AWS Lambda

05:31 PM

AWS Lambda

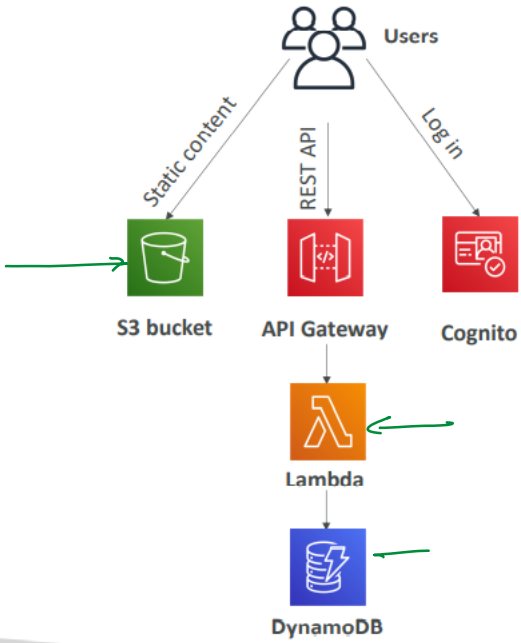
It's a serverless world

What's serverless?

- Serverless is a new paradigm in which the developers don't have to manage servers anymore…
- They just deploy code
- They just deploy… functions !
- Initially… Serverless == FaaS (Function as a Service)
- Serverless was pioneered by AWS Lambda but now also includes anything that's managed: "databases, messaging, storage, etc."
- **Serverless does not mean there are no servers…**
  it means you just don't manage / provision / see them

Serverless in AWS

- AWS Lambda
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS & SQS
- AWS Kinesis Data Firehose
- Aurora Serverless
- Step Functions
- Fargate



Why AWS Lambda

8 RAM, 12 vcpc,

- Virtual Servers in the Cloud
- Limited by RAM and CPU

Amazon EC2

- Continuously running
- Scaling means intervention to add / remove servers

Amazon Lambda

- Virtual **functions** – no servers to manage!
- Limited by time - **short executions**
- Run **on-demand**
- **Scaling is automated!**

Benefits of AWS Lambda

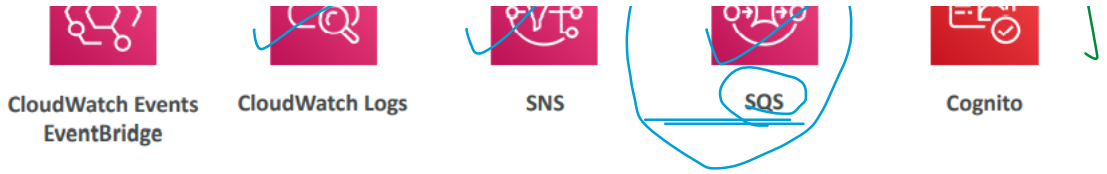
- Easy Pricing:
  - Pay per request and compute time
  - Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
- Integrated with the whole AWS suite of services
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per functions (up to 10GB of RAM!)
- Increasing RAM will also improve CPU and network!

AWS Lambda language support

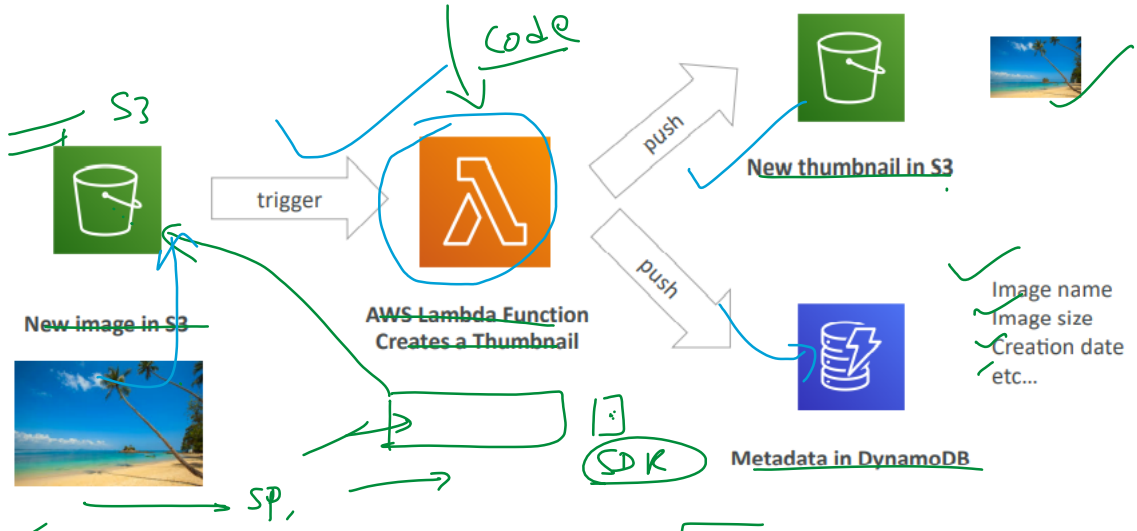
- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell
- Ruby
- Custom Runtime API (community supported, example Rust)
- Lambda Container Image
  - The container image must implement the Lambda Runtime API
  - ECS / Fargate is preferred for running arbitrary Docker images

AWS Lambda Integrations Main ones

API Gateway | Kinesis | DynamoDB | S3 | CloudFront



Example: Serverless Thumbnail creation



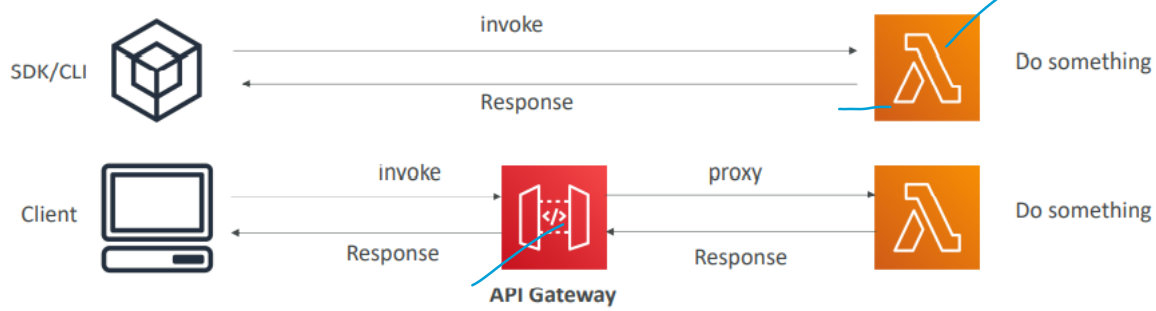
Example: Serverless CRON Job

AWS Lambda Pricing: example

- 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 1 ms)
  - 400,000 GB-seconds of compute time per month for 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
- It is usually very cheap to run AWS Lambda so it's very popular

Lambda – Synchronous Invocations

- Synchronous: CLI, SDK, API Gateway, Application Load Balancer
  - Results is returned right away
  - Error handling must happen client side (retries, exponential backoff, etc…)



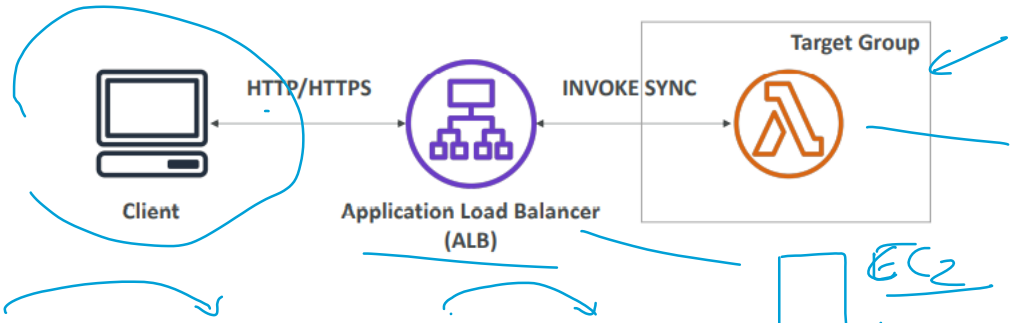
Lambda - Synchronous Invocations - Services

- User Invoked:
  - Elastic Load Balancing (Application Load Balancer)
  - Amazon API Gateway
  - Amazon CloudFront (Lambda@Edge)
  - Amazon S3 Batch
- Service Invoked:
  - Amazon Cognito
  - AWS Step Functions
- Other Services:
  - Amazon Lex
  - Amazon Alexa
  - Amazon Kinesis Data Firehose

Lambda Integration with ALB

Proxy http JSON

- To expose a Lambda function as an HTTP(S) endpoint…
- You can use the **Application Load Balancer** (or an API Gateway)
- The Lambda function must be registered in a **target group**



EC2

Lambda

ALB to Lambda: HTTP to JSON

Request Payload for Lambda Function

```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:12
            49e9d65c45c6791a"
        }
    },
    "httpMethod": "GET",
    "path": "/lambda",
    "queryStringParameters": {
        "query": "1234ABCD"
    },
```

ELB information

HTTP Method & Path

Query String Parameters as Key/Value pairs

```
"headers": {
    "connection": "keep-alive",
    "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
    Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http",
},
"body": "",
"isBase64Encoded": false
}
```


Headers as Key/Value pairs

Body (for POST, PUT...) & isBase64Encoded

Lambda to ALB conversions: JSON to HTTP

Response from the Lambda Function


```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "headers": {
        "Content-Type": "text/html; charset=utf-8"
    },
    "body": "<h1>Hello world!</h1>",
    "isBase64Encoded": false
}
```

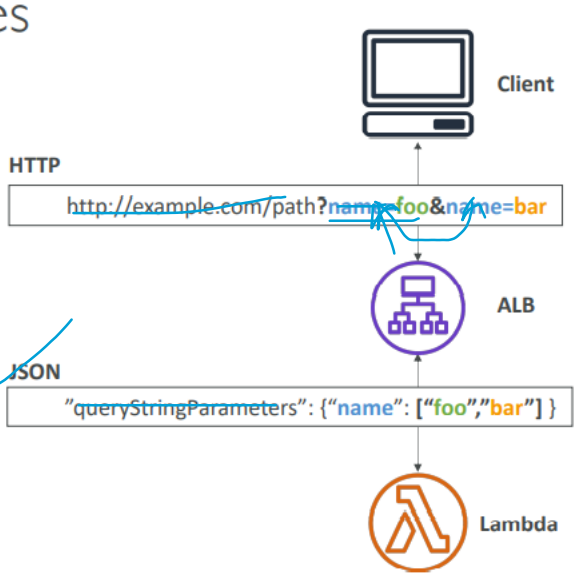

Status Code & Description

Headers as Key/Value pairs

Body & isBase64Encoded

ALB Multi-Header Values

- ALB can support multi header values (ALB setting)
- When you enable multi-value headers, HTTP headers and query string parameters that are sent with **multiple values** are shown as arrays within the AWS Lambda event and response objects.



Lambda - Asynchronous Invocations - Services

- Amazon Simple Storage Service (S3)
- Amazon Simple Notification Service (SNS)
- Amazon CloudWatch Events / EventBridge
- AWS CodeCommit (CodeCommit Trigger: new branch, new tag, new push)
- AWS CodePipeline (invoke a Lambda function during the pipeline, Lambda must callback)

----- other -----

- Amazon CloudWatch Logs (log processing)
- Amazon Simple Email Service
- AWS CloudFormation
- AWS Config
- AWS IoT
- AWS IoT Events

Lambda Execution Role (IAM Role)

Lambda Execution Role (IAM Role)

- Grants the Lambda function permissions to AWS services / resources
- Sample managed policies for Lambda:
  - AWSLambdaBasicExecutionRole – Upload logs to CloudWatch.
  - AWSLambdaKinesisExecutionRole – Read from Kinesis
  - AWSLambdaDynamoDBExecutionRole – Read from DynamoDB Streams
  - AWSLambdaSQSQueueExecutionRole – Read from SQS
  - AWSLambdaVPCAccessExecutionRole – Deploy Lambda function in VPC
  - AWSXRayDaemonWriteAccess – Upload trace data to X-Ray.
- **When you use an event source mapping to invoke your function, Lambda uses the execution role to read event data.**
- Best practice: create one Lambda Execution Role per function

S3

Lambda Resource Based Policies

- Use resource-based policies to give other accounts and AWS services permission to use your Lambda resources
- Similar to S3 bucket policies for S3 bucket
- An IAM principal can access Lambda:
  - if the IAM policy attached to the principal authorizes it (e.g. user access)
  - OR if the resource-based policy authorizes (e.g. service access)
- When an AWS service like Amazon S3 calls your Lambda function, the resource-based policy gives it access.

Lambda Environment Variables

- Environment variable = key / value pair in "String" form
- Adjust the function behavior without updating code
- The environment variables are available to your code
- Lambda Service adds its own system environment variables as well
- Helpful to store secrets (encrypted by KMS)
- Secrets can be encrypted by the Lambda service key, or your own CMK

prod

stage

Lambda Logging & Monitoring

[AWS/]

- CloudWatch Logs:
  - AWS Lambda execution logs are stored in AWS CloudWatch Logs
  - **Make sure your AWS Lambda function has an execution role with an IAM policy that authorizes writes to CloudWatch Logs**

basic

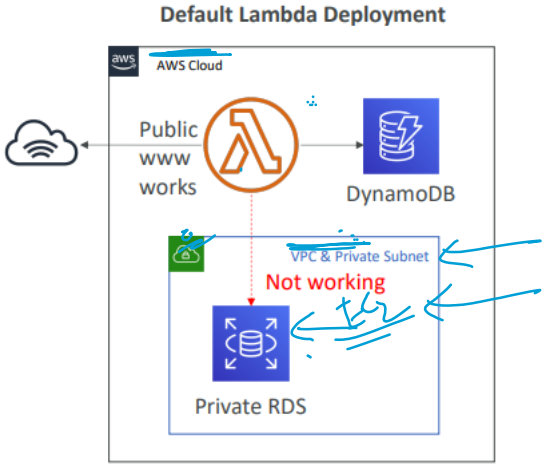
- CloudWatch Metrics:
  - AWS Lambda metrics are displayed in AWS CloudWatch Metrics
  - Invocations, Durations, Concurrent Executions
  - Error count, Success Rates, Throttles

- Error count, Success Rates, Throttles
- Async Delivery Failures
- Iterator Age (Kinesis & DynamoDB Streams)

Lambda by default

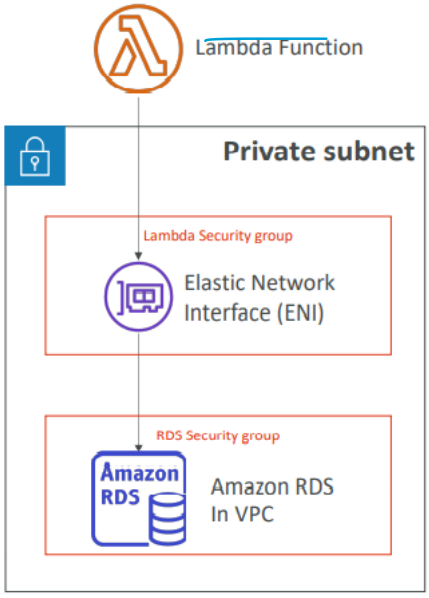
VPC

- By default, your Lambda function is launched outside your own VPC (in an AWS-owned VPC)
- Therefore it cannot access resources in your VPC (RDS, ElastiCache, internal ELB…)



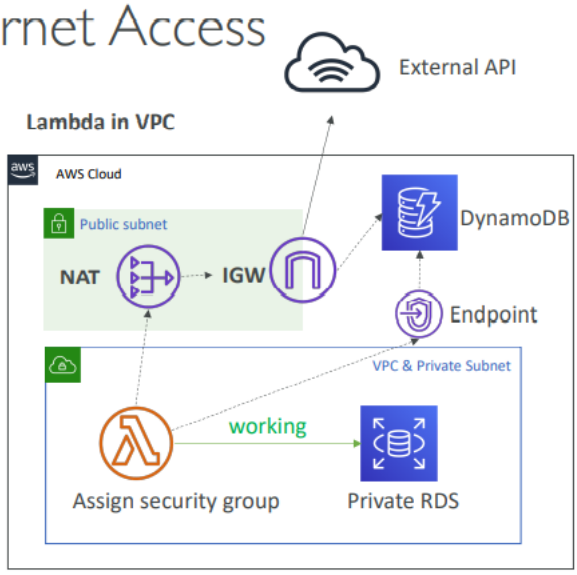
Lambda in VPC

- You must define the VPC ID, the Subnets and the Security Groups
- Lambda will create an ENI (Elastic Network Interface) in your subnets
- AWSLambdaVPCAccessExecutionRole



Lambda in VPC – Internet Access

- A Lambda function in your VPC does not have internet access
- **Deploying a Lambda function in a public subnet does not give it internet access or a public IP**
- Deploying a Lambda function in a private subnet gives it internet access if you have a **NAT Gateway / Instance**
- You can use **VPC endpoints** to privately access AWS services without a NAT



Note: Lambda - CloudWatch Logs works even without endpoint or NAT Gateway

without endpoint or NAT Gateway

Lambda Function Configuration

- RAM:
  - From 128MB to 10GB in 1MB 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 multi-threading in your code to benefit from it (up to 6 vCPU)
- If your application is CPU-bound (computation heavy), increase RAM
- Timeout: default 3 seconds, maximum is 900 seconds (15 minutes)

Lambda Execution Context

- The execution context is a temporary runtime environment that initializes any external dependencies of your lambda code
- Great for database connections, HTTP clients, SDK clients…
- The execution context is maintained for some time in anticipation of another Lambda function invocation
- The next function invocation can "re-use" the context to execution time and save time in initializing connections objects
- The execution context includes the */tmp* directory

Initialize outside the handler

BAD!

```
import os

def get_user_handler(event, context):

    DB_URL = os.getenv("DB_URL")
    db_client = db.connect(DB_URL)
    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established
At every function invocation

GOOD!

```
import os

DB_URL = os.getenv("DB_URL")
db_client = db.connect(DB_URL)

def get_user_handler(event, context):

    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established once
And re-used across invocations

Lambda Functions /tmp space

- If your Lambda function needs to download a big file to work…
- If your Lambda function needs disk space to perform operations…
- You can use the /tmp directory
- Max size is 512MB

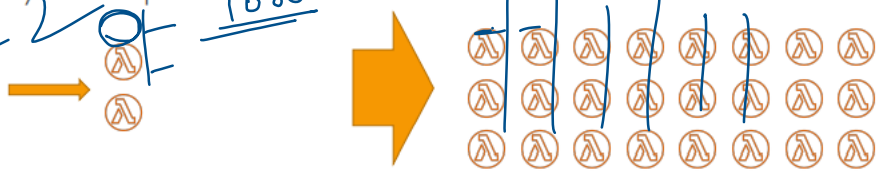
- Max size is 512MB
- The directory content remains when the execution context is frozen, providing transient cache that can be used for multiple invocations (helpful to checkpoint your work)
- For permanent persistence of object (non temporary), use S3

Lambda Concurrency and Throttling

throttling

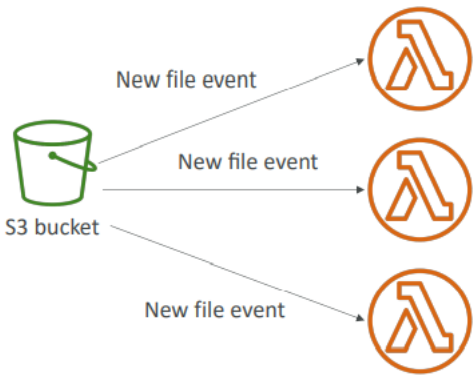
- Concurrency limit: up to 1000 concurrent executions

AZB: 2 of 1000



- Can set a "reserved concurrency" at the function level (=limit)
- Each invocation over the concurrency limit will trigger a "Throttle"
- Throttle behavior:
  - If synchronous invocation => return ThrottleError - 429
  - If asynchronous invocation => retry automatically and then go to DLQ
- If you need a higher limit, open a support ticket

Concurrency and Asynchronous Invocations



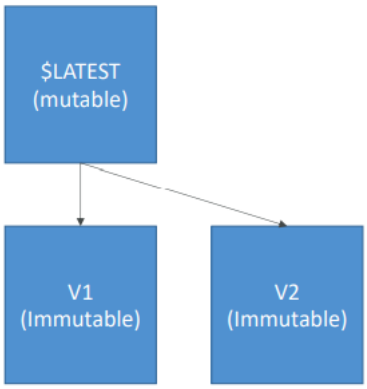
- If the function doesn't have enough concurrency available to process all events, additional requests are throttled.
- For throttling errors (429) and system errors (500-series), Lambda returns the event to the queue and attempts to run the function again for up to 6 hours.
- The retry interval increases exponentially from 1 second after the first attempt to a maximum of 5 minutes.

Lambda Function Dependencies

- If your Lambda function depends on external libraries: for example AWS X-Ray SDK, Database Clients, etc…
- You need to install the packages alongside your code and zip it together
  - For Node.js, use npm & "node_modules" directory
  - For Python, use pip --target options
  - For Java, include the relevant .jar files
- Upload the zip straight to Lambda if less than 50MB, else to S3 first
- Native libraries work: they need to be compiled on Amazon Linux
- AWS SDK comes by default with every Lambda function

AWS Lambda Versions

- When you work on a Lambda function, we work on **$LATEST**
- When we're ready to publish a Lambda function, we create a version
- Versions are immutable
- Versions have increasing version numbers
- Versions get their own ARN (Amazon Resource Name)
- Version = code + configuration (nothing can be changed - immutable)
- Each version of the lambda function can be accessed



AWS Lambda Limits to Know - per region

- Execution:
  - Memory allocation: 128 MB – 10GB (1 MB increments)
  - Maximum execution time: 900 seconds (15 minutes)
  - Environment variables (4 KB)
  - Disk capacity in the "function container" (in /tmp): 512 MB
  - Concurrency executions: 1000 (can be increased)
- Deployment:
  - Lambda function deployment size (compressed .zip): 50 MB
  - Size of uncompressed deployment (code + dependencies): 250 MB
  - Can use the /tmp directory to load other files at startup
  - Size of environment variables: 4 KB

AWS Lambda Best Practices

- Perform heavy-duty work outside of your function handler
  - Connect to databases outside of your function handler
  - Initialize the AWS SDK outside of your function handler
  - Pull in dependencies or datasets outside of your function handler
- Use environment variables for:
  - Database Connection Strings, S3 bucket, etc… don't put these values in your code
  - Passwords, sensitive values… they can be encrypted using KMS
- Minimize your deployment package size to its runtime necessities.
  - Break down the function if need be
  - Remember the AWS Lambda limits
  - Use Layers where necessary
- Avoid using recursive code, never have a Lambda function call itself