AWS Lambda

Before we learn about the Lambda, we need to know the term serverless and its meaning

Serverless

Developers no need to manage/provision/see the servers anymore {Amazon going to do that for the developers}, Developers just deploys the code {background amazon takes care of the runtime engines ..etc}

Services which are serverless (main ones)

- AWS Lambda & S3
- DynamoDB and Aurora Serverless
- API Gateway and Cognito
- Amazon SNS & SQS

Now we will come to the Lambda Service

AWS Lambda: Serverless Compute Service

https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/begin

Play and see the – How the Typical Lambda works

Developers can write their functions and deploy them directly

- Virtual Functions – No Servers to manage or provision
- Run on-demand
- Limited by time – only till 15 Min executions, that it can run
- Automated Scaling

We can write our functions in

- Node.js
- Python
- Java (few versions)
- C#
- Golang
- PowerShell
- Ruby
- Custom Runtime API (we can bring other languages)

Now we know Lambda – How it is going to be run, that's where we need to learn about the term event, event is the triggering point which can revoke the lambda to run or start, now who will create events or what are the sources of the events which can in turn trigger the Lambda.

Source → Trigger → Event → Lambda

- Function: A function is a resource that you can invoke to run your code in Lambda.
- Trigger: A trigger is a resource or configuration that invokes a Lambda function.
- Event: An event is a JSON-formatted document that contains data for a Lambda function to process. The runtime converts the event to an object and passes it to your function code.

Lambda 01: Invocations

Invocation – the way we woke the lambda so that it starts running

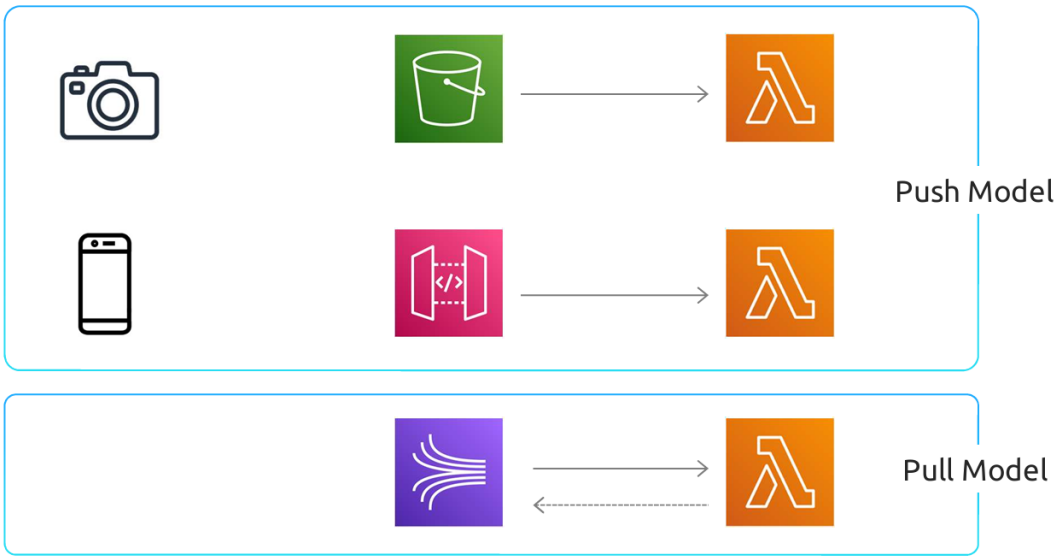
Simple we have 2 ways – PUSH and PULL Models

Push Model –

In this model another service directly triggers the Lambda and tells it to activate when something happens

Pull Model -

In this model – information is flowing through a stream or put in a queue that lambda periodically pole and it then goes into action when certain events are detected



In the Push Model Invocation, we have 2 ways again

Those are - Synchronous and Asynchronous

Let's assume you are the source of the event -

Synchronous –

when you trigger the Lambda, if the Lambda responds back to you with the results – keep you updated, moreover it will ask your permission to retry if it's failed.

Here, both the ends are in SYNC, what is going on.

Asynchronous –

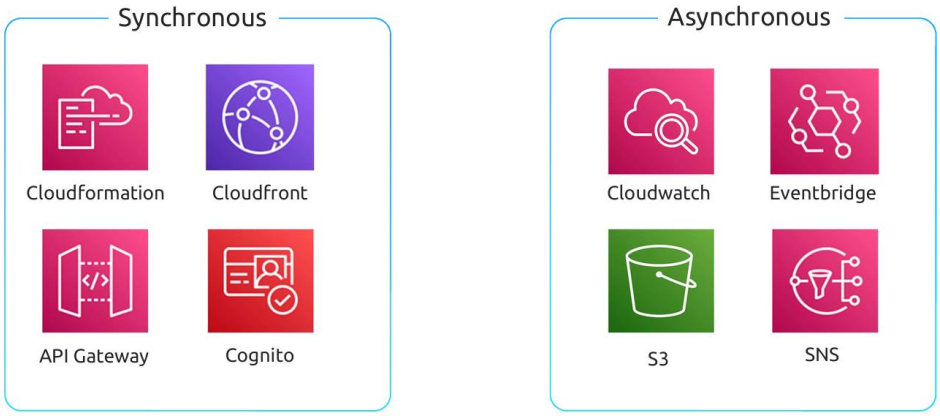
When you trigger the lambda, lambda will not respond back with the results to you – will not keep you updated, it will automatically retry 2 times, after 1 min and 2 min. {If you want to see what is going on, you have to and go look at the logs, then only we can know what is going on}

Here, both the ends are NOT in SYNC, what is going on, Developer should be smart enough with the coding to handle the retry mechanism (means you can have destinations specified like event bridge, SNS or dead letter queue) configured to send the messages to you when the lambda faces certain errors or based on the retry count.

Who decides the which invocation we are going to be on the path, based on the event source that you integrate the lambda with, that decides our invocation call, whether it's SYNC or ASYNC

For your information – few services will be categorised like this

AWS Lambda



Now we saw Push Model and it’s 2 sub types -- < Uhooooooo 👀

Now we will look at the Pull Model →

In the pull model we can divide the event sources into 2 types –

Those are Streams and Queues

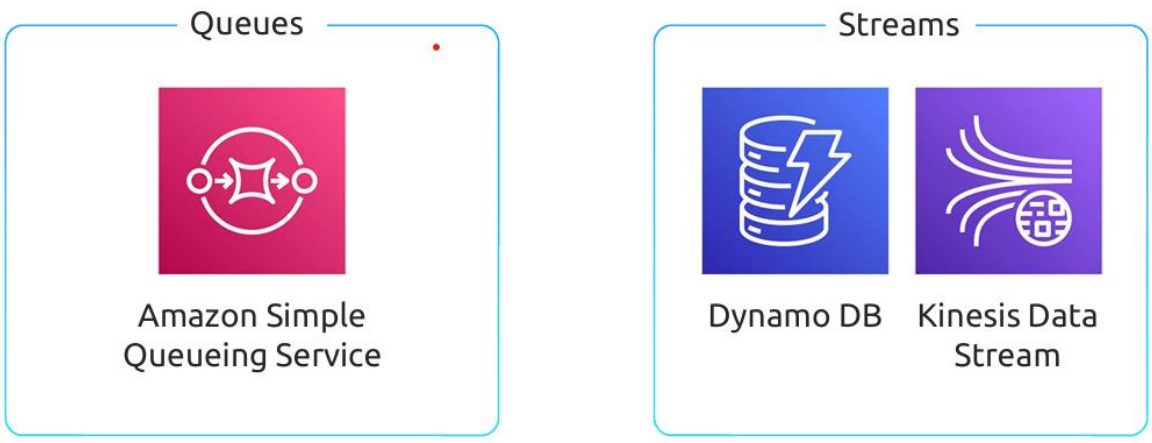
In both the types: Lambda continuously pulls and we can configure what lambda should watch for in the stream or queue, when Lambda sees a march – function will be invoked

Streams

Streams are the applications that have data that is continuously flowing at a steady rate from the one or many sources at the same time.

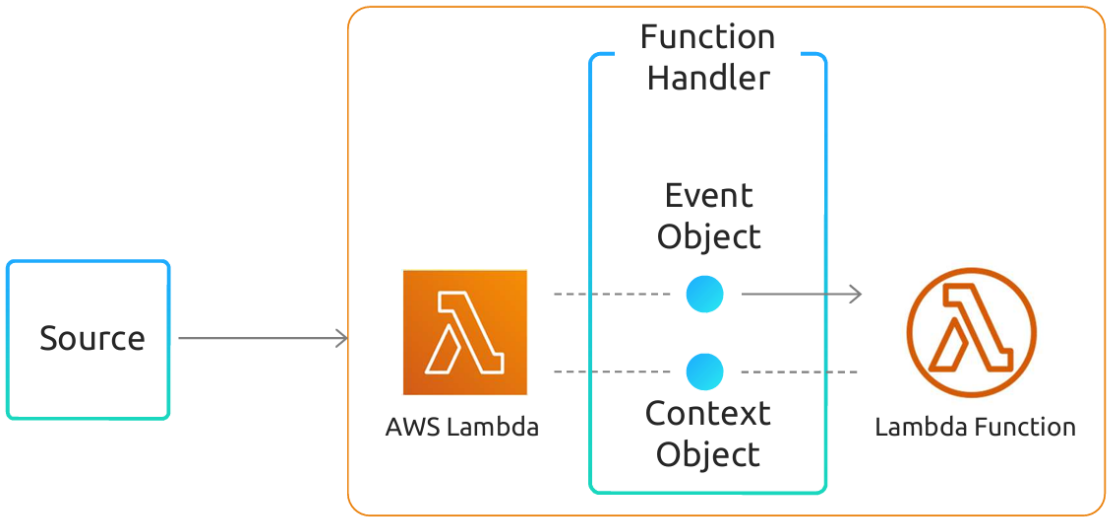
Queues

Queues are the applications that collects messages and place them in a bucket queue to be processed.



Lambda 02: Working with Lambda

Till now we saw the sources which can trigger Lambda, Now let's say we trigger the lambda, we need to look at how Lambda function operates.



Let's take example Lambda Function – which part does what we need to know then we will go deep and understand the functionality

Function Handler:

- The function handler is the entry point for your Lambda function. It's the method in your function code that processes events.
- When your Lambda function is invoked, Lambda runs the handler method.
- Your function continues executing until the handler returns a response, exits, or times out.
- The handler's name is derived from the name of the file where the handler function is located (default: lambda_function. lambda_handler).

Event Object:

- The event object is the first argument passed to the handler function.
- It contains data for your Lambda function to process.
- The event is usually a JSON-formatted document, but it can also be a list, string, integer, float, or None.
- You define the structure and contents of the event when invoking the function.
- For example, an API request event holds the HTTP request object.

Context Object:

- The context object is the second argument passed to the handler function.
- It provides information about the invocation, function configuration, and execution environment.

Useful properties include:

- context.invoked_function_arn: Lambda function ARN
- context.log_stream_name: CloudWatch log stream name
- context.log_group_name: CloudWatch log group name
- context.aws_request_id: Lambda Request ID

  You can access these properties within your function code.

```
def handler_name(event, context):
    # Your code here
    return some_value
```

Enough with theory ..! these are basic & foundational understandings of Lambda.

⚔ Lambda 02: Introvert & Extrovert Lambda

AWS Lambda EcoSystem we can divide into 2 parts

- Introvert Path
- Extrovert Path

Extrovert Path:

- Application Load Balancer → Lambda
- Event Bridge → Lambda
- S3 → Lambda
- SQS → Lambda

This Path will be handled via Practical hands on – before we go there, we need to understand the Introvert Path better

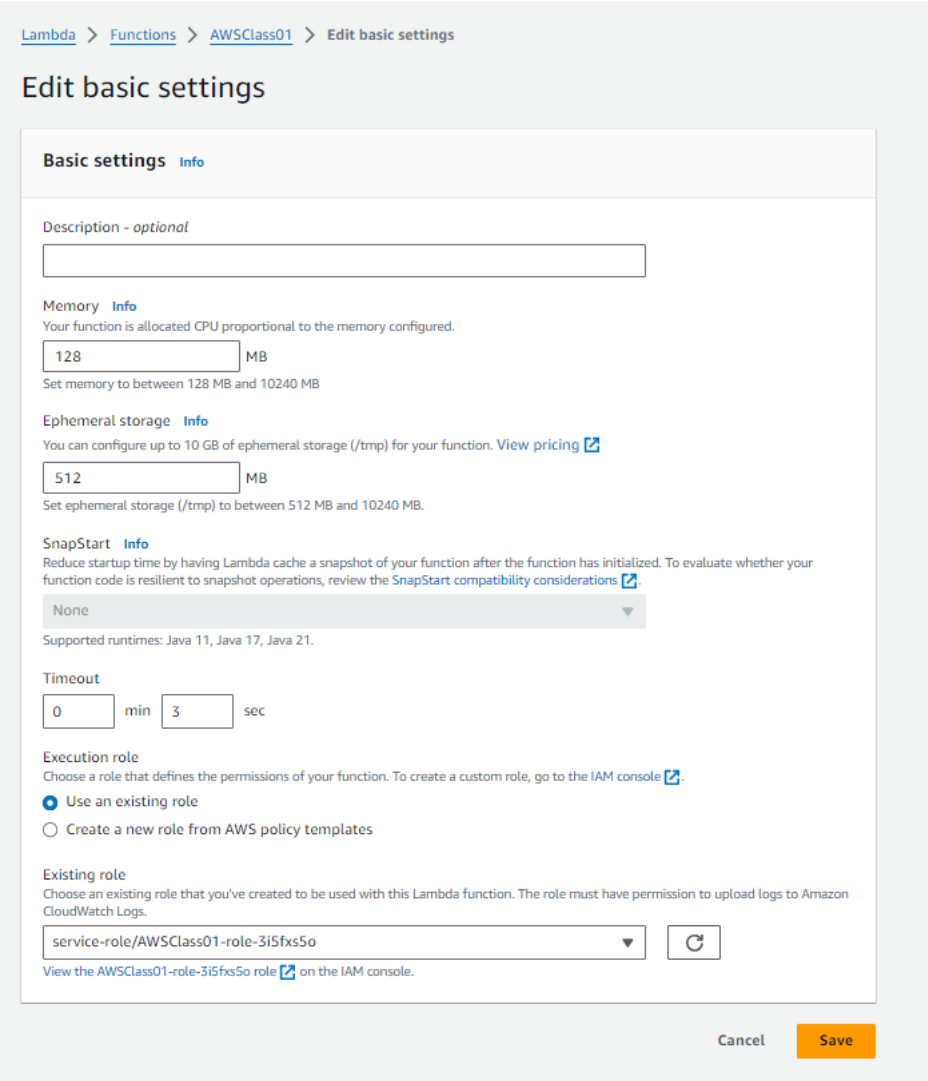
Introvert Path:

- General Configurations
- Performance
- Triggers & Permissions
- Destinations
- Function URL
- Environment Variables
- Tags & VPC
- RDS Databases – Monitoring & Operational Tools
- Concurrency and Throttle
- Asynchronous Invocation & Code Signing
- File Systems & State Machines
- Qualifier
- Layers
- Deployment Methods (Zip Files or Containers)
- Cloud Formation and Code Deploy

🚄 Introvert Path:

Along with the Programs, we need to remember few things in this path ..!

👍 *General Configurations / Basic Settings*



@ Description: Your Lambda Configs or Usage or Functionality as per the user

@ Memory: as we are Learning about memory – we will look at the performance as well

We need to look at the 2 main pointers for the performance – one is RAM and other one is execution context

The First one is ***RAM***

- From 128 MB to 10 GB in 1 MB Increments
- The more RAM you add, the more vCPU credits you get
- At 1,792 MB, a function has the equivalent of one full vCPU
- After 1,792 MB, you get more than one CPU, and need to use the multi-threading in your code to benefit from it.
  {what is vCPU credit – Virtual CPU, vCPU credits are tied to the amount of memory allocated to a function. Here's how it works: When you configure a Lambda function with memory, it also determines the amount of virtual CPU (vCPU) available to that function}

If your application is CPU bound (Computation Heavy) – Increase the RAM

Timeout: default 3 seconds, maximum is 900 seconds (15 Min)

The Second one is ***Execution Context***

- The execution context is the temporary runtime environment that initializes any external dependencies of your lambda code
- Great for the database connections, HTTP Clients and SDK Clients
- The execution context is maintained for some time in the anticipation of another lambda function invocation
- The next time function invocation – can reuse the context to execution time and save time in initializing connections objects
- It includes /tmp directory

Any connections – which takes much time for the initialization, put it outside of the handler function so that it can be used for the multiple lambda invocations and executions

Now we are speaking about /tmp directory – it's also called as Ephemeral Storage

@ Ephemeral Storage: We are talking about the /tmp folder and space – at what instances we need to use this /tmp folder

- If your lambda function needs to download a big file to work
- If your lambda function needs disk space to perform operations
- Max size of the /tmp directory is 10 GB
- The directory content remains when the execution context is frozen, providing transient cache that can be used for the multiple invocations

@ Timeout: The timeout field defines the upper limit of time (in seconds) that your Lambda function can execute. If the function exceeds this time, AWS Lambda stops the function execution and raises a TimeoutError

The timeout can be set from **1 second** up to **15 minutes** (900 seconds).

Remember that **3 Seconds is the default one** ..not the Minimum One ….! Minimum is 1 second 😉.

Why we need to consider this field very carefully – Look at the below aspects

Cost Management - Lambda functions are billed based on the compute time they consume

Resource Management - Long-running functions can hold onto resources like memory and CPU for extended periods, which might impact other functions if resources are limited.

Error Handling - A well-defined timeout helps in catching situations where the function might enter an infinite loop or become unresponsive

User Experience - For applications that require real-time or near-real-time responses, having an appropriate timeout ensures that the function doesn't take too long to return results, thereby improving user experience.

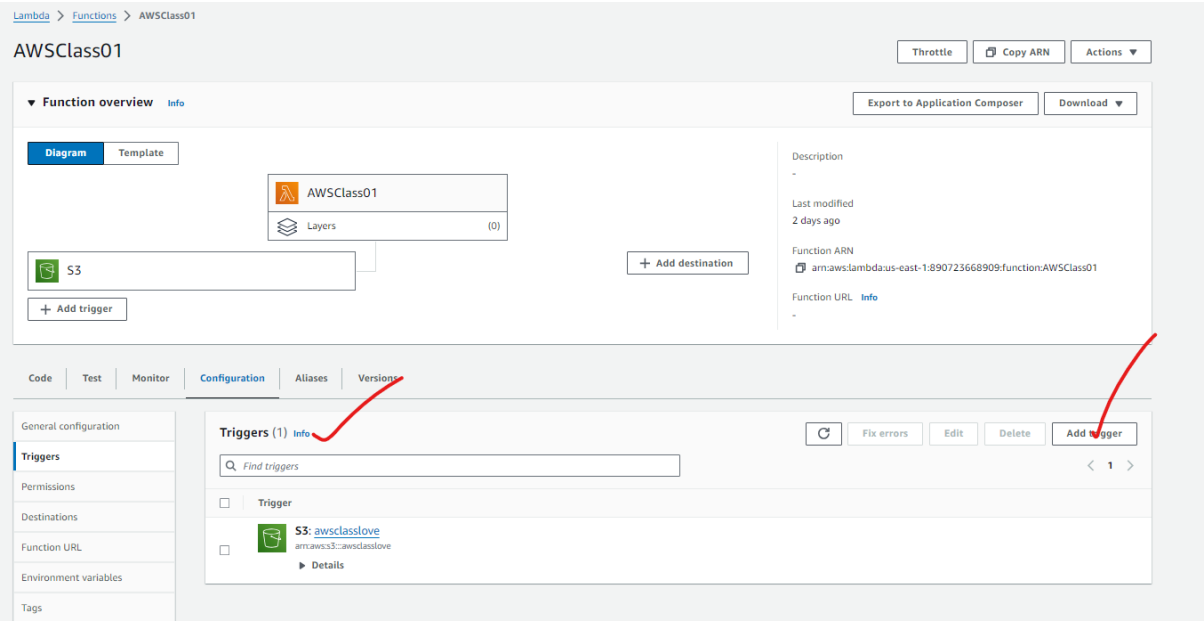
Service Limits - If your Lambda function is triggered by API Gateway, the default timeout for API Gateway is 29 seconds. Therefore, setting a Lambda timeout greater than this would not be useful as the API Gateway would timeout first.

@ SnapStart - AWS Lambda SnapStart is a feature that improves the startup performance of Lambda functions, particularly for latency-sensitive applications

The largest contributor to startup latency (often referred to as cold start time) is the time that Lambda spends initializing the function, which includes loading the function's code, starting the runtime, and initializing the function code1. With SnapStart, Lambda initializes your function when you publish a function version.

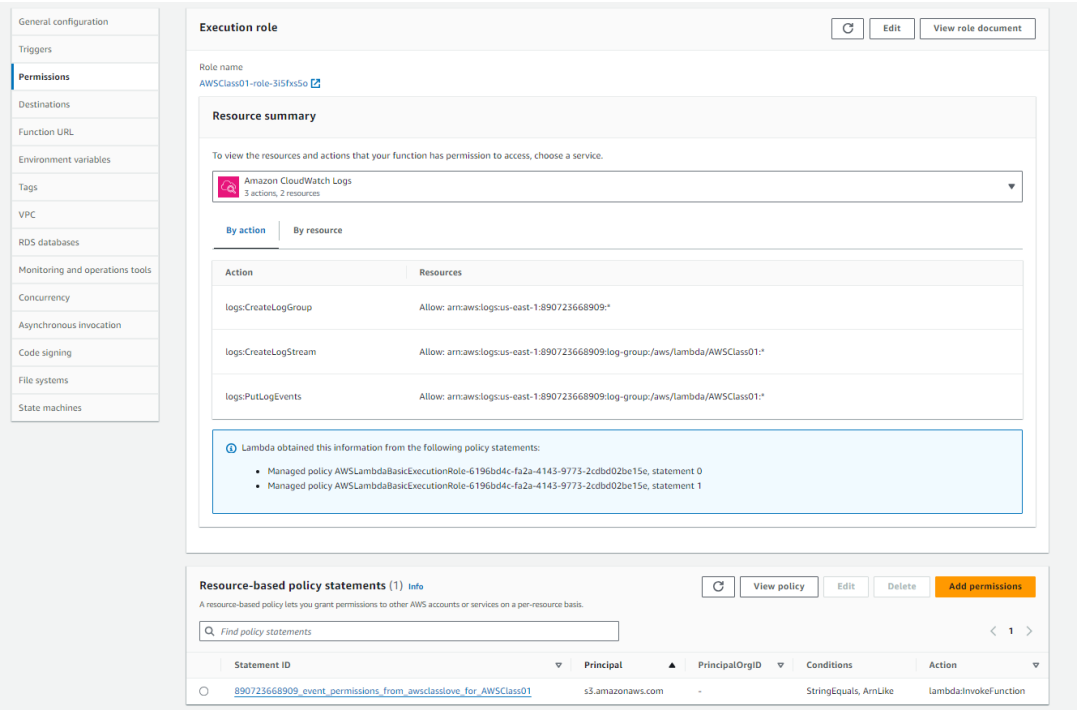
@ Execution role - The role with which, we are going to run this Lambda, we can choose from the existing or create new one as well (We will learn more about this in the IAM Chapter, Keep it aside)

👍 *Triggers*



This is the area – where we learn in extrovert path as well, so for now we can add the triggers from here as well or at the source side as well.

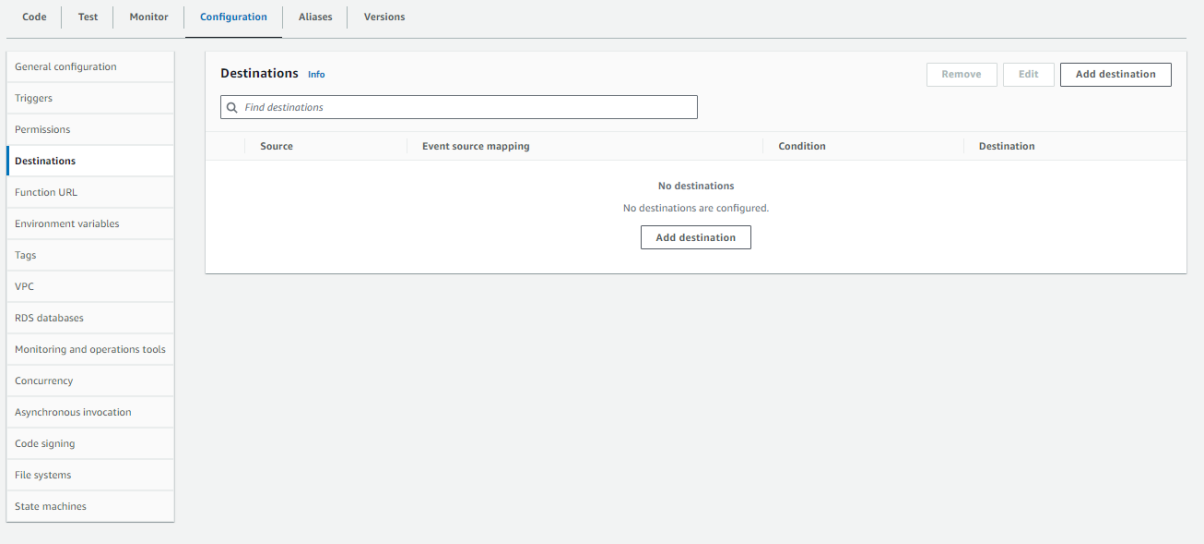
👍 *Permissions*



This is the Tab where define all the Lambda Roles and Its neighbours' permissions (IAM we will learn)

👍 *Destinations*

A destination is an AWS resource where Lambda can send events from an asynchronous invocation. You can configure a destination for events that fail processing. Some services also support a destination for events that are successfully processed.

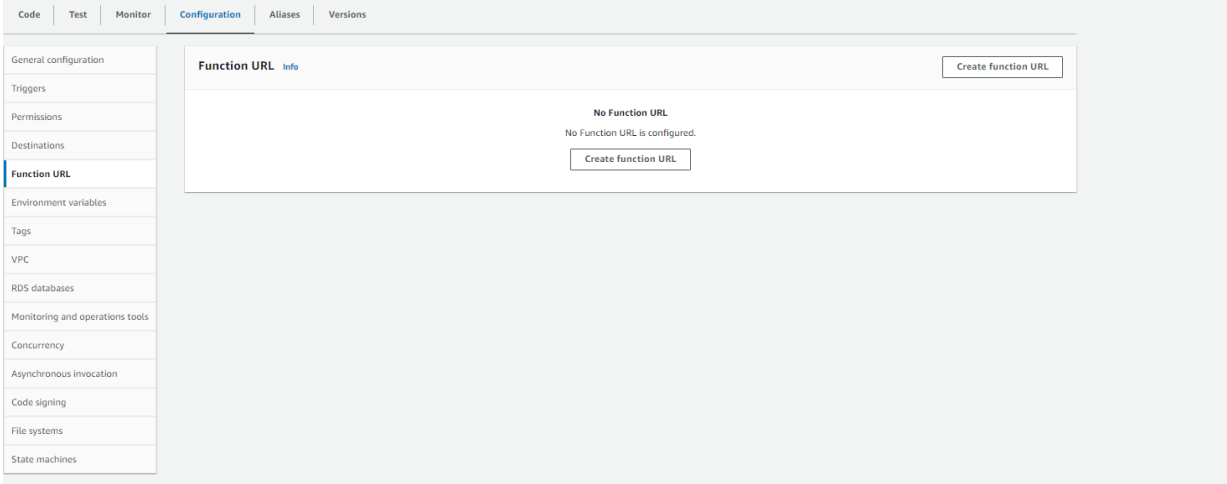


👍 *Function URL*

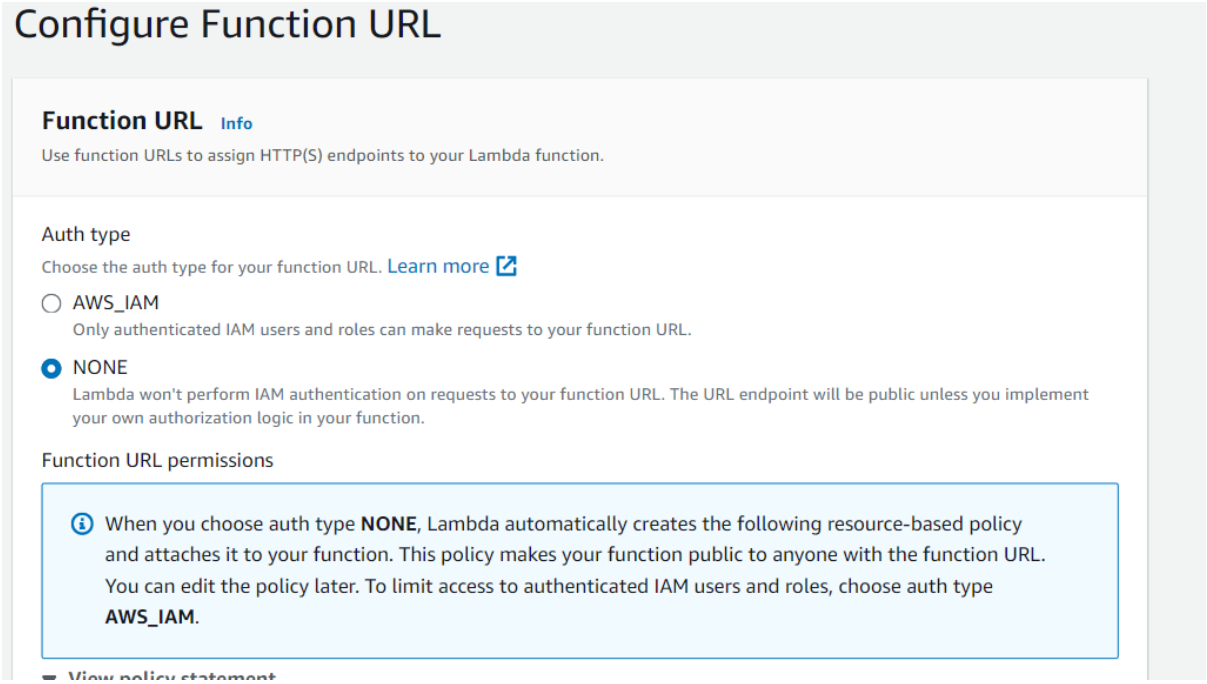
Provides an Endpoint: A Function URL gives your Lambda function a unique URL endpoint, allowing you to invoke the function over HTTP(S) without needing additional infrastructure like API Gateway.

Direct Invocation: You can call the Lambda function directly by sending HTTP requests (GET, POST, etc.) to this URL.

Supports Authentication: You can configure the Function URL to require AWS IAM authentication or to be open for public access, depending on your security needs.

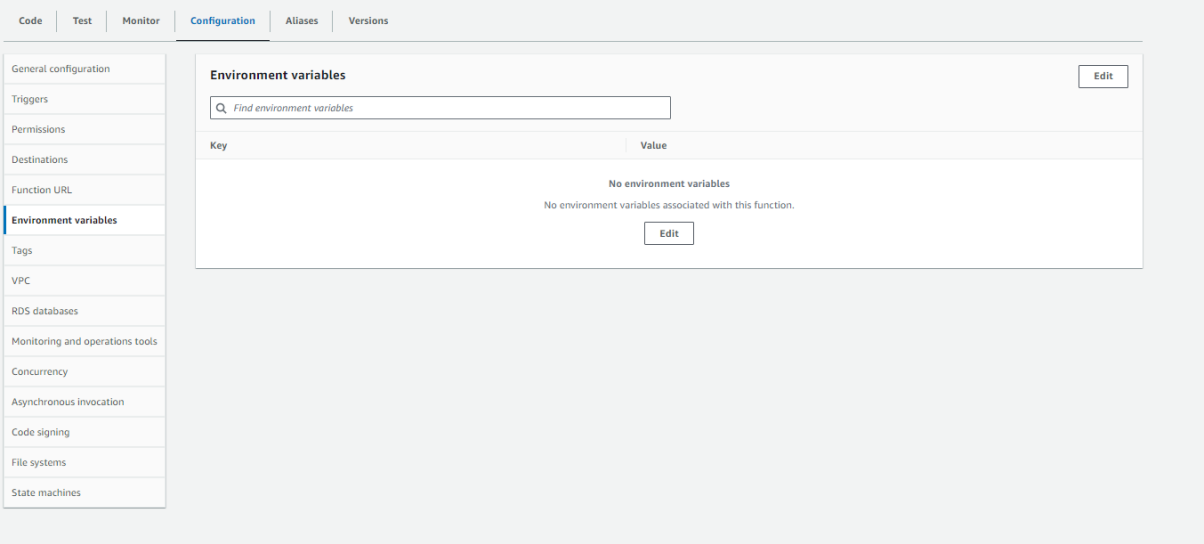


Now – Function URL is having 2 types of auth mechanism, one is via AWS IAM and Other one is None {usually we will not go deep into this} as we these things in the Load Balancer and API Gateway end



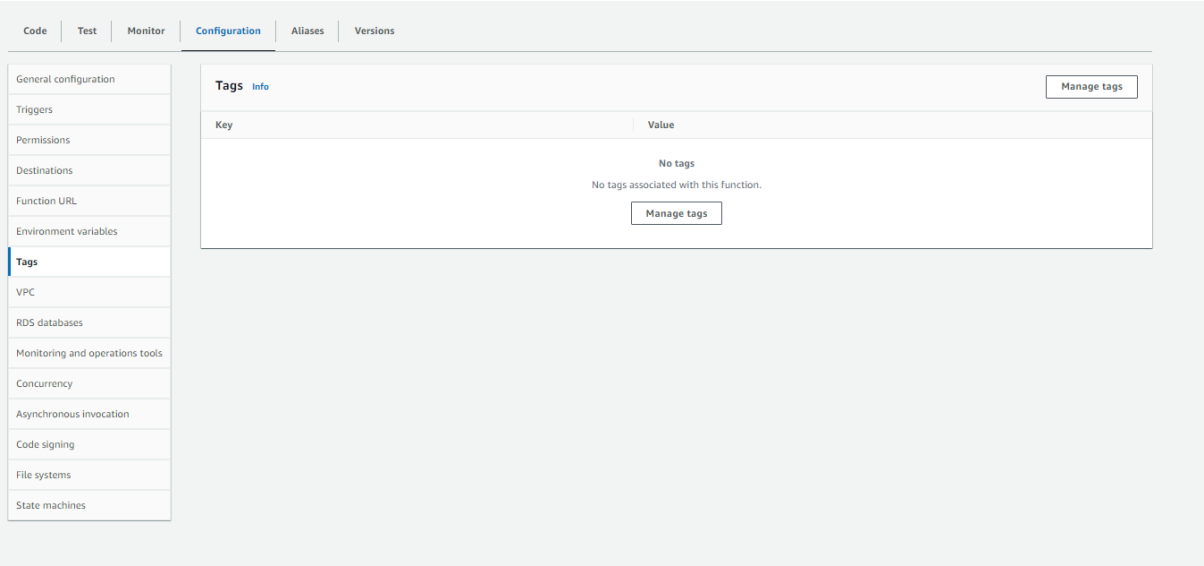
👍 Environment Variables

- Environment variable = Key / Value pair in the String form
- Adjust the function behaviour without updating code
- The environment variables are available to your code
- Lambda service adds its own system environment variables as well
- Helpful to store the secrets
- Importing the OS Module is necessary in-order to retrieve the Environment variables



👍 Tags

Tags are key-value pairs that you can associate with your Lambda functions. They are used for identification, organization, and management purposes.

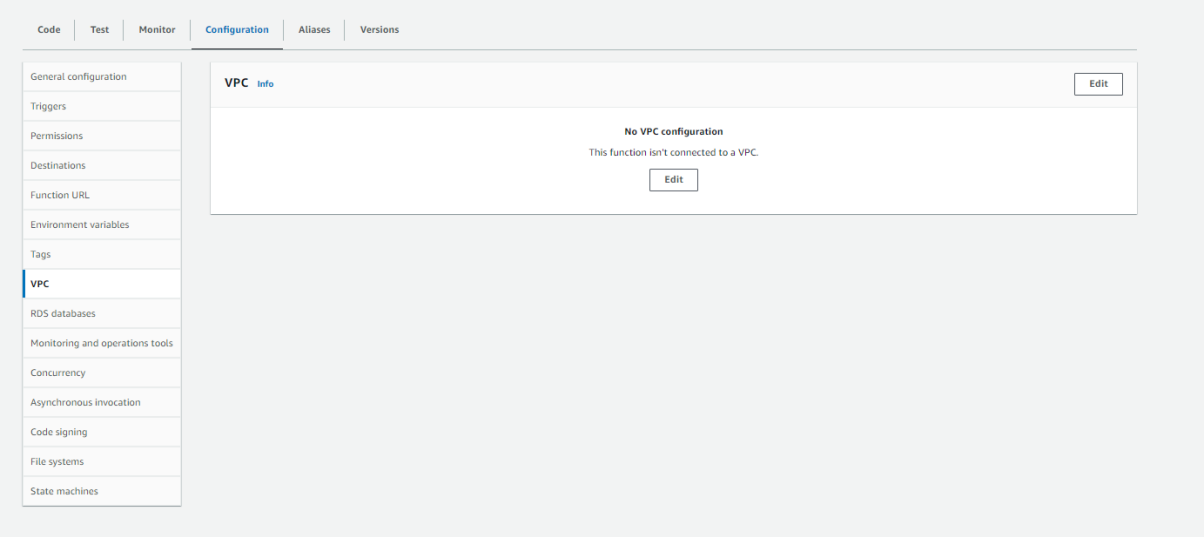


👍 VPC

VPC (Virtual Private Cloud) Config setting allows you to connect your Lambda function to a VPC, providing access to resources within that VPC

➔ *Connects Lambda to VPC:* It specifies the VPC, subnets, and security groups that your Lambda function can use to connect to other AWS resources within your VPC.

➔ *Controls Network Access:* By placing your Lambda function in a VPC, you can control its network access to resources such as databases, cache clusters, and other services running within the VPC.

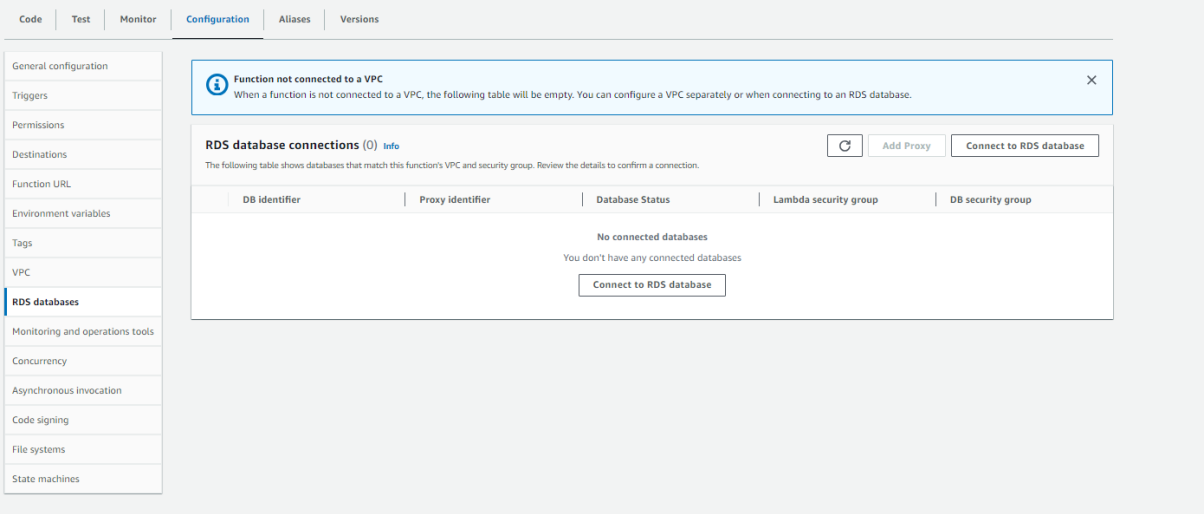


👍 RDS Databases

RDS (Relational Database Service) Databases Config is not a direct configuration option within Lambda itself but involves configuring your Lambda function to securely connect to an RDS database. This setup is significant for building applications that require database interactions, such as querying, updating, and managing relational data.

➔ *Connects Lambda to RDS:* Configuring Lambda to connect to an RDS database allows your Lambda functions to interact with the database to perform operations like executing SQL queries, reading from tables, and writing data.

➔ *Ensures Secure Access:* It involves setting up secure access to the RDS database by configuring VPC settings, security groups, and IAM roles/policies.



👍 Monitoring and Operational Tools

Monitoring and Operational Tools Config refers to the settings and integrations that enable you to monitor, debug, and maintain the health of your Lambda functions. These tools are crucial for ensuring the reliability, performance, and security of your serverless applications.

➔ Enable Logging and Monitoring

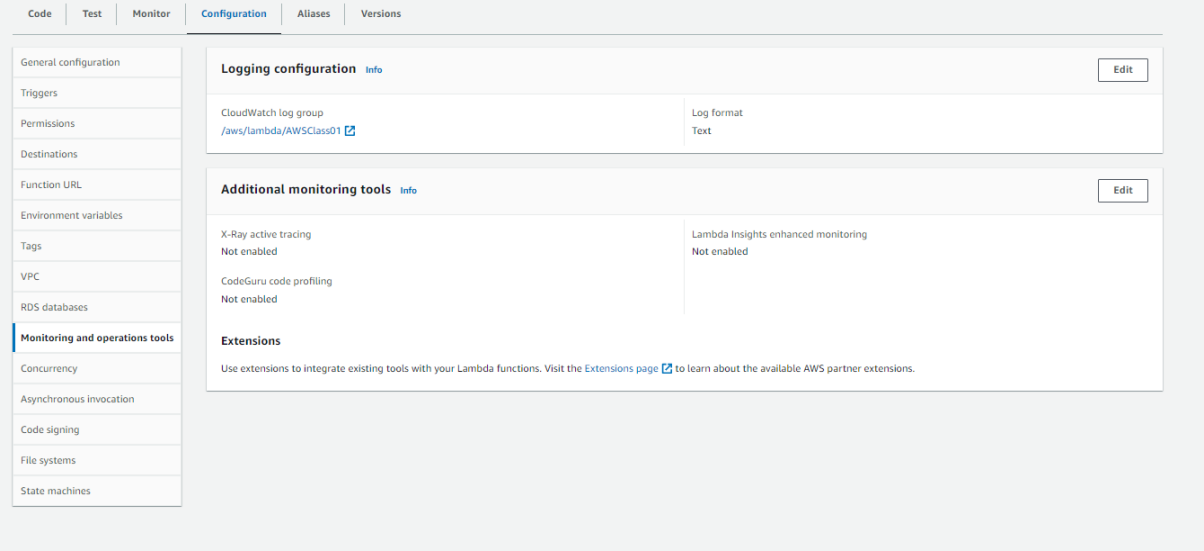
CloudWatch Logs: Automatically captures and stores logs generated by your Lambda function, such as invocation logs, error messages, and custom logs.

CloudWatch Metrics: Provides detailed metrics about your Lambda functions, such as invocation counts, durations, error rates, and throttling.

➔ Set Up Alarms and Notifications

CloudWatch Alarms: Allows you to create alarms based on CloudWatch metrics. These alarms can trigger notifications or automated actions when specified thresholds are met.

SNS Notifications: Integrates with Amazon SNS (Simple Notification Service) to send alerts via email, SMS, or other communication channels.



Cloud Watch Logs you are aware that, they are default for the Lambda Executions, along with that we have something called AWS X Ray (Just keep in mind , sometimes you might get questions on this , we will see about that later)

👍 Concurrency

Concurrency is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.

Minimum 2 and max of 1000 concurrent executions (the max one keep on changing based on the quotas) , now it is 10 for us…! , This will be set based on the usage or we can request AWS to increase it.

As a best practice we need to set the concurrent limit – it is called the reserved concurrency and we set the limit at the Functional level

Each Invocation over a concurrency limit, will trigger a throttle

- If it is synchronous invocation → ***return ThrottleError -429***
- If asynchronous invocation → ***retry automatically and then go to DLQ (Dead letter queue)***

If we need the higher limit, we can raise the support ticket to the AWS

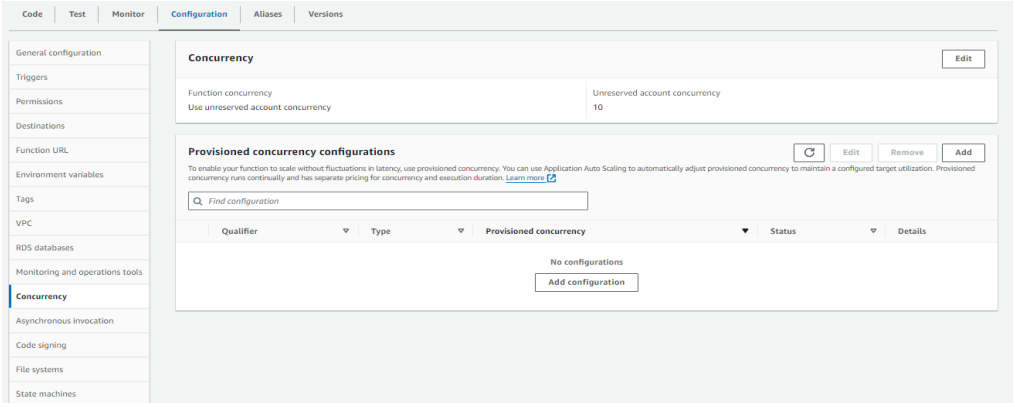
Now we are learning about the Concurrency, now we need to understand below terms as well

Cold Start:

- New Instance → code is loaded and code outside the handler run (init)
- If the init is large (Code, dependencies, SDK …) this process can take sometime
- First request served by new instances has higher latency than the rest

Provisioned Concurrency:

- Concurrency allocated before the function is invoked (in advance)
- So that code start never happens and all invocations have low latency
- Application Auto scaling can manage the concurrency (schedule or target utilization)



👍 Asynchronous invocation

Asynchronous Invocation Config allows you to manage how your Lambda function handles asynchronous invocations. This configuration includes settings such as the retry behaviour and the destination for events when the function is invoked asynchronously

Key Components of Asynchronous Invocation Config

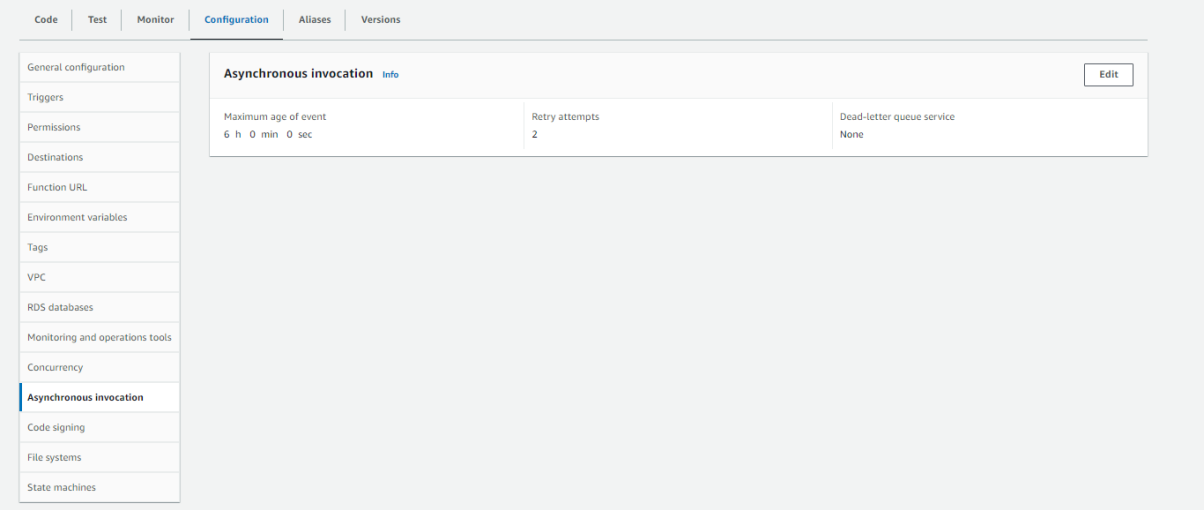
Maximum Retry Attempts: Specifies how many times AWS Lambda should retry an asynchronous invocation if it fails. The default is 2 retries.

Maximum Event Age: Sets the maximum age (in seconds) of an event that Lambda retains and attempts to process. If the event exceeds this age, it will be discarded or sent to the failure destination.

Dead Letter Queue (DLQ): Can be an Amazon SQS queue or an Amazon SNS topic where failed events are sent after all retries are exhausted.

Event Destinations:

- Success Destination: Where to send events after successful asynchronous processing.
- Failure Destination: Where to send events that fail to be processed after retries.



👍 Code signing

Code signing configuration provides a mechanism to verify the integrity and authenticity of the code deployed to your Lambda functions

Integrity Verification: Code signing ensures that the code deployed to Lambda functions has not been altered or tampered with after it was signed.

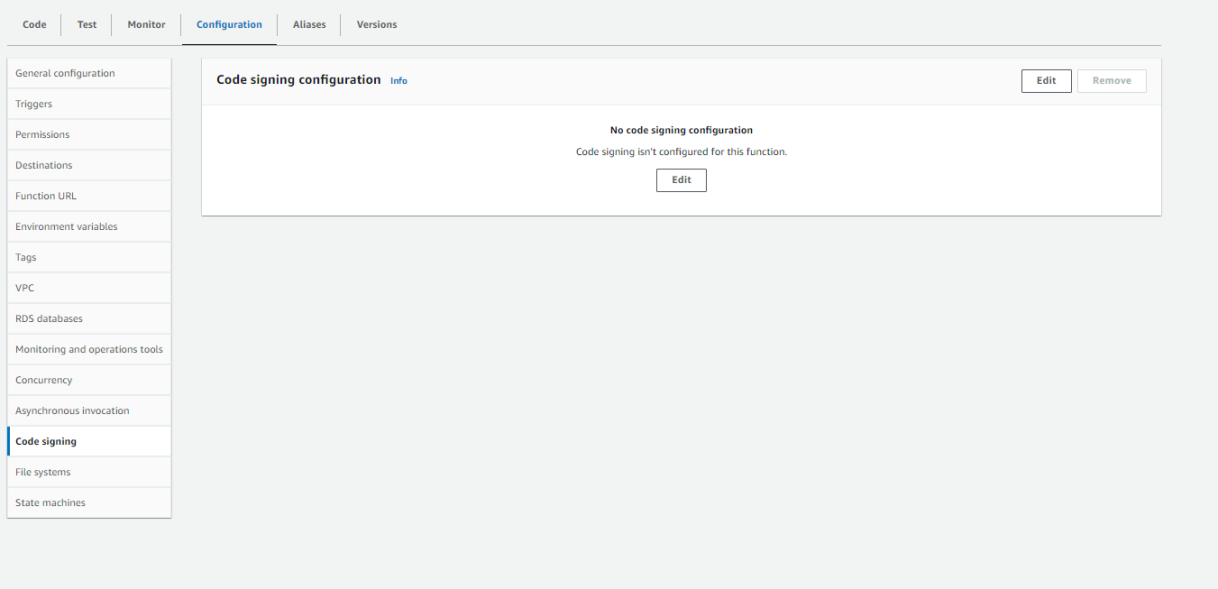
Each deployment package (ZIP file) is signed with a digital signature, which includes a cryptographic hash of the code. AWS Lambda verifies this signature before executing the function.

Authentication of the Publisher: Code signing certificates used to sign Lambda functions are issued by trusted Certificate Authorities (CAs) or AWS Identity and Access Management (IAM) users.

Verifying the signature ensures that the code was indeed signed by an authorized entity, establishing trust in the source of the code.

Enforcement of Security Policies: Organizations can enforce policies that require all Lambda functions to be signed before deployment, ensuring compliance with security standards and mitigating risks associated with unauthorized or malicious code changes.

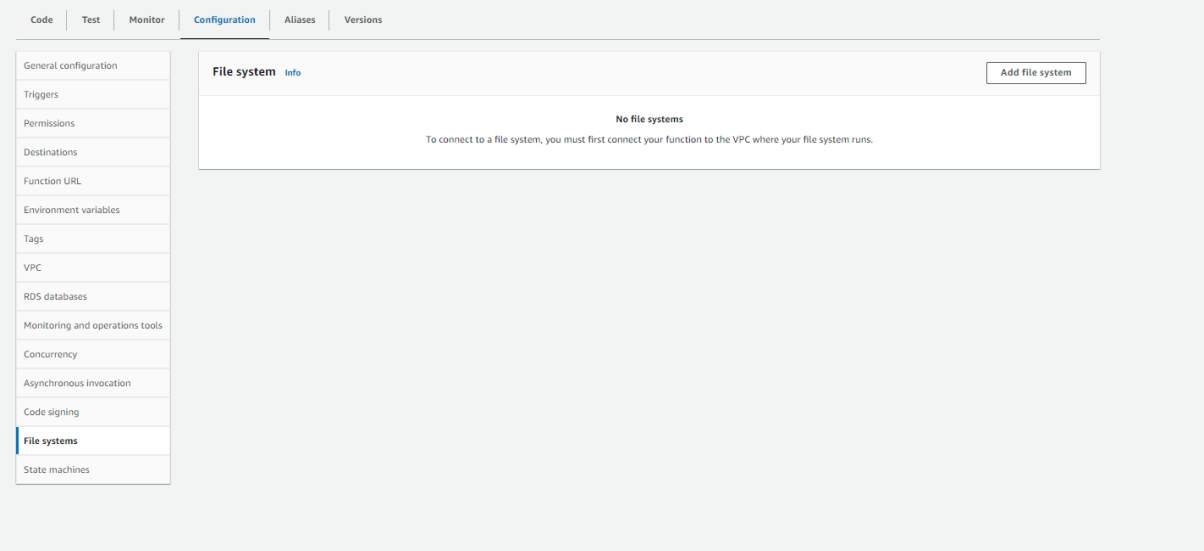
Non-Repudiation: Code signing provides non-repudiation, meaning the entity who signed the code cannot later deny their involvement. This is crucial for accountability in software supply chains.



👍 File Systems

File systems configuration (often referred to as "File System Config") allows Lambda functions to access file systems like Amazon EFS (Elastic File System) or FSx for Windows File Server.

➔ Provides Persistent Storage: Integrates Lambda functions with Amazon EFS or FSx file systems, offering persistent and scalable storage options beyond the temporary storage provided by Lambda's /tmp directory.
➔ Enables Shared File Access: Allows multiple Lambda function instances to access and share files stored in the file system concurrently, facilitating coordination and collaboration between instances.
➔ Supports Stateful Applications: Facilitates the development of stateful applications by enabling Lambda functions to read and write data to a shared file system, maintaining state across function invocations.
➔ Enhances Data Processing: Supports scenarios where Lambda functions need to process large volumes of data stored in a file system, such as processing media files, log files, or database backups.



👍 State Machines

State Machines configuration refers to integrating Lambda functions with AWS Step Functions.

Workflow Orchestration:

➔ AWS Step Functions: AWS Step Functions is a serverless orchestration service that allows you to coordinate multiple AWS services, including Lambda functions, into scalable workflows or state machines.
➔ Integration: The State Machines configuration in Lambda allows you to specify a Step Functions state machine as the orchestrator for controlling the execution flow of your Lambda functions.

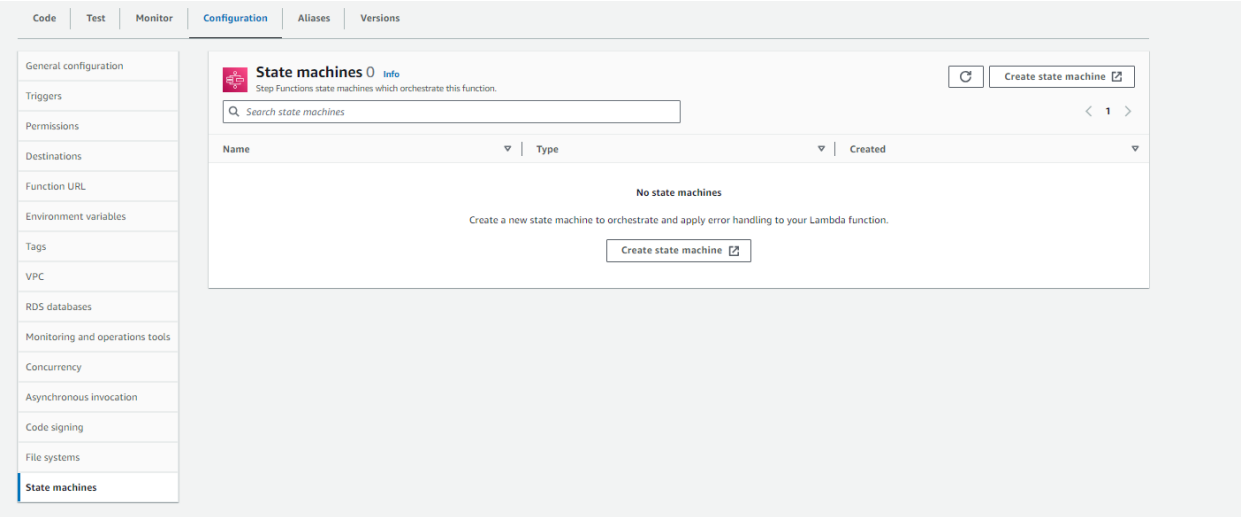
Define Workflow Logic:

➔ States: Step Functions lets you define states, transitions, and conditions to specify how different components of your application interact and react to events.
➔ Lambda Integration: You can invoke Lambda functions from Step Functions as individual steps within a state machine, passing input data and capturing output for subsequent steps.

Error Handling and Retry Logic:

➔ Step Functions provides built-in error handling capabilities, including retries, catch clauses, and fallback states, allowing you to build robust workflows that can recover from failures or errors.
➔ Visibility and Monitoring:

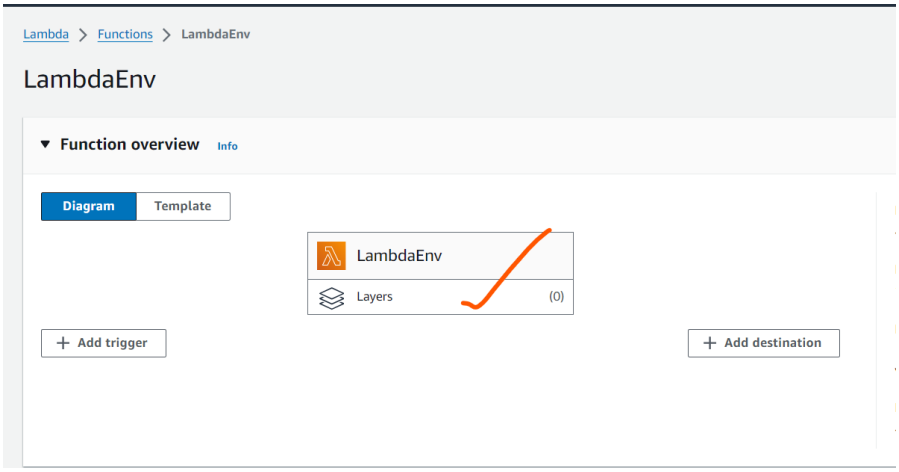
Step Functions offers real-time visibility into the execution status and progress of workflows, including detailed logs and metrics for monitoring and troubleshooting purposes.



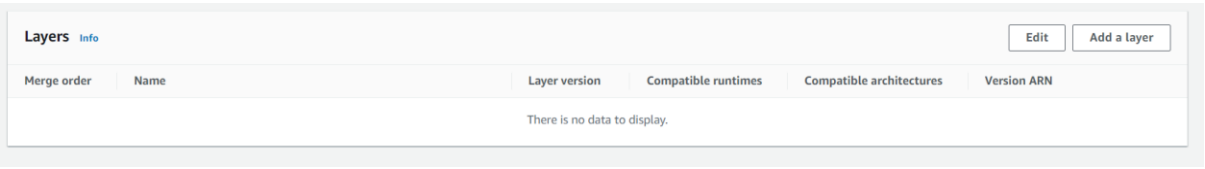
👍 Layer

A Lambda layer is a .zip file archive that can contain additional code or other content. A layer can contain libraries, a custom runtime, data, or configuration file.

Layers provide a convenient way to package libraries and other dependencies that you can use with your Lambda functions. Using layers reduces the size of uploaded deployment archives and makes it faster to deploy your code. Layers also promote code sharing and separation of responsibilities so that you can iterate faster on writing business logic.



;



👍 Qualifier

When you invoke or view a function, you can include a qualifier to specify a version or alias. A version is an immutable snapshot of a function's code and configuration that has a numerical qualifier.

👍 Deployment Package

Lambda supports two types of deployment packages:

- A .zip file archive that contains your function code and its dependencies. Lambda provides the operating system and runtime for your function.
- A container image that is compatible with the Open Container Initiative (OCI) specification. You add your function code and dependencies to the image. You must also include the operating system and a Lambda runtime.

🚄 Extrovert Path:

We have integrated Lambda with below services

- S3 → Lambda
- Event Bridge → Lambda
- Lambda → SQS
- Lambda → SNS
- Application Load Balancer → Lambda

Other examples like kinesis and SQS triggering Lambda we will see when we are learning about those services in specific

- Small programs related to the Introvert Path

Questions to practice

Numerical Questions:

https://lisireddy.medium.com/aws-lambda-numerical-questions-cd640f880009

Scenario Questions:

https://lisireddy.medium.com/aws-lambda-scenario-based-questions-86cb7e9207e1

Wiish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)