

Case Study : Serverless Computing AWS Lambda

CSL7091 - Cloud Computing and Virtualization

Dr. Sumit Kalra

Shashwat Kathuria - B17CS050

Abhinav Pandey - B17CS001

Serverless Computing

Introduction

- Cloud Computing Model
- Cloud provider runs the server and dynamically manages the allocation of machine resources
- Pricing is based on the actual amount of resources consumed by the application, rather than on pre-purchased units of capacity
- A form of utility computing

Benefits

- Easy to Deploy
- Low Cost
- More time available for coding, UI, UX, etc
- Better scalability
- Improved latency
- Improved flexibility

AWS Lambda

Introduction

- Event - driven, serverless computing platform provided by Amazon
- A computing service that runs code in response to events and automatically manages the computing resources required by that code

Features

- No servers to manage - only write and upload code
- Continuous Scaling - scaling precisely with the size of the workload
- Subsecond metering - pay only for compute time
- Consistent performance - optimize code execution time

Our Case Studies

Used Lambda to **parse a response from an API Gateway** as an example

Very easy to **set up, scale** and **manage resources** with **only compute time cost, no servers to maintain**, everything done on cloud

The screenshot displays the AWS Lambda console interface. On the left, the 'https-example' function configuration is visible, showing the code entry type as 'Edit code inline' and the runtime as 'Node.js 12.x'. The code is a JavaScript function that parses an HTTP response. The right pane shows the 'Execution result: succeeded (logs)' for the function. The response body is a JSON object: `{"userId": 1, "id": 3, "title": "fugiat veniam minus", "completed": false}`. The summary table indicates the function executed successfully with a duration of 1191.82 ms and a billed duration of 1200 ms. The log output shows the function logs for the execution.

https-example Throttle Qualifiers Actions

Code entry type: Edit code inline Runtime: Node.js 12.x

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{"userId": 1, "id": 3, "title": "fugiat veniam minus", "completed": false}
```

Summary

Code SHA-256	Request ID
7gJLDM8w6SvykOLwYVeWwAsDrPWuPsZ9/H0BP9QVTKs=	5b22969f-c768-4b17-b088-ec932900dd28
Duration	Billed duration
1191.82 ms	1200 ms
Resources configured	Max memory used
128 MB	76 MB Init Duration: 137.75 ms

Log output

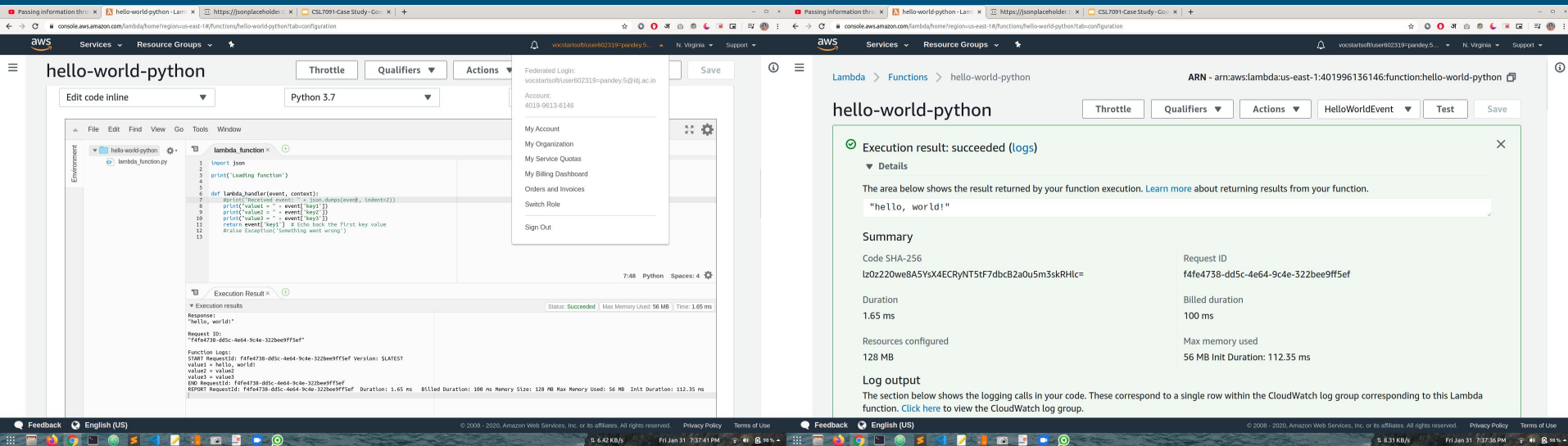
Function Logs

```
START RequestID: 5b22969f-c768-4b17-b088-ec932900dd28 Version: SLATEST
2020-01-31T14:04:02.866Z 5b22969f-c768-4b17-b088-ec932900dd28 INFO Status: 200/200-31-33174:04:02.866Z 5b22969f-c768-4b17-b088-ec932900dd28 INFO Headers: [7
REPORT RequestID: 5b22969f-c768-4b17-b088-ec932900dd28 Duration: 137.75 ms Billed Duration: 1200 ms Memory Size: 128 MB Max Memory Used: 76 MB Init Duration: 137.75 ms
```

Our Case Studies

Used Lambda to **print a statement (or any metric as an example after processing event data)**

Very easy to **set up, scale** and **manage resources** with **only compute time cost, no servers to maintain**, everything done on cloud



The image displays two side-by-side screenshots of the AWS Lambda console interface, illustrating the setup and execution of a simple Python function named 'hello-world-python'.

Left Screenshot (Code Editor):

- The function is named 'hello-world-python' and is configured for Python 3.7.
- The code in the editor is as follows:

```
1 import json
2
3 print('loading function')
4
5
6 def lambda_handler(event, context):
7     print('Received event: {}'.format(json.dumps(event, indent=2)))
8     print('value1 = {}'.format(event['key1']))
9     print('value2 = {}'.format(event['key2']))
10    print('value3 = {}'.format(event['key3']))
11    return event['key1'] # Echo back the first key value
12    #raise Exception('Something went wrong')
13
```
- The 'Execution Result' tab shows the function executed successfully, returning 'hello, world!'.

Right Screenshot (Execution Details):

- The function is named 'hello-world-python' and is configured for Python 3.7.
- The 'Execution result: succeeded (logs)' section shows the function executed successfully, returning 'hello, world!'.
- The 'Summary' section provides details about the execution:
 - Code SHA-256: lz0z220e8A5yS4ECRyNT5f7dcb2a0u5m3kRHlc=
 - Request ID: f4fe4738-dd5c-4e64-9c4e-322bee9ff5ef
 - Duration: 1.65 ms
 - Billed duration: 100 ms
 - Resources configured: 128 MB
 - Max memory used: 56 MB
 - Init Duration: 112.35 ms
- The 'Log output' section shows the logging calls in the code, corresponding to the execution logs.

Cons of Serverless Computing

→ Vendor Lock-In

- ◆ Have to **conform to the rules** of the vendor
- ◆ Currently **less freedom of languages** to choose from

→ Learning curve

- ◆ Learning curve for FaaS (Function-as-a-Service) tools is **pretty steep**
- ◆ For easy migration, have to **split your monolith into microservices**, a **complicated** task

→ Unsuitable for long term tasks

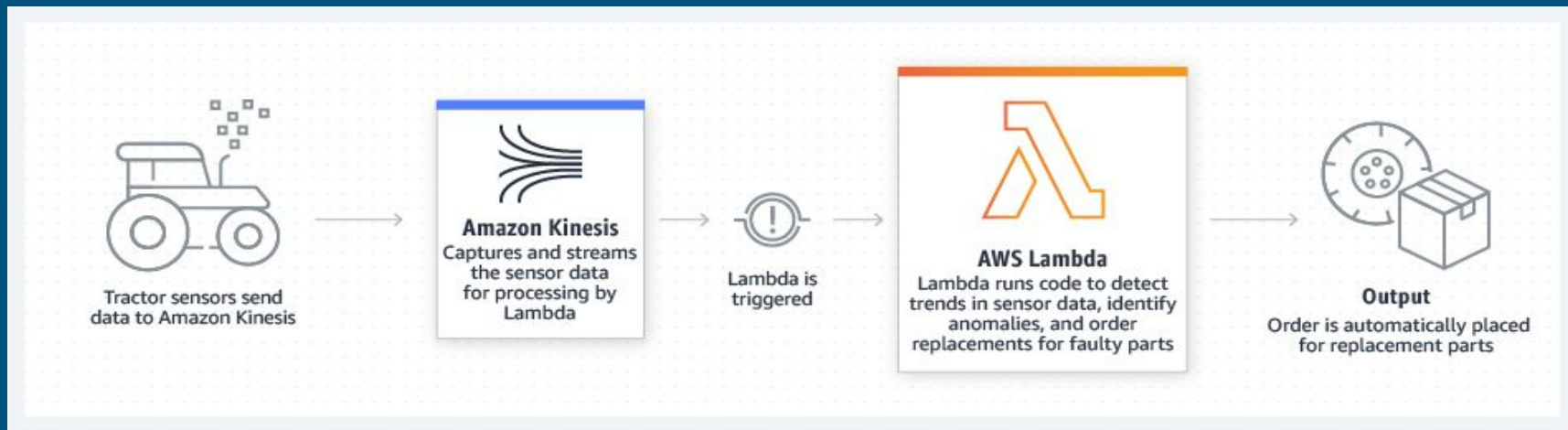
- ◆ Lambda gives you five minutes to execute a task and if it takes longer, you'll have to **call another function**
- ◆ Long duration operations such as uploading video files would require **additional FaaS functions** or be better with **“server-ful” architecture**

Real World Applications



An application of **Real Time File Processing**
Lambda **triggers on photo upload** and **runs resizing code**

Real World Applications



An application of **IoT Backend**

Lambda **triggers on sensor data collection** and runs **sensor maintenance code**

Real World Case Study

The Coca-Cola Company

Context & Problem

- Wanted to update customer bonus points of their vending machines.
- The issue was that there needed to be introduced a delay for the same which built up costs due to delay of 90 seconds that was required for updating the records.

Solution

- Used AWS Lambda to update the bonus points and pay only for the consume time.
- Introduced AWS Step State of 90 second delay which was not considered in costs.
- Efficiently and cost effectively solved the solution without managing servers.

Real World Case Study



Context & Problem

- Wanted to scale their new connected IoT robot vacuum cleaners cloud application.
- The issue was that they wanted an easy solution and did not want to go up with subscription services and the headache of scaling and maintaining.

Solution

- Used AWS Lambda and AWS IoT platform to update their IoT backend and provide connectivity layer between the robots and the iRobot cloud platform.
- Were also able to compute the metrics of their devices.
- Scaled their applications in a hassle-free way by using serverless computing.

Real World Case Study



Benchling

Context & Problem

- Wanted to scale their applications and fasten their genome search queries (biology related).
- The issue was that their servers were taking a lot of time, on the other hand they were also expecting more and more users so they needed to scale and the servers were being used in off hours leading to higher costs.

Solution

- They split their searches across several AWS Lambda by parallelizing and storing genome heavy data in Amazon S3.
- Reduced search times by 90%
- Were able to increase the number of genomes
- Reduced their costs and increased scalability

Competitors



Google Cloud
Functions



IBM Cloud
Functions



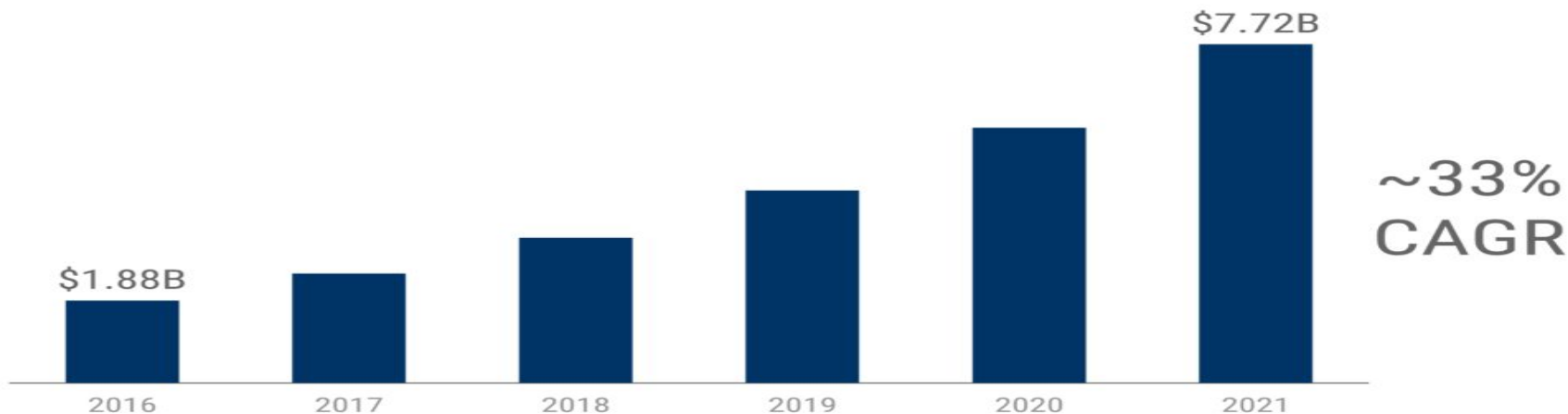
Microsoft

Azure
Functions

Expected Trends

The serverless market is expected to reach \$7.7B by 2021

Estimated size of the serverless & function-as-a-service market annually, 2016 – 2021



Source: CB Insights Market Sizing Tool; Research and Markets

 CBINSIGHTS



Thank You!

