Overview

This document outlines best practices when considering AWS Serverless Lambda.

When to Consider For Use

AWS Lambdas functional use cases continue to grow. Below are some recommended use cases:

Small Event Based Processing

```
- Scan tags on resources
- Backup configuration settings
```

Data Transformation

```
- Unity Logging - Data from Kinesis stream to Athena.
- Cloudwatch log shipping to SumoLogic
```

Small Jobs - Data Backups, Security checkups, log parsing

```
- Trigger Database Maintenance
- Security Scanning
```

Restful APIs - (With Warming pattern)

```
  - TELS - Capital planning
  - TELS - Site Visits
```

When Not to Use

AWS Lambdas are not a fit for everything, and should avoid the following:

Long running batch processes

- Lambdas are intended for short executions of small sets of logic.

Stateful applications

- Applications that require a persisted state in memory state. These processes are short living in duration.

Large Monolithic sets of logic/Highly complex applications

- Intended for small sets of functionality, not an entire monolithic application in one Lambda.

Reserved concurrency Vs Provisioned concurrency

- Reserved concurrency guarantees the maximum number of concurrent instances for the function. When a function has reserved concurrency, no other function can use that concurrency. There is no charge for configuring reserved concurrency for a function.
- Provisioned concurrency initializes a requested number of execution environments so that they are prepared to respond immediately to your function's invocations. Note that configuring provisioned concurrency incurs charges to your AWS account.

Benefits of Reserving concurrency

Other functions can't prevent your function from scaling

- All of your account's functions in the same Region without reserved concurrency share the pool of unreserved concurrency. Without reserved concurrency, other functions can use up all of the available concurrency. This prevents your function from scaling up when needed.

Your function can't scale out of control

- Reserved concurrency also limits your function from using concurrency from the unreserved pool, which caps its maximum concurrency. You can reserve concurrency to prevent your function from using all the available concurrency in the Region, or from overloading downstream resources.

Other Considerations

Cold Starts

- If you cannot afford to have your lambda go cold you can implement provisioned concurrency. Provisioned concurrency initializes a requested number of execution

environments so that they are prepared to respond immediately to your function's invocations. Note that configuring provisioned concurrency incurs additional costs. Only provision to accommodate the max perceivable requests that would need to be handled in parallel.

Reserved Concurrency

- If your lambda needs to run when targeted, you should configure reserved concurrency.
- Cloudwatch Alarms should be added on your lambdas concurrent executions notifying when thresholds start to near set reserved concurrency limits.
- No lambda should be trying to right size reserved concurrency. It's meant to be set as an upper bound that should never need to be exceeded.
- Lambda versioning - Reserved concurrency counts against the lambda as a whole. Meaning if you deploy 5 versions of the same lambda with a provisioned concurrency of 2, you are already consuming 10 hot instances of this lambda. The reserved concurrency must be set higher than the total provisioned instances.
- If it is acceptable for your lambda to get execution time when available (you have retries built in or another mechanism to handle this), you can stay in the shared pool. Your lambda will only run when executions are available in the shared pool.

Deprecated Runtimes

- Language versions can be deprecated, requiring existing Lambdas to be updated to a newer version
- This happened recently - Node.js 6.x was deprecated, requiring teams to update to a newer version

Portability of FaaS

Function-as-a-Service

- Well organized code can be portable with few changes
- Separate the Lambda handler from your core logic (Adapter Pattern/ DS best practice)
- Lambda handler is a wrapper for your business logic.

Recommendations

- Use provisioned concurrency when building interactive mobile or web backends, latency sensitive microservices, and synchronously invoked APIs.
- Auto-scale your provisioned concurrency for on & off peak loads.
- Add reserved concurrency limits to ensure execution environment is available.

- Set Max Runtime/timeout on your lambda. This will help detect issues early & avoid large monetary mistakes.
- Always follow portability design pattern (Adapter Pattern).
- Understand the architectural dependencies of serverless and how not to directly couple business logic with proprietary aws services.
- Maintenance of up to date runtimes should be considered as part of the maintenance plan.

Example: Terraform module used to schedule lambdas

Scheduled AWS Lambda Function This module can be used to deploy an AWS Lambda function which is scheduled to run on a recurring basis.

Example: Implementation of Solr Cloud backup validation

Scheduled AWS Lambda function Implementation Validate the SolrCloud backup