Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44000 Internet Applications

Week 10 Lecture 1, Serverless

Conor Sheedy

Container-as-a-Service

- **Amazon Elastic Container Service (ECS).**
  - Your container instance must be running the Amazon ECS container agent.
  - The container agent is able to register the instance into one of your clusters.
  - If you use an Amazon ECS-optimized AMI the agent is already installed.
  - To use a different operating system you must install the agent.
  - An ECS container instance is an EC2 instance that runs the ECS Container Agent.
  - The EC2 instance is owned and managed by you.
  - The instance appears in the list of EC2 instances like any other EC2 instance.

Container-as-a-Service

- **Amazon Elastic Container Service (ECS).**
  - Usually, you run a cluster of container instances in an auto-scaling group.
  - ECS is free of charge.
  - You only pay for the EC2 instances.
  - The downside is that you have to scale, monitor, patch, and secure the EC2 instances yourself.
  - An ECS container instance can run on Linux or Windows.
  - Unused CPU shares can be used by other containers if available.

Serverless Computing – Container-as-a-Service

- **Fargate**
  - users are freed from thinking about the nature of machine resources that should be reserved for the task.
  - Instead, users just draw the resources they need from a seemingly infinite pool
  - AWS Fargate manages the task execution.
  - No EC2 instances to manage.
  - You pay for running tasks.
- **Serverless**
  - there are servers involved, but service users don't need to be concerned with them.
- **The packaging of functionality is a LINUX container**
  - still has quite a large granularity

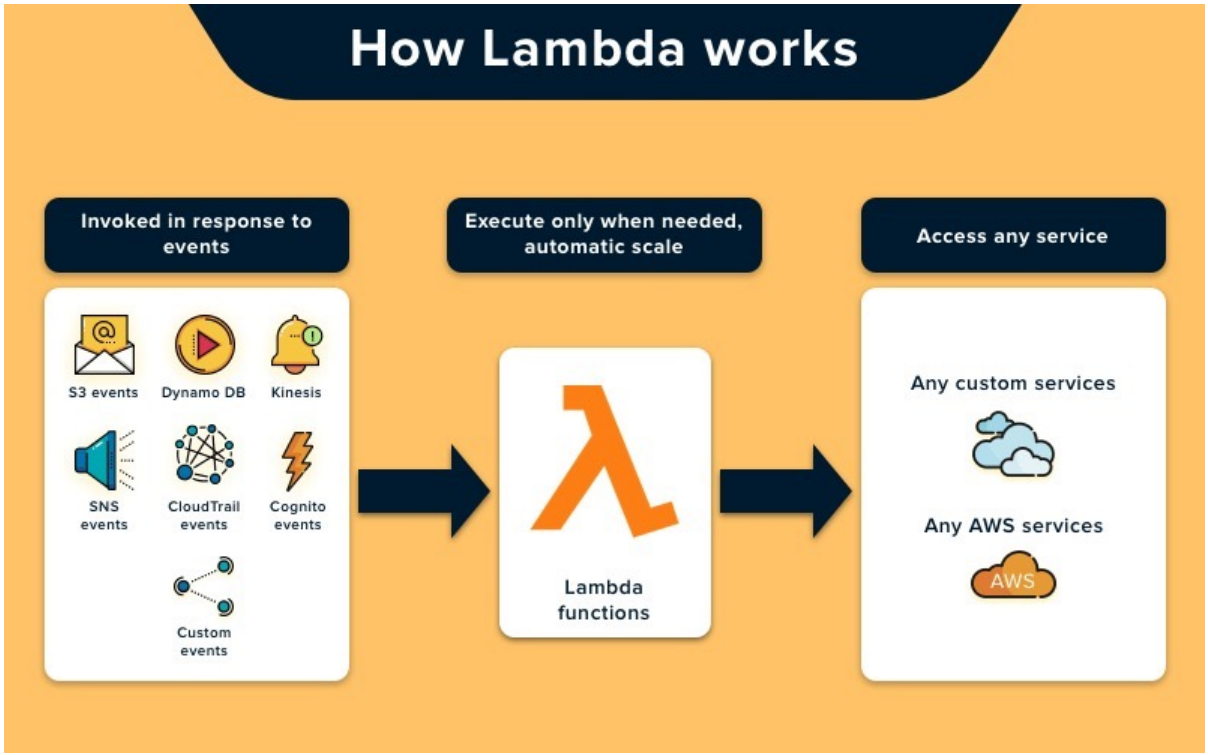
Serverless Computing – Function-as-a-Service

- **AWS Lambda**
  - Event driven
  - Serverless
  - November 2014
  - Node.js, Python, Java, Go, Ruby, C#
  - Name comes from “Lambda functions”, anonymous functions
- **Google Cloud Functions**
- **Azure Functions**

Reduces complexity to the bare minimum – a single function!

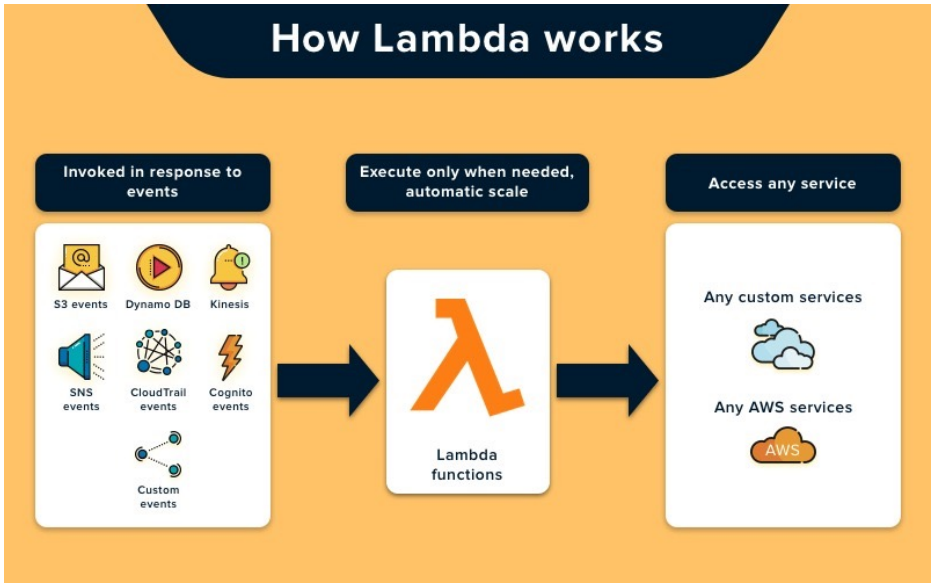
The AWS Lambda Architecture

- **A variety of services - including incoming http requests trigger AWS Lambda requests**
- **Code is launched into a container on an EC2 instance with an Amazon Linux AMI**
  - The point is that the implementation is not the developers concern
- **Event is passed into the code**



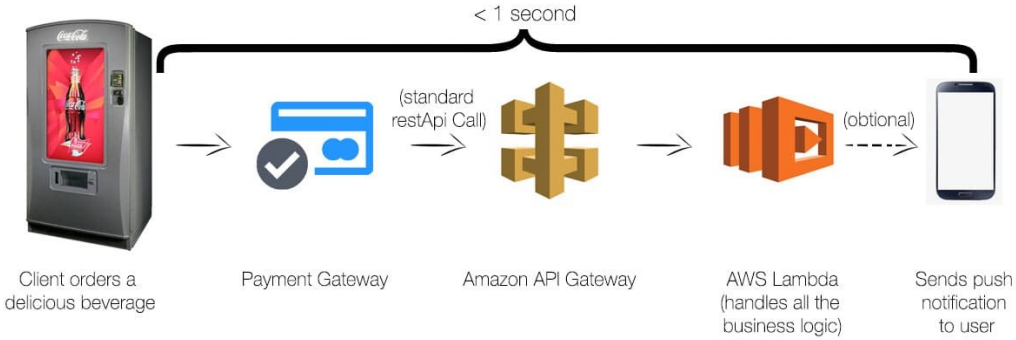
The AWS Lambda Use Cases

- **Event Driven**
  - handling image and object uploads to Amazon S3
  - updating DynamoDB tables
  - responding to website clicks
  - reacting to sensor readings from IoT connected devices
- **custom HTTP requests**
  - configured in AWS API Gateway
  - which can handle authentication and authorisation



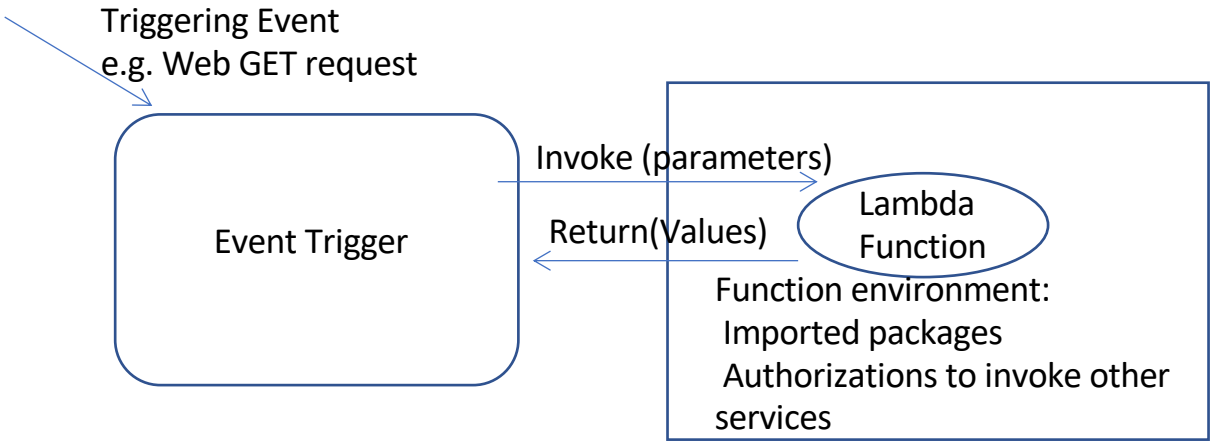
Case Study: CocaCola's use of Serverless

https://dashbird.io/blog/serverless-case-study-coca-cola/



- **Up to 2016 using 6 EC2 T2.medium**
  - cost $12,864 per year
- **Serverless**
  - $4,490 per year
  - 30 million requests per month
- **Infrastructure as a Service would start to get cheaper at 80 million requests per month**

Lambda Function Components



- **Much of the complexity in Serverless is not in writing the Lambda Function, but in deploying it in the right environment, with correct event triggers and authorizations**
- **Frameworks like Claudia.js and "Serverless" simplify this**
  - create functioning Node. Js program, locally, then invoke the "Deploy" step which packages everything into a .ZIP file and uploads to AWS Lambda
- **We may use a more low-level approach for educational purposes, but hard to get anything complex done**
  - In practice use JavaScript Frameworks which hide the complexity
  - Reduce the time to develop and deploy

Context

Events are passed in with a *Context & an Event*

We will look at an example event:

GET /lambda?query=1234ABCD

Which arrives at an Application Load Balancer

Example Context & Event from an Application Load Balancer


```
{
   "requestContext": {
      "elb": {
         "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
      }
   },
   "httpMethod": "GET",
   "path": "/lambda",
   "queryStringParameters": {
      "query": "1234ABCD"
   },
   "headers": {
      "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
      "accept-encoding": "gzip",
      "accept-language": "en-US,en;q=0.9",
      "connection": "keep-alive",
      "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
      "upgrade-insecure-requests": "1",
      "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 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",
      "x-imforwards": "20"
   },
   "body": "",
   "isBase64Encoded": false
}
```

Exercise:
Create a simple function on AWS Lamda that sends the event parameters back in JSON format

- **The Lambda function is invoked and passed in an event**
- **If it is a HTTP GET, then it passes back JSON containing all the event parameters**

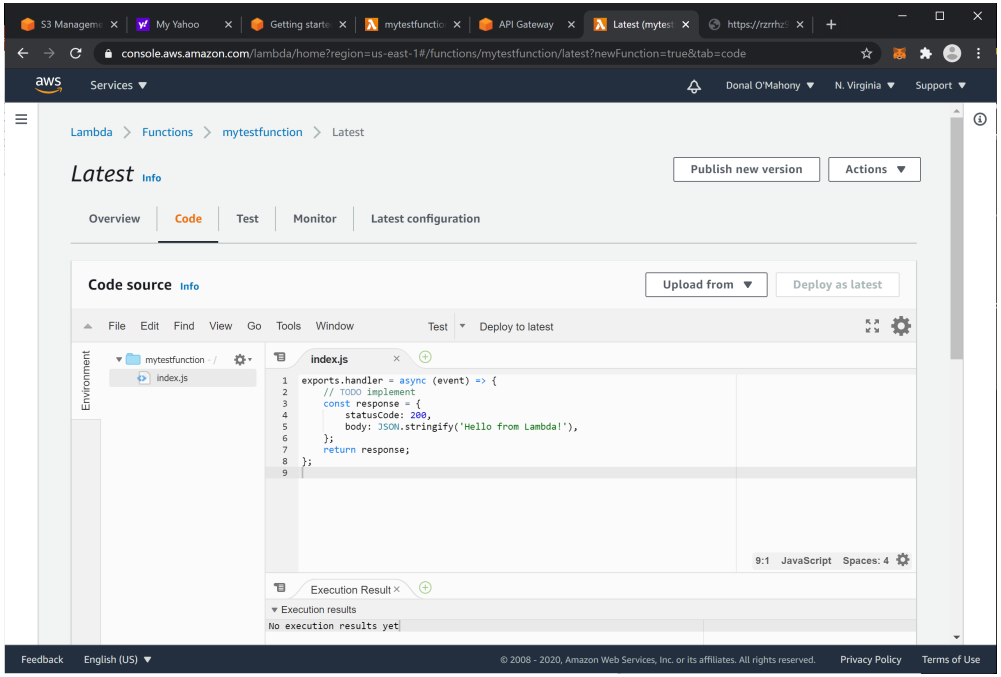
```
exports.handler = async (event) => {
 let response = { statusCode: 200,
body: JSON.stringify('Hello from Lambda!'),    };

 if (event.httpMethod == 'GET')     {
    response = { statusCode :200,
                    body: JSON.stringify(event)}}

return response;}
```

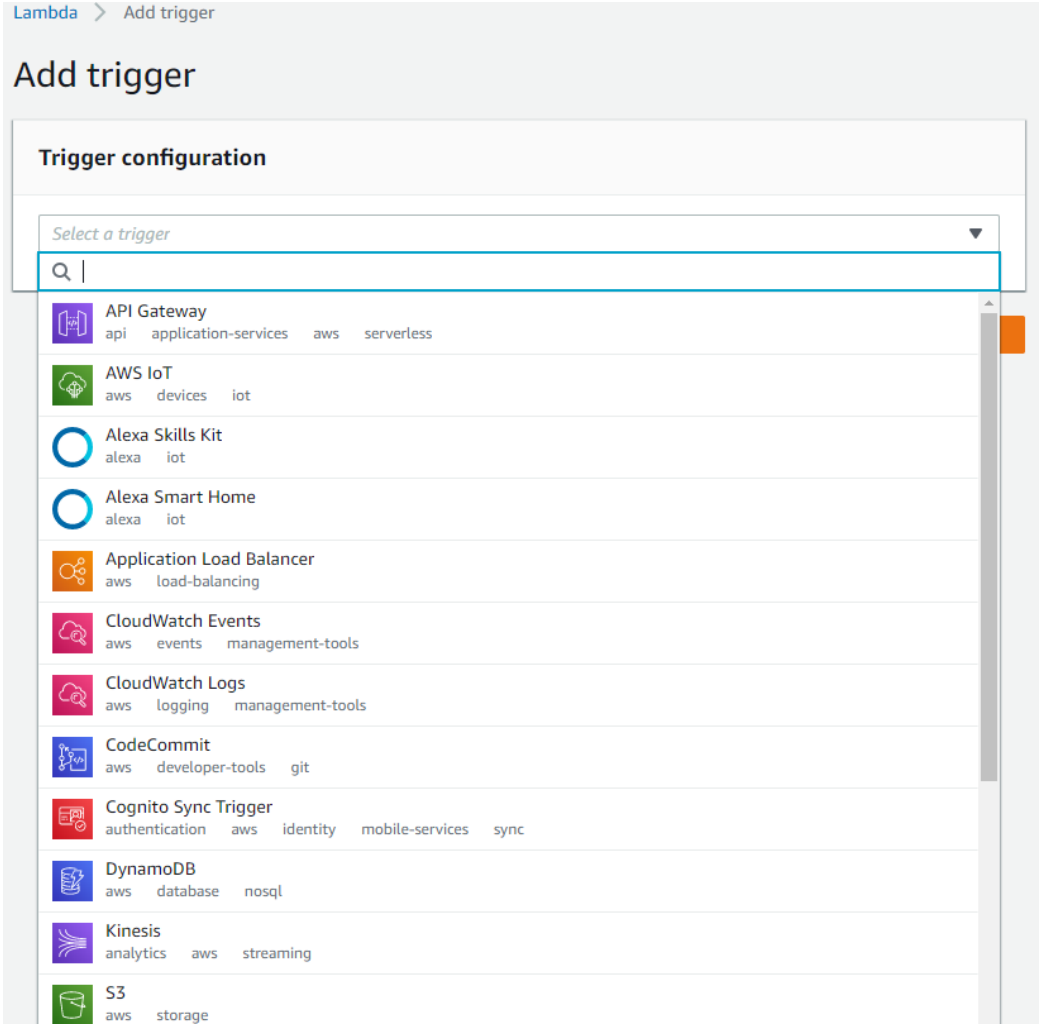
Creating a Lambda Function

- **The interface has changed quite a lot over the years**
- **First specify function name and execution environment**
  - e.g. Node, Ruby, Python etc
- **Invokes Cloud9 Editor**
  - Online IDE
  - Acquired by AWS in 2016
- **Allows function to be input and simple tests to be performed**



Next step is to add Event Triggers

- **There are a large selection of things happening within AWS environment that can trigger the calling of a Lambda Function**
- **If programming a web REST API, then API gateway will field the HTTP requests and convert them into events before calling the Lambda function**



Using API Gateway as Event Trigger

https://www.youtube.com/watch?v=DSrg7hG-jV4&list=PLzvRQMJ9HDiSQMe68cti8cupl0mzLk1Gc&index=3

[5 min]

Result of invoking API to get event data

{"resource":"/DOMfirstfunc","path":"/DOMfirstfunc","httpMethod":"GET",“
headers":{"accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3",
"accept-encoding":"gzip, deflate, br","accept-language":"en-US,en;q=0.9",
"Host":"4y94vcgivg.execute-api.eu-west-1.amazonaws.com",
"sec-fetch-mode":"navigate","sec-fetch-site":"none","upgrade-insecure-requests":"1",
"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36","X-Amzn-Trace-Id":"Root=1-5dbf2bc5-d40b3965baba4de43211824d",
"X-Forwarded-For":"37.228.252.43","X-Forwarded-Port":"443","X-Forwarded-Proto":"https"},
"multiValueHeaders":{"accept":["text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3"],
"accept-encoding":["gzip, deflate, br"],"accept-language":["en-US,en;q=0.9"],"Host":["4y94vcgivg.execute-api.eu-west-1.amazonaws.com"],"sec-fetch-mode":["navigate"],
"sec-fetch-site":["none"],"upgrade-insecure-requests":["1"],"User-Agent":["Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36"],
"X-Amzn-Trace-Id":["Root=1-5dbf2bc5-d40b3965baba4de43211824d"],"X-Forwarded-For":["37.228.252.43"],"X-Forwarded-Port":["443"],"X-Forwarded-Proto":["https"]},"queryStringParameters":null,
"multiValueQueryStringParameters":null
,"pathParameters":null,"stageVariables":null,"requestContext":{"resourceId":"i2tmc4“
,"resourcePath":"/DOMfirstfunc",

"httpMethod":"GET","extendedRequestId":"CmPG7EDZDoEF8KA=","requestTime":"03/Nov/2019:19:34:29 +0000",
"path":"/prod/DOMfirstfunc","accountId":"989357430853",
"protocol":"HTTP/1.1","stage":"prod","domainPrefix":"4y94vcgivg","requestTimeEpoch":1572809669872,"requestId":"a904c3ec-3cce-4d92-ad9f-6018bf6ede9a",
"identity":{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"sourceIp":"37.228.252.43","principalOrgId":null,"accessKey":null,
"cognitoAuthenticationType":null,"cognitoAuthenticationProvider":null,"userArn":null,
"userAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36","user":null},
"domainName":"4y94vcgivg.execute-api.eu-west-1.amazonaws.com","apiId":"4y94vcgivg"},"body":null,"isBase64Encoded":false}

Lambda Pricing

- **Charges are based on the number of invocations and the intensity of each invocation**
- **Invocations:**
  - First 1 million invocations/month are free
  - $0.20 PER 1M Requests thereafter: $0.0000002 per request.
- **Intensity**
  - measured in GB-seconds
    - product of memory allocated and seconds of CPU time
  - First 400,000 GB-seconds per month is free
  - $0.0000166667 for every GB-SECOND used thereafter
  - Minimum memory of 128MB per instance
- **S3 or DynamoDB charges are separate**

Pricing Example

If your Lambda@Edge function executed 10 million times in one month, and it ran for 50ms each time, your charges would be calculated as follows:

Monthly compute charges

The monthly compute price is $0.00000625125 per 128MB-second

Total compute (seconds) = 10M * (0.05sec) = 500,000 seconds
Monthly compute charges = 500,000 * $0.00000625125 = $3.13

Monthly request charges

The monthly request price is $0.60 per 1 million requests.
Monthly request charges = 10M * $0.6/M = $6.00

Total monthly charges

Total charges = Compute charges + Request charges = $3.13 + $6.00 = $9.13 per month

Serverless Lambda Function-as-a-Service FaaS Summary

- **Intention is to package up useful work as a ‘Function’**
- **Cloud environment facilitates connecting events to the function (web, IoT, Messaging)**
- **User does not have to worry about compute resources or scaling**
- **Pricing is based on function invocations i.e. useful work done**

Serverless: Function-as-a-Service FaaS Differences from Platforms-as-a-Service

- **Both hide "servers" from developers**
- **PaaS typically always has at least one server process running that receives external requests**
- **scaled by booting up more server processes**
  - charged for
- **FaaS no server process constantly being run**
  - ˇpay for function execution time
  - no process idle time

Alternative Approach, manage the hardware like software

- **Infrastructure as code (IaC)**
  - machine-readable definition files
  - managing and provisioning computer data center resources
    - AWS CloudFormation
      - JSON or YAML
    - Terraform
      - HashiCorp Configuration Language (HCL), or JSON
      - August 2023 moved from Open source to source available
        - » Business Source License