AWS Lambda

If you are studying for AWS Developer Associate Exam, this guide will help you with quick revision before the exam. it can use as study notes for your preparation.

Dashboard Other Certification Notes

AWS Lambda

Why use AWS Lambda?

- Avoid limitations of EC2 Servers:
  - Virtual Servers in the Cloud
  - Limited by RAM and CPU
  - Continuously running
  - Scaling means intervention to add / remove servers
- Lambda provides:
  - Virtual functions
  - no servers to manage
  - Quick, short executions
  - Run on-demand
  - Scaling is fully automated

Benefits of AWS Lambda

- Easy Pricing:
- Pay per request and compute time
- Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
- Integrations with the entire AWS Stack
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per functions (up to 3GB of RAM)
- Increasing RAM will also improve CPU and network

AWS Lambda language support:

- Node.js (JavaScript)
- Python
- Java (Java 8 compatible) • C# (.NET Core)
- Golang
- C# / Powershell

AWS Lambda pricing

- You can find overall pricing information here: https://aws.amazon.com/lambda/pricing/
- Pay per calls:
- First 1,000,000 requests are free
- $0.20 per 1 million requests thereafter ($0.0000002 per request)
- Pay per duration: (in increment of 100ms)
- 400,000 GB-seconds of compute time per month if FREE
- == 400,000 seconds if function is 1GB RAM
- == 3,200,000 seconds if function is 128 MB RAM
- After that $1.00 for 600,000 GB-seconds

- After that $1.00 for 600,000 GB-seconds
- It is usually very cheap to run AWS Lambda which makes it very popular

AWS Lambda configuration

- Timeout: default 3 seconds, max of 300s (Note: new limit 15 minutes) - Environment variables
- Allocated memory (128M to 3G)
- Ability to deploy within a VPC + assign security groups
- IAM execution role must be attached to the Lambda function

AWS Lambda Synchronous Invocation

- Result of the function is returned right away
- Synchronous invocation happens when the function is invoked as follows:
  - User invoked:
    - ELB (Application Load Balancer)
    - API Gateway
    - CloudFront
    - S3 Batch
    - CLI
    - SDK
  - Service invoked:
    - Cognito
    - Step Functions
  - Invoked from other services:
    - Amazon Lex
    - Alexa
    - Kinesis Data Firehose
- Error handling must be done on the client side
- Invoke function from a command line:

```
aws lambda invoke --function-name hello-world --cli-binary-format raw-in-base64-out --payload '{}' --re
```

Lambda integration with ELB (ALB)

- Used for exposing a HTTP(S) endpoint
- The function must be registered in a target group
- HTTP request is converted by the ALB to JSON and the JSON is passed to the function an input
- The ALB converts the lambda response back to JSON
- Multi-header values: by default they are disabled. When enabled HTTP headers and query string parameters that are sent with multiple values are shown as arrays within the Lambda event and response object

Lambda@Edge

- Lambda function deployed synchronously from CDN using CloudFront
- Used for running global AWS Lambda function alongside CDN
- Can be used for request filtering before reaching the final application
- Can change CloudFront requests and responses:
  - After CloudFront receives a request from viewer (view request)
  - Before CloudFront forwards a request to origin (origin request)
  - After CloudFront receives the response from the origin (origin response)
  - Before CloudFront forwards the response to the viewer (viewer response)
- Use cases:
  - Website security and privacy
  - Dynamic web application at the Edge
  - Search engine optimization (SEO)
  - Intelligent route across origins and data centers
  - Bot mitigation at the Edge
  - Real-time image transformation
  - A/B testing
  - User authentication/authorization
  - User prioritization
  - User tracking and analytics

AWS Lambda Asynchronous Invocation

- Mainly invoked after a certain event
- The events are placed in an internal event queue
- Automatic retry: 3 retries: first one right after failure, second one waits 1 minute, third one after 2 minutes (exponential back-off)
- Lambda function should be idempotent - in case of retries the result should be the same
- Retries can cause duplicate log entries in CloudWatch logs
- DQL: in case of failure after retries the lambda function can send an event to SNS or SQS
- Services that invoke async lambda functions:

    - S3
    - SNS
    - CloudWatch Events / EventBridge
    - CodeCommit
    - CodePipeline
    - CloudWatch Logs
    - Amazon Simple Email Service
    - CloudFormation
    - Amazon Config
    - Amazon IoT / Iot Events
- Invoke a function asynchronously from CLI:

```
aws lambda invoke --function-name hello-world --cli-binary-format raw-in-base64-out --payload '{}' --re
```

- Invocation from CloudWatch Events/EventBridge:
    - Serverless CRON job
    - CodePipeline state change
- Invocation from S3 events:
    - Events: *S3:ObjectCreated*, *S3:ObjectRemoved*, etc.
    - Possible to de object filtering by name (*.jpg)
    - Possible use case: generate thumbnails for newly uploaded images
    - S3 event notification typically deliver events in seconds, sometimes it can take longer
    - If two writes are happening on a single, non-versioned object at the same time, it is possible that only one event gets delivered (solution: enable versioning on the S3 bucket)

AWS Lambda Event Source Mapping

- Records needs to be polled from the source
- Applies to SQS queues, Kinesis Data Streams, DynamoDB Streams
- Lambda function is invoked synchronously
- 2 categories of event source mapper:
    - Streams (Kinesis Data Streams, DynamoDB):
        - Event source creates an iterator for each shard so data can be processed in order
        - New items, from beginning or timestamp
        - Processed items are not removed from the stream
        - Low traffic: use batch window
        - High traffic: multiple batches can be processed in parallel. Up to 10 batch processes per shard, in-order processing for each partition key
        - Errors: the entire batch is reprocessed in case of an error until the function succeeds or the items in the batch expire
        - Configure event source mapping to handle errors:
            - discard old events
            - restrict number of retries
            - split the batch on error (mainly used in case of timeout issues)
            - discarded events can go to a *Destination*
    - Queues (SQS):
        - Event source mapping uses long polling to poll the queue
        - Batch size: 1-10 messages
        - Recommendation: set the queue visibility to *6x the timeout of the Lambda function*
        - DLQ: set-up the DLQ on the queue, not on the Lambda
        - Lambda supports FIFO queues, scaling up to the number of active message groups
        - Lambda scales up to process a standard queue as fast as possible
        - In case of error batches are returned as individual items => they might be process as part of different batch
        - Idempotent processing in case of duplicates
        - Lambda deletes the processed items from the queue
- Scaling:
    - Data streams:
        - One Lambda invocation per each shard
        - Parallelization: up to 10 batches per shard in the same time
    - Queues:
        - Lambda adds 60 more instances per minute to scale up
        - Up to 1000 batches of messages simultaneously
    - FIFO queues:
        - Messages within the same group are processed in order
        - Lambda scales to the number of the message groups

AWS Lambda Destinations

- Configure to send result of a function to a destination
- Async invocations destinations for successful and failed events can be sent to:
    - SQS
    - SNS
    - Other Lambda
    - EventBridge bus

- Use destination instead of DLQ (even if both can be used at the same time)
- In case of event source mapping: for discarded batches, event baches can be sent to SQS and SNS

IAM Roles for Lambda

- Grants permission to Lambda to access other AWS services (obviously :)
- When we use an event source mapping to invoke a function, Lambda uses the execution role to read event data.
- Best practice: one execution role per function
- Lambda resource based policy: give other accounts of AWS services to use Lambda resources
- IAM principal can access Lambda:
  - if the IAM policy attached to the principal authorizes it
  - if the resource based policy authorizes

Environment Variables

- Env. variable = key / value pair
- Adjust the system behavior
- Env. variables are available to the function code
- Lambda has its own env. variables
- Env. variables can be encrypted (KMS), useful for secrets

Logging / Monitoring

- Integration with CloudWatch logs if the lambda has the IAM policy
- CloudWatch Metrics:
  - Information about invocation, duration, concurrent execution
  - Error count, success rate, throttles
  - Failures
  - Iterators for streams
- Tracing with X-Ray using X-Ray SDK in code
- Env. variables for X-Ray:
  - **_X_AMZN_TRACE_ID**: contains the tracing header, which includes the sampling decision, trace ID, and parent segment ID. If Lambda receives a tracing header when our function is invoked, that header will be used to populate the *_X_AMZN_TRACE_ID* environment variable. If a tracing header was not received, Lambda will generate one
  - **AWS_XRAY_CONTEXT_MISSING**: the X-Ray SDK uses this variable to determine its behavior in the event that our function tries to record X-Ray data, but a tracing header is not available. Lambda sets this value to *LOG_ERROR* by default
  - **AWS_XRAY_DAEMON_ADDRESS**: exposes the X-Ray daemon's address in the following format: *IP_ADDRESS:PORT*

Networking

- By default: Lambda is launched outside of user VPC, so it can not access resources from inside of a private VPC
- Lambda can be deployed in a VPC:
  - Lambda will create a ENI (Elastic Network Interface)
  - Needed *AWSLambdaVPCAccessExecutionRole*
- By default: Lambda in an user defined VPC can not access internet (even if the VPC is public). Solution: NAT Gateway / Instance, Lambda deployed in a private subnet
- DynamoDB can be accessed via VPC endpoints (or via public internet)
- CloudWatch logs work in private subnet without NAT

Configuration / Performance

- RAM:
  - Can be scaled from 128MB to 3008MB in 64MB increments
  - The more RAM we add, the more vCPU credit we get (vCPU can not be configured separately)
  - Ath 1792MB a function has the equivalent of a full vCPU
  - After this value we get more than one vCPU, multi-threaded code can leverage this
- Timeout: by default 3 seconds, max 900 seconds (15 minutes)
- Execution context:
  - Temporary runtime env. that initializes any external dependency
  - Great for DB connections, HTTP clients
  - Is maintained for some time in anticipation of another Lambda invocation
  - Does include the **/tmp** directory (512MB disk space)
  - Recommended: initialize DB connections outside of function handler, it can be reused for next execution
  - For permanent persistence use S3 (not /tmp)!
- Concurrency:
  - Limit: up to 1000 concurrent executions per account
  - Reserved concurrency creates a pool of requests that can only be used by its function, and

also prevents its function from using unreserved concurrency
    - Invocation over concurrency limit will trigger throttle
    - Throttle behavior:
        - Sync invocation => return ThrottleError - 429
        - Async invocation => automatic retry and DLQ
    - If higher limit is required per account, open a support ticket
    - In case of reserved concurrency not set, one function could throttle other function, max concurrency limit is applied globally to a single account
    - In case of async invocation Lambda will attempt to run the function again for 6 hours. The retry interval increases exponentially
    - Cold start: in case of a new function instance, first request can take longer because of initialization
    - Provisioned concurrency initializes a requested number of execution environments so that they are prepared to respond to your function's invocations => no cold start!
    - Application Auto Scaling can manage concurrency (schedule or target utilization)

Lambda concurrency

- Reserved: Reserved concurrency creates a pool of requests that can only be used by its function, and also prevents its function from using unreserved concurrency
- Provisioned: Provisioned concurrency initializes a requested number of execution environments so that they are prepared to respond to your function's invocations
- Estimate reserved concurrency: **the average number of requests per second multiplied by the average duration in seconds**. For example, if a Lambda function takes an average 500 ms to run with 100 requests per second, the concurrency is 50

External Dependencies

- Dependencies, packages need to be installed together with Lambda code (zip package)
- Zip can be uploaded to Lambda if it has less than 50MB. Otherwise use S3
- Native libraries need to be compiled on Amazon Linux
- AWS SDK comes together with every Lambda function

CloudFormation

- Code can be defined in-line (for very simple functions)
- Inline functions can not include dependencies
- For more complex functions:
    - We can use the **Code.ZipFile** property
    - We can use S3 where we should refer to the location of the zip file in a bucket
        - Note: if we update the code in S3 but don't update the S3Bucket, S3Key or S3ObjectVersion in the template, CloudFormation wont update the function, **versioning is import**

AWS Lambda Layers

- Custom Runtimes for Lambda, example:
    - C++ aws-lambda-cpp
    - Rust aws-lambda-rust
- Externalize dependencies to re-use them in case of bigger dependency packages
- Another function could reuse layers

AWS Lambda Versions and Aliases

- When we work on a Lambda function, we work on *$LATEST* version
- When we are ready to publish, we create a new version
- Versions are immutable (can not change to code, env. variables of anything after a version is published)
- Versions have increasing number
- Each version is independent and it gets its own ARN
- Each version of the lambda can be accessed
- Alias:
    - Pointer to a Lambda function version
    - We can defined dev, test, prod aliases which can point to different versions
    - Aliases are mutable
    - Aliases enable Blue/Green deployment by assigning weights to Lambda functions
    - Aliases enable stable configurations of our event triggers/destinations
    - Aliases have their own ARN
    - **Aliases can not reference other aliases!**

CodeDeploy

- Can help automate traffic shift for Lambda aliases
- Feature is integrated within SAM framework
- Traffic shifting methodologies:

- Linear growth: traffic grows every N minutes until 100%
- Canary: try X percent then 100%
- AllAtOnce: immediate (quickest, most dangerous)
- Rollback: can create pre and post traffic hooks to check the health of the Lambda function

AWS Lambda Limits

- Execution:
  - Memory: 128MB - 3008MB (64MB increments)
  - Max execution time: 900 seconds (15 minutes)
  - Env. variables: 4KB
  - Disk space (/tmp): 512MB
  - Concurrent executions 1000 per account
- Deployments:
  - Compressed: 50MB
  - Uncompressed (code + dependencies): 250MB
  - Can use /tmp to load other files at startup

Best practices

- Perform heavy-duty work outside of the function handler
- Use env. variables for anything that changes over time
- Minimize deployment package size to its runtime necessities
- **Avoid recursive code, never have a Lambda function call itself!**