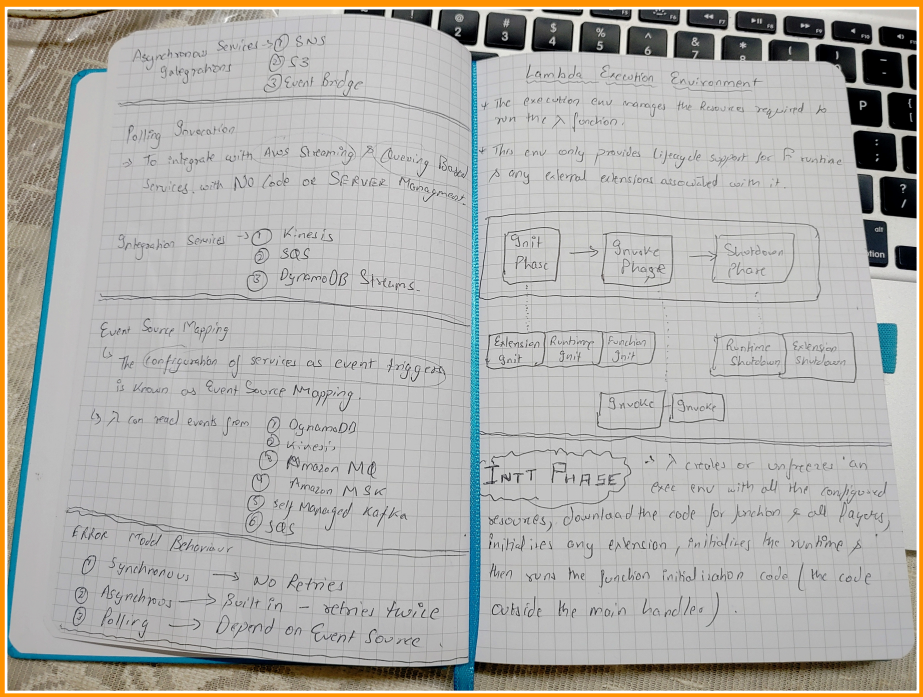
AWS

SERVERLESS DIARY



BY BHAVUK BHARDWAJ

CONTENT

1.
JUST STARTING

- **Lambda** - It's service provided by AWS that allows you to run your code without provisioning or managing servers.
- This technology - run your code without provisioning or managing servers is called as "**Serverless Computing**"
- **Support multiple languages** like Node.js, Python, Java, Go, .NET etc.
- Lambda Function - λ are stateless.
- **Q** - **What does it mean when we say λ are stateless ?**

A - In simple terms, it means that they don't remember anything between different invocations.

Example - Imagine you have a Lambda function that adds two numbers. When you invoke the function with the numbers 3 and 4, it will return the result, which is 7. But the Lambda function doesn't remember this result for the next invocation. If you invoke the function again with different numbers, say 5 and 2, it will calculate the result as 7 (5 + 2) without any knowledge of the previous invocation.

Note - If you need to store and retrieve data between invocations, you would typically use external storage services like databases or file systems, rather than relying on the

Lambda function itself to maintain any kind of memory or state.

- **Q** - **How many different kinds of invocations are there for λ ?**

  A - Three kinds of Invocations:-

  1. Synchronous Invocation
  2. Asynchronous Invocation
  3. Polling Invocation

- **Synchronous Invocation**:
  - **λ** runs the function (code) & wait for response.
  - **λ** completes the execution and return the Response with Additional Data.
  - This type of invocation is required when there is a immediate Response is required. (Like Payment System)

- **Asynchronous Invocation**: In this, events are **queued** and Requestor does not wait for λ to complete.

- **Polling Invocation:**
  - In context of λ, it refers to a method of triggering or invoking a λ Function based on periodic polling or checking for a specific condition.

  - **Example**, let's say you have a Lambda function that needs to retrieve new data from a database every 5 minutes. Instead of waiting for an external event or trigger to occur, you can configure the Lambda function

with a polling invocation. It will then automatically run the function every 5 minutes to check for any new data in the database.

List of Integration Services Based on Invocation

Invocation Types	Integration Services
Polling Invocation	SQS, Kinesis Data Stream (KDS), DynamoDB Streams, Managed Kafka Streams, Amazon MQ etc
Synchronous Invocation	API Gateway, AppSync, Cognito, Alex, Lex, CloudFront (Lambda@Edge)
Asynchronous	SNS, S3, EventBridge, SES, CloudFormation, CodeCommit, CodePipeline, CodeDeploy,

Error Model Behaviour

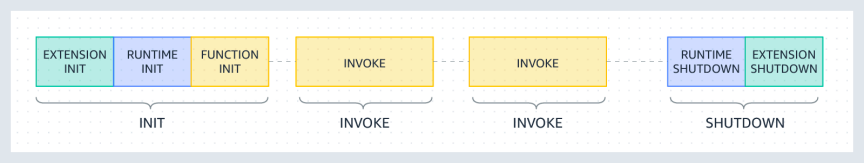
Invocation Types	Behaviour
Polling Invocation	Depend on Event Source
Synchronous Invocation	Built-in ; Retires Twice
Asynchronous	No Retires

- **Q - What is Event Source Mapping in λ ?**
  - **A -** When you configure AWS services as event trigger that is called as Event Source Mapping.
    - Example - SQS as an event trigger.
  - λ can read from :-
    - SQS
    - DynamoDB

- Kinesis
- Amazon MQ
- Amazon MSK (Managed Streaming Kafka)
- Self managed Kafka

2.
LAMBDA EXECUTION ENVIRONMENT

- The execution environment only managed the resources that are required to run the λ function.
- This env only provides lifecycle support to λ function runtime and any external extensions associated with it.



Lambda execution env lifecycle

- **3 Phases** - Init Phase, Invoke Phase and Shutdown Phase

INIT PHASE

- In this init phase, λ creates an execution env with
  - all the configured resources (memory etc);
  - download the code for function & all layers;
  - Initialises any extension then
  - Initialises the runtime and then
  - Run the function initialisation code.

- Init Phase happens either during the First invocation OR before function invocation (in case of enabled Provisioned Concurrency)
- **Init Sub Phases:-**
  - Extension Init - Starts all execution
  - Runtime Init - Bootstraps the runtime
  - Function Init - Runs the function static code
- The three sub phases ensures that all extension & their runtime complete their setup tasks before the function code runs.

INVOKE PHASE

- In this phase λ invokes the function handler.

SHUTDOWN PHASE

- If λ does not receive any invocations for a period of time, then this phase gets initiated.
- λ shutdown the runtime, alerts the extensions to let them stop cleanly and then removes the environment.
- λ send the shutdown event to each extension, which tells the extension that the env is about to shutdown.

- **Shutdown Sub Phases**:-
  - Runtime Shutdown
  - Extension Shutdown

3.
PERFORMANCE OPTIMISATION

- Serverless uses below to improve the performance:
  - Parallelisation
  - Concurrency
  - Auto Scaling

COLD START

A cold start occurs when a new execution env is required to run a λ. When the λ receive the request to run the function, the service first prepares an execution env. During this step, the service download the code then creates the execution environment with the specified Memory, Runtime and Configuration. Once completes, λ runs any initialisation code outside of the event handler before finally running the handler code.

WARM START

The λ retains the environment instead of destroying it immediately. This allows the function to run again within the same execution env. This saves Time by not needing to

initialise the new env. In warm start, everything is already setup; λ just needs to run the code.

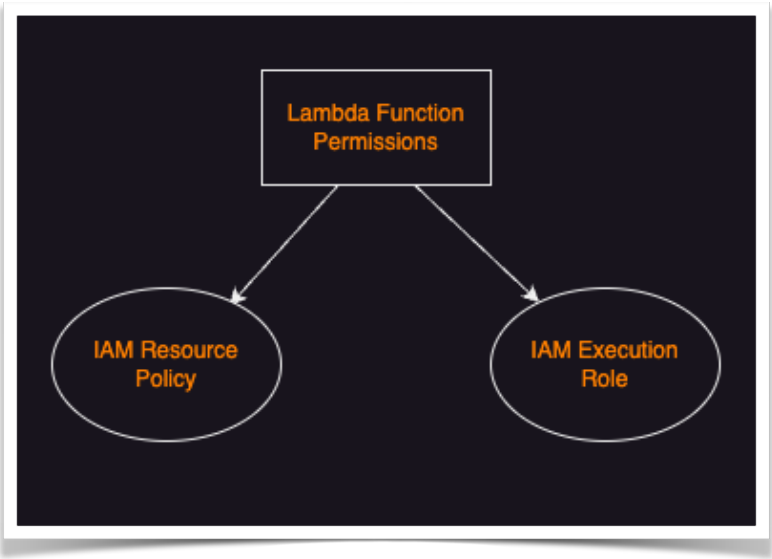
Comparison b/w Cold and Warm

	COLD	WARM
Defintion	The initial invocation of λ function insitance	Subsequent invocation of a λ Function
Invocation Time	Longer execution time due to instance initialisation	Shorter execution time as the instance is already warm.
Instance Creation	New Instance is created to handle the first innovation.	Reuses an existing instance from previous invocation
Initial Overhead	Additional time required for initialisation and setup	No additional initialisation overhead is required.
Resource Allocation	Resources are provisioned for the instance	They are already allocated & ready for execution
Cold Start Penalty	Higher Latency; Increased Response Time	Low Latency; Improved Response Time
Cost Implications	Potentially higher cost due to instance creation	Potentially lower cost as instances are re-used.
Occurence	Happens on the first invocation OR when scaling up	Happens after the initial invocation OR during scaling.

- In cases, where there is NO available Warm Instance then a Cold Start might still occur.

- **Cold Start Latency** - The time taken When an invocation is routed to a new execution env, λ must initialise the env, download the code, initialise the runtime & initialise any packages or dependencies required by function is called as “Cold Start Latency”.

4.
PERMISSIONS



- 2 Types of permissions can be attached to
  - Resource Based Policy
  - Execution Role

- **Resource Based Policy** - is also called as **Function Policy** that tells the λ service which principals have permission to invoke λ function.
  - It makes easy to grant access to the λ Function across separate AWS account.

- It does have a size limit. If you hit that limit then use IAM Roles instead of resource policies (IAM assume role).

Feature	Resource Policy	Execution Role
Purpose	Control access to the λ Function's AWS Resources.	Grant permission to the λ function during execution.
Permission	Define what actions can be performed on λ	Define what action λ function can perform.
Cross Account Access	Can allow access from other AWS Account	Can assume IAM Role in other AWS Account
Usage Example	Restricting Access based on IP, VPC or other condition	Accessing other AWS Resources or services during Runtime.

5.
VPC AND PRIVATE LINK

LAMBDA VPC

- This functionality allow λ to access resources in VPC.
- It does not change How the function is secured.
- It uses VPC Subnet Id's and Security Group Id's.
- λ also requires - create, describe and delete ENI's permission.IAM Policy - 'AWSLambdaVPCAccessExecutionRole'

PRIVATE LINK

- To establish a private connection between VPC and Lambda, create an **Interface VPC Endpoint**.
- Interface VPC Endpoint are powered by AWS Private Link, which enable us to privately access Lambda API's without an Internet Gateway ; NAT Device ; VPN Connection ; Or AWS Direct Connection.
- Instances in your VPC does not need to be in Public IP Address to communicate with λ API's. Traffic between VPC & λ does not leave the AWS Network.

6.
BEST PRACTICES

LAMBDA DESIGN BEST PRACTICES

- Minimise Cold Start Time using Provisioned Concurrency -
  - Use **Provisioned Concurrency**. Its a feature that prepares concurrent execution environment before invocation.
  - Provisional Concurrency ensures the **Lowest Possible Latency**.
  - It keeps the function initialised & warm & ready to respond in **double-digit millisecond**.
  - Unlike with ON-DEMAND, all setup activities happen before invocation, including running the initialisation code.

- **Separate Business Logic** - Separate Core business Logic from the Handler Event. This makes code more Portable and you can target unit tests on your code without worrying about the configuration of the Function.

- **Write Modular Function** - It will reduce the amount of time that it takes for your deployment package to be downloaded and unpacked before invocation.

- **Treat Function as Stateless** - No information about state should be saved within the context of the Function itself because your function only exist when there is work to be done. If you have to store state data, consider below options -
  - DynamoDB
  - Elasticache
  - S3

- **Only Include what you need** - Minimize deployment package Dependencies and its size. This improves significant startup time.

LAMBDA WRITING CODE BEST PRACTICES

- Use Logging Statement
- Use Return Coding (function must give λ info about the result of their action)
- Use λ Environment Variables for operational Parameters
- λ can also encrypt the env variables with a key that it creates in your account (CMK).
- Use Secrets Manager or Parameter store to store secrets or credentials or sensitive information.
- Avoid Recursive code.

- NOTE - If you accidentally deploys recursive code, you can quickly set the concurrent execution limit to zero by using console or CLI immediately.

- Reuse execution context - Take advantage of an existing execution context when you get a warm start by doing the below:-
  - Store Dependencies locally
  - Limit - reinitialisation of variables
  - Reuse existing connections
  - Use ‘tmp’ space in Lambda as transient cache.
  - Always check the background process have completed.

7.
AWS SAM

- Open source framework for building serverless framework.
- It provides short hand syntax express functions, API's DB, Event Source Mapping.
- Its in YAML Format
- If Cloud Formation Template mentions "transform" keyword in it. It specifies AWS SAM Template
- You can use AWS SAM CLI to test and Deploy. SAM Cli uses docker container that you can interact with to test and debug λ function.
- If you have to locally test your serverless app, validate the SAM Template and streamline your deployments; then use AWS SAM CLI.
- **Commands** :-
  - Init - Initialise the Serverless Application
  - Local - Runs your application locally
  - Validate - Validate the SAM Template
  - Deploy - deploy's the SAM Application
    - NOTE - Deploying λ function via Cloud Formation Template requires S3 Bucket. But SAM Cli creates and manage this Amazon S3 Bucket for you.

- Build - It process your SAM Template file, application code and any dependencies. This command also copies the *Artifacts*.

8. BALANCE BETWEEN POWER & DURATION

- MEMORY
- TIMEOUT
- CONCURRENCY

- **MEMORY** - you can allocate upto 10gb's of memory; then λ will allocates CPU and other resources linearly in proportion to the amount of memory configured. (If men goes high then CPU goes high)
  - NOTE - To know if you have rightly configured the memory; you can use '**AWS Lambda Power Tuning Tool**'
- **TIMEOUT** - How long Function can run. Max 15 minutes.

NOTE - Balance between Power and Duration

- Sometimes higher memory might actually cost less because Function can complete much more quickly than a lower configuration.

- **CONCURRENCY** - Number of λ function invocations running at a single time.
- 3 Types of concurrency:-
  - **Unreserved -** The amount of concurrency that is NOT allocated to any specific set of functions. You should always have **minimum 100** unreserved concurrency to server the other requests.
  - **Reserved** - Guarantees the maximum number of concurrent instances for the function. No charges is incurred for configuring the reserved concurrency.
  - **Provisioned** - Initialises a requested number of runtime environments so that they are prepared to respond immediately to function invocations. It provides High Performance and Low Latency.

CONCURRENCY BURST

- A burst is when there is a sudden increase in number of instances needed to fulfil the request.
- NOTE - Burst Concurrency quota is NOT applicable on per function. It applies to all function in that region.
  - 3000 - US West, US East, Ireland
  - 1000 - Tokyo, Frankfurt, Ohio
  - 500 - Other Regions
- Means after the initial burst, function concurrency can scale by adding 500 instances each minute (other regions).
- This continues until there are enough instances to server all the request or until a concurrency limit is reached.

Q:- What are some reasons a developer would set a concurrency limit ?

- Match the limit with a downstream resources. Example, assume it, you have an application that connects to Database. Now, the DB is the downstream resource which you want to limit the number of connections.
- Manage Cost
- Regulate how long does it takes to process a batch of events.

9.

REDUCING RISKS USING VERSION & ALIASES

- **VERSIONING** - You can use versions to manage the deployment of λ functions. You can publish a new version of a function for Beta Testing without affecting users of stable production version.

- **PUBLISH** - Publish makes a snapshot copy of $LATEST version.
  - Publish as many versions as you need.
  - Each version result in a new sequential version number.
  - Add the version number to the function ARN to reference it.
  - The snapshot becomes the new version and is immutable. (Immutable refers to an object or Data Structure that can not be modified or created.

- **ALIASES** - λ alias is like a pointer to a specific function version. You can access the function version using the Alias ARN.
  - An alias can only point to a function version not to another alias.
  - You can update an alias Point-to-new Version.

TEST USING ALIAS ROUTING

- We can also use Routing configuration on an alias to send a portion of traffic to a second version
- NOTE - You can point an alias to a maximum of 2 Lambda function version. The version must meet the following criteria:-
  - Both Version must have same runtime role
  - Both version must have same DLQ Config OR No DLQ at all.
  - Both version must be published. Alias can't point to the $LATEST Version.

INTEGRATE WITH AWS CODE DEPLOY

- **Traffic Shifting Patterns**
  - **CANARY** - Traffic is shifted in Two increments. If the first increment is successful , the second is completed based on the time specified in the Deployment.
- **LINEAR** - Traffic is slowly shifted in a pre-determined %age every 'x' minute based on you have it configured.
- **ALL-AT-ONCE** :- Shifts all traffic from original λ function to the updated λ function version at once.

10.
MONITORING

MONITORING METRICS

- **Invocation** - The number of times λ function runs including Success or Function Errors (it does not cover)

- **Duration -** The amount of time function code spends processing an event.

- **Errors** - The no. of invocation that result in a function error. Function Error include:-
  - Exceptions thrown by code
  - Exceptions thrown by Lambda Runtime
- Runtime ERROR - occurs for issues such as Timeout and Configuration Errors.

- **Throttle** - The number of time that a process failed because of Concurrency Limit. Throttling is not a function error.

- **Iterator Age** - The age is the amount of time b/w when the stream receives the record and when the event source mapping send the event to the function.

- **Dead Letter Error** - For Asynchronous invocations, this is the number of times λ attempts to send an event to DLQ but fails.
- **Concurrent Execution** - The number of λ instances running at the same time.

LAMBDA INSIGHTS

- It uses a new cloud watch λ extension, which is provided as Lambda Layer. When enabled,
  - it collects system level metrics;
  - it summaries diagnostic info such as cold start and lambda worker shutdown to help isolate issues with λ Function.
- You can also use Lambda Insights to identify Over and Under Utilised Lambda's.

AWS X-RAYS

- It Visualises the component of your application, identify performance bottlenecks & troubleshoot requests that resulted in an error.
- λ function send Trace Data to X-Ray and it process the data to generate the a SERVICE MAP & SEARCHABLE TRACE SUMMARIES.
- **USES**:
  - Tuning Performance
  - Identify the call flow of λ Function and API Calls
  - Tracing Path & Timing of an invocation to locate bottleneck and failure.

CLOUDTRAIL

- Audit your application by recording all the API actions made.
- It can also be used auditing serverless deployments & rolling back unplanned deployments.

DEAD LETTER QUEUE (DLQ)

- Helps to capture **Application Errors**, that must receive a response, such as try to understand with a basic e-commerce application that processes orders. If an order fails, you can not ignore that order error. You move that error into the DLQ and manually look at the queue and fix that problem.
- To analyse the failures for follow up and corrections.
- NOTE - DLQ are available for Asynchronous and Non-Stream Polling Events.
- DLQ can be SQS (normally) Or you can configure SNS Topic as a λ DLQ.

ABOUT THE AUTHOR

My name is Bhavuk Bhardwaj. I am an experienced AWS DevSecOps Engineer and Infrastructure Engineer. I am having around 5 Years of experience in this field and had the privilege of contributing to the growth of 3 **Product Based Startups**, and designed and implemented multiple system based on AWS Cloud.
Connect - https://www.linkedin.com/in/bhavukbhardwaj/
Website - https://www.notfromiit.com/