Introduction to AWS LAMBDA

Presentation by:
Ratna Kameswari Prabhala (1750211)
Spoorthy Balasubrahmanya (1740013)
Tejaswi Gorrepati (1750154)

AGENDA

- **Introduction to AWS Lambda**
- **History**
- **Key Features**
- **Use Cases**
- **Advantages**
- **Disadvantages**
- **Usability**
- **Cost Discussion**
- **Possible alternatives**

INTRODUCTION

- AWS Lambda is Function As A Service(FaaS).
- It is an event-driven, cloud computing platform service.
- Microservice without servers.
- Allows developers to provision resources for a programming function on a pay per use basis.
- Runs your function code without you managing or scaling servers.
- Lambda function can be written in Java, Python , Node.js and C#.

Technology Invention

- The first 'pay as you go' code execution platform was Zimki in 2006.
- Iron.io has coined the term serverless in 2011 and 2012.
- AWS Lambda, introduced by Amazon in November 2014, was the first major provider considered to have a serverless offering.
- IBM announced OpenWhisk as an open source serverless platform.
- Microsoft followed up in 2016 by announcing Azure Functions.

Motivation

- Resources were under-utilized and left idle for longer periods.
- No infrastructure to manage.
- Resource centric service.
- Focus on program execution.

Key Features

- **Build Custom Back-end Service :**

Create new back-end services for your applications that are triggered on-demand

- **Run Code at Edge Locations :**

Run code without provisioning or managing server with lowest network latency.

- **Automatic Scaling :**

Automatically scales to support the rate of incoming requests without requiring you to configure anything.

- **Integrated Security Model :**

Allows your code to securely access other AWS services through its built-in AWS SDK and integration with IAM.

- **Flexible Resource Model :**

Choose the amount of memory you want to allocate to your functions and AWS Lambda allocates proportional CPU power, network bandwidth, and disk I/O.

Key Features(1)

- **Pay Per Use:**

Pay only for the requests served and the compute time required to run the code.

- **Build your own code:**

We can use any third party library, even native ones.

- **Completely Automated Administration:**

Manages all the infrastructure to run your code on highly available, fault-tolerant infrastructure

- **Built-in Fault Tolerance:**

Protect your code against individual machine or data center facility failures.

Use Cases

- **Event Based Triggers:**
  - Real Time Stream processing (IOT applications)
  - Processing Images uploaded to S3
  - Chat Bot applications , e.g. user input(search for weather) is a trigger.
- **Scheduled Events:**
  - Hourly/daily large scale batch processing (Rebuilding indexes for search engine)
  - Automated back-ups.
- **Mobile Applications Back End :**
  - Since mobile apps have resource constraints, expensive processing can be offloaded to AWS Lambda
- **Business Use Cases:**
  - Netflix
  - Zillow (AWS Lambda and AWS Kinesis for calculating mobile metrics)
  - Amazon Alexa

Advantages

- **Cost**
  - Sub second billing -
  - For e.g. you are charged for every 100ms your function executes and number of times it executes.
  - Better than renting EC2 machines that are billed on hourly basis.
  - First 1 million requests per month are free. Subsequent requests are billed at 0.2$ per million requests
  - No need to maintain expensive servers - Handled by AWS platform
  - Avoid under-utilization of resources
- **Auto Scaling**
  - Easily scale up and down function instances based on demand. Handled by AWS platform
  - Compare and contrast with EC2 where one has to set up Auto Scaling Groups and configure Load Balancer.
- **Ease of Development and Deployment**
  - Reduced software complexity
  - Easy to deploy and iterate on functions using command line and UI

Disadvantages

- **Resource Limits:**
  - Maximum Execution time cannot exceed ***300*** seconds
  - Concurrent Executions limited to ***600*** per region
  - **Deployment Limits:**
  - Size of Lambda Function Deployment package cannot exceed ***50*** MB
  - Total size of all the deployment packages that can be uploaded per region is 75GB
- **Cold Start:**
  - Delay in executing first request since Lambda has to provision a new instance of the function for the first time.
- **Non persistent:**
  - Lambda functions are short lived. Any state has to be stored in S3 or AWS relational databases for extra cost.
- **Limited Monitoring Support**
  - Cloud Watch is the only viable option currently to monitor and debug AWS lambda function.

Usability

- **Ease of Use:**
  - Configuring different event sources has a bit of learning curve.
- **Language Support:**
  - Good coverage: Java, Node.js, Python, .NET
- **Documentation**
  - Documentation is limited for some topics including mobile back end applications and configuring event sources

Cost Discussion (1)

- **Pricing Example:**

Memory = 512 MB
No of Executions = 3M
Time per Execution = 1 sec
Total charges (Monthly) = Compute charges + Request charges

Compute charges (Monthly):
compute price is $0.00001667 per GB-s and the free tier provides 400,000 GB-s.
Total compute (seconds) = 3M * (1s) = 3,000,000 seconds
Total compute (GB-s) = 3,000,000 * 512MB/1024 = 1,500,000 GB-s
Total compute – Free tier compute = Monthly billable compute GB- s
1,500,000 GB-s – 400,000 free tier GB-s = 1,100,000 GB-s

Compute charges = 1,100,000 * $0.00001667 = **$18.34**

Cost Discussion (2)

- **Request charges (Monthly) :**

1. Price =$0.20 per 1 million requests and the free tier provides 1M requests per month.

- Total requests – Free tier requests = Monthly billable requests
- 3M requests – 1M free tier requests = 2M Monthly billable requests
- Monthly request charges = 2M * $0.2/M = $0.40

Total charges = Compute charges + Request charges = $18.34 + $0.40 = $18.74 per month

Possible Alternatives

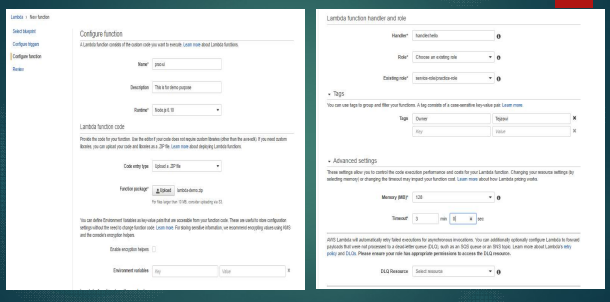
- Azure Functions (General Availability)
- Google Cloud Function (Beta Release)
- Open Source Alternatives:
  - IBM OpenWhisk
  - Iron.Io (Alpha 2)

Summary

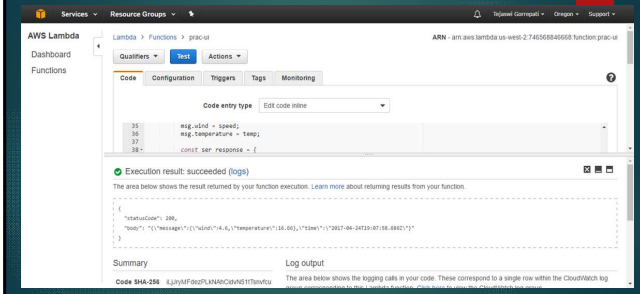
- AWS Lambda is Function As a Service(FaaS)
- Sub second costs for running functions
- On-Demand Execution and auto scaling
- Faster development and deployment
- Not suitable for long running functions

DEMO

Creating and configuring a lambda function in GUI:



Invoking the lambda function in GUI:



Creating a lambda function using AWS CLI:

$ **aws lambda create-function** --function-name prac-cli --runtime nodejs4.3 --role arn:aws:iam::123456:role/service-role/practice-role --handler handler.hello --zip-file fileb://lambda-demo.zip (*Replace 123456 with your AWS Account ID*)

```
C:\Users\Tejaswi\Documents\Spring_2017\CloudComputing\workspace\prac-ui>aws lambda create-function --function-name prac-cli --runtime nodejs4.3 --role arn:aws:iam::[redacted]role/service-role/practice-role --handler handler.hello --zip-file fileb://lambda-demo.zip
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "Ij3ryM0dezPLkNAhCidvNS1tTsnvfcutYpRg9sGCew=",
    "FunctionName": "prac-cli",
    "CodeSize": 1556649,
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:[redacted]function:prac-cli",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::[redacted]role/service-role/practice-role",
    "Timeout": 3,
    "LastModified": "2017-04-24T19:23:40.193+0000",
    "Handler": "handler.hello",
    "Runtime": "nodejs4.3",
    "Description": ""
}
```

Updating & Invoking the lambda function using AWS CLI:

$ **aws lambda update-function**-configuration --function-name prac-cli --timeout 30

```
C:\Users\Tejaswi\Documents\Spring_2017\CloudComputing\workspace\prac-ui>aws lambda update-function-configuration --function-name prac-cli --timeout 30
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "Ij3ryM0dezPLkNAhCidvNS1tTsnvfcutYpRg9sGCew=",
    "FunctionName": "prac-cli",
    "CodeSize": 1556649,
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:[redacted]function:prac-cli",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::[redacted]role/service-role/practice-role",
    "Timeout": 30,
    "LastModified": "2017-04-26T03:29:07.323+0000",
    "Handler": "handler.hello",
    "Runtime": "nodejs4.3",
    "Description": ""
}

C:\Users\Tejaswi\Documents\Spring_2017\CloudComputing\workspace\prac-ui>
```

$ **aws lambda invoke** --function-name prac-cli --payload file://input.txt lambda-output.txt

```
C:\Users\Tejaswi\Documents\Spring_2017\CloudComputing\workspace\prac-ui>aws lambda invoke --function-name prac-cli --payload file://input.txt output.txt
{
    "StatusCode": 200
}

C:\Users\Tejaswi\Documents\Spring_2017\CloudComputing\workspace\prac-ui>
```