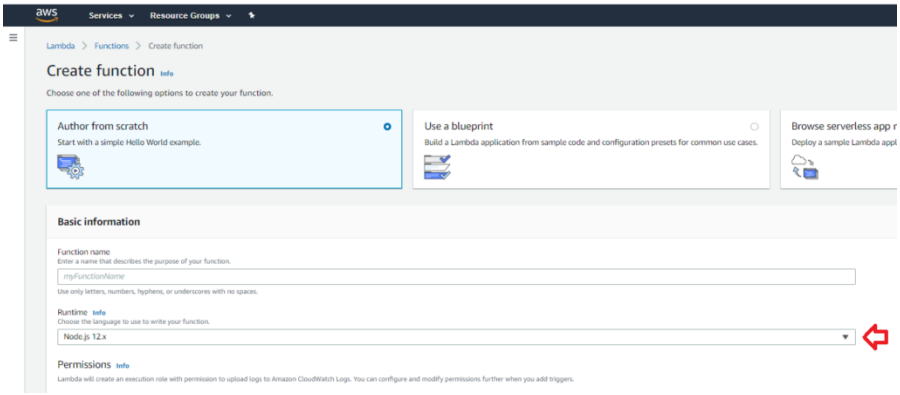
View all posts

POPULAR TAGS

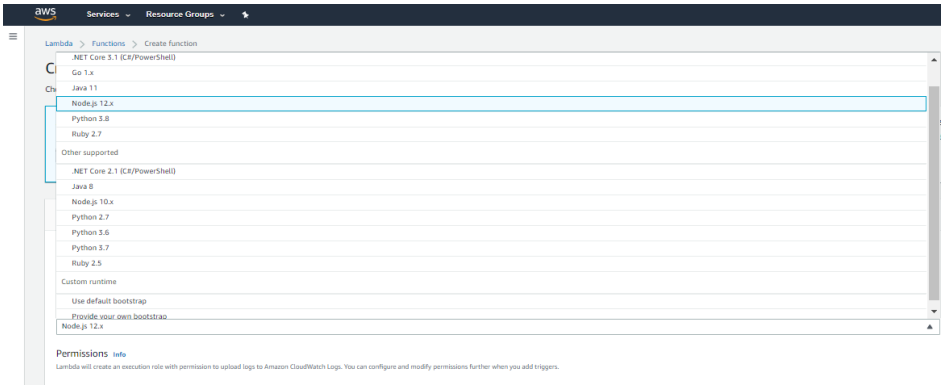
Agile analytical function apex api application container clou service Architecture Azure BPEL bpm cloud container Database docker DVT h integration iot Java javascript jms json kafka kubernetes linux maven monitoring node oci oracle cloud oracle cloud infrastructu oracle database oracle xml db OSB paas performance tuning plsql provisioning puppet push python RES saas Scrum soa sql Vagrant virtual box visualization vm XML



Click now on the arrow down image under Runtime:



You will see the list of all runtimes that are available now:



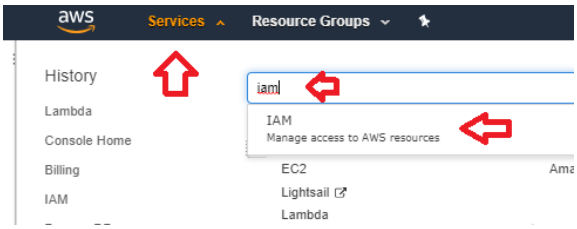
The code I present here, is written in Python version 3.8. Some libraries (for example: default system libraries or the libraries to access AWS services) are already present, other libraries have to be send with your function code to let your function work. In this example, we only use default system libraries and the boto3 library to access AWS functions, so the functions in this example are pretty simple.

Connection with AWS Identity and Access Management (IAM)

In general, your Lambda functions will also need permissions to use other AWS services. In our shop example, all lambda functions use AWS CloudWatch for logging. The accept and decrypt functions will send data to SNS, the process function will send data to DynamoDB. The decrypt function also uses AWS Key Management Services (KMS) to do the decryption of the data.

To give the Lambda function access to these services, the access function can *assume* a role. This role is written in AWS Identity and Acccess management.

We'll look into that now: go to AWS IAM:



In the left menu, click on roles:

DIY Kafka Topic Watcher tool - Node, Express, ..
This article can be read in at least two different ways: as a somewhat lengthy introduction of a
technology.amis.nl

AMIS Conclusion
@AMISnl
New Article : A Free Apache Kafka Cloud Service – an how to quickly get started with it ift.tt/2y4ZmMF by Luc Jellema

A Free Apache Kafka Cloud Service - and how ..
Last week I presented on Apache Kafka – twice. Once to a group of over 100 students, once to 30+
technology.amis.nl

Apr 27, 2(

AMIS Conclusion
@AMISnl
The latest The AMIS Oracle Technology Weekly! paper.li/AMIS_Services/… #orclapex

Example application in AWS using Lambda - A..
technology.amis.nl I have to admit: I love serverless Serverless computing is using the cloud as it is
paper.li

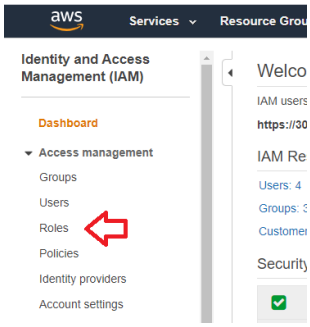
Apr 27, 2(

AMIS Conclusion
@AMISnl
New Article : Example application in AWS using Lamb ift.tt/3cSy1w9 by Frederique Retsema

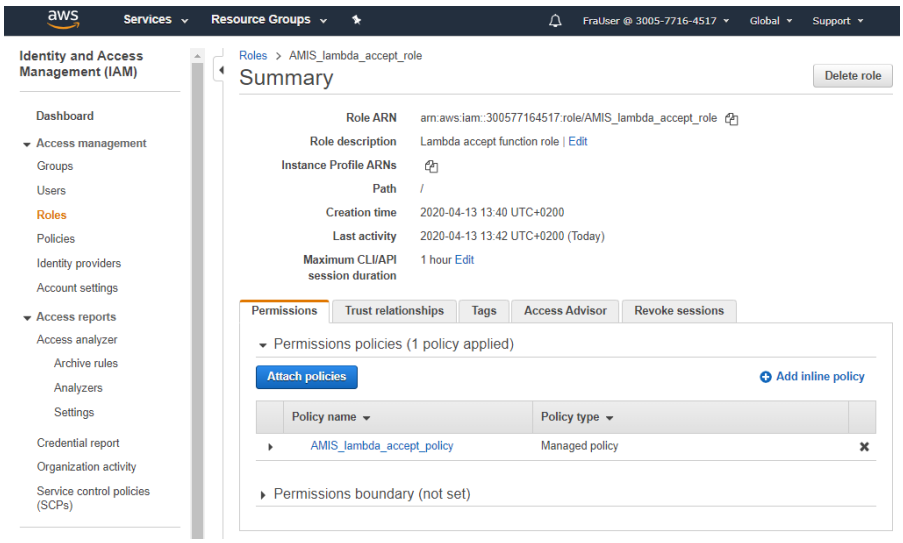
Example application in AWS using Lambda - A..
Introduction I have to admit: I love serverless. Serverless computing is using the cloud as it is
technology.amis.nl

Apr 26, 2(

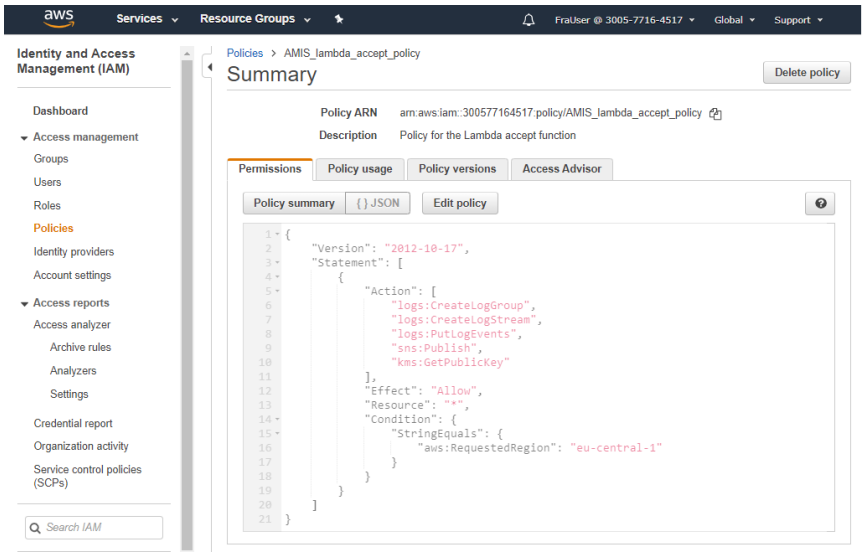
AMIS Conclusion
@AMISnl
New Article : Windows Sandbox – light weight playground for R&D, tutorials and workshops ift.tt/2znr by Lucas Jellema



Let's look at one role: click on the link AMIS_lambda_accept_role:



You see that there is one policy attached: the AMIS_blog_lambda_access_policy. When you click on this policy, you see that the accept access policy allows for the creation of AWS CloudWatch groups, the creation of AWS CloudWatch log streams and the permission to add (put) AWS CloudWatch log events. The policy also allows for publishing SNS events and to get public KMS keys.



When you look at the policies for the other Lambda functions, you will see that all policies are slightly different. It is good practice to create a role and a policy for every function: you then use the least privilege security principle.

Inside the accept function

Let's go back to Lambda and look at the definition of the accept function. Go to the Lambda service (see the first screen image in this blog if you need to), and click on the link of the AMIS_accept Lambda function (not on the radio button in front of it):

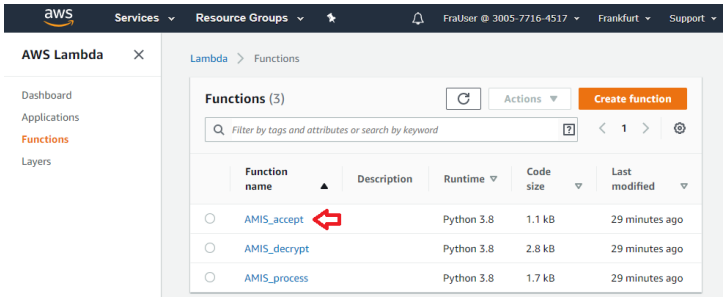
Windows Sandbox - light weight playground f…
Windows Sandbox to me is a light weight Windows 10 virtual machine that I can quickly start and stop
technology.amis.nl

Apr 25, 2(

Embed View on Twit

META

Log in
Entries feed
Comments feed
WordPress.org

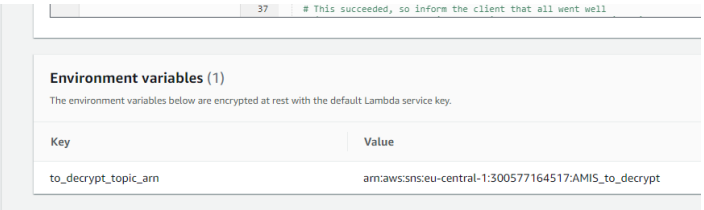


You will now see the code of AMIS_accept:

```
import json
import boto3
import os

# Main function
# -------------
def lambda_handler(event, context):

  from botocore.exceptions import ClientError

  try:

    # Log content of the data that we received from the API Gateway
    # The output is send to CloudWatch
    #

    print("BEGIN: event:"+json.dumps(event))

    # Initialize the SNS module and get the topic arn.
    # These are placed in the environment variables of the accept function by the Terraform script
    #

    sns = boto3.client('sns')
    sns_decrypt_topic_arn = os.environ['to_decrypt_topic_arn']

    # Publish all the incomming data to the SNS topic
    #
    message = json.dumps(event)

    print ("Message to to_decrypt: " + message)

    sns.publish(
      TopicArn = sns_decrypt_topic_arn,
      Message = message
    )

    # This succeeded, so inform the client that all went well
    # (when there are errors in decrypting the message or dealing with the data, the client will NOT
    #

    statusCode = 200
    returnMessage = "OK"

  except ClientError as e:

    # Exception handling: send the error to CloudWatch
    #

    print("ERROR: "+str(e))

    # Inform the client that there is an internal server error.
    # Mind, that the client will also get a 500 eror when there is something wrong in the API gatewa
    # In that case, the text is "Internal server error"
    #
    # To be able to make the difference, send a specific application text back to the client
    #

    statusCode = 500
    returnMessage = "NotOK: retry later, admins: see cloudwatch logs for error"

  # To make it possible to debug faster, put anything in one line. Also show some meta data that is
  #

  print("DONE: statusCode: " + str(statusCode) + \
            ", returnMessage: \"" + returnMessage + "\"" + \
            ", event:"+json.dumps(event) + \
            ", context.get_remaining_time_in_millis(): " + str(context.get_remaining_time_in_millis(
            ", context.memory_limit_in_mb: " + str(context.memory_limit_in_mb) + \
            ", context.log_group_name: " + context.log_group_name + \
            ", context.log_stream_name: "+context.log_stream_name)

  return { "statusCode": statusCode,
           "headers" : { "Content-Type" : "application/json" },
           "body": json.dumps(returnMessage) }
```

When you use print statements, these will automatically be send to CloudWatch. If necessary, a new group and a new log stream will be created.

To be able to send a message to the SNS topic, the boto3 library is used. We have to know what the Amazon Resource Name (ARN) for the pipeline is. To keep the code clean, the ARN of this pipeline is not in the code itself, but in the environment variables of this function. You can see this when you scroll down: the environment variables are directly under the code for the function:

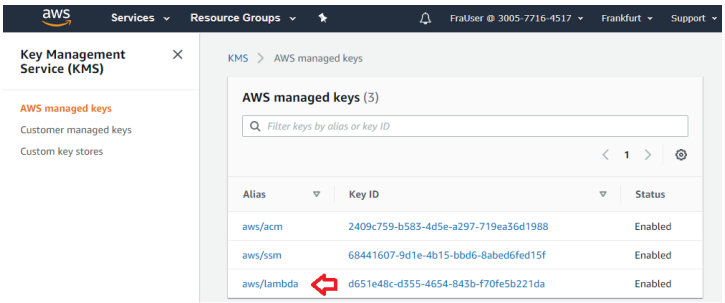


In my example, Terraform is used to deploy the AWS objects. Both the SNS topics and the lambda functions are deployed in the same script, the ARN from the SNS topic is added as an environment variable to the lambda function.

In the code, you see how the environment variable is retrieved from the environment, and then the whole event as it is sent to the accept function is used as a message to the decrypt topic.

```
# Initialize the SNS module and get the topic arn.
# These are placed in the environment variables of the accept function by the Terraform script
#

sns = boto3.client('sns')
sns_decrypt_topic_arn = os.environ['to_decrypt_topic_arn']

# Publish all the incomming data to the SNS topic
#
message = json.dumps(event)

print ("Message to to_decrypt: " + message)

sns.publish(
  TopicArn = sns_decrypt_topic_arn,
  Message = message
)
```

When I discussed the policy for the accept function, you might have asked yourself why we needed KMS keys in this function: we don't seem to use encryption in this function. Well, the environment variables are always encrypted, using a KMS keys. Lambda uses a default key for this, you can see this by going to the KMS service and click on AWS managed keys in the left window. You will see that one of the keys is aws/lambda:

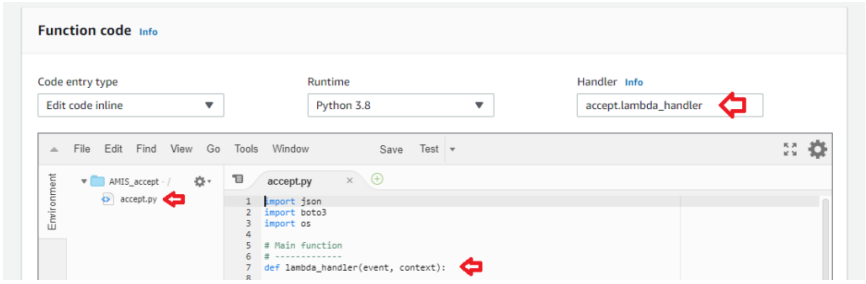


The decryption of the environment variable is done in the background, we don't have to add code for this ourselves.

Handler

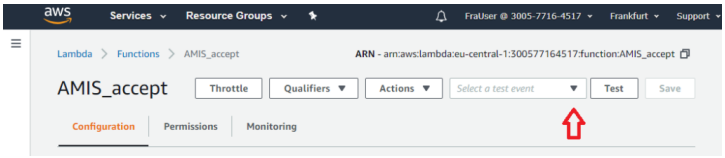
In this example, it is quite clear where the Lambda function starts: this code just has one function. It is however possible to have multiple functions in your code. The AWS environments needs to know what the name of the function is that will be the starting point for the execution. This is called the handler.

You can see the handler name just above the code: in our case it is called "accept.lambda_handler". In this name, accept refers to the name of the file with the code, in our case accept.py. The part behind the dot, refers to the name of the function: in our case lambda_handler.

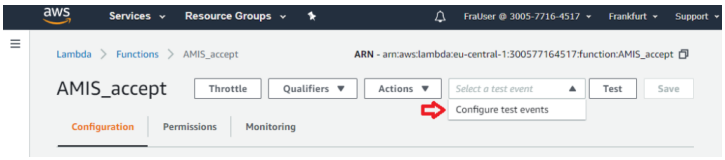


Testing Lambda functions

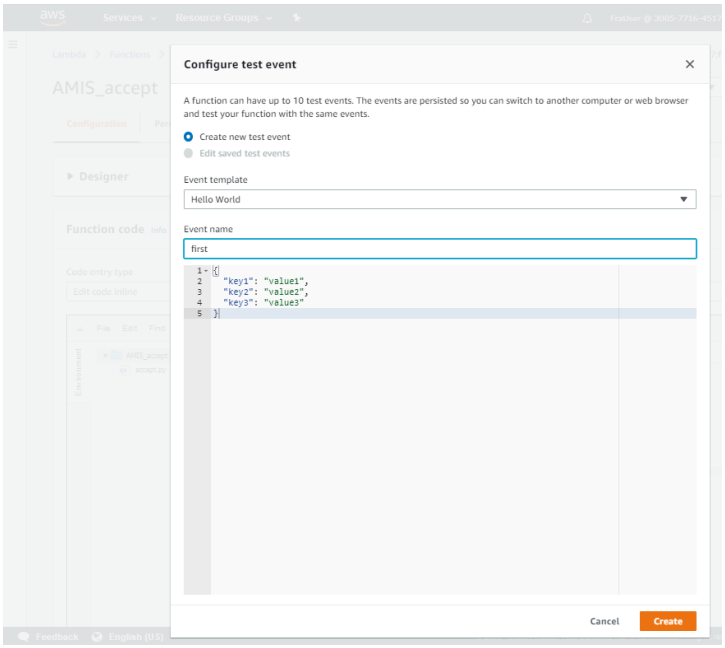
In the top of the screen, you see some options to test our Lambda function. Let's try them out: click on the button-down arrow next to "Select a test event":



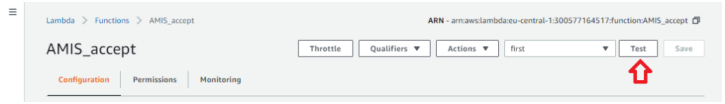
You can now select “Configure test events”:



You see pretty straightforward test json. You can change this in any way you want. When you are ready, change the event name (f.e. to “first”) use the Create button to create the test template:



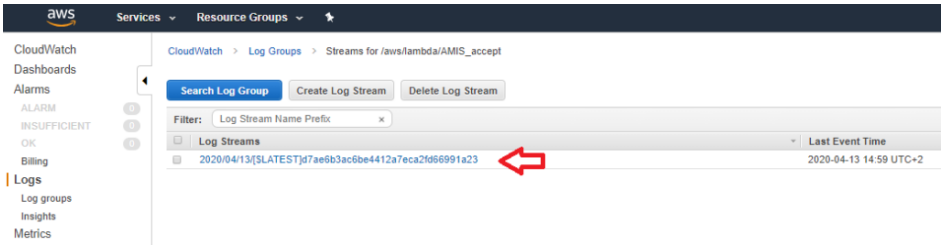
You can fire off the event, by clicking on the Test button:



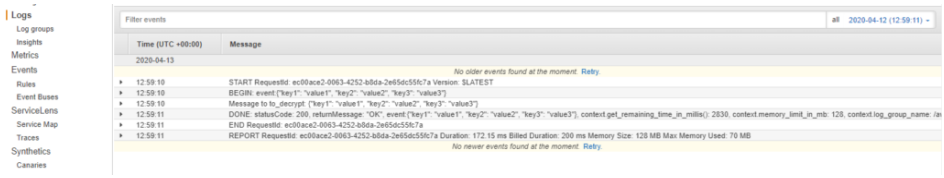
Cloudwatch

The test was successful. Let’s look what information is send to CloudWatch: click on the link “logs” (next to “Execution result: succeeded):

A new tab opens, with the CloudWatch logs. You can see that there is a log stream, with the date in it. It also has a recent Last Event Time. Click on the most recent Log Stream in this screen:

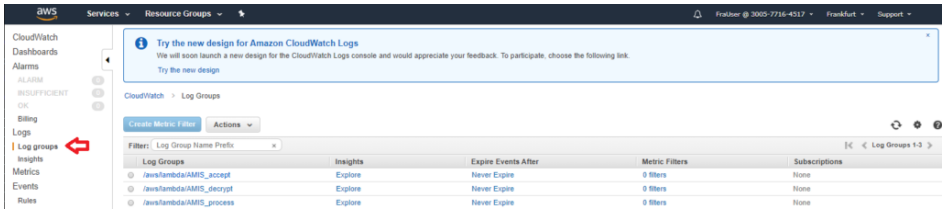


You can see that the command print("BEGIN: event:"+json.dumps(event)) sent out our test event. The last line is also interesting: it will always be added to every Lambda call, and it contains the Duration, the Billed Duration, the memory size and the max memory size that is used.

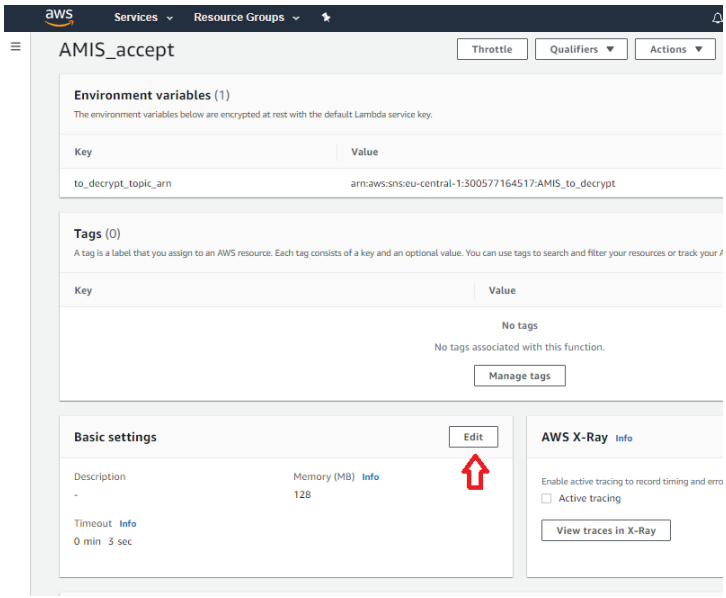


Lambda's are billed both based on the number of milliseconds that the function has run, and on the amount of memory that has been assigned. In our case, we assigned the minimum amount of memory possible.

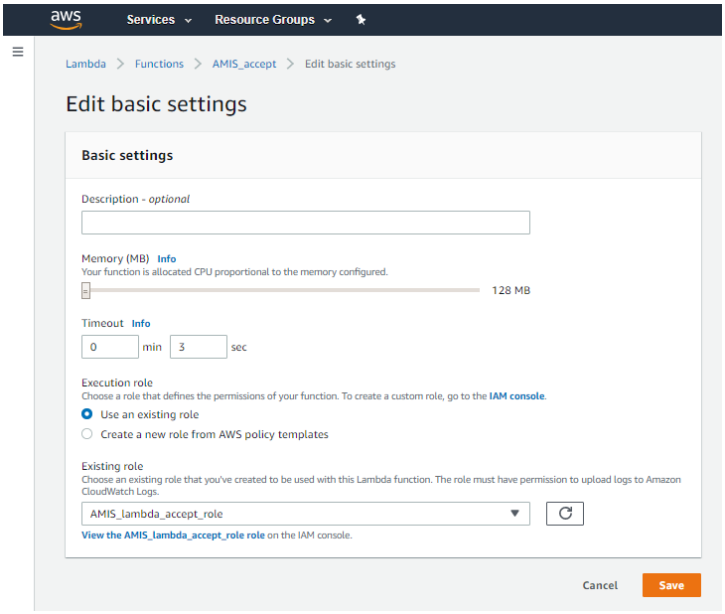
If you want to go to Cloudwatch logs without sending a test message, then go to the CloudWatch service and choose the Logs > Log groups item in the left menu.



Let's go back to the tab of the Lambda function and scroll down to where these settings are configured: these settings are below the code, below the environment variables we looked at before. Click on Edit:



When you look at the default settings, you can see that the amount of memory is by default 128 MB and the timeout is by default 3 seconds. You can also see the name of the role that we saw earlier. When you need more memory, or more time than you can change these settings. The maximum timeout value is 15 minutes. You will see that when you ask for more memory, then the amount of time that your Lambda function uses will be lower. AWS will use better performing servers for Lambda functions that ask for more memory.



When you play along, you will see a different duration for the same Lambda function. The first time will take much more time than the second or the third time. Sometimes, however, the function will create a new log group and then start again with a first invocation which again will take relatively long.

In my environment, the first invocation takes more than 1000 milliseconds (one second), where the second or third one takes 100 – 300 milliseconds. This difference is there, because the first time the Lambda function is called, it has to be retrieved and to be put in memory. When this is done and the Lambda is executed, following events can use the same Lambda function.When the function isn't used for some time, it will be swapped out of memory. We will see more about this in a later blog about the testing of the shop example.

Play along

I scripted the solution [2]. You can follow along and create this solution in your own environment, see the previous blog [1] and the README.md file in the vagrant directory.

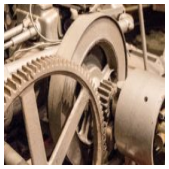
Links

[1] https://technology.amis.nl/2020/04/26/example-application-in-aws-using-lambda/

[2] Link to github account: https://github.com/FrederiqueRetsema/AWS-Blog-AWS . For the example in this blog, look in the shop-1 directory.

Related Posts:

Example application in AWS using



Differences between CloudFormation,

Creating policy's, groups and users in

Policies in AWS (2)

Oracle OpenWorld 2015 – a customer's

ABOUT AUTHOR

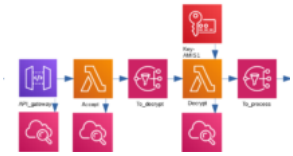
FREDERIQUE RETSEMA

Frederique Retsema is active in IT since 1993. Senior Consultant and developer on diverse areas including SQL and Java. She likes to work with automation tools like Bamboo, Jenkins, Ansible, Terraform and CloudFormation.

RELATED POSTS

APRIL 27, 2020 0

A Free Apache Kafka Cloud Service – and how to quickly get started with it



APRIL 26, 2020 0

Example application in AWS using Lambda

APRIL 9, 2020 2

Migrating an old (10.2.0.4) database to Oracle Cloud with minimal downtime

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. Learn how your comment data is processed.