**Purpose**: As part of the serverless architecture research here I will include interesting and relevant topics and features involving lamba, api gateway, sqs, dynamo, ec2 and some other service that can provide an interesting feature within this type of architecture.

**AWS LAMBDA (ESSENTIALSS) :**

The concept is simple, a function (container with a runtime) that can be invoked whenever is necessary (an external **event** ex: uploading a file to s3) in which we only get charged for the execution and the CPU consumption. It can scale to a lot of executions. It supports different runtimes like Node, python, c, c++ etc and it is very well integrated with other services like SQS or API Gateway.

***Note: Usually this kind of infrastructure is not handled through the console of AWS, it is considered a good practice to use CloudFormation, SAM or other frameworks that allows you to deploy infrastructure using templates. I’m going to do a corresponding demo on different features using CloudFormation and SAM.***

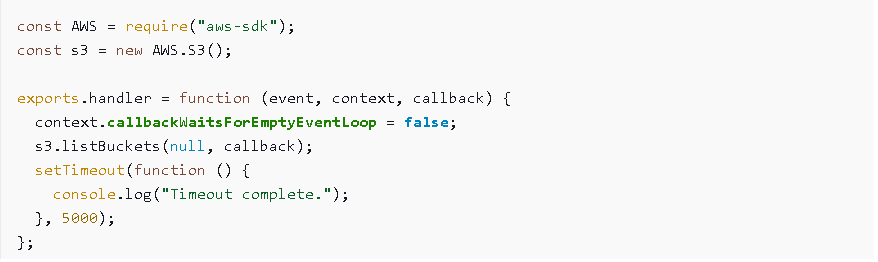
***Keep in mind that SAM is just an abstraction of CloudFormation that is more oriented towards serverless design patterns but anything that can be done within SAM can be done within CFN and vice versa (there are some features within SAM like local testing and easy CLI deployments that can be useful so I will use SAM but inside of SAM I will also be using CFN syntax which is very similar)***

The code within a lambda function can be written within the AWS console (there is a limit to the size of the function 10mb**\*?**) or It can be written locally and then uploaded to an S3 bucket. (Alternatively it can be written on our local pc and deployed using SAM which creates the bucket and uploads the code in an automated way)

***Note: Aside from CFN and SAM there are third party tool providers like the serverless framework, some of the advantages of that framework is that it is designed to work with a multi-cloud architecture, and it has a community that develops plugins for it with extra functionality, one drawback is that the new features that AWS implement could be late for that framework.***

**Concepts:**

A lambda function can be written using callbacks (useful if using older versions of node) or using async/await syntax. The following is what is called a handler and that’s like the main part of a function in which all the code will be executed but other functions and even imports can be don outside of the handler.

**Callback example:**================================================================================ **Async Example**

**General Features (Console):**

**Environment Variables: \***

Lambda Functions supports environment variables, and they can even be encrypted with a very customizable flow.

**Tags:**

These generally applies for most services and resources, and it is just a way of tagging the resources so they can be grouped and filtered when needed.

**Execution Role:**

A lambda function most likely will need to have a role with a policy attached so it has permissions to access other services. (It is a good practice to have a different role for each function/resource even if they seem to need the same one)

**Settings: \***

We can set a description, the amount of memory and the timeout for a function (the timeout has a max value of 15 minutes, previously it was 10 minutes). The resources allocated to a function depends on the memory size selected (it will be proportionally to that setting)

**VPC: \***

A lambda function can be attached to a VPC so it can only access resource within that VPC.(It needs at least two VPC to be specified)

**DLQ’s (Dead letter queues):**

When a lambda function fails or ends with and error even after multiple retries a SQS queue can be configured to receive the event payload (Useful to debug or implement retry logic)

**Concurrency: \***

All accounts have by default a concurrency limit of 1000 lambda functions running at the same time. (This can be expanded communicating with AWS support)

For any function you can reserve/assign a specific amount of concurrency within the AWS account limit.

**Auditing:**

Using CloudTrail we can log lambda invocations for control in case the organization needs it.

**Testing: \***

Test events (Static json objects) can be configured to test the lambda function. This tests can be saved and even shared with other functions. AWS provides also test events for different services like for example the event that s3 generates when a file gets uploaded.

**Version and Aliases:**

Each lambda function can have versioning control so if a lambda function stops working we can just go back to a version that was working, it can also have aliases which is a way to reference a function dynamically so it can be scalable and reusable withing other services like API gateway.

**Export:**

Lambda functions can be Exported as SAM files or even as a Zip file.

**Throttle: \*\***

In case of emergency there is a button which will set the lambda function concurrency to zero, it is recommended to use it only in case of emergency (need to abruptly stop invocating a function).

**Monitoring:**

These are just some graphs which shows things like the number of invocations over a period of time, the duration of the average maximum and minimum invocation, errors etc. We can also go to CloudWatch to the log Stream of the specific execution and see the logs of the lambda execution.

**IAM permission model:**

The lambda service uses a decouplable permission model which means that the event which triggers the function only needs the necessary permission to invoke the function and then the function needs a separate role/permissions to invoke any other service that it needs to use.

***Note: Different events can invoke the same lambda function, the policy or role to do this is called lambda invocation policy or function policy. The function policy is used by the service which invokes the function and the Execution role is for the lambda to access different AWS services.***

**Best Practices:**

Each function should have a different role only with the specific permissions needed to execute.

Each function should be limit to only one task.

**HANDLER**

The handler of a function consists of two arguments and one extra optional depending on if we are using callbacks of async/await syntax.

**Event**: This parameter contains all the information about the event that triggered the function.

The structure of this event depends on the event source. (s3, sns, etc)

**Context**: Has metadata related to the context of the execution of the function like the name the runtime. <https://docs.aws.amazon.com/lambda/latest/dg/nodejs-context.html>

The context parameters aside from the attributes it can contain (see the previous link) it also provides some methods specific to this object.

**Callback (Node 6.10)**: The function that gets executed, usually here goes the “return” statement.

***Note: The async/await handler syntax is only supported on Node 8.10 and above.***

**Invocation Type:**

The invocation type of a lambda function depends on the event source, for example s3 event will invoke a lambda in a Asynchronous way, while Cognito will invoke a lambda in a Synchronous way.

***In the topic of interest keep in mind that API gateway can invoke a lambda function Asynchronously and Synchronously.***

[***https://aws.amazon.com/es/premiumsupport/knowledge-center/api-gateway-invoke-lambda/#:~:text=You%20can%20configure%20the%20Lambda,then%20choose%20your%20REST%20API***](https://aws.amazon.com/es/premiumsupport/knowledge-center/api-gateway-invoke-lambda/#:~:text=You%20can%20configure%20the%20Lambda,then%20choose%20your%20REST%20API)***.***

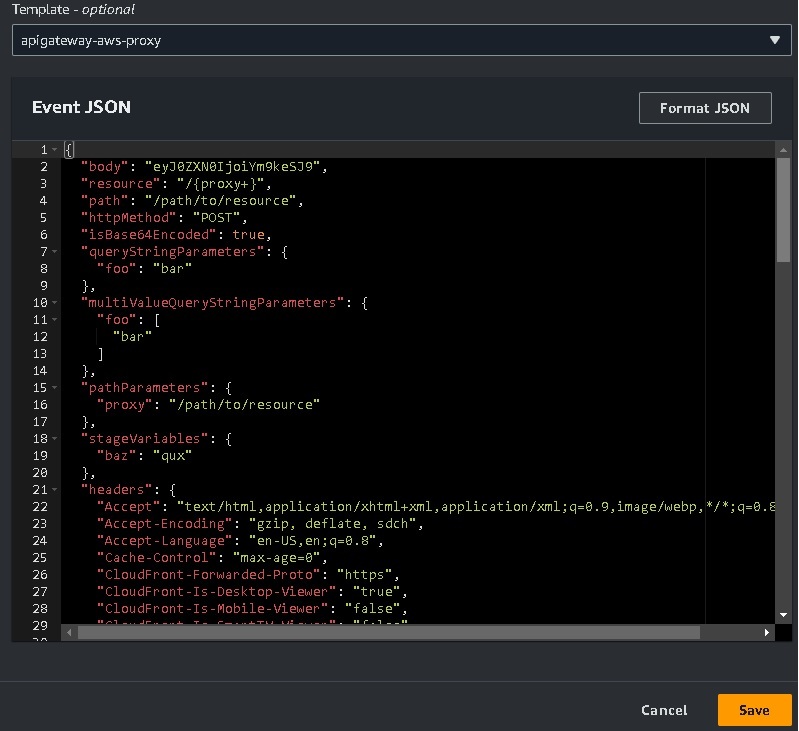
We can also decide how to invoke a lambda function if we invoke it from within our own application like using the invoke function from the JavaScript SDK.

Event source type:

**Push events:** The invoking service PUSH the event to the lambda function to invoke it. (It’s like the source passes the event payload and then the function lambda uses it) (Ex: API Gateway)

**Pull/Poll events:** Here the source only invokes the lambda but is the lambda function which pull the event from the source (Work like a queue of events from the source) (Ex: SQS, DynamoDB)

***Note: There are also custom events if we invoke a lambda from our own application (SDK)***



The context argument/object is optional but it provides useful runtime information about the context of the execution of a lambda function, is useful to get the id, the log Stream or the remaining time of execution for the lambda function.

***Atributte: context.functionName;--Method: context.getRemainingTimeInMillis();***

Login and error Handling:

**Specific runtime error log handling and debugging:**

As usual we can throw errors within our code, in the callback style the callback first argument would take the error, so for example, **callback(error)** and when using async/await just using the **throw** reserved keyword we could ‘return/throw’ errors or even better wrapping the code within a **try/catch** block.

***Note: All lambda logs are written to CloudWatch. (It could take some minutes to display in CW)***

Any print logs (console.log.error.info etc) will be send to the CloudWatch Log Stream of the lambda execution.

**AWS LAMBDA (DEEP DIVE) :**