

AWS Lambda and Python - Full Course

Absolute Beginner to Advanced

No previous Coding Experience Required



AWS Lambda and Python – Beginner to Advanced

12 Months on....

900+ Learners

110+ Ratings



DineshEzeikel

Updated 2 months ago



Excellent course.



basam nath

Updated 4 months ago



Just superb videos



Ebrahim Kamari

Updated 7 months ago



This course helped me alot. Thanks Rahul



Terence Atarah

Updated 2 months ago



Yes it was. It gives a clear insight on how to use lambda with python. I have struggled with this over the years and i am finally understanding it from a 3 hour course...Amazing



Shashank j

Updated 5 months ago



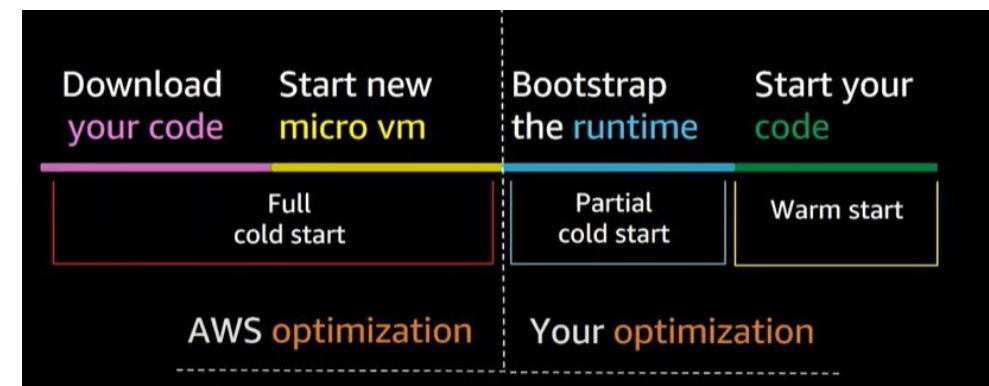
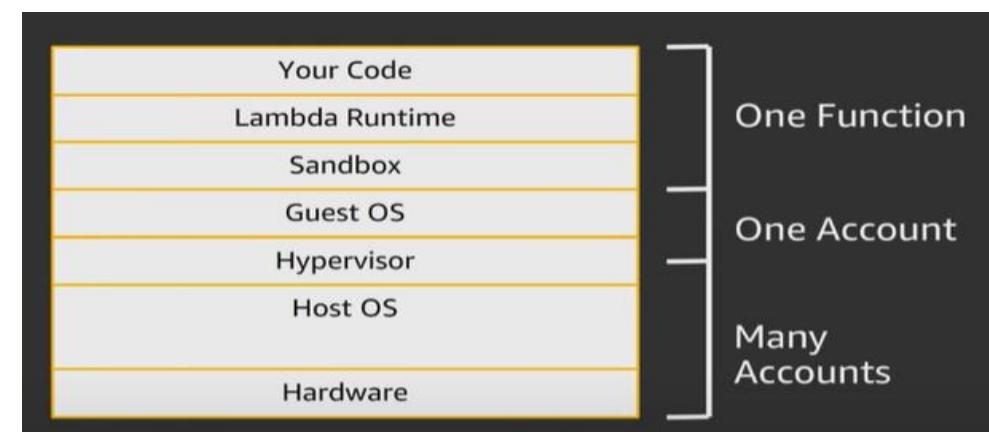
best course i have ever taken on AWS lambda. The instructor taught me lot of good stuff that i use every day



AWS Lambda and Python – Beginner to Advanced

Section 2 : AWS Lambda – Basic Concepts – Part 1

- Evolution from Physical Servers to AWS Lambda
- What is AWS Lambda - Architecture and Use Cases
- Lambda Console Walkthrough
- Lambda Execution Role
- AWS Lambda Limits



AWS Lambda and Python – Beginner to Advanced

Section 3 : AWS Lambda - Python Basics Refresher



Python Basics_Refresher_Part 1 (Python Print Function, Variables, Data Types Intro, Data Type – Dictionary)

Python Basics_Refresher_Part 2 (Nested Dictionary , List and Nested List , Data Type and Python Functions)

```
####Nested Dictionary

nesteddictionary = {1: 'Python', 2: {'books': 'cloud', 'aws': 'Lambda'}}
print(nesteddictionary[2])
print(nesteddictionary[2]['books'])

### Loops and Dictionary Methods

for k in nesteddictionary.values():
    print(k)

for k in nesteddictionary.keys():
    print(k)
###### user input

name = input('Please enter your name')
grade = input('Please enter your grade')
print('My name is {} and my grade is {}'.format(name, grade))

##### 9 String Slicing
data = 'ASTRING'
print(data[2:4:1])

for a in data:
    print(a)

# A simple Python function to check
# whether x is even or odd
```

```
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code to call the function
evenOdd(2)
evenOdd(3)

def square_value(num):
    """This function returns the square
    value of the entered number"""
    return num ** 2

print(square_value(2))
print(square_value(-4))
```

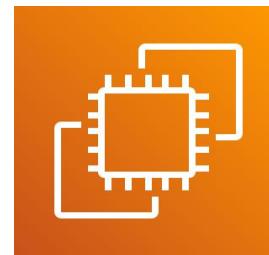
AWS Lambda and Python – Beginner to Advanced

Section 4 : AWS Lambda – Create S3, EC2 and DynamoDB using AWS SDK for Python(Boto3)

- AWS Lambda Basics – Boto3, Client and Resource, Lambda function handler
- AWS Lambda with S3 (List all the buckets, Create new Bucket and Delete Bucket)
- AWS Lambda with EC2 (Create EC2 and Stop)
- AWS Lambda with DynamoDB (Create Table and Put Items)



S3



EC2

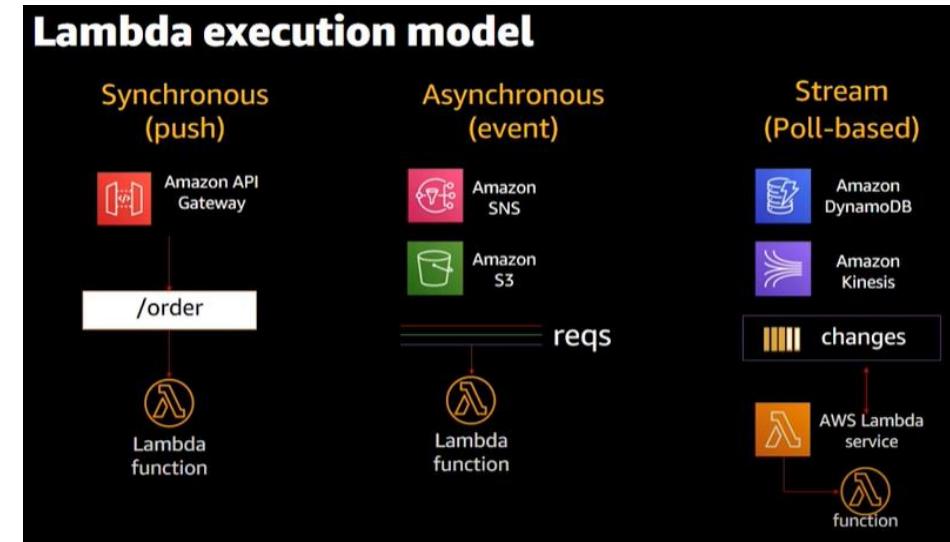


DynamoDB

AWS Lambda and Python – Beginner to Advanced

Section 5 : AWS Lambda – Basic Concepts – Part 2

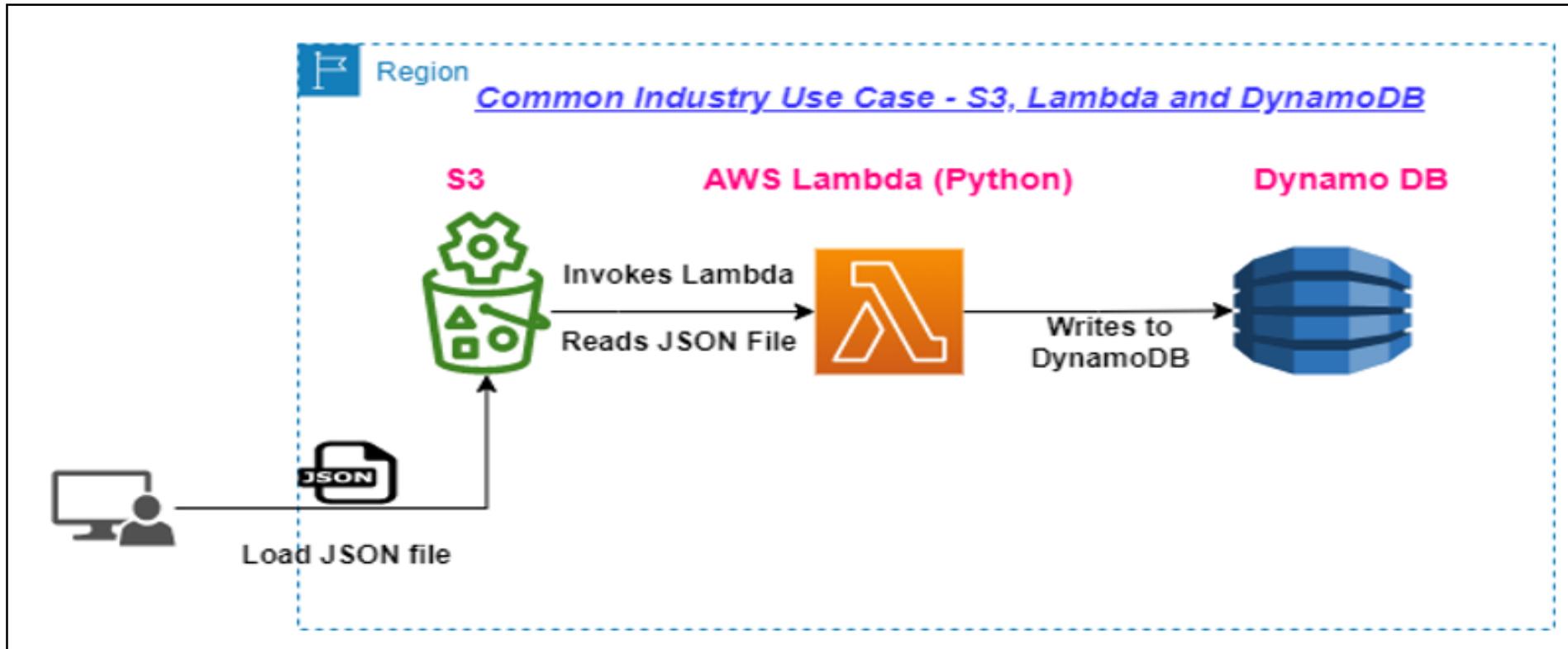
- AWS Lambda Invocation Models – Theory
- AWS Lambda Invocation Models – Hand-on
- AWS Lambda Limits - Memory



Memory	Duration	Cost
128 MB	11.722 s	\$0.024628
256 MB	6.678 s	\$0.028035
512 MB	3.194 s	\$0.026830
1024 MB	1.465 s	\$0.024638

AWS Lambda and Python – Beginner to Advanced

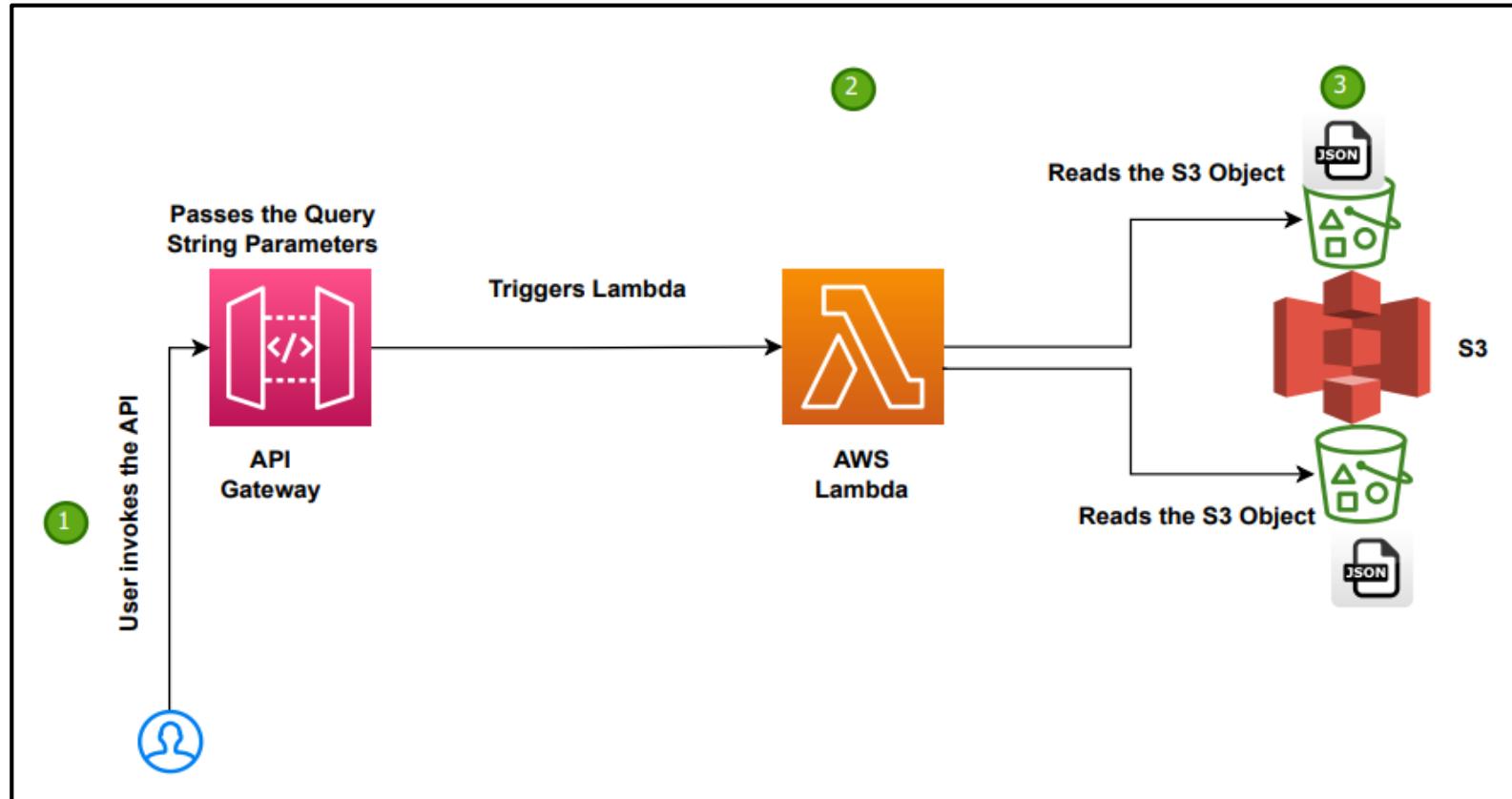
Section 6 : Real World Serverless Use Case 1 - using S3, AWS Lambda and DynamoDB



- *S3*
- *AWS Lambda*
- *DynamoDB*

AWS Lambda and Python – Beginner to Advanced

Section 7 : Real World Serverless Use Case 2 - using API Gateway, AWS Lambda and S3

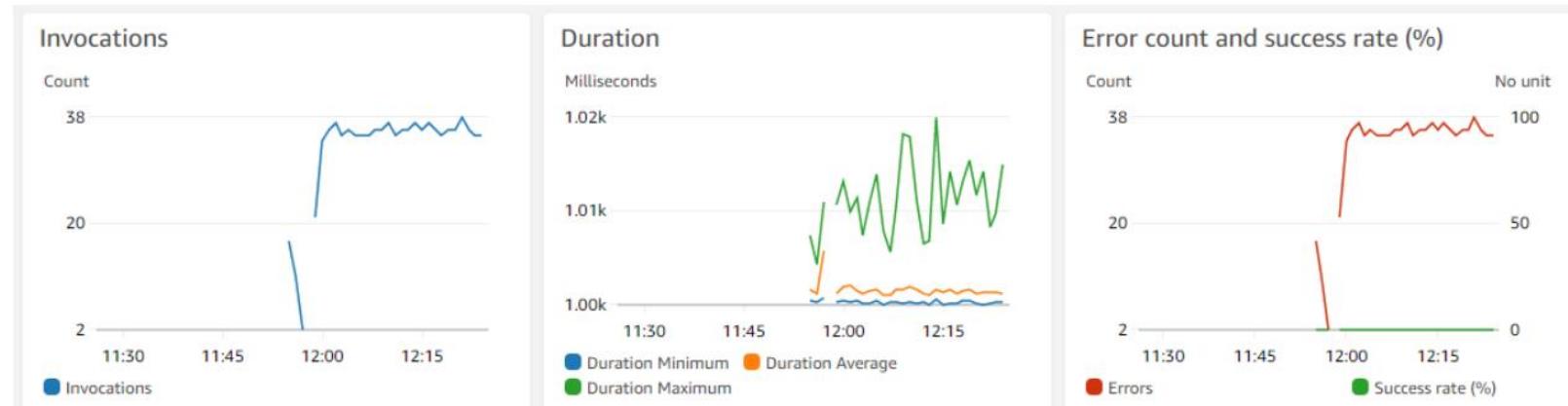
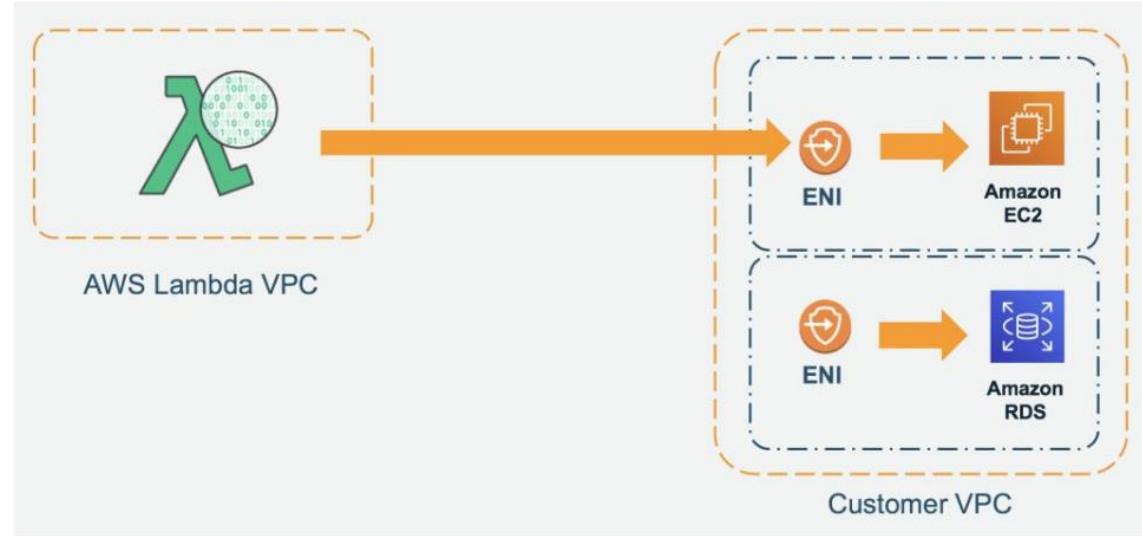


- *API Gateway*
- *AWS Lambda*
- *S3*

AWS Lambda and Python – Beginner to Advanced

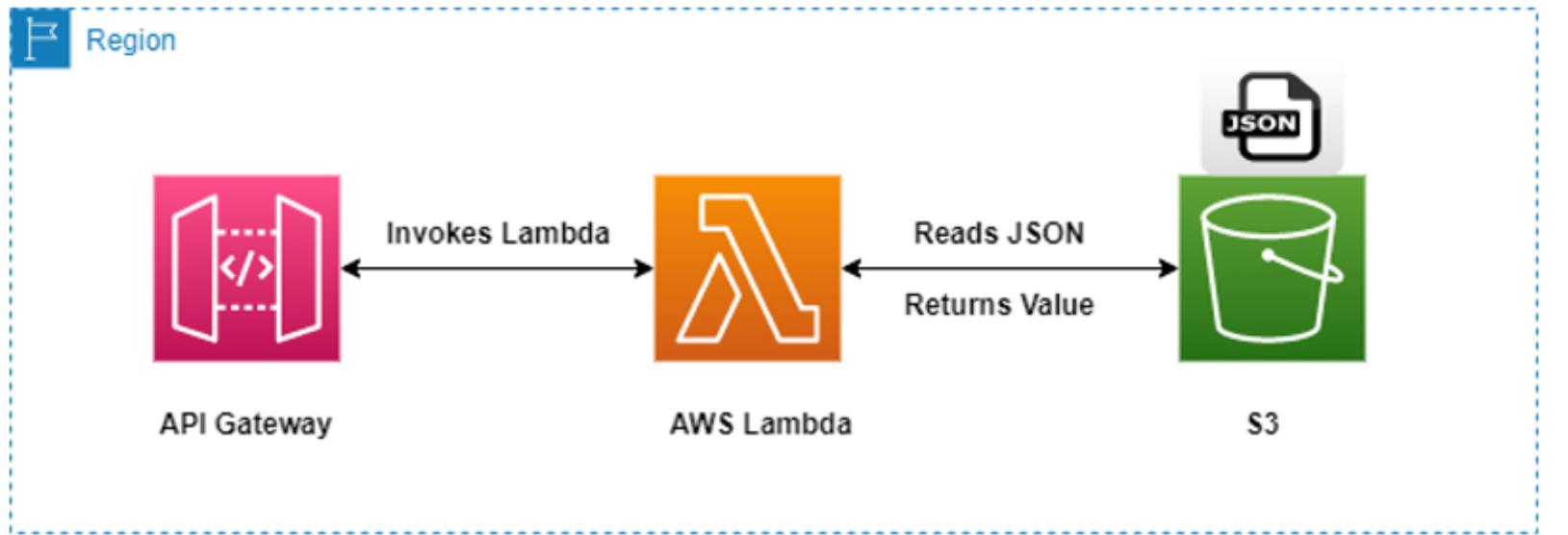
Section 8 : AWS Lambda – Advanced Concepts (Theory and HandsOn)

- Lambda Execution and Concurrency
- Lambda - Reserved and Provisioned Concurrency
- Lambda - VPC Networking Configuration
- Lambda - Environment Variables
- Lambda Versions
- Lambda Aliases
- Lambda Monitoring - CloudWatch Metrics
- Lambda Monitoring - CloudWatch Logs



AWS Cloud Development Kit (CDK) v2 – Serverless Use Case

Section 9 : Serverless Use Case - using API Gateway, AWS Lambda and S3 using AWS CDK v2

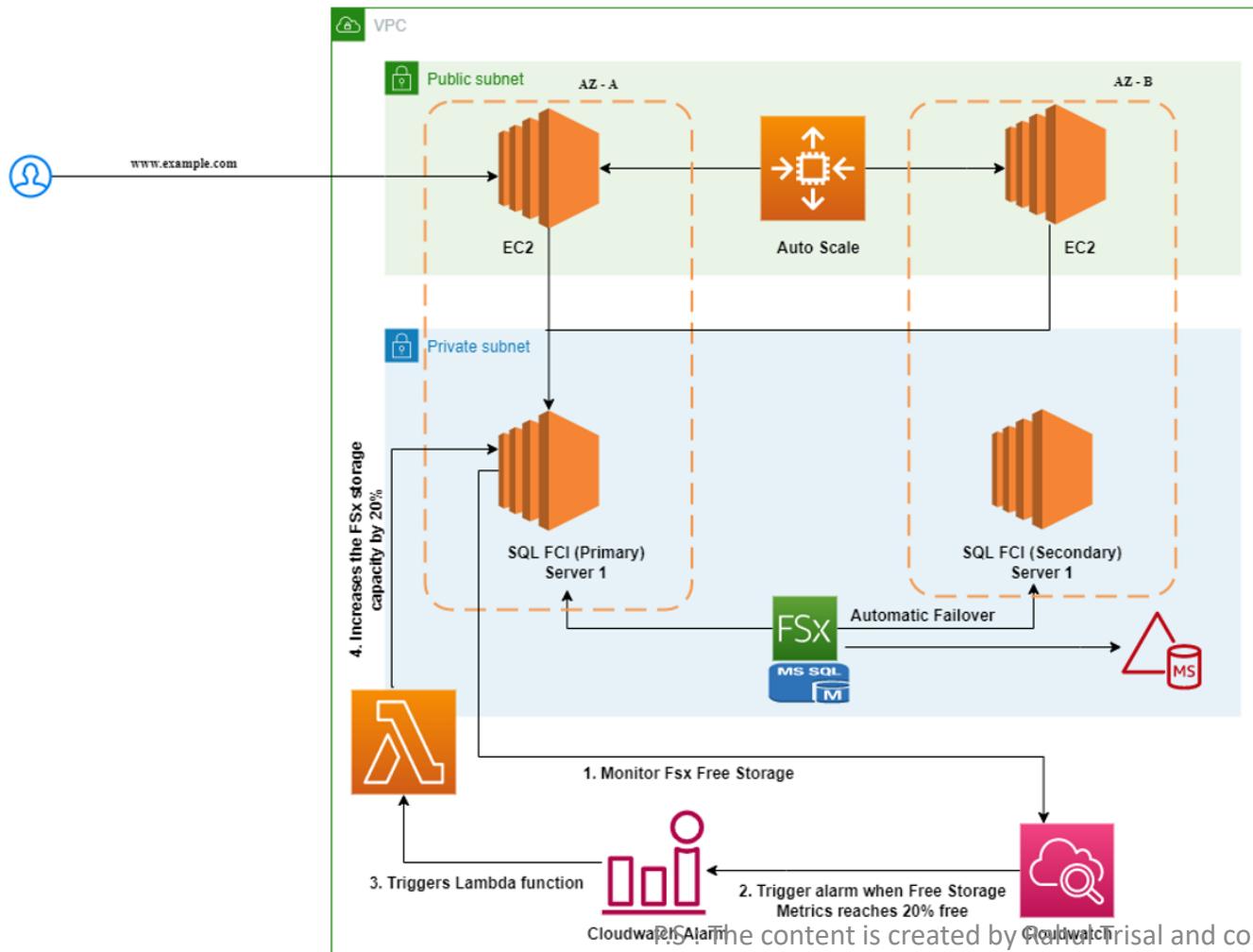


- *S3*
- *IAM Role*
- *AWS Lambda*
- *API Gateway*

Implement the use case using AWS CDK

AWS Lambda and Python – Beginner to Advanced

Section 10 : Use Case 3 – Lambda Automation Monitor & increase free storage for SQL Server FCI Cluster using AWS Lambda- MS AD, EC2, FSx, CloudWatch and CloudWatch Alarm



- Microsoft AD
- EC2
- FSx
- CloudWatch Metrics
- CloudWatch Alarm

Connect with me and share your AWS Learning Experiences

- I am **Rahul Trisal** working as an **AWS Solution Architect** in a Fortune 500 Organization
- Co-Founder of a AWS focused Startup – TechCloudByte – www.techcloudbyte.com
- 6X Cloud Certified
 - **AWS Solution Architect – Professional**
 - AWS Solution Architect – Associate
 - AWS Certified SysOps
 - AWS Cloud Practitioner
 - Azure Fundamental
 - IBM Bluemix Developer
- 200+ applications migrated working with Fortune 100 customers on large AWS Cloud Migration Programs



I would love to Connect with you and together learn AWS. Connect with me at below links:

- Instagram - https://www.instagram.com/aws_with_rahul/
- Linkedin - <https://www.linkedin.com/in/rahul-trisal-7709628/>
- Youtube - <https://www.youtube.com/@trisalrahul/videos>
- AWS Startup - www.techcloudbyte.com
- Follow our LinkedIn page for latest on AWS <https://www.linkedin.com/company/tech-cloud-byte-techclobyte>

AWS Lambda and Python – Beginner to Advanced

Section on

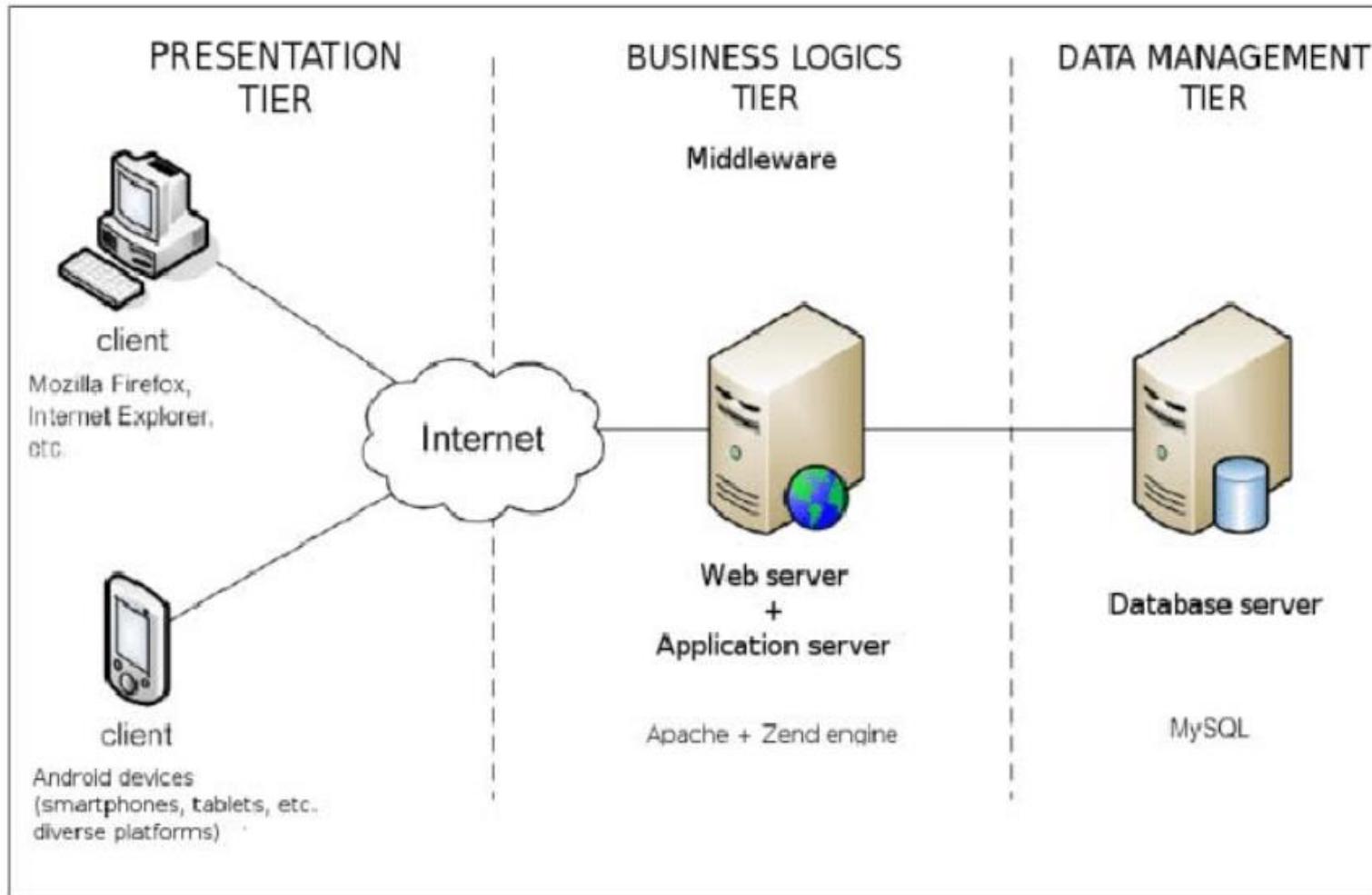
AWS Lambda – Basic Concepts (Part 1)

AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Basic Concepts – Part 1

- ❖ Evolution from Physical Servers to AWS Lambda
- ❖ What is AWS Lambda - Architecture and Use Cases
- ❖ Lambda Console Walkthrough
- ❖ Lambda Execution Role
- ❖ AWS Lambda Limits - Timeout

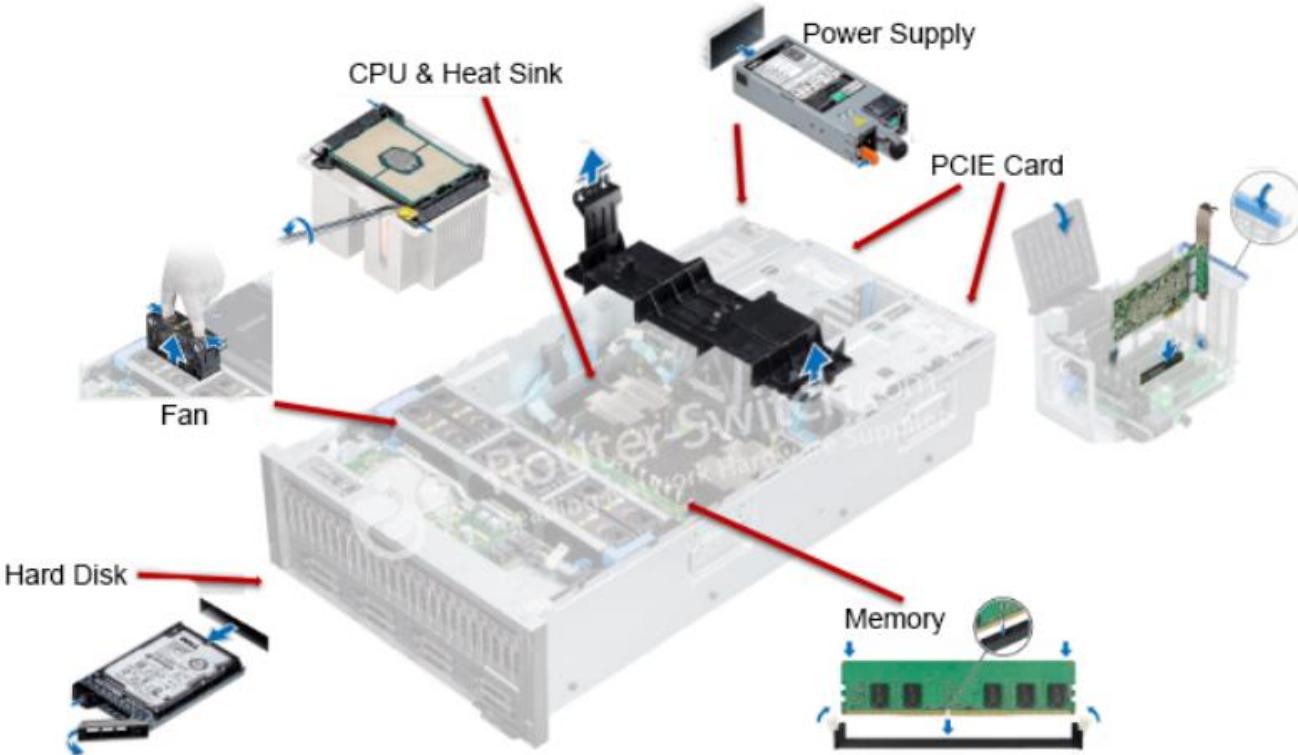
Three-Tier Architecture



3-tier architecture

Source : Internet

Server



Source : Internet



Source : Internet

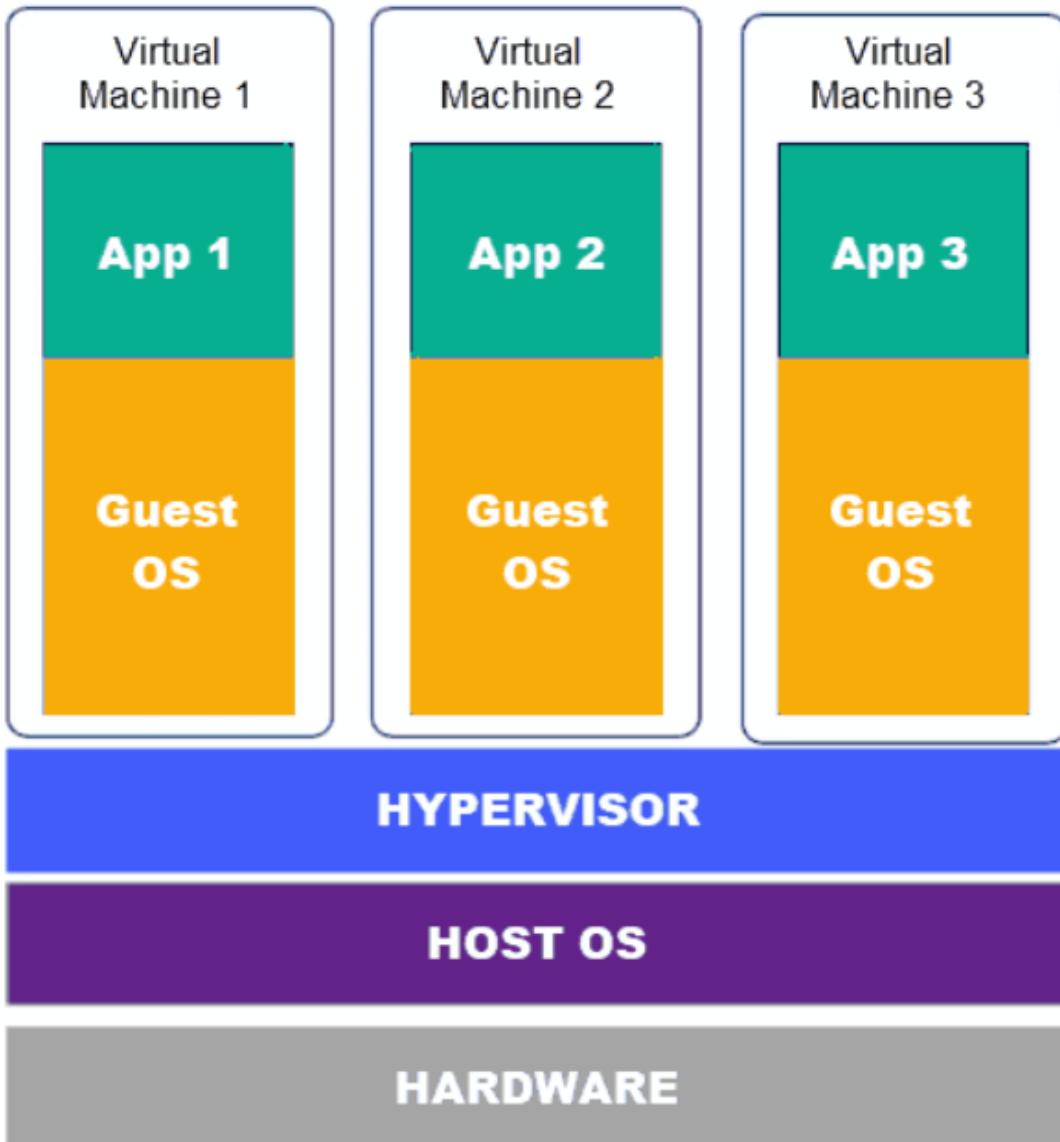


Source : Internet



Google data center in Eemshaven, Netherlands | Image credit: Google

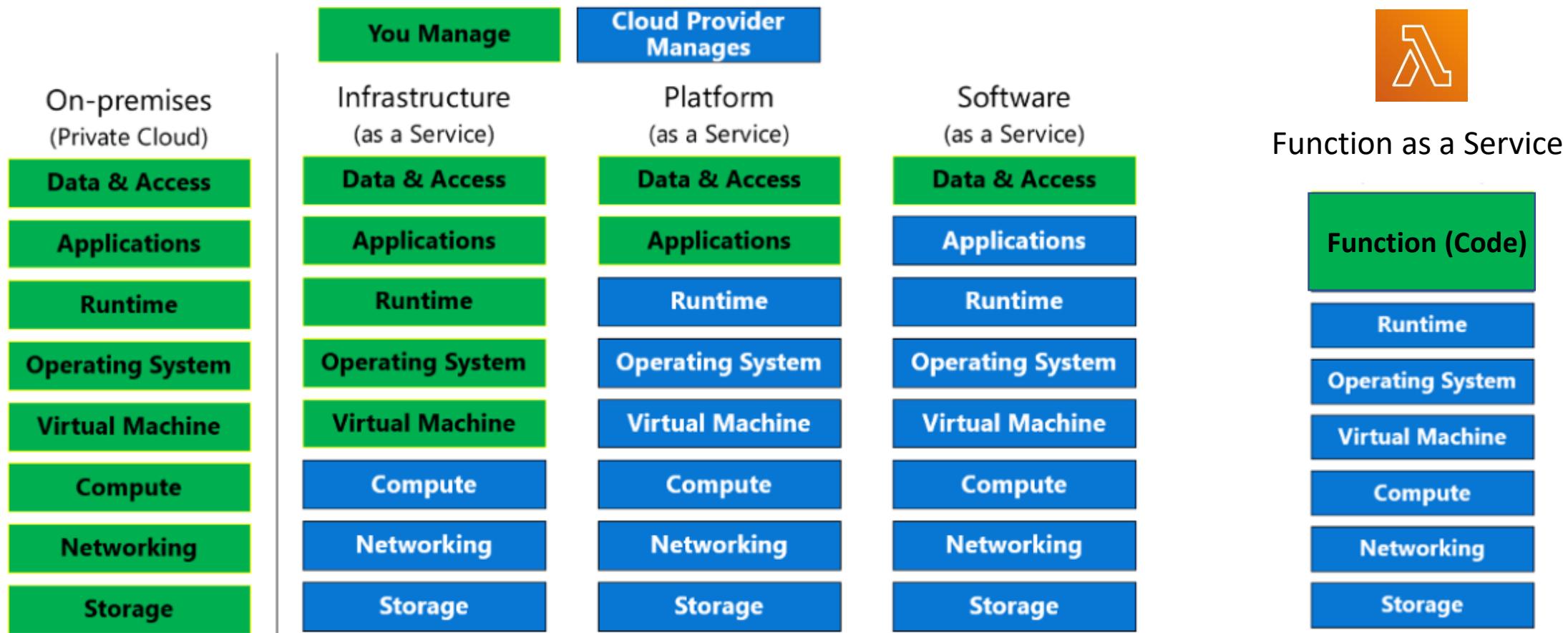
Virtualization



What is Cloud Computing ?

Cloud computing is **the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing.**

IaaS vs PaaS vs SaaS and FaaS



Source : Internet

AWS Lambda and Python – Beginner to Advanced

AWS Definition of Lambda

Lambda is a **compute service** that lets you **run code without provisioning or managing servers**. Lambda **runs your code on a high-availability** compute infrastructure and performs all of the **administration of the compute resources**, including server and **operating system maintenance, capacity provisioning and automatic scaling, and logging**.

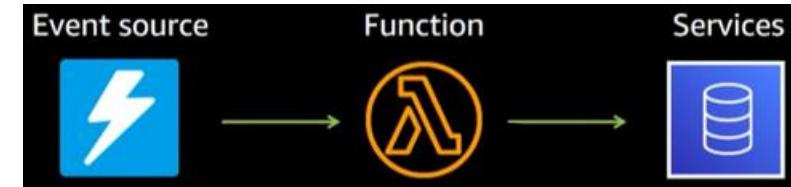
Key Features of Lambda

- Compute Service
- Highly Available
- All of the administration of the compute resources, including server and operating system maintenance, automatic scaling, and logging.
- Provisioning done by AWS and Pay as you use
- Bring your code and run on Lambda
- **Lambda is a serverless, event-driven compute service**

AWS Lambda and Python – Beginner to Advanced

Use Case for AWS Lambda

- Event Driven
- Unpredictable demand



When not to use a AWS Lambda

- When you need to manage the Infrastructure
- Need to run the Server Continuously instead of event driven

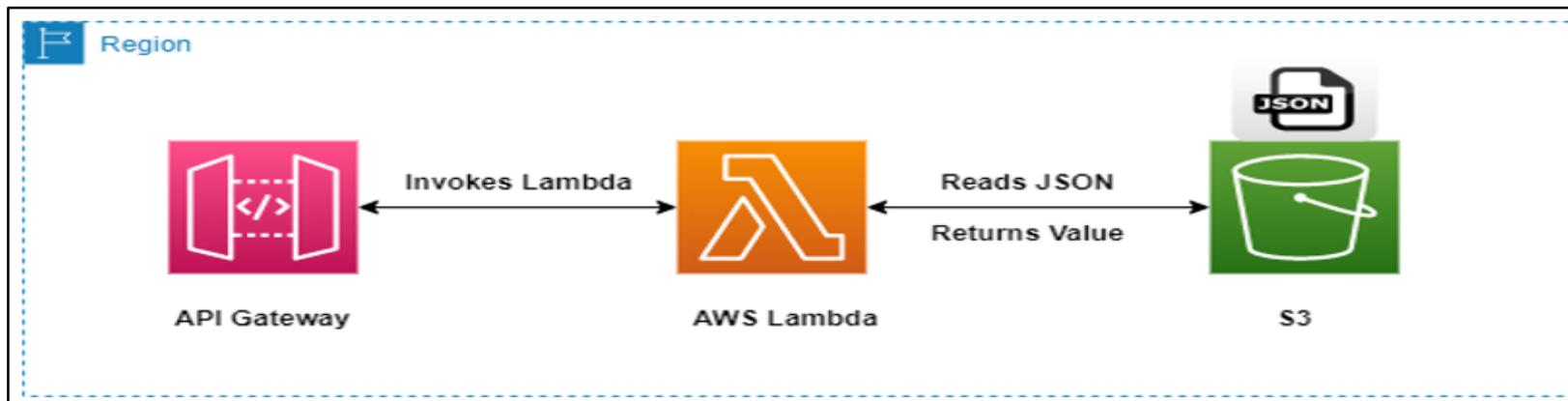


AWS Lambda and Python – Beginner to Advanced

Image Processing – S3 and AWS Lambda

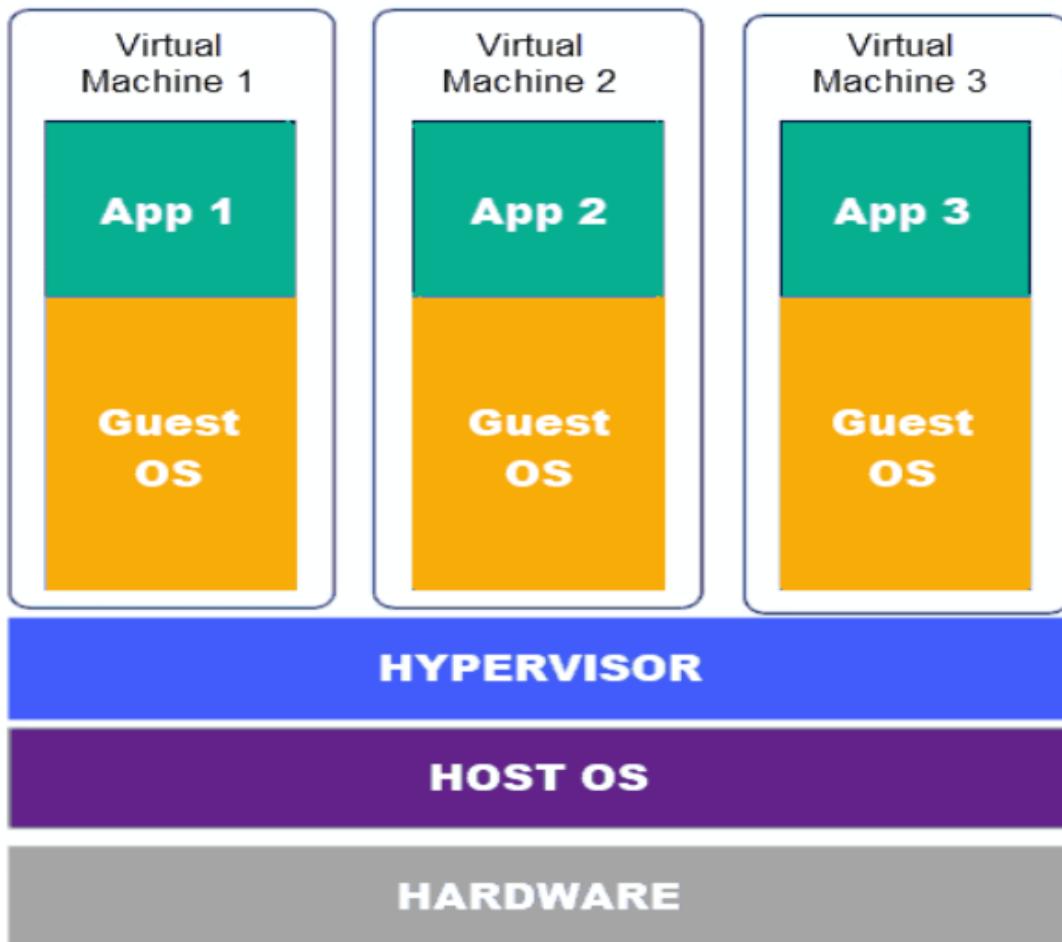


Build REST API with API Gateway, Lambda and S3

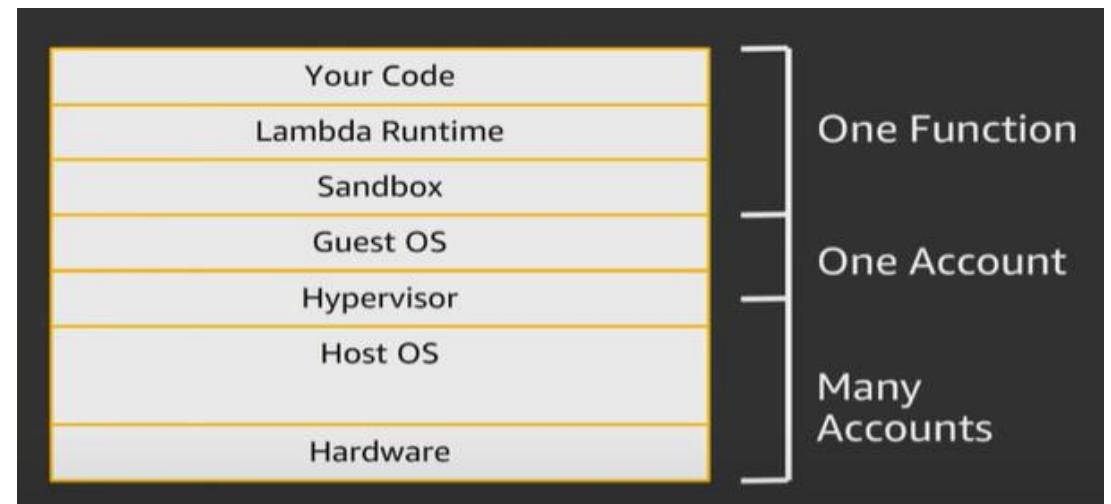


Virtualization

EC2 Instance



AWS Lambda

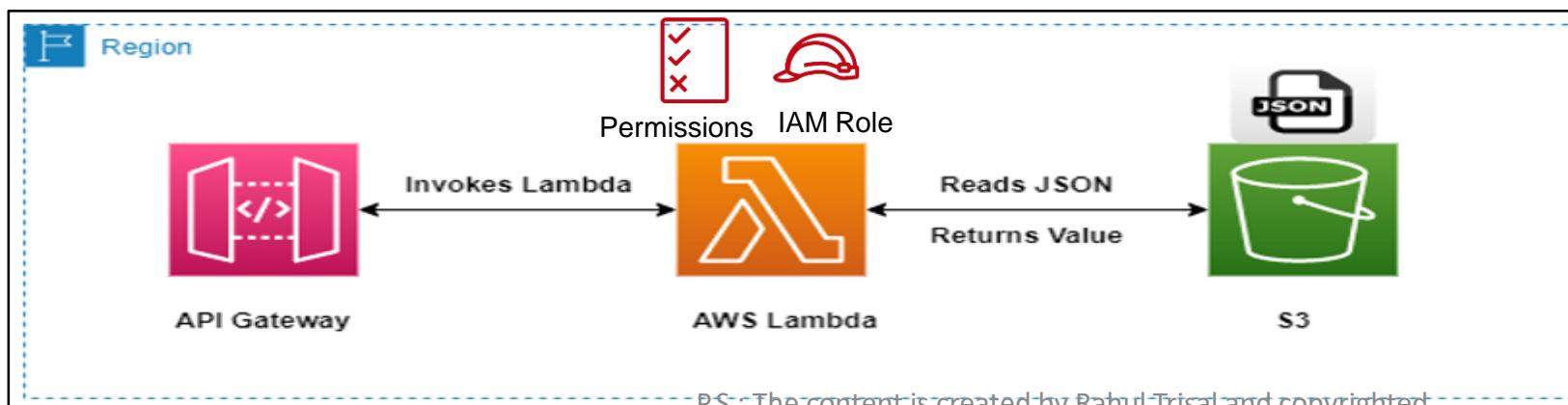


Source : Internet

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Execution Role

- A **Lambda function's execution role** is an AWS Identity and Access Management (**IAM**) role that grants the function permission to access AWS services and resources.
- **Default Lambda Function Role Permissions :**
- **AWSLambdaBasicExecutionRole** : AWSLambdaBasicExecutionRole grants permissions to **upload logs to CloudWatch**.
- Need to provide an execution role when a function is created. Invoke your function, Lambda automatically provides your function with temporary credentials by assuming this role.



AWS Lambda and Python – Beginner to Advanced

AWS Lambda Execution Role

AWSLambdaMSKExecutionRole – Lambda added the kafka:DescribeClusterV2 permission to this policy.	<code>AWSLambdaMSKExecutionRole</code> grants permissions to read and access records from an Amazon Managed Streaming for Apache Kafka (Amazon MSK) cluster, manage elastic network interfaces (ENIs), and write to CloudWatch Logs.
AWSLambdaBasicExecutionRole – Lambda started tracking changes to this policy.	<code>AWSLambdaBasicExecutionRole</code> grants permissions to upload logs to CloudWatch.
AWSLambdaDynamoDBExecutionRole – Lambda started tracking changes to this policy.	<code>AWSLambdaDynamoDBExecutionRole</code> grants permissions to read records from an Amazon DynamoDB stream and write to CloudWatch Logs.
AWSLambdaKinesisExecutionRole – Lambda started tracking changes to this policy.	<code>AWSLambdaKinesisExecutionRole</code> grants permissions to read events from an Amazon Kinesis data stream and write to CloudWatch Logs.
AWSLambdaMSKExecutionRole – Lambda started tracking changes to this policy.	<code>AWSLambdaMSKExecutionRole</code> grants permissions to read and access records from an Amazon Managed Streaming for Apache Kafka (Amazon MSK) cluster, manage elastic network interfaces (ENIs), and write to CloudWatch Logs.
AWSLambdaSQSQueueExecutionRole – Lambda started tracking changes to this policy.	<code>AWSLambdaSQSQueueExecutionRole</code> grants permissions to read a message from an Amazon Simple Queue Service (Amazon SQS) queue and write to CloudWatch Logs.
AWSLambdaVPCExecutionRole – Lambda	<code>AWSLambdaVPCExecutionRole</code> grants permissions to manage

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Limits - Timeout

- Lambda runs your code for a **set amount of time** before timing out.
- When the specified timeout is reached, Amazon Lambda terminates execution of your Lambda function.
- The **default value** for this setting is **3 seconds**
- **Maximum value of 900 Sec (15 minutes)**



Best Practice:

- Performance and cost are two key parameters for setting the Timeout limit
- Edge Cases can impact cost

AWS Lambda and Python – Beginner to Advanced

Section on

AWS Lambda - Python Refresher

(For detailed refresher go to last section of Course)

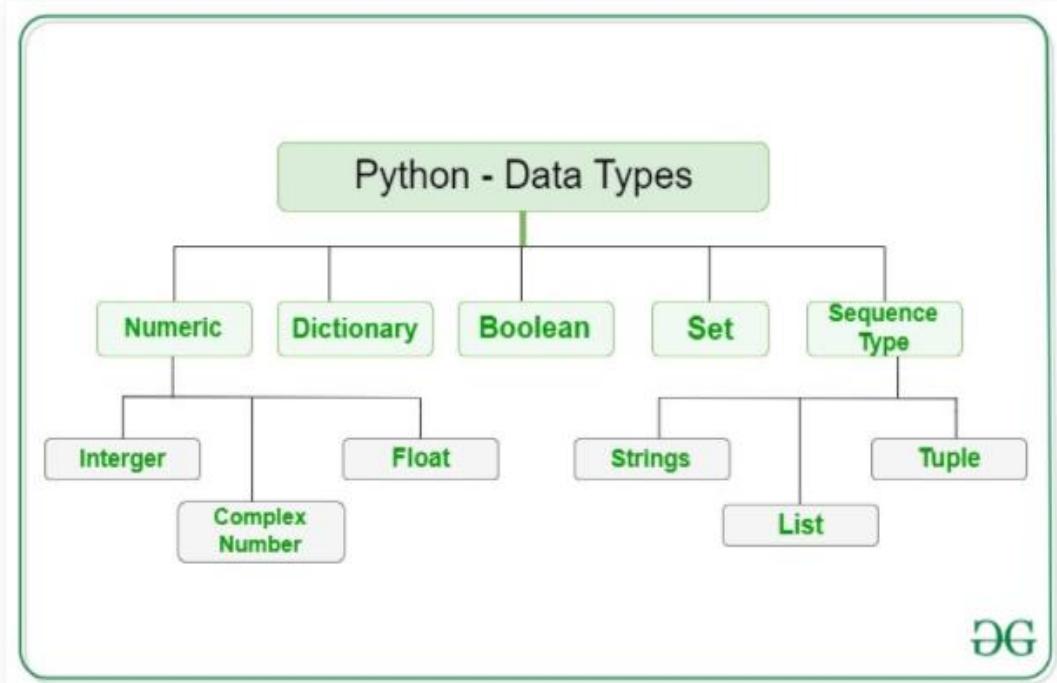
Python Basics—Refresher



1. Print Function – print the message to screen or any interface; Syntax : `print()`

2. Variables - Containers for storing data values string, float or integers and no need to declare; Syntax : `x = 3, greeting = "hello"` etc.

3. Data Type



Source : Internet

P.S : The content is created by Rahul Trisal and copyrighted

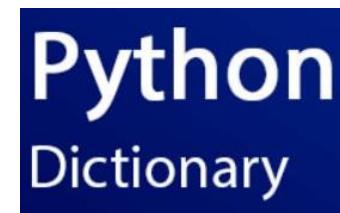
Image Source - Internet

Data Type	Example
Int	<code>x = 20</code>
Float	<code>x = 20.5</code>
Dictionary	<code>x = {"name": "John", "age": 36}</code>
Strings	<code>x = "Hello World"</code>
List	<code>x = ["apple", "banana", "cherry"]</code>

Python Basics—Refresher

3. Data Types – Dictionary

- curly brackets
- key: pair values
- Nested Dictionary



Dict

```
response = {1: 'Rahul', 2: 'John', 3: 'Joy'}
```

Nested Dict

```
response = {1:'Python', 2:{'books': 'arch', 'aws':'Lambda'}}}
```

4. Nested Dictionary

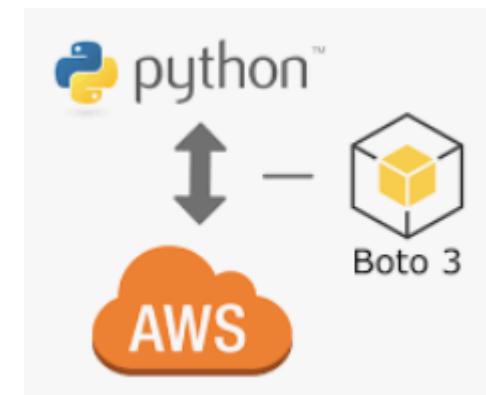


Python Basics–Refresher

5. Sample Example from Boto3

Response Syntax

```
{  
    'Buckets': [  
        {  
            'Name': 'string',  
            'CreationDate': datetime(2015, 1, 1)  
        },  
    ],  
    'Owner': {  
        'DisplayName': 'string',  
        'ID': 'string'  
    }  
}
```



https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3/client/list_buckets.html#

Python Basics—Refresher

6. Data Types – List

- Lists in Python can be created by just placing the sequence inside the **square brackets []**
 - A single list may contain Data Types like Integers, Strings, as well as Objects.
 - List in Python are ordered and have a definite count. The elements in a list are indexed with 0 being the first index.
 - **slice(start, stop, step)**
 - Reverse [:: -1]
-

```
list = [1, 4, 'For', 6, 'Anisha']
```

.....

7. Nested List

Nested List => nestedList = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Python Basics—Refresher

8. Data Types – List and Dictionary

Python
Dictionary

9. Data Type Determination

```
response = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
print(type(response))
```

Response Syntax

```
{  
    'Buckets': [  
        {  
            'Name': 'string',  
            'CreationDate': datetime(2015, 1, 1)  
        },  
        ],  
    'Owner': {  
        'DisplayName': 'string',  
        'ID': 'string'  
    }  
}
```



https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/s3/client/list_buckets.html#

Python Basics—Refresher

10. Function

Python Function

A function is a block of code which runs when it is called.

Syntax:

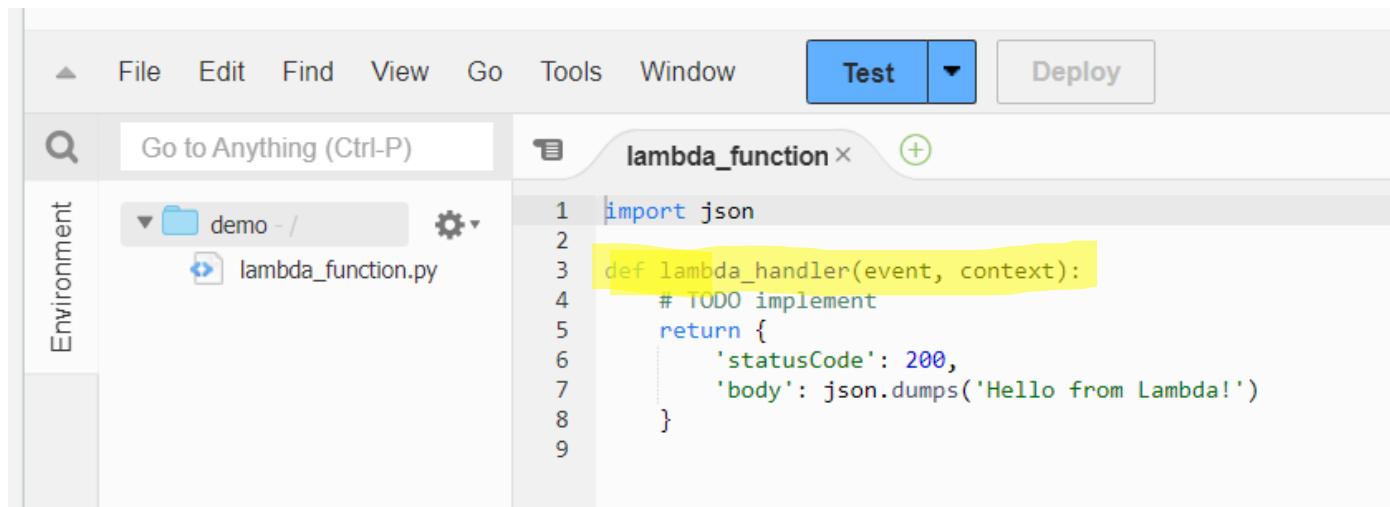
```
def function_name (argument/parameters):
```

```
    return expression or value
```

Example

```
# A simple Python function to check whether x is even or odd
```

```
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
# Driver code to call the function
evenOdd(2)
evenOdd(3)
```



The screenshot shows a Python IDE interface with the following details:

- Toolbar:** File, Edit, Find, View, Go, Tools, Window, Test (highlighted in blue), Deploy.
- Search Bar:** Go to Anything (Ctrl-P).
- Environment:** Shows a folder named "demo - /" and a file named "lambda_function.py".
- Code Editor:** A tab titled "lambda_function" is open. The code is as follows:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

AWS Lambda and Python – Beginner to Advanced

Section 4

*AWS Lambda – Create S3, EC2 and DynamoDB
resources using Lambda (Boto3)*

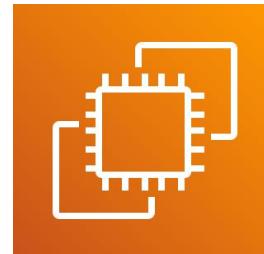
AWS Lambda and Python – Beginner to Advanced

Section 4: AWS Lambda – Create S3, EC2 and DynamoDB resources using Lambda (Boto3)

- AWS Lambda with S3 (Create new Bucket , List all the Buckets and Delete Bucket)
- AWS Lambda with EC2 (Create EC2 and Start/Stop)
- AWS Lambda with DynamoDB (Create Table and Put Items)



S3



EC2



DynamoDB

AWS Lambda Hands on - Basics of Lambda Function

AWS SDK for Python (Boto3)

- The **AWS SDK for Python is Boto3**
- **Boto3 library makes it easy to integrate with AWS services** including Amazon S3, Amazon EC2, Amazon DynamoDB, and more **using Python.**

Boto3 has two distinct levels of APIs.

Client APIs

- Clients provide a **low-level interface to the AWS service.**
- All AWS service operations supported by clients

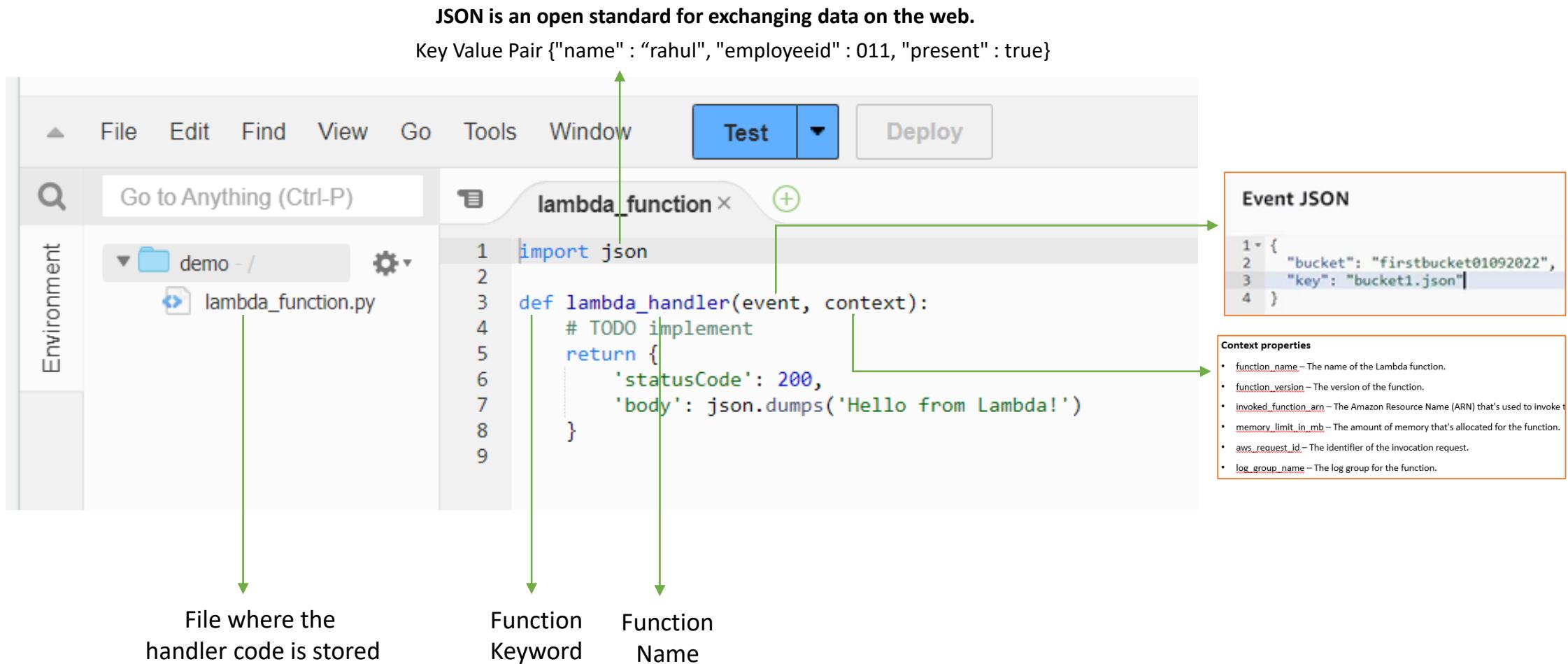
```
import boto3  
  
client = boto3.client('s3')
```

Resource APIs

- Resources are a higher-level abstraction compared to clients.
- Few AWS service operations not supported

```
import boto3  
  
# Let's use Amazon S3  
s3 = boto3.resource('s3')
```

AWS Lambda Hands on - Basics of Lambda Function



The **Lambda function *handler*** is the method in your function code that processes events. When your **function is invoked**, Lambda runs the **handler method**.



Runtime
Python 3.9

Handler Info
lambda_function.lambda_handler

Architecture Info

x86_64

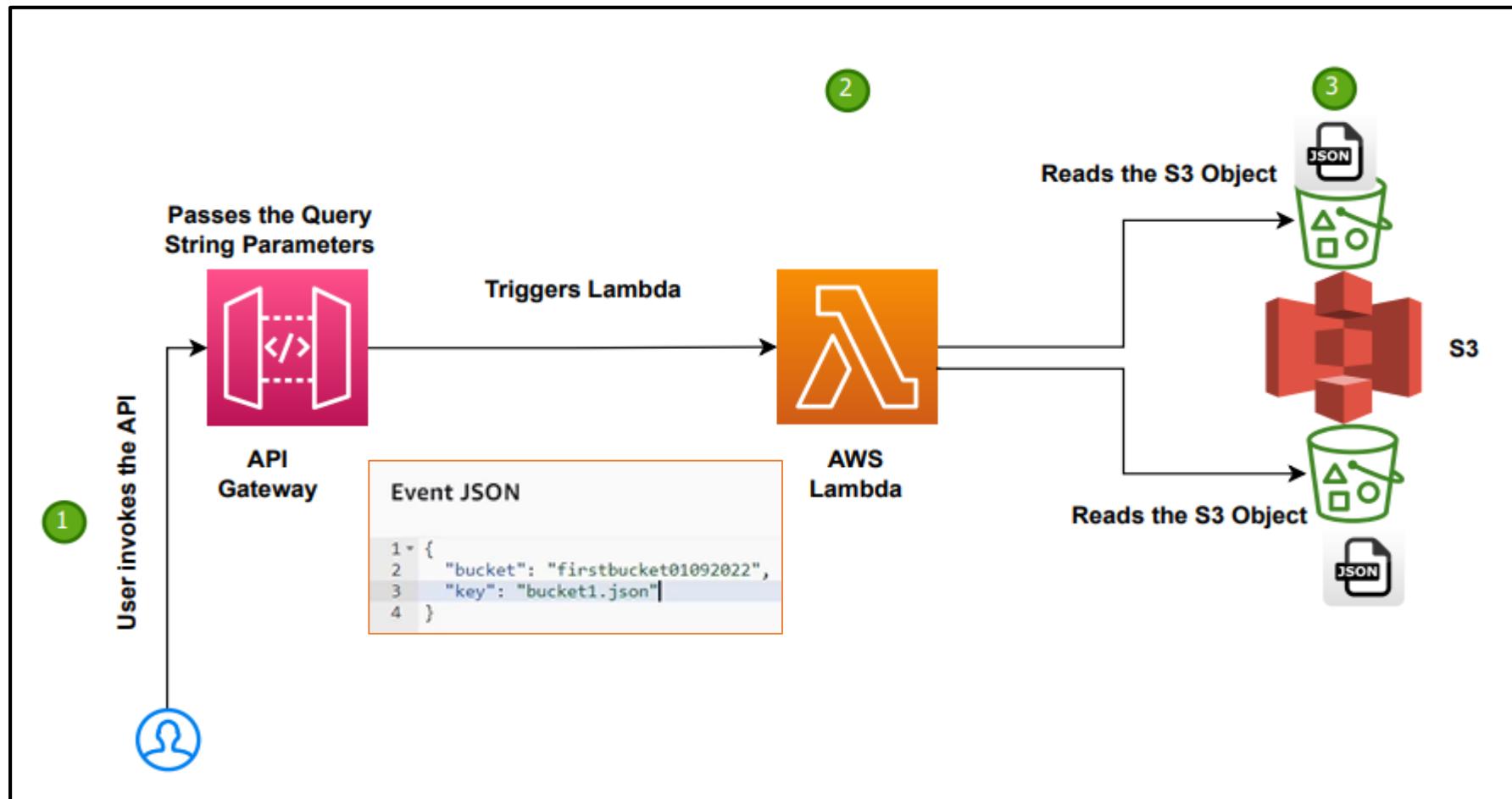
P.S : The content is created by Rahul Trisal and copyrighted

AWS Lambda Hands on - Basics of Lambda Function

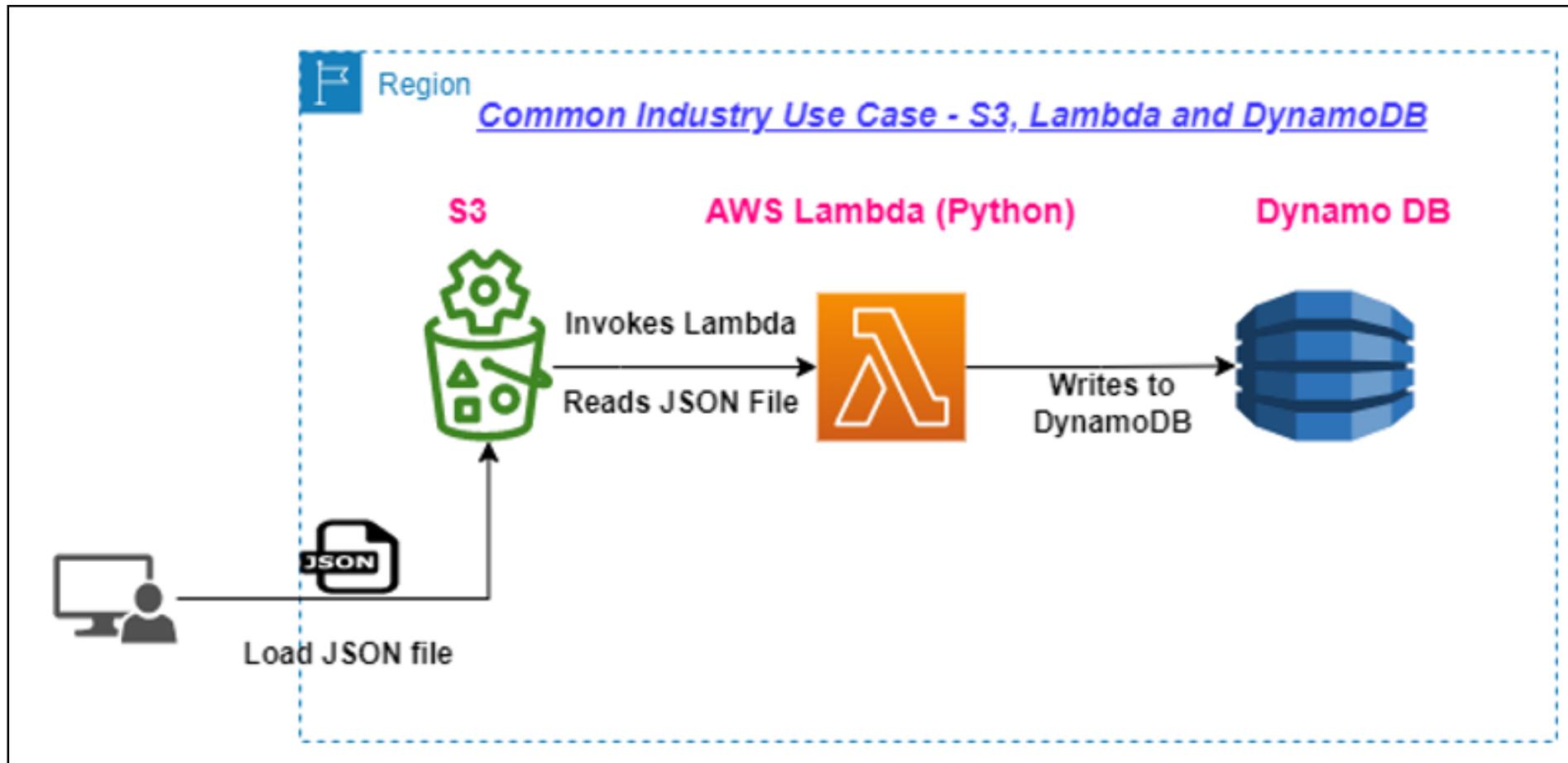
Context properties

- `function_name` – The name of the Lambda function.
- `function_version` – The version of the function.
- `invoked_function_arn` – The Amazon Resource Name (ARN) that's used to invoke the function. Indicates if the invoker specified a version number or alias.
- `memory_limit_in_mb` – The amount of memory that's allocated for the function.
- `aws_request_id` – The identifier of the invocation request.
- `log_group_name` – The log group for the function.
- `log_stream_name` – The log stream for the function instance.
- `identity` – (mobile apps) Information about the Amazon Cognito identity that authorized the request.
- `cognito_identity_id` – The authenticated Amazon Cognito identity.
- `cognito_identity_pool_id` – The Amazon Cognito identity pool that authorized the invocation.
- `client_context` – (mobile apps) Client context that's provided to Lambda by the client application.
- `client.installation_id`
- `client.app_title`
- `client.app_version_name`
- `client.app_version_code`
- `client.app_package_name`
- `custom` – A dict of custom values set by the mobile client application.
- `env` – A dict of environment information provided by the AWS SDK.

AWS Lambda Hands on - Basics of Lambda Function



AWS Lambda Hands on - Basics of Lambda Function



Create S3 using Lambda and Python Boto3 SDK

Create Below Resources through Lambda

S3

- Create new Bucket
- Delete Bucket
- List all the buckets (Inventory Collection within an AWS Account)



S3

Use Case Scenario

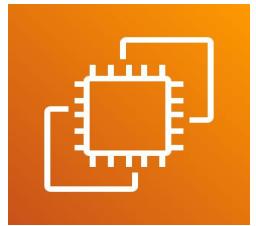
- List the inventory of all the S3 buckets in the Organization AWS Account (Sandbox, Dev etc.)

Create EC2 using Lambda and Python Boto3 SDK

Create Below Resources through Lambda

EC2

- Create new EC2
- Stop EC2 Instance
- Start EC2 Instance



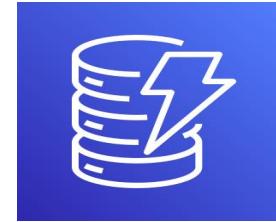
EC2

Create DynamoDB Table using Lambda & Python Boto3 SDK

Create Below Resources through Lambda

DynamoDB

- Create DynamoDB Table
- Insert data into the DynamoDB Table using Put Item method



DynamoDB

AWS Lambda Hands on – Step by Step Guide - 1

1. Go to AWS Account, Search for AWS Lambda Service from Console and click on ‘Create Function’

- Select Author from Scratch
- Function Name – Meaningful name of the function based on the usage
- Runtime – The programming language you intent to write the code in
- Architecture - x86_64
- Default Execution Role – Permission only to CloudWatch logs
- Create Function

AWS Lambda Hands on – Step by Step Guide - 2

2. Increase timeout limit

- Go to Configuration --- > General Configuration --- > Change Timeout limit to – 1 min

3. Create/Enhance the IAM Role

- Go to Configuration --- > Permission--- > Click on Role name --- > Add Permission

4. Go to ‘Code’ Tab and Search for Boto3 Documentation

5. ‘import boto3’ in the lambda function code

```
import boto3  
  
client = boto3.client('s3')
```

6. Create the Client connection with the service being provisioned

AWS Lambda Hands on – Step by Step Guide - 3

7. Search the API method for AWS Service being created

These are the available methods:

- [abort_multipart_upload](#)
- [can_paginate](#)
- [close](#)
- [complete_multipart_upload](#)
- [copy](#)
- [copy_object](#)
- [create_bucket](#)
- [create_multipart_upload](#)
- [delete_bucket](#)
- [delete_bucket_analytics_configuration](#)

8. Write the code for AWS Service to be created

- Copy the Request Syntax
- Identify the relevant attributes to be used to create the AWS Service based on your requirement
- Review the mandatory and optional attributes

Request Syntax

```
response = client.create_bucket(  
    ACL='private'|'public-read'|'public-read-write'|'authenticated-read',  
    Bucket='string',  
    CreateBucketConfiguration={  
        'LocationConstraint': 'af-south-1'|'ap-east-1'|'ap-northeast-1'|'ap-northeast-2'|'ap-nort  
    },  
    GrantFullControl='string',  
    GrantRead='string',  
    GrantReadACP='string',  
    GrantWrite='string',  
    GrantWriteACP='string',  
    ObjectLockEnabledForBucket=True|False,  
    ObjectOwnership='BucketOwnerPreferred'|'ObjectWriter'|'BucketOwnerEnforced'  
)
```

AWS Lambda Hands on – Step by Step Guide - 4

9. Evaluate the Response

- Response parameters
- Response Type such as Dictionary

RETURN TYPE:

dict

RETURNS:

Response Syntax

```
{  
    "Location": "string"  
}
```

10. Identify the Response parameters to be printed

- Evaluate the attributes that will be returned
- Print the response based on type (Dictionary and so) using print function

AWS Lambda Hands on – Step by Step Guide - 5

11. Click on Deploy to save changes

12. Create a Test Event

13. Test the Function

14. Verify from console if the AWS Service is created as expected.

AWS Lambda and Python – Beginner to Advanced

Section on

AWS Lambda – Basic Concepts (Part 2)

AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Basic Concepts – Part 2

- ❖ Evolution from Physical Servers to AWS Lambda
- ❖ What is AWS Lambda - Architecture and Use Cases
- ❖ Lambda Console Walkthrough
- ❖ Lambda Execution Role
- ❖ AWS Lambda Limits - Timeout
- ❖ **AWS Lambda Invocation Model - Theory**
- ❖ **AWS Lambda Invocation Model – Hands-On**
- ❖ **AWS Lambda Limits – Memory**

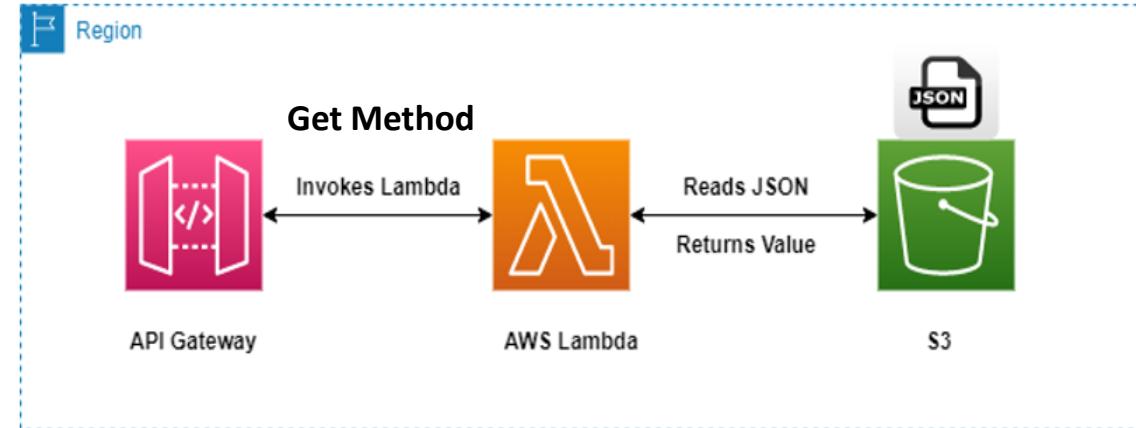
AWS Lambda and Python – Beginner to Advanced

AWS Lambda Execution/Invocation Model

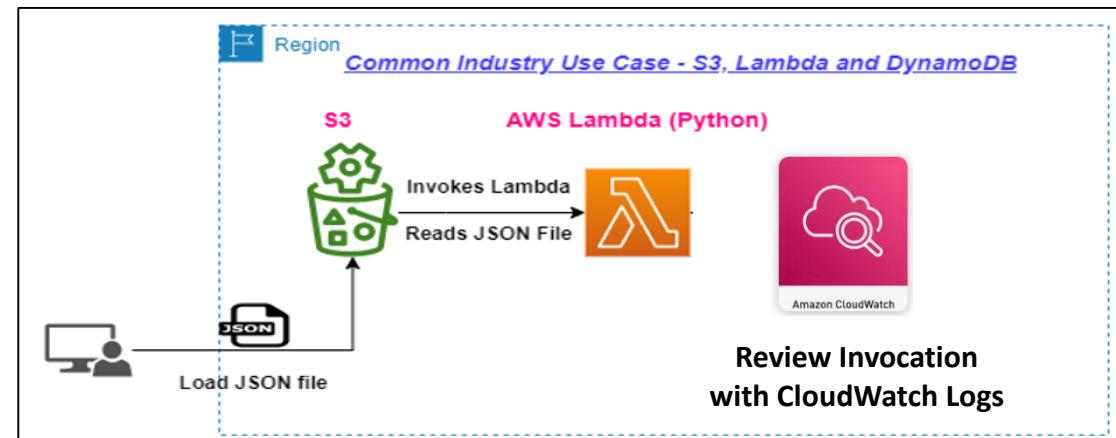
Lambda execution model



1. Synchronous Invocation – API Gateway with AWS Lambda

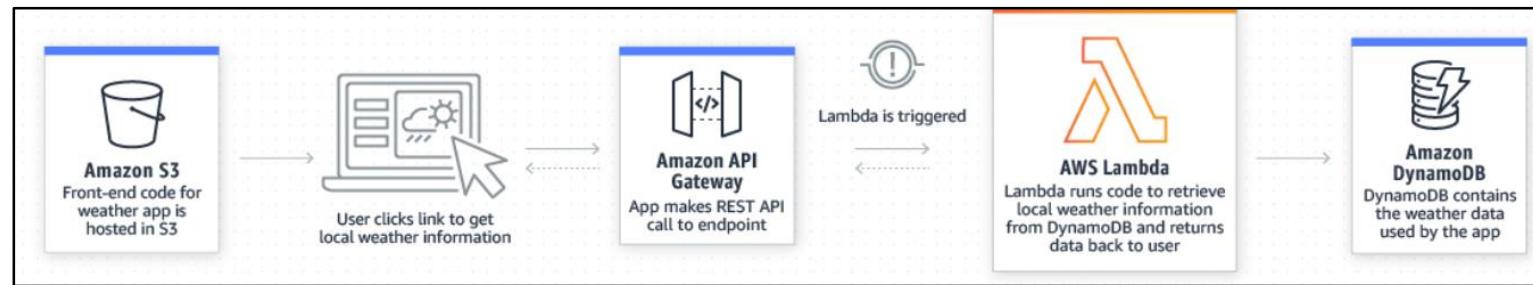


2. Asynchronous Invocation – S3 with AWS Lambda



AWS Lambda and Python – Beginner to Advanced

1. Synchronous Invocation – API Gateway with AWS Lambda



2. Asynchronous Invocation – S3 with AWS Lambda



3. Stream based (Polling) Invocation – Kinesis or SQS with AWS Lambda



AWS Lambda and Python – Beginner to Advanced

AWS Lambda Limits - Memory

- Lambda allocates **CPU power in proportion to the amount of memory configured.**
- Default Memory – **128 MB**
- Maximum Memory - **10,240 MB**, in 1-MB increments.
- At **1,769 MB**, a function has the equivalent of **one vCPU**.
- More memory = more CPU
- **Lambda Pricing** – **Number of invocations * Memory Allocated * Execution Time**



AWS Lambda and Python – Beginner to Advanced

AWS Lambda Limits - Memory

Chart based on per 1000 Invocations

- One-thousandth of a cent cost difference, the function has a 10-fold improvement in performance.
- Choosing the memory allocated to Lambda functions is an optimization process that balances speed (duration) and cost.
- [AWS Lambda Power Tuning](#) tool allows to automate the process.

Memory	Duration	Cost
128 MB	11.722 s	\$0.024628
256 MB	6.678 s	\$0.028035
512 MB	3.194 s	\$0.026830
1024 MB	1.465 s	\$0.024638

Best Practice

- Derive the balance between Function Time out and Memory allocated (think about the edge cases)
- Over-Provision Memory, If required but Not Function Timeout

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Limits - Memory

Chart based on per 1000 Invocations

- One-thousandth of a cent cost difference, the function has a 10-fold improvement in performance.
- Choosing the memory allocated to Lambda functions is an optimization process that balances speed (duration) and cost.
- [AWS Lambda Power Tuning](#) tool allows to automate the process.

Memory	Duration	Cost
128 MB	11.722 s	\$0.024628
256 MB	6.678 s	\$0.028035
512 MB	3.194 s	\$0.026830
1024 MB	1.465 s	\$0.024638

Best Practice

- Derive the balance between Function Time out and Memory allocated (think about the edge cases)
- Over-Provision Memory, If required but Not Function Timeout

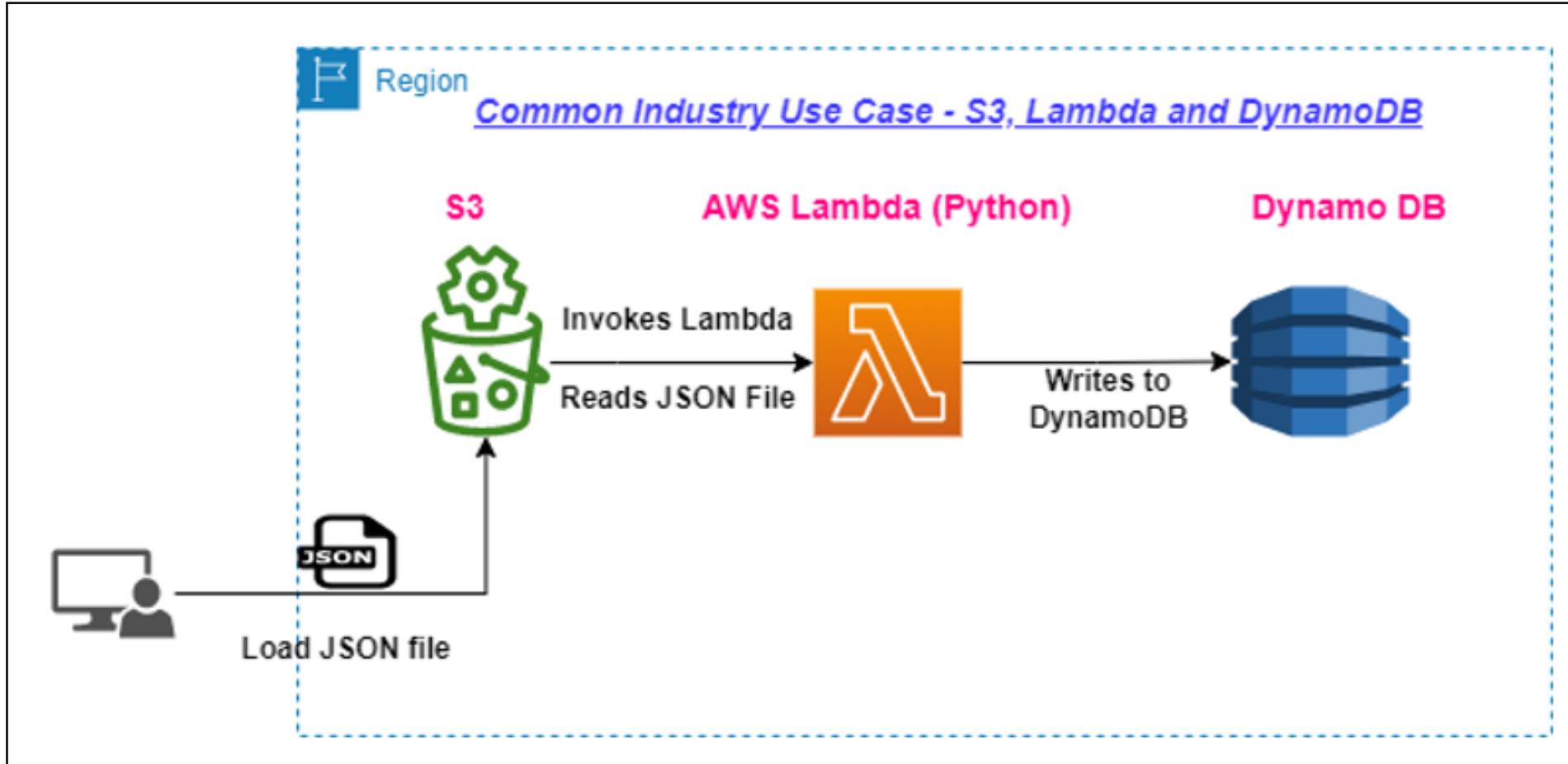
AWS Lambda and Python – Beginner to Advanced

Section on

*Real World Serverless Use Case 1 - using S3, AWS
Lambda and DynamoDB*

Serverless Architecture – 1

AWS Lambda and Python – Target Architecture



Very Important :

Please change the name of the S3 Bucket and DynamoDB Table with your own and substitute the names in the Lambda Python Code.

The S3 bucket names need to be globally unique and same names that were highlighted in the video will not work.

P.S. The content is created by Rahul Trivedi and copyrighted.

AWS Lambda and Python – Beginner to Advanced

Section on
API Gateway Overview

AWS API Gateway - Overview

What is API Gateway

Amazon API Gateway is an AWS service for **creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale.**



Sample Architecture - RESTful microservices



AWS API Gateway - Overview

Key Features Of API Gateway

1. API Types

1. HTTP API

Build **low-latency and cost-effective REST APIs** with built-in features such as OIDC and OAuth2, and native CORS support.

2. WebSocket API

Build a WebSocket API using **persistent connections for real-time use cases such as chat applications or dashboards**.

3. REST API



Develop a REST API where you gain complete control over the request and response along with **API management capabilities**.

4. REST API (Private)

Create a REST API that is **only accessible from within a VPC**.

AWS API Gateway - Overview

Key Features of API Gateway

REST API's vs HTTP API's

- REST APIs support **more features** than HTTP APIs, while HTTP APIs are designed with **minimal features** and offered at a **lower price**.
 - Choose **REST APIs** if you need features such as **API keys, per-client throttling, request validation, AWS WAF integration, or private API endpoints**.
 - Choose **HTTP APIs** if you don't need the features included with **REST APIs**.
-
- <https://aws.amazon.com/blogs/compute/introducing-amazon-api-gateway-private-endpoints/>

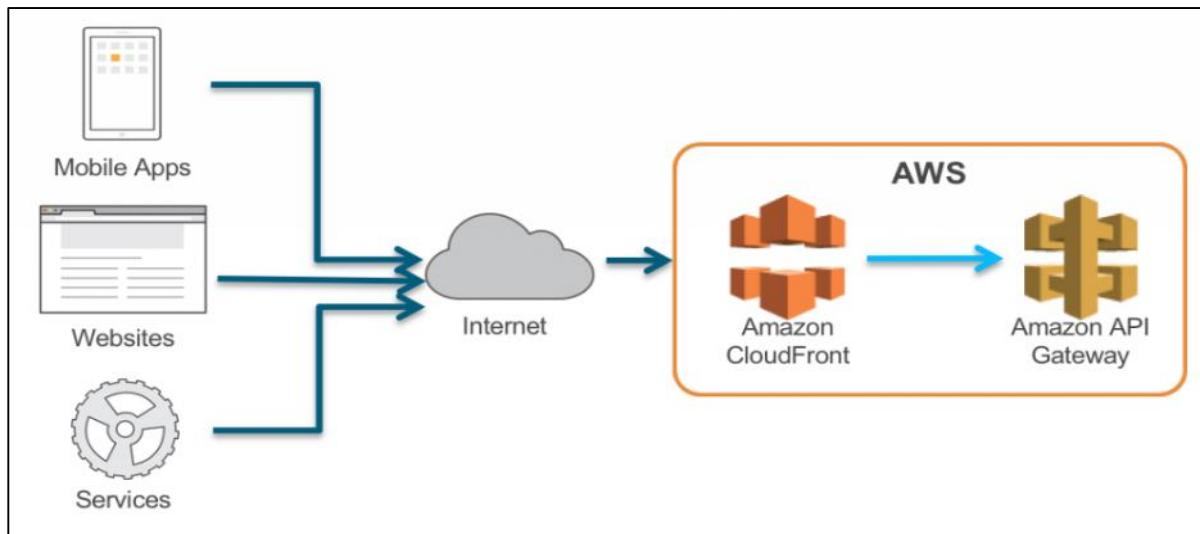
API Gateway - Overview

2. API Endpoint Types

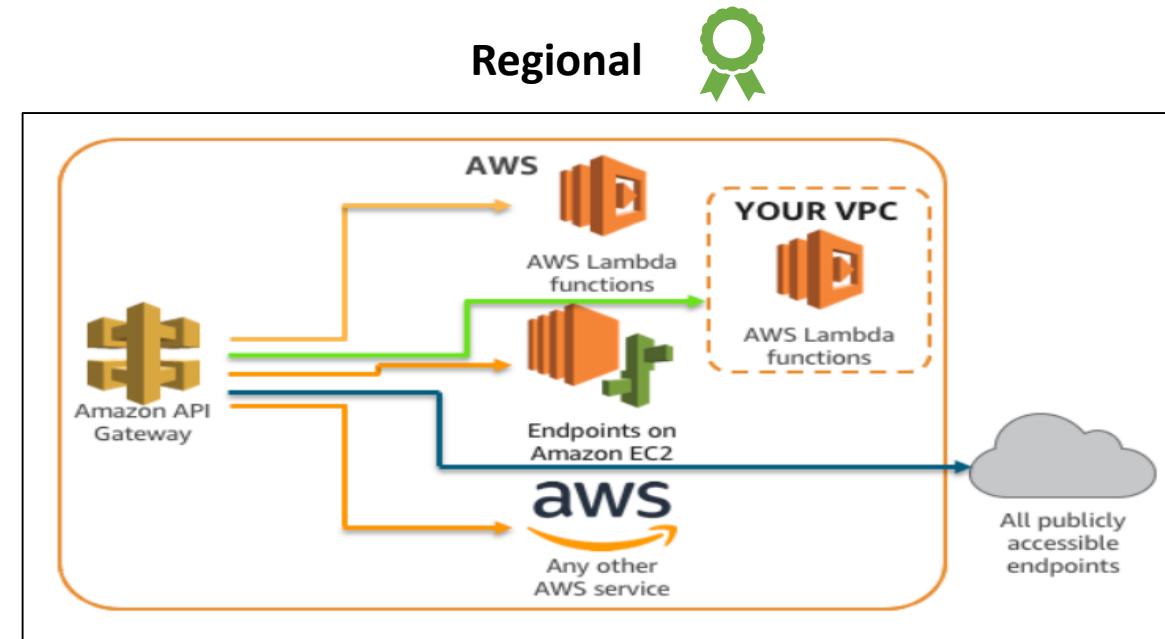
An **API endpoint type** refers to the **hostname of the API**.

The API endpoint type can be ***edge-optimized*, *regional*, or *private***, depending on where the majority of your API traffic originates from.

Edge Optimized

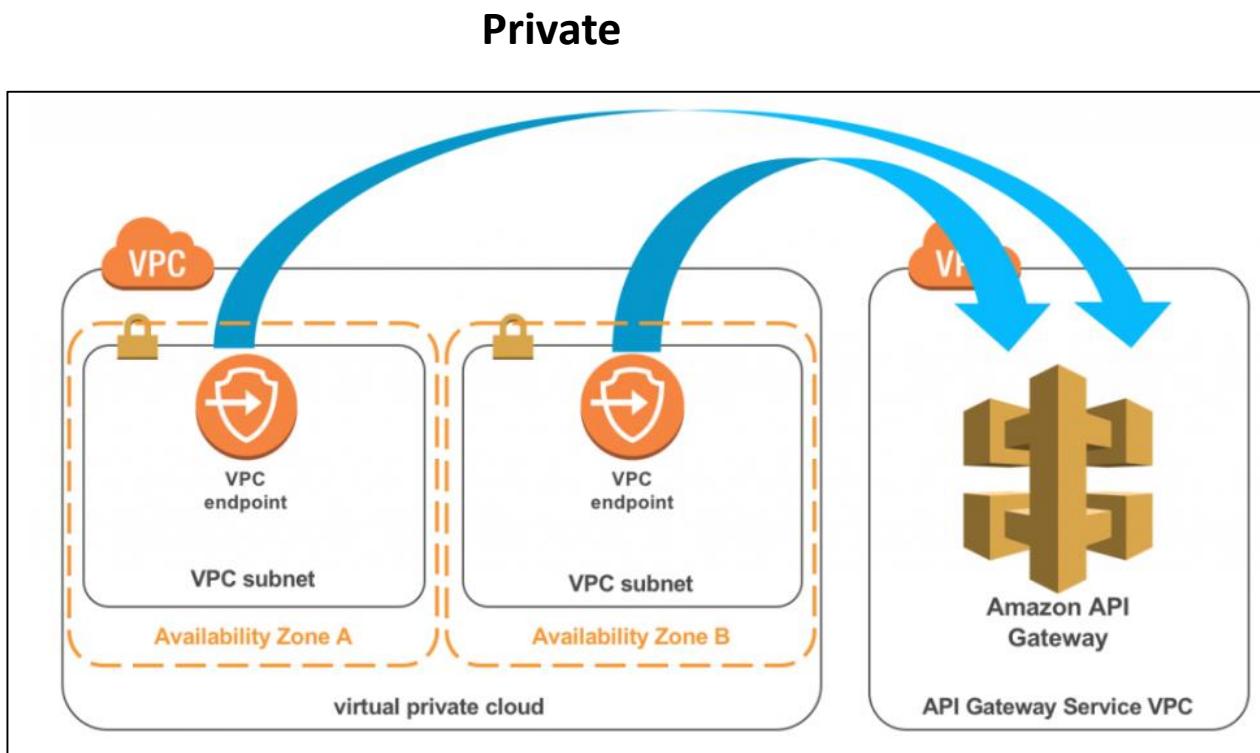


Regional



API Gateway - Overview

2. API Endpoint Types



API Gateway - Overview

3. Resources



REST architecture treats every content as a **resource**.

These resources can be Text Files, Html Pages, Images, Videos or Business Data.

4. Methods



Method	Description
GET	Retrieve information about the REST API resource
POST	Create a REST API resource
PUT	Update a REST API resource
DELETE	Delete a REST API resource or related component

API Gateway - Overview

5. Integration Type

Lambda Function



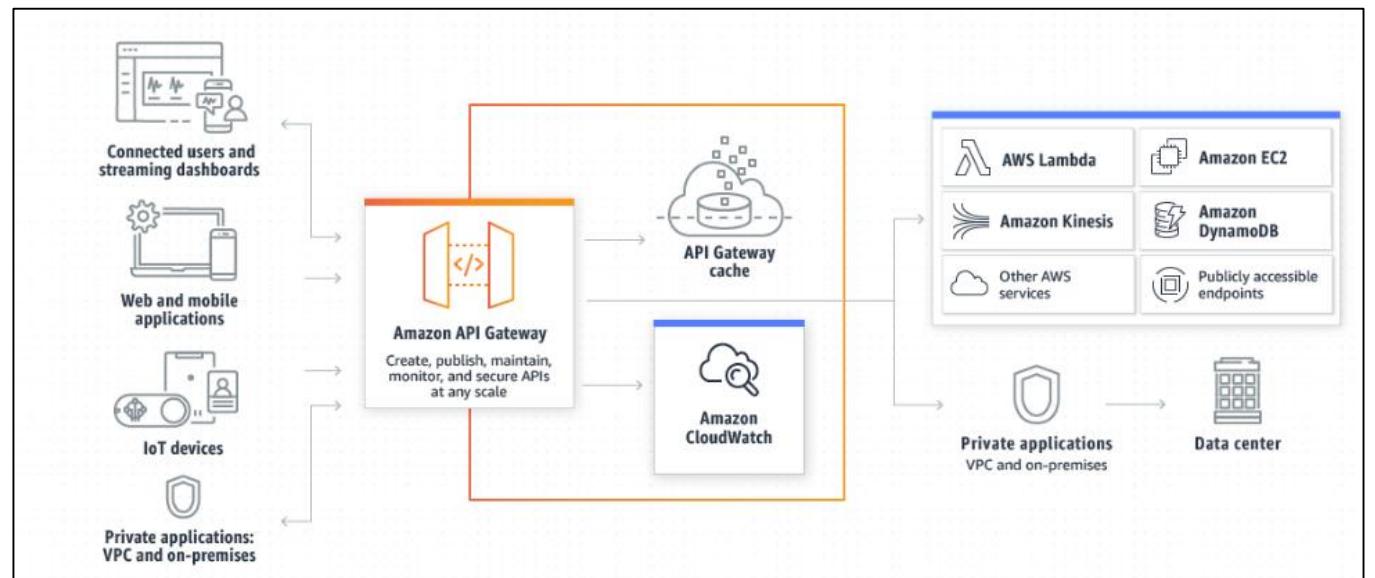
This type of integration lets an API method be integrated with the Lambda function

HTTP

- Allows integration with existing HTTP endpoint
- This type of integration lets an API expose HTTP endpoints in the backend.

Mock

- This type of integration lets API Gateway return a response without sending the request further to the backend.



API Gateway - Overview

5. Integration Type

AWS Service

This type of integration lets an API expose AWS service actions.

VPC Link

A VPC link is a resource in [Amazon API Gateway](#) that allows for connecting API routes to private resources inside a VPC

API Gateway - Overview

5. Deployment



After **creating your API**, you must **deploy it** to make it callable by your users.

To **deploy an API**, you **create an API deployment** and **associate it with a stage**.

Important Tip :

Every time you update an API, you must redeploy the API to an existing stage or to a new stage.

Updating an API includes modifying routes, methods, integrations, authorizers, and anything else other than stage settings.

6. Stages



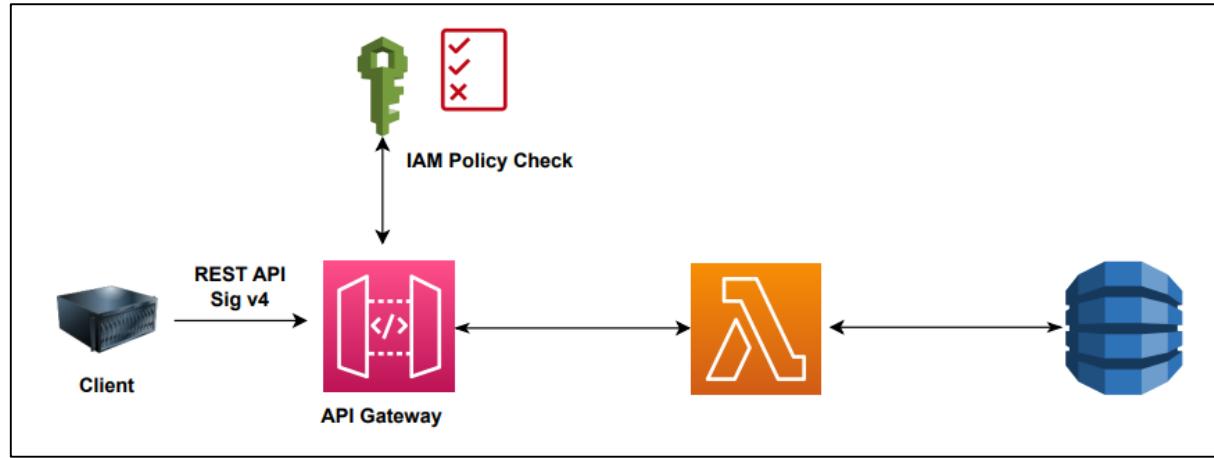
A logical reference to a **lifecycle state of your API** (for example, 'dev', 'prod', 'beta', 'v2').

<https://{{restapi-id}}.execute-api.{{region}}.amazonaws.com/{{stageName}}>

<https://ent94mc14j.execute-api.us-east-1.amazonaws.com/dev/students>

API Gateway - Overview

7. AWS API Gateway - Authentication and Authorization

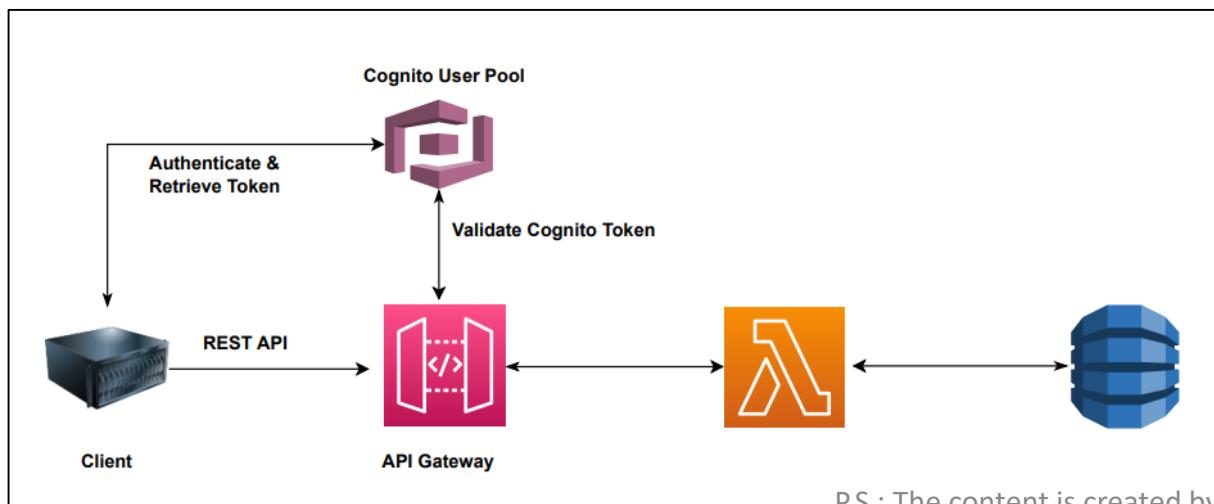


1. API Security – IAM

Authentication – IAM

Authorization – IAM Policy

Use Case : Internal AWS Services, Cross Account Access



2. API Security – Cognito

Authentication – Cognito User Pool

Authorization – API Gateway Methods

Use Case : External Users for Web/Mobile Apps

API Gateway - Overview

7. AWS API Gateway - Authentication and Authorization



3. API Security – Lambda Authorizer

Authentication – External

Authorization – Lambda Function

Use Case : Third Party Identity Provider such as OAuth 2.0

API Gateway - Overview

8. Usage Plans



Usage Plans

Use Case : Differentiate between Basic and Premium Customers

Sets the **target request rate - Throttling, Burst and Quota Limit for each API Key**

9. API Keys



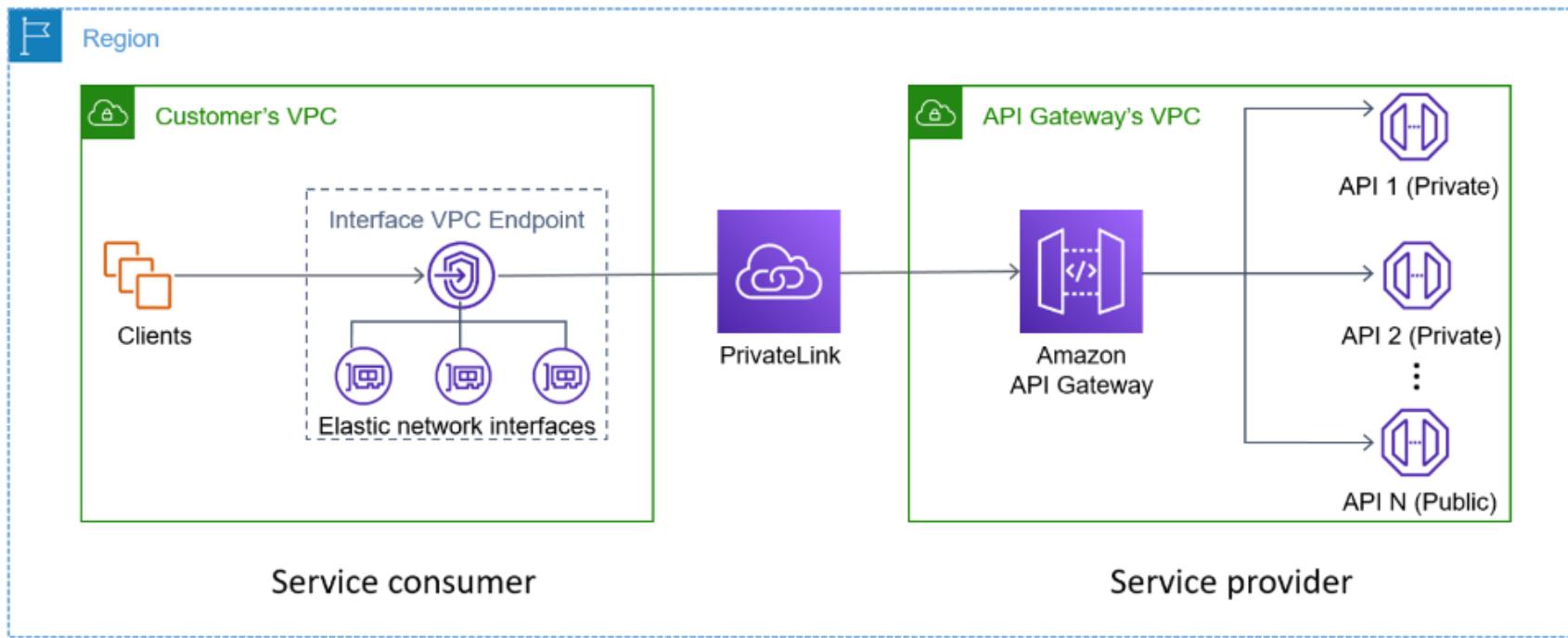
- An **alphanumeric string** that API Gateway uses to identify an app developer who uses your REST or WebSocket API.
- You can use API keys together with [Lambda authorizers](#) or [usage plans](#) to control access to your APIs.

API Gateway - Overview

10. API Integration and Private API's

Private API's

A private API means that the **API endpoint is reachable only through the VPC**. Private APIs are **accessible only from clients within the VPC or from clients that have network connectivity to the VPC**.

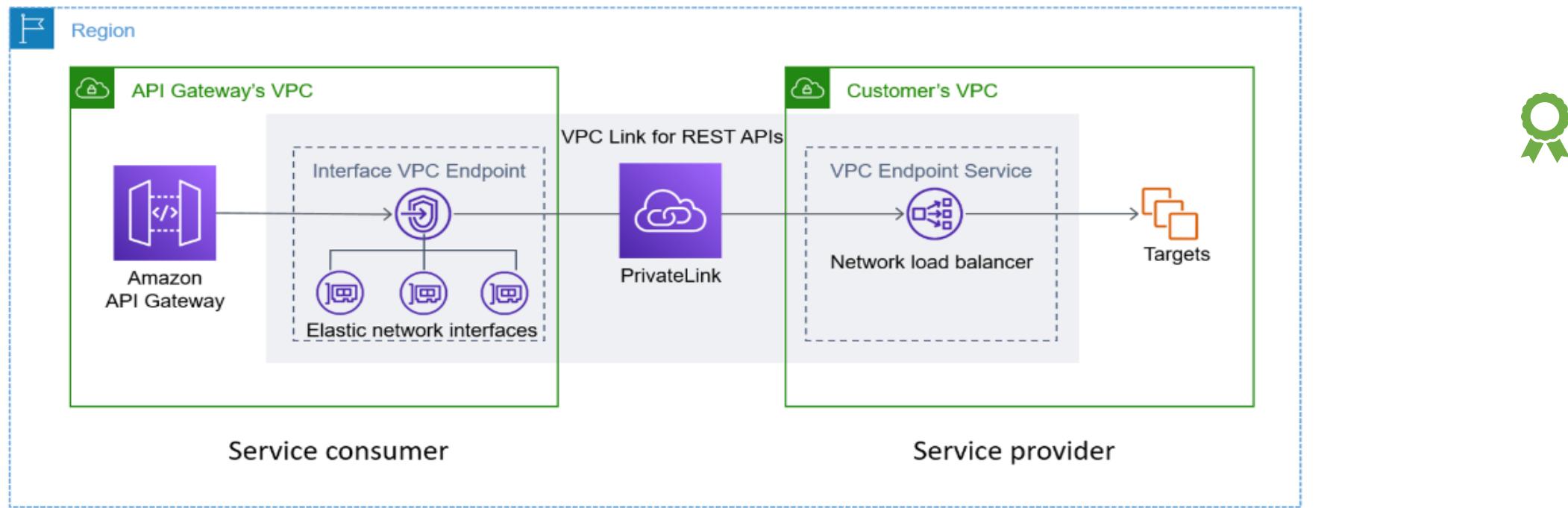


API Gateway - Overview

10. API Integration, Private API's and VPC Link

Private Integration

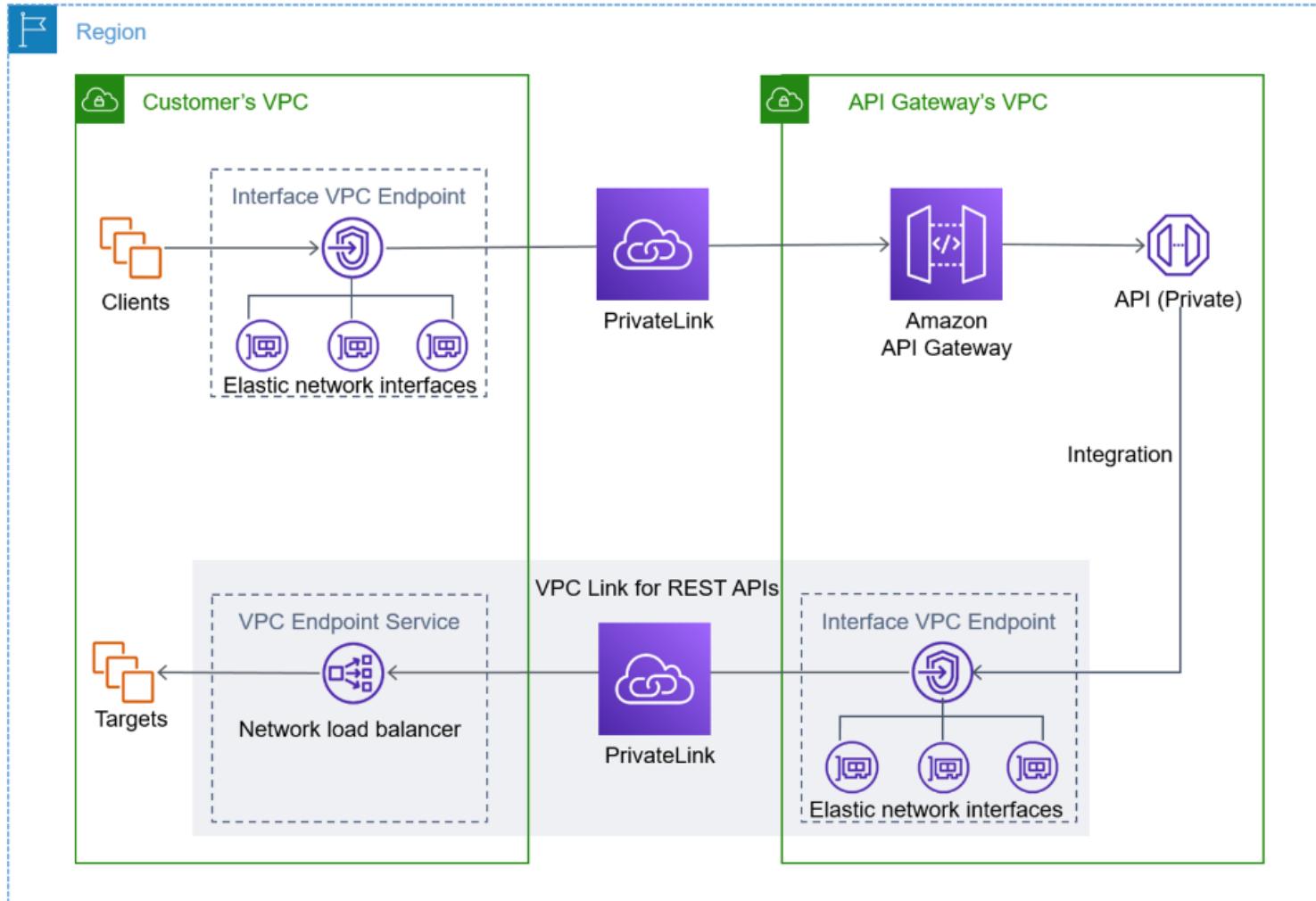
An API Gateway integration type for a client to access resources inside a customer's VPC through a private REST API endpoint without exposing the resources to the public internet.



Source : <https://aws.amazon.com/blogs/compute/understanding-vpc-links-in-amazon-api-gateway-private-integrations/>

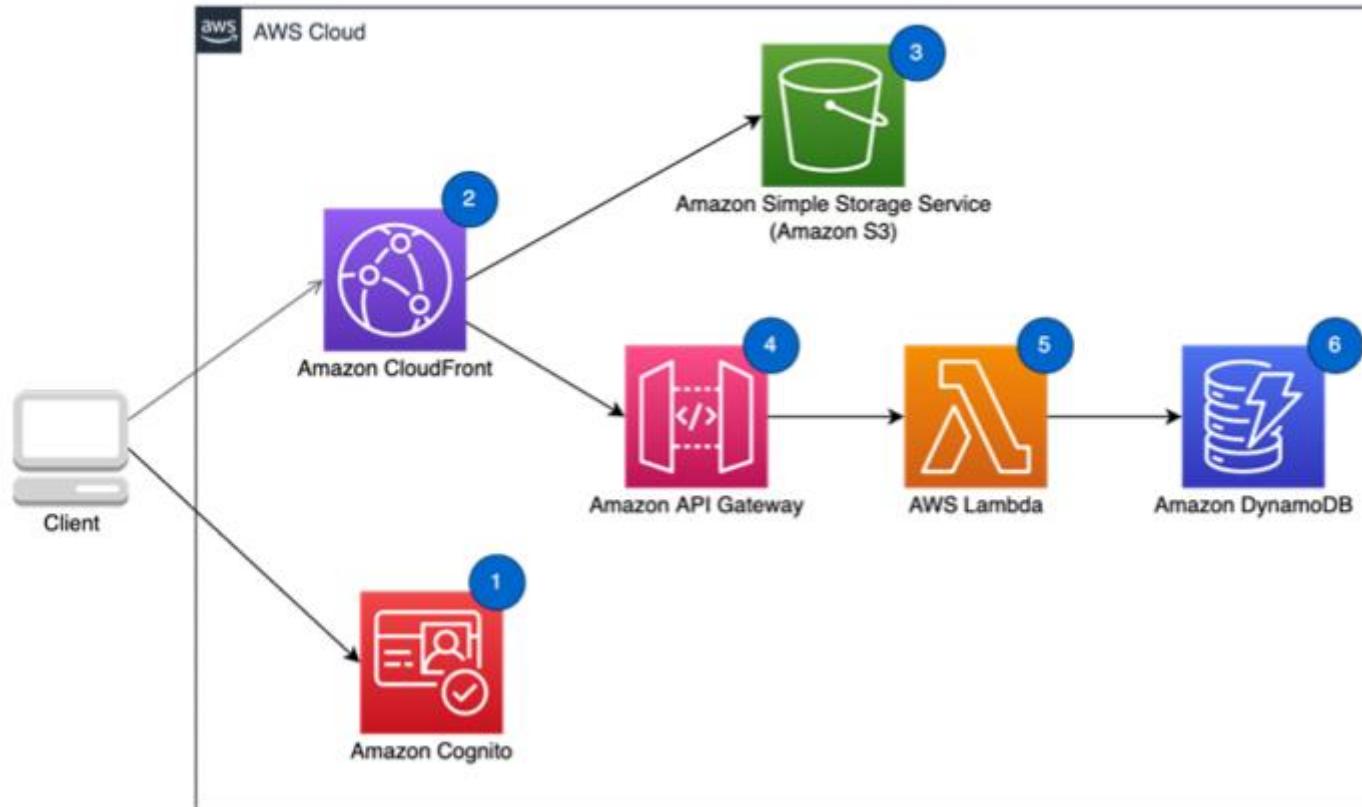
API Gateway - Overview

10. API Integration and Private API's



API Gateway - Overview

Sample Architecture

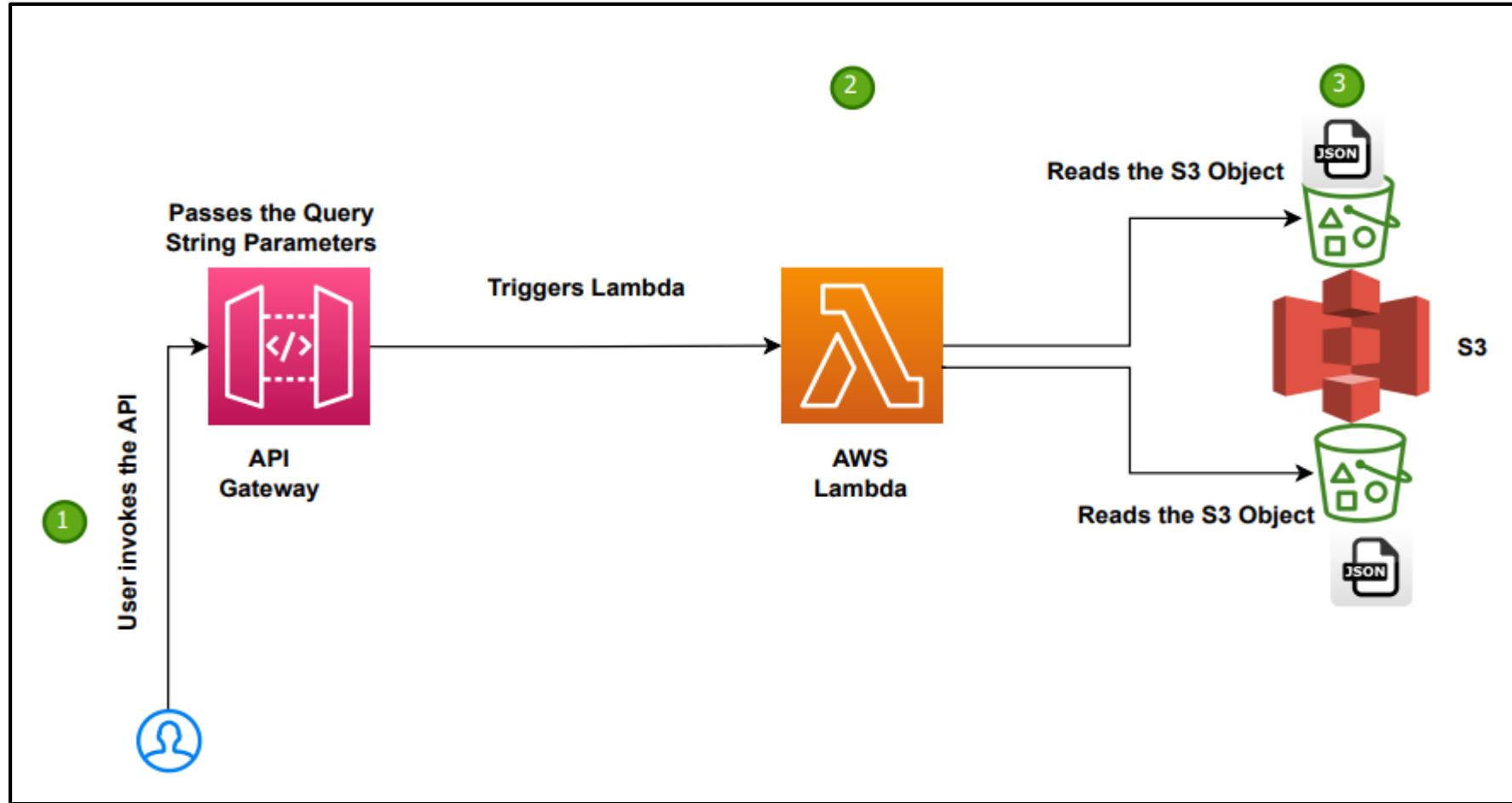


AWS Lambda and Python – Beginner to Advanced

Section on
*Real World Serverless Use Case 2 - using API Gateway,
AWS Lambda and S3*

Serverless Architecture – 2:

Target Architecture – API Gateway, Lambda and S3



Resources to be Created

- **S3 buckets – 2**
- **Lambda Function**
- **API Gateway**

Very Important :

Please change the name of the S3 Bucket with your own and substitute the names in the Lambda Python Code.
The S3 bucket names need to be globally unique and same names that were highlighted in the video will not work.

P.S. The content is created by Rohit Trivedi and copyrighted.

AWS Lambda and Python – API Keys

API Keys

- API keys are **alphanumeric string values** that you **distribute to application developer customers** to grant **access to your API**.
- An API key has a **name and a value**.

Params	Authorization	Headers (6)	Body	Pre-request Script	Tests	Settings
Headers	5 hidden					
		KEY		VALUE		
<input checked="" type="checkbox"/>	x-api-key			3uqsR25hz7a0R7BZxjVpo3XHFyl2tGQsPj6ncswb		

- API Gateway can **generate API keys on your behalf** or **import it** into API Gateway from an **external source**.
- API Key can be used along with **Usage Plan** for Throttling and Quotas to differentiate between Basic and Premium Users
- You can use API keys together with **Lambda authorizers** or **usage plans** to control access to your APIs.
- **Don't use API keys for authentication or authorization for your APIs.**
- Instead, use an **IAM role**, **a Lambda authorizer**, or **an Amazon Cognito user pool**.

AWS Lambda and Python – Throttling and Quota

Throttling Limit

- API throttling is the process of **limiting the number of API requests a user can make in a certain period**.
- This can be set at the **API or API method level**.

The screenshot shows the AWS API Gateway Throttling configuration interface. It is divided into two main sections: **Throttling** and **Quota**.

Throttling:

- Enable throttling:
- Rate*: 50 requests per second
- Burst*: 10 requests

Quota:

- Enable quota:
- 500 requests per Month

Quota

- Quota limit sets the **target maximum number of requests with a given API key** that can be submitted within a specified time interval.
- Throttling and quota limits apply to requests for **individual API keys that are aggregated across all API stages within a usage plan**

AWS Lambda and Python – Usage Plans

A usage plan

Use Case : Differentiate between Basic and Premium Customers

- Sets the **target request rate - Throttling, Burst and Quota Limit for each API Key**
- Specifies who can **access one or more deployed API stages and methods through the API Keys**

API Key -- >Usage Plan --- > Throttling, Burst and Quota ---- >Set at [API --- > Deployment Stage -- > Resources -- > Method]

AWS Lambda and Python – Usage Plans Sample Scenario

Sample Enterprise Scenario

1. Basic Plan

- Throttling Limit – 5 Requests/Second
- Burst – 2 Requests/Second
- Quota – 150 Requests/Month
- API (Resource – Students, Method – GET)
- Stage – Beta
- API Key – x-api-key = xyz

2. Premium Plan

- Throttling Limit – 50 Requests/Second
- Burst – 20 Requests/Second
- Quota – 1500 Requests/Month
- API(Resource – Students, Method – GET)
- Stage – Prod
- API Key - abc

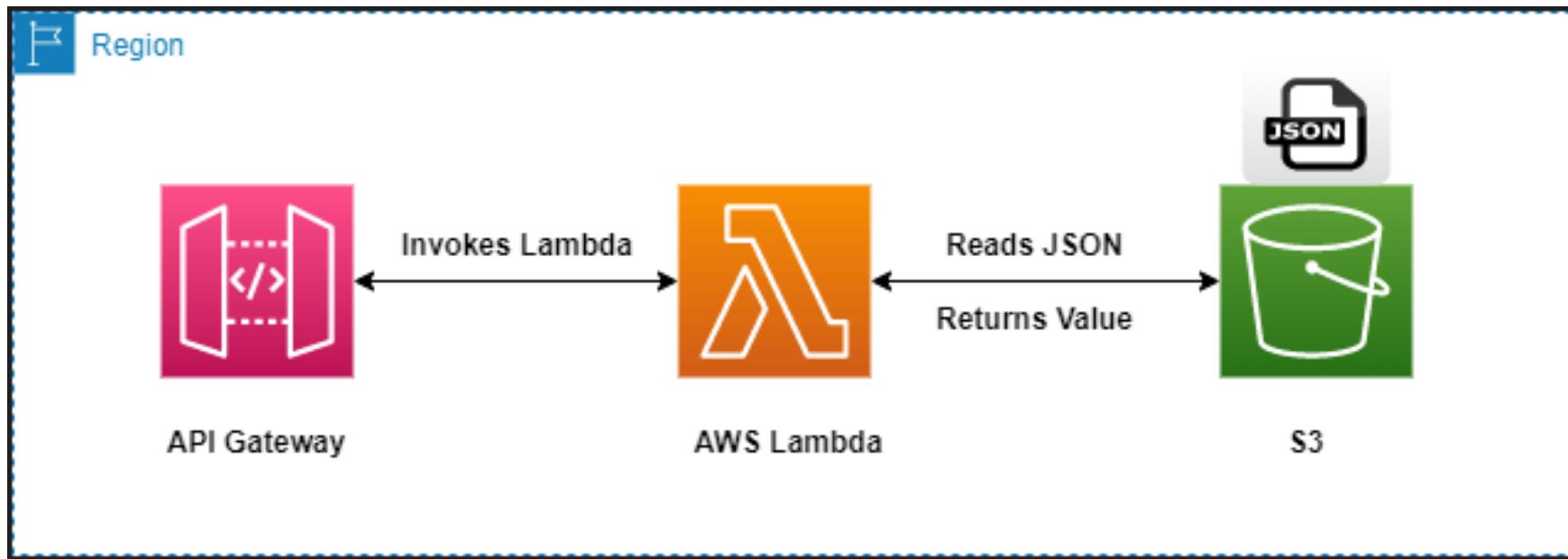
AWS Lambda and Python – Usage Plans

Steps to configure a usage plan

- Create API Gateway with one or more APIs
- Configure the Resources and methods
- Deploy the APIs to stages.
- Generate or import API keys to distribute to application developers (your customers) who will be using your API.
- Create the usage plan with the desired throttle and quota limits.
- Associate API stages and API keys with the usage plan.
- Callers of the API must supply an assigned API key in the x-api-key header in requests to the API.

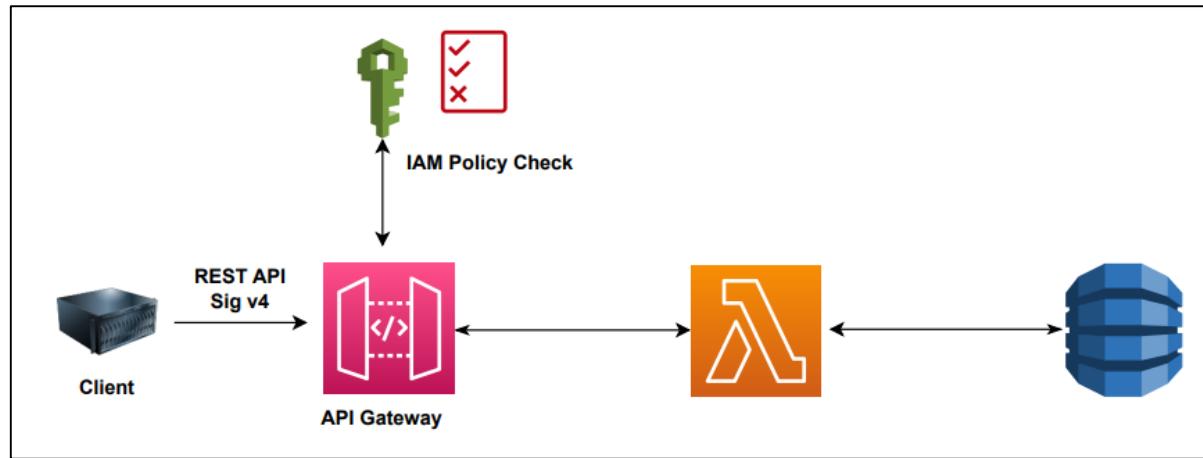
AWS Lambda Authorizer

AWS API Gateway - Authentication and Authorization using Lambda Authorizer



AWS Lambda Authorizer

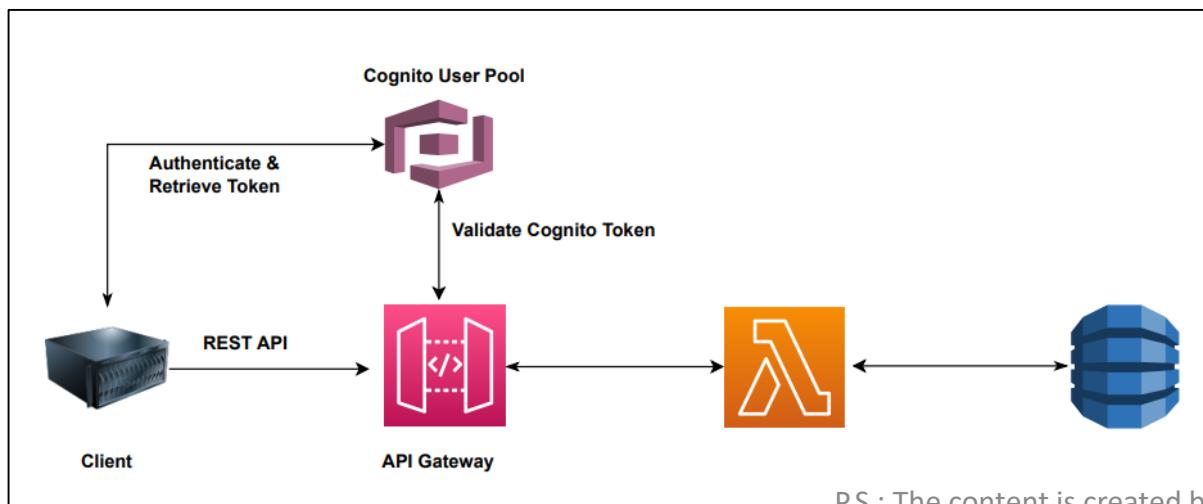
AWS API Gateway - Authentication and Authorization



1. API Security – IAM

Authentication – IAM
Authorization – IAM Policy

Use Case : Internal AWS Services, Cross Account Access



2. API Security – Cognito

Authentication – Cognito User Pool
Authorization – API Gateway Methods

Use Case : External Users for Web/Mobile Apps

AWS Lambda Authorizer

AWS API Gateway - Authentication and Authorization



3. API Security – Lambda Authorizer

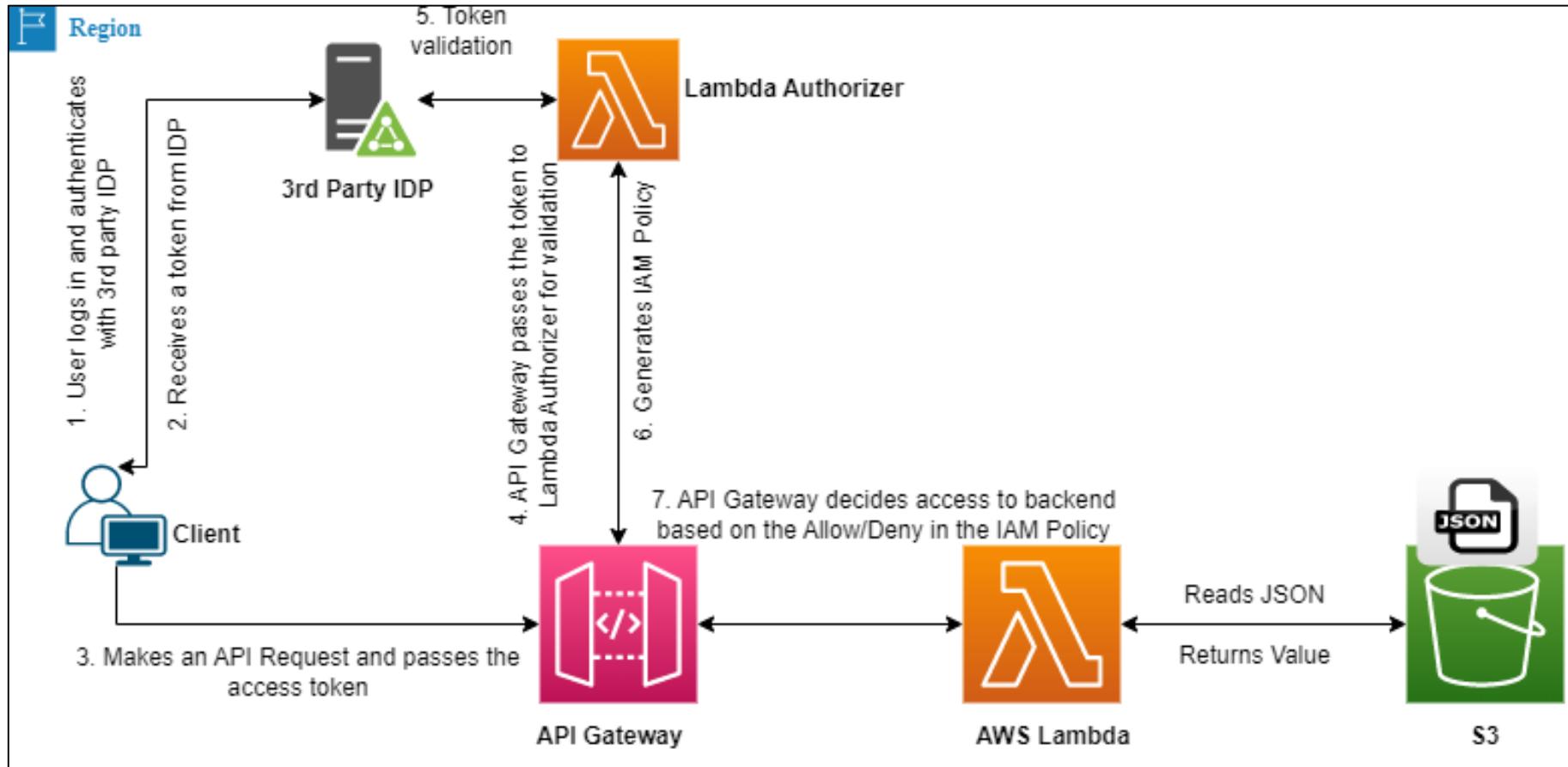
Authentication – External

Authorization – Lambda Function

Use Case : Third Party Identity Provider such as OAuth 2.0

AWS Lambda Authorizer

AWS API Gateway - Authentication and Authorization using Lambda Authorizer



Lambda Authorizer

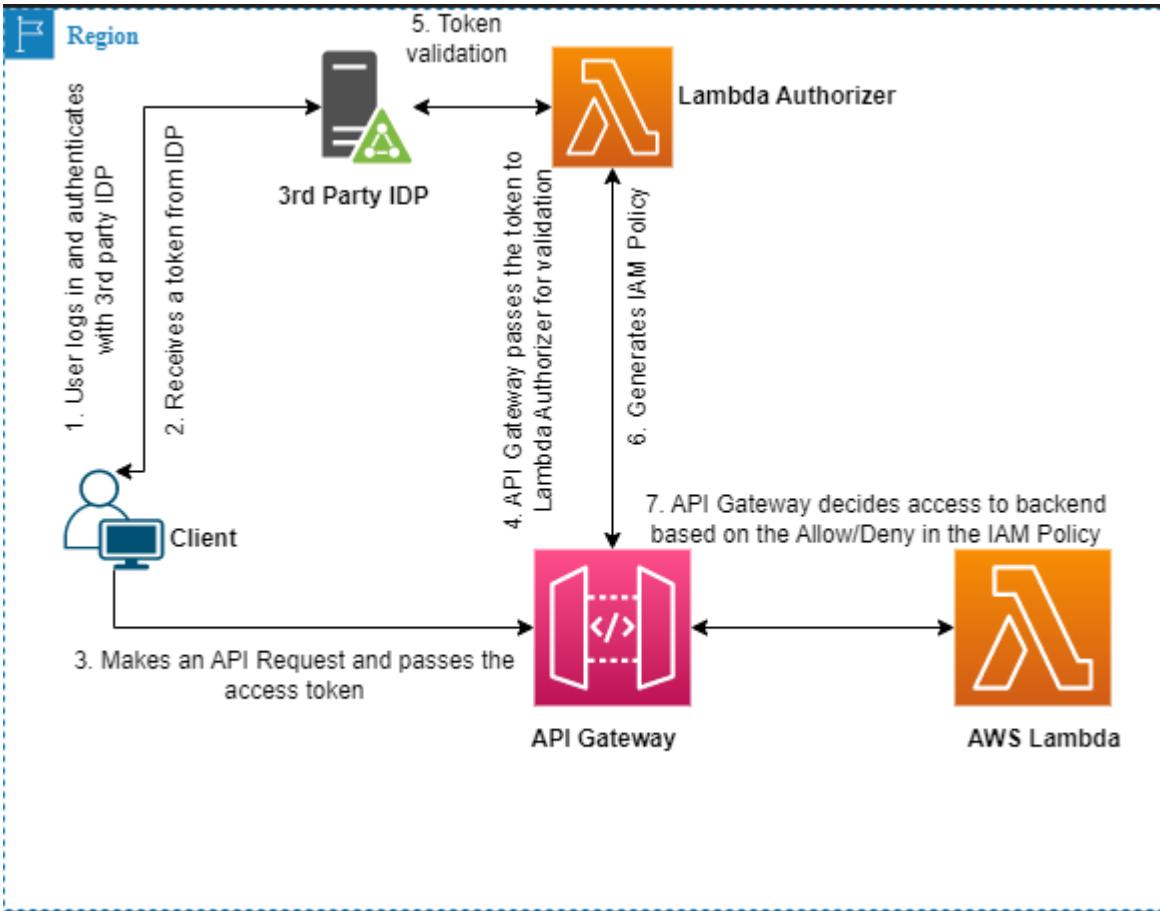
Authentication – External

Authorization – Lambda Function

Use Case : Third Party Identity Provider such as OAuth 2.0

AWS Lambda Authorizer

AWS API Gateway - Authentication and Authorization using Lambda Authorizer



AWS Lambda Authorizer - Steps

1. Create backend service which client needs to access such as Lambda - **Backend Lambda – demolambda_authorizer**
2. Create the API Gateway
3. Resource - **students**
4. Method - **GET**
5. Deploy to a Stage - **dev**
6. Test the API GW + Lambda
7. Create the Authorizer in the API Gateway Console (Authorization Token : **Key – authorizationToken, Value - 123456**)
8. Create the Lambda function corresponding to the Authorizer –
 - **Lambda Authorizer Function – lambdaauthtest** (1. Log event, 2. Validate Token, 3. **Return the Policy**)
9. Test from the API Gateway Authorizer
10. Test end to end flow using Postman

AWS Lambda Authorizer - Steps

1. Create backend service which client needs to access such as Lambda - **Backend Lambda - apigatewaylambdas31025**
2. Create the API Gateway
3. Resource - **students**
4. Method - **GET**
5. Deploy to a Stage - **dev**
6. Test the API GW + Lambda
7. Create the Authorizer in the API Gateway Console (Authorization Token : **Key – authorizationtoken, Value - 123456**)
8. Create the Lambda function corresponding to the Authorizer –
 - **Lambda Authorizer Function – lambdaauthtest** (1. Log event, 2. Validate Token, 3. **Return the Policy**)
9. Test from the API Gateway Authorizer
10. Test end to end flow using Postman

AWS Lambda Authorizer – Policy Document

```
.....  
{  
  "principalId": "yyyyyyyy", // The principal user identification associated with the token sent by the client.  
  "policyDocument": {  
    "Version": "2012-10-17",  
    "Statement": [  
      {  
        "Action": "execute-api:Invoke",  
        "Effect": "Allow|Deny",  
        "Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apild}/{stage}/{httpVerb}/[{resource}]/[{child-resources}]"  
      }  
    ]  
  },  
}  
.....
```

```
"Resource": "arn:aws:execute-api:{regionId}:{accountId}:{apild}/{stage}/{httpVerb}/[{resource}]"
```

```
.....  
"Resource": ["arn:aws:execute-api:regionId:accountId:apild/stage/httpVerb/resource"]
```

```
{ "principalId": "demorahul", "policyDocument": { "Version": "2012-10-17", "Statement": [ {"Action": "execute-api:Invoke", "Resource": ["arn:aws:execute-api:us-east-1:196715057542:kiwmj0bs7e/test/GET/students"], "Effect": "auth"} ] }
```

```
.....  
Source : https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-lambda-authorizer-output.html
```

P.S : The content is created by Rahul Trisal and copyrighted

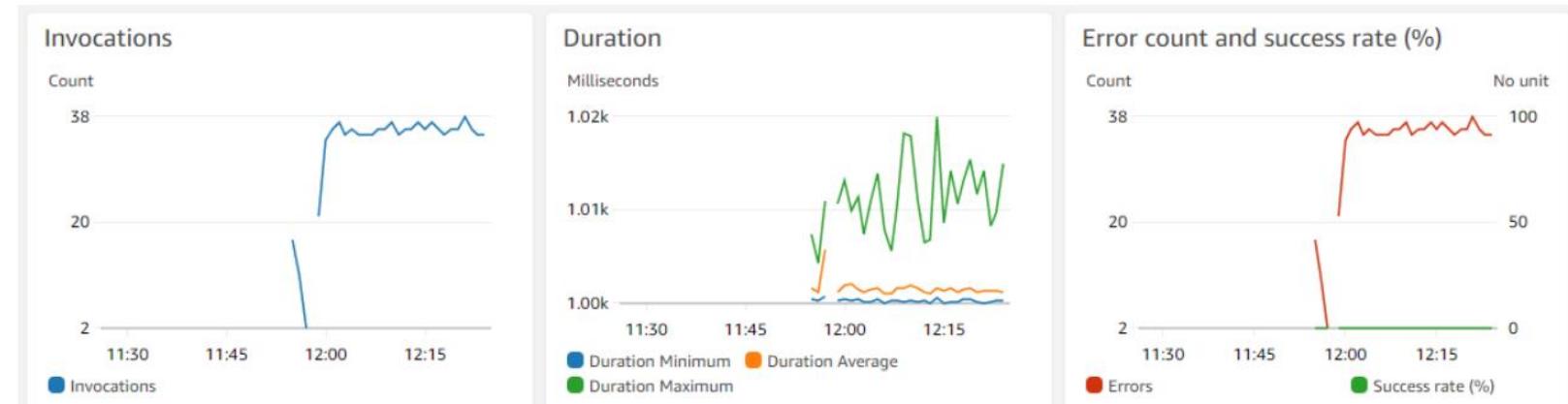
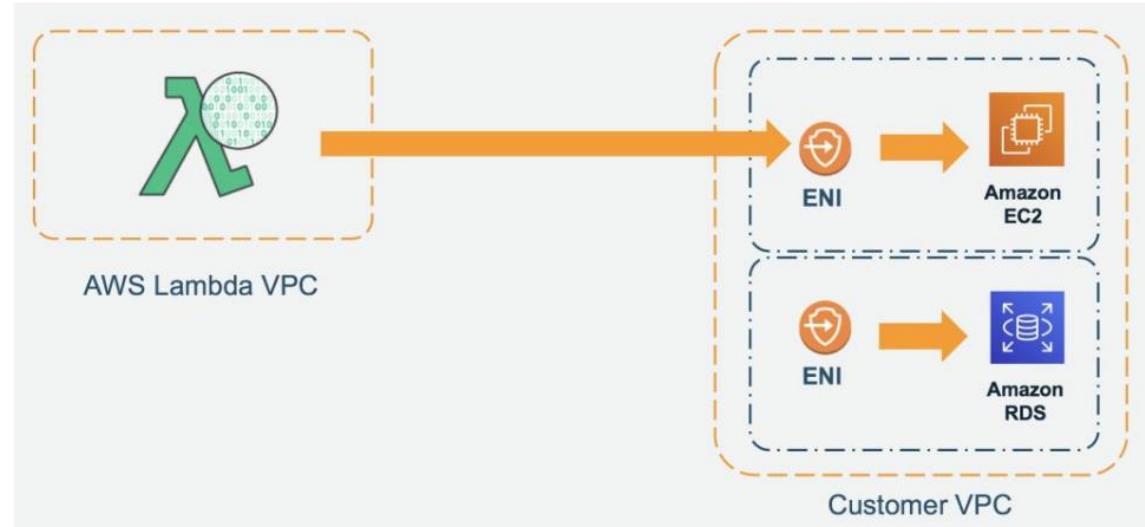
AWS Lambda and Python – Beginner to Advanced

Section on
Lambda Advanced Concepts

AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Advanced Concepts (Theory and HandsOn)

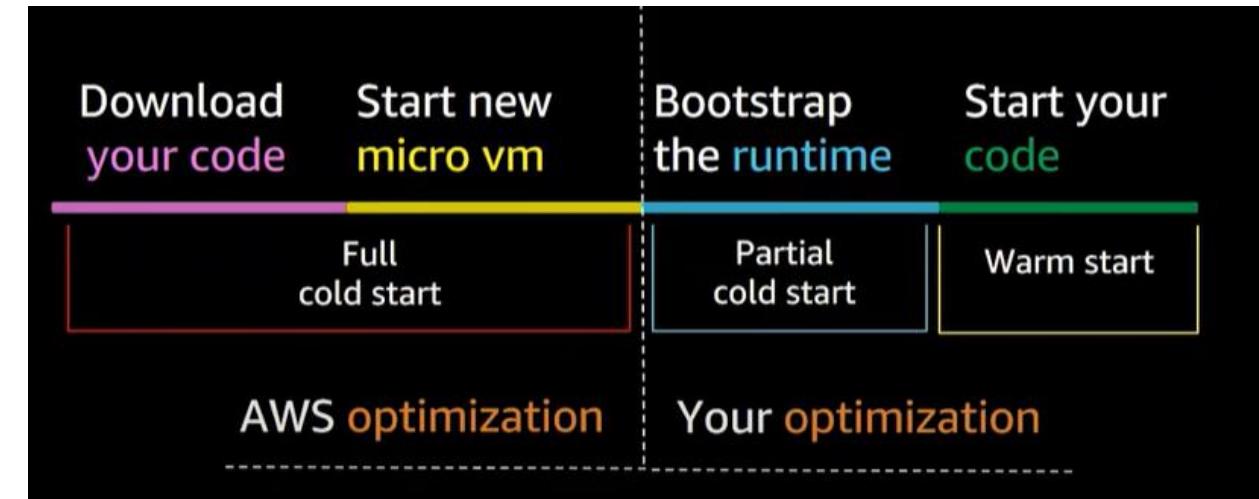
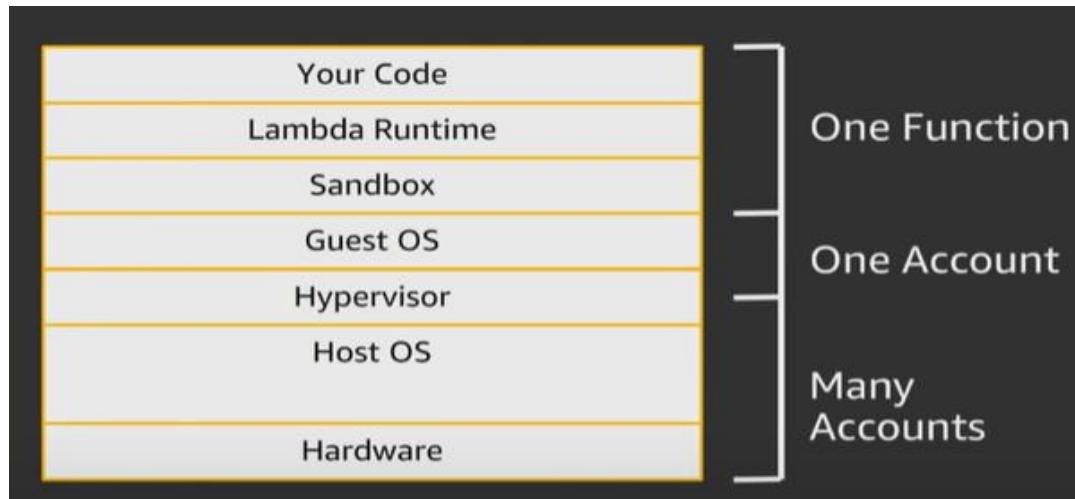
- Lambda Execution and Concurrency
- Lambda - Reserved and Provisioned Concurrency
- Lambda - VPC Networking Configuration
- Lambda Monitoring - CloudWatch Metrics
- Lambda Monitoring - CloudWatch Logs
- Lambda Versions
- Lambda Aliases
- Lambda - Environment Variables



AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Concurrency, Reserved Concurrency and Provisioned Concurrency

AWS Lambda Function Execution and Cold Start:



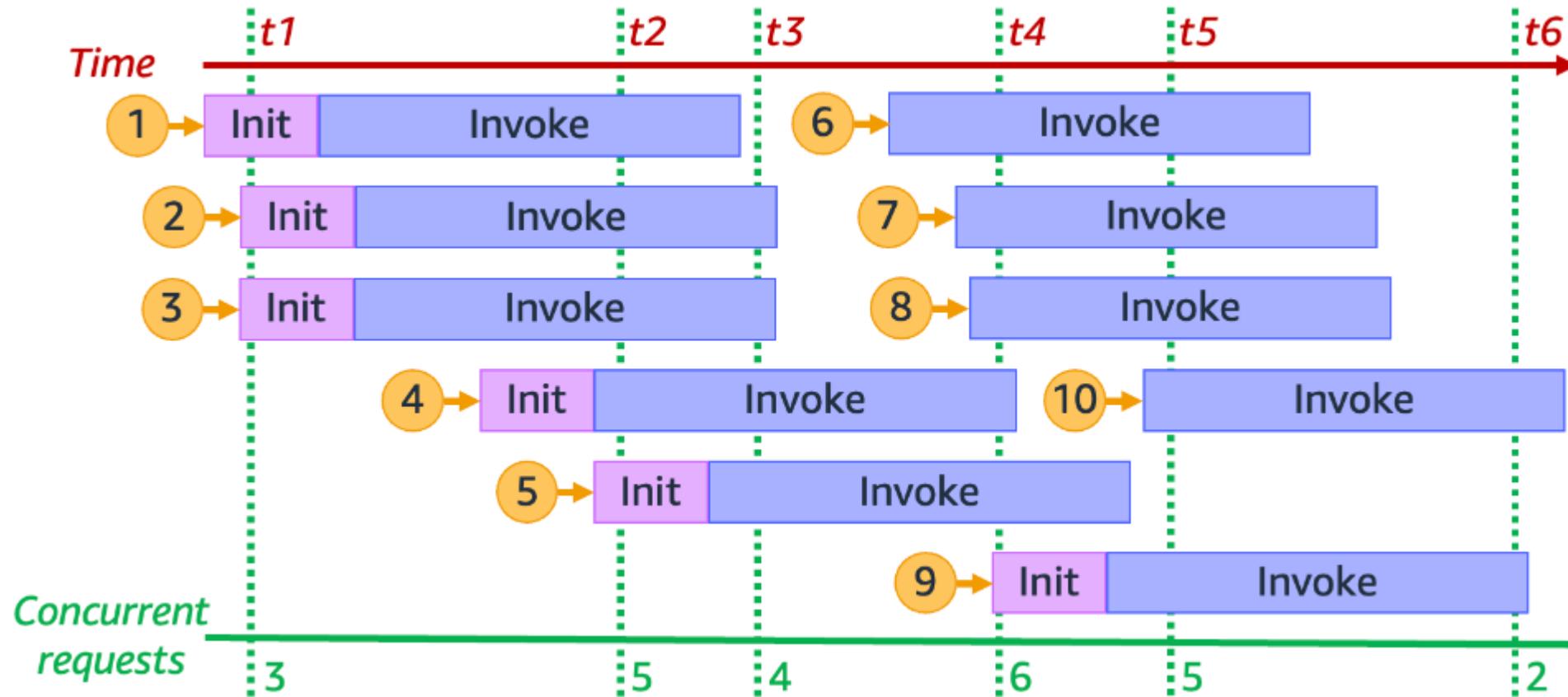
AWS Lambda and Python – Beginner to Advanced

AWS Lambda Concurrency

- **Concurrency** is the **number of in-flight requests** your AWS Lambda function is handling at the same time.
- For each concurrent request, Lambda provisions a **separate instance of your execution environment**.
- As the functions receive more requests, Lambda automatically handles **scaling the number of execution environments** until account's concurrency limit.
- By default, Lambda provides account with a **total concurrency limit of 1,000** across all functions in a region.

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Concurrency



AWS Lambda and Python – Beginner to Advanced

AWS Lambda - Concurrency

How to calculate Concurrency

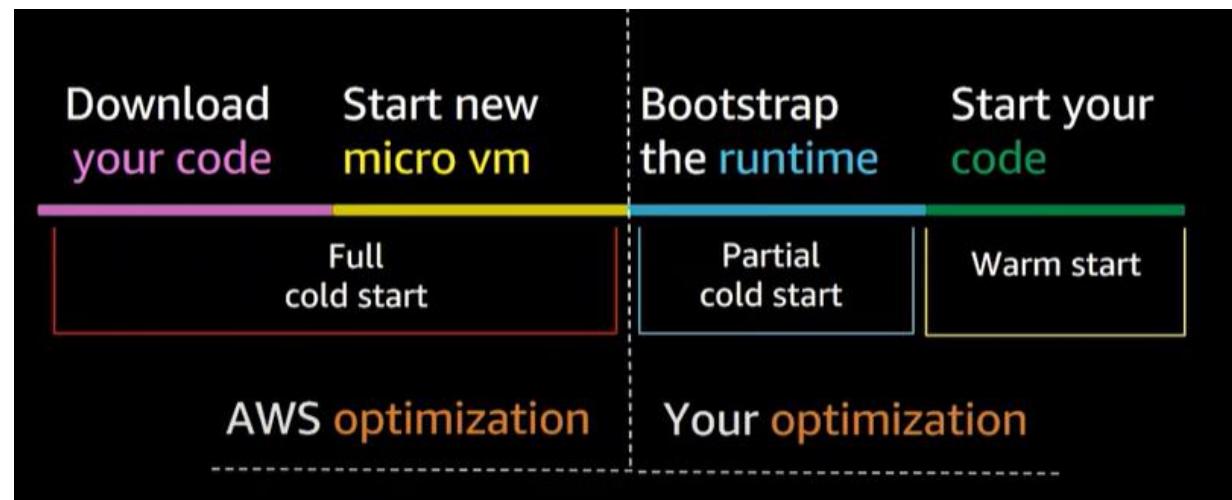
- **Concurrency** = (average requests per second) * (average request duration in seconds)
- If the average request duration is **500 ms**, the concurrency is :
- **Concurrency** = (100 requests/second) * (0.5 second/request) = **50**
- Concurrency Implication :
- A concurrency of 50 means that Lambda must provision **50 execution environment instances** to efficiently handle this workload without any throttling.

AWS Lambda and Python – Beginner to Advanced

Provisioned Concurrency and Reserved Concurrency

Provisioned Concurrency

- Set number of **pre-provisioned execution environment** instances for your function.
- Use **provisioned concurrency** to pre-initialize a number of environment instances for a function. This is useful for **reducing cold start latencies**.

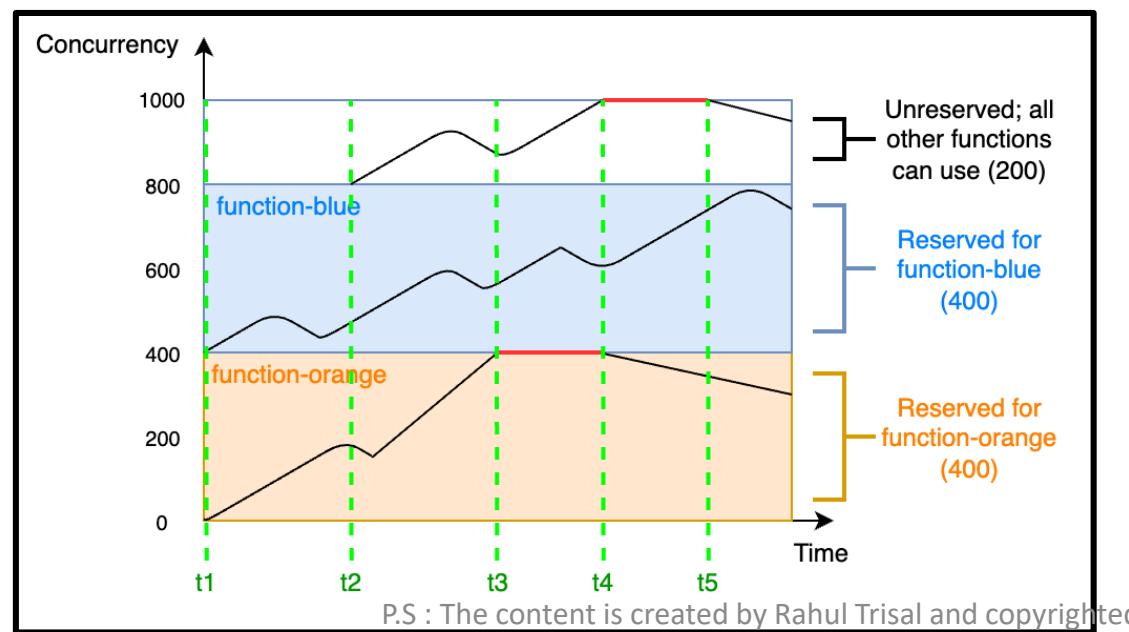


AWS Lambda and Python – Beginner to Advanced

Reserved concurrency vs. Provisioned concurrency

Reserved Concurrency

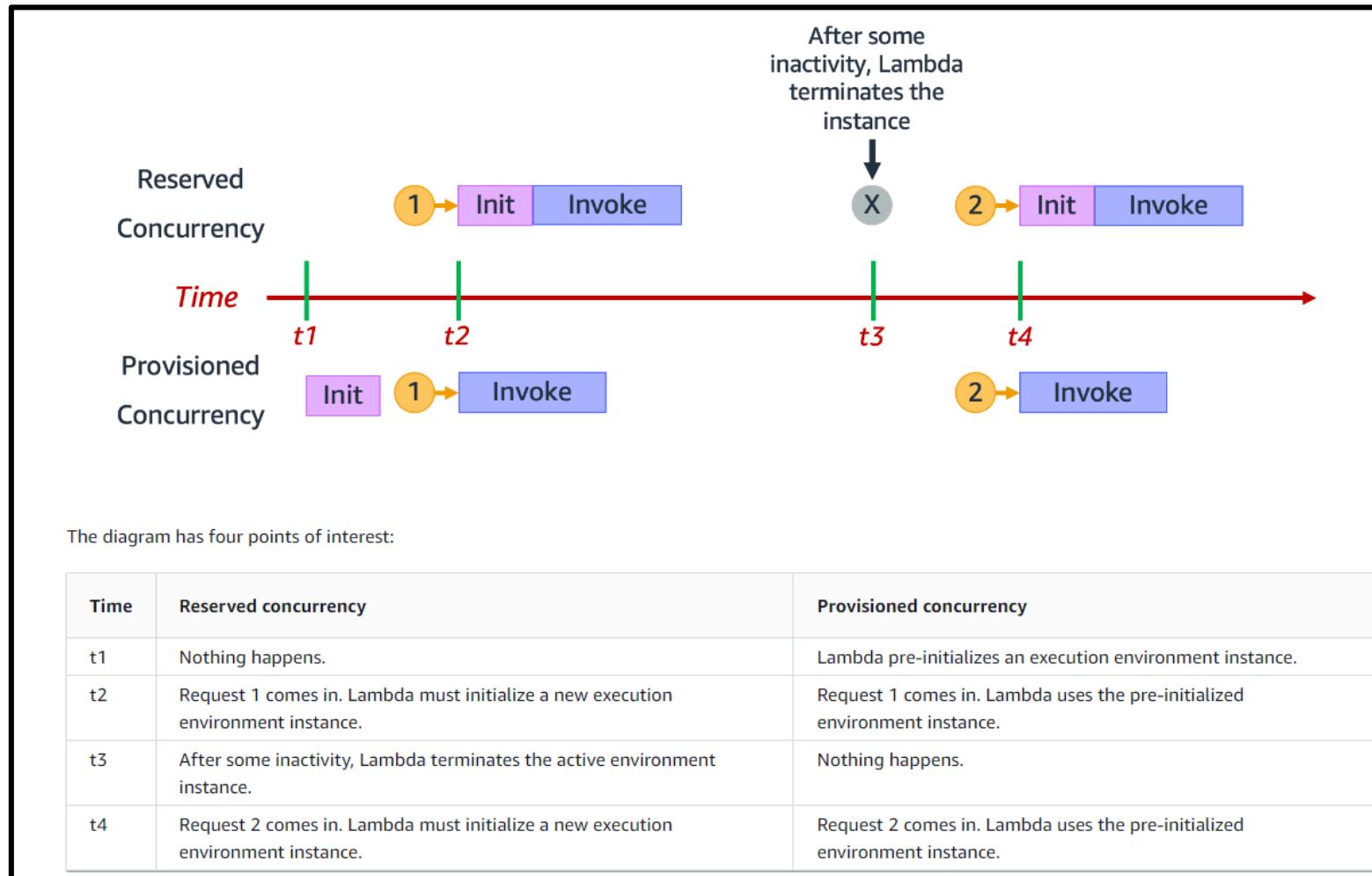
- Maximum number of execution environment instances for a **particular function**.
- Use **reserved concurrency** to reserve a portion of your account's concurrency for a function. This is useful if you don't want other functions taking up all the available unreserved concurrency.



Source : AWS

AWS Lambda and Python – Beginner to Advanced

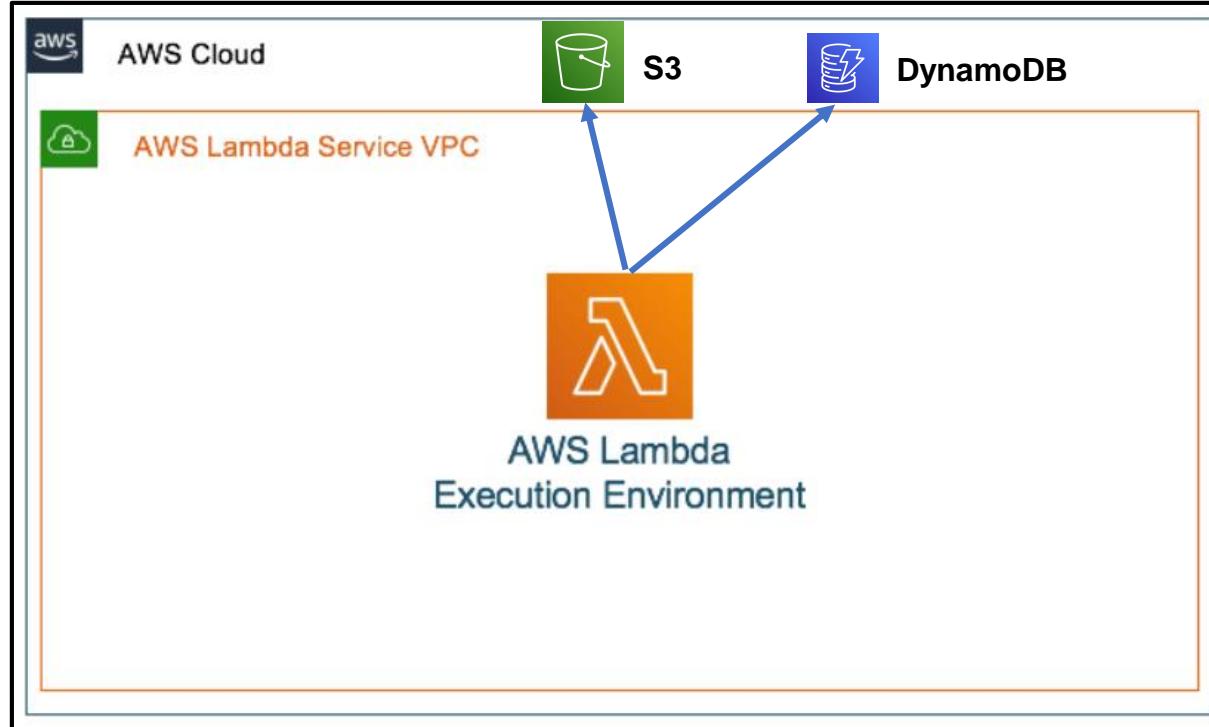
Provisioned concurrency vs Reserved concurrency



AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Deployment Options – Default and VPC Private Resources

Where is AWS Lambda Deployed – Option 1

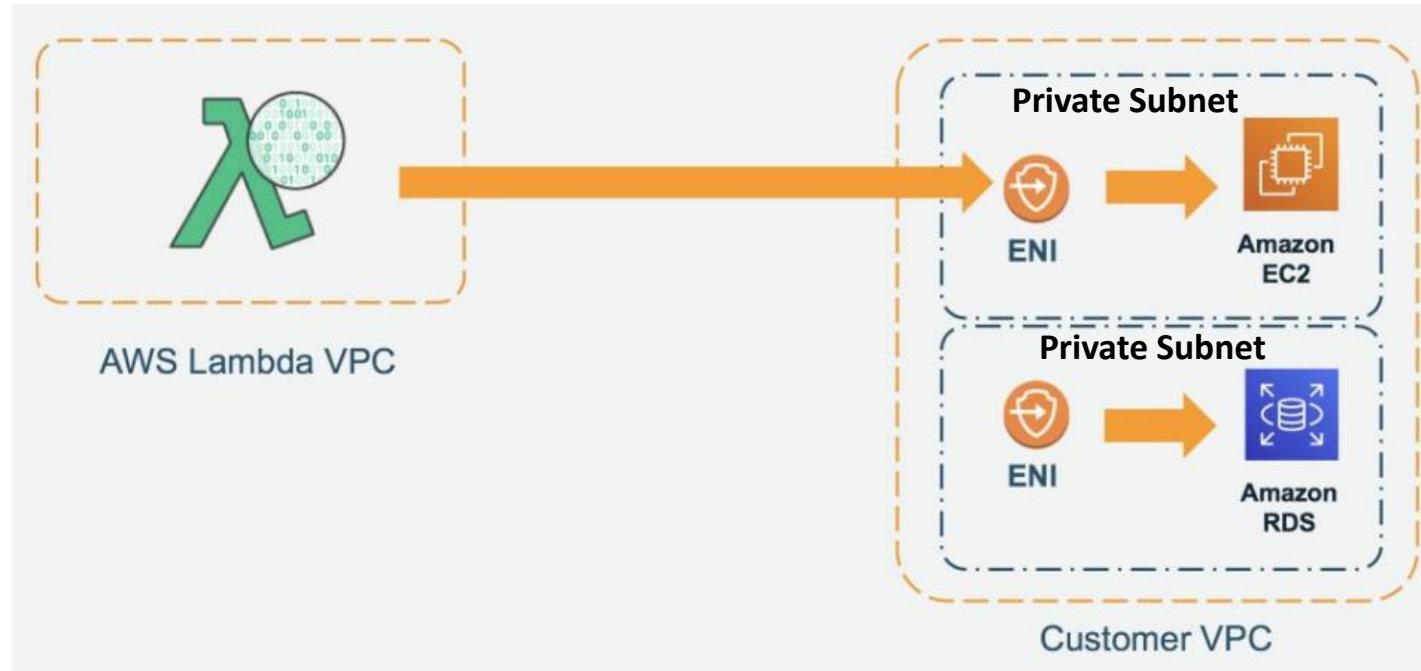


- All Lambda functions run securely inside a **default system-managed virtual private cloud (VPC)**.
- **Not** in Customer VPC
- Has access to all the services that can be accessed on **Internet – S3, DynamoDB etc.**

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Deployment and access to Private Subnet Resources

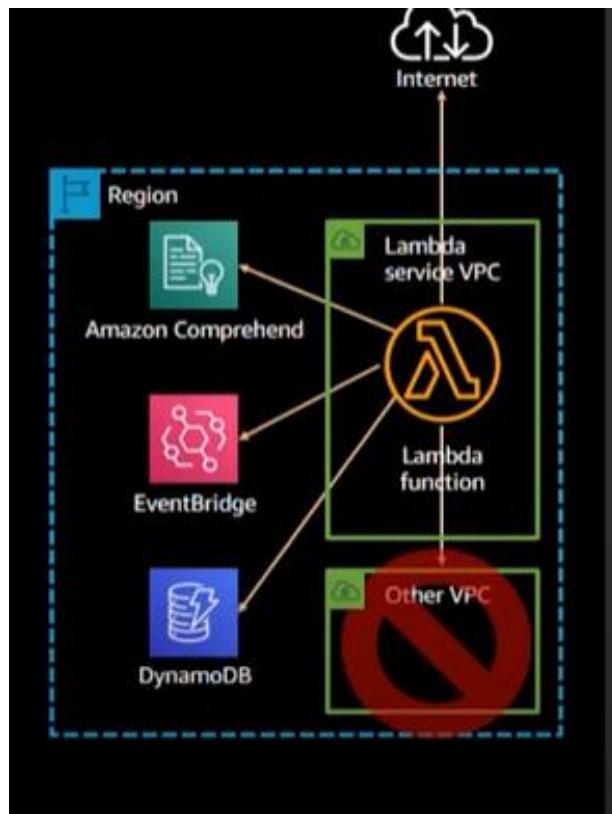
How does Lambda access AWS Services in Customer VPC in Private Subnet such as RDS DB or EC2



- Lambda need the **IAM permissions** required to create and delete network interfaces in your VPC - [AWSLambdaVPCAccessExecutionRole](#)
- Control the **subnet and security group configurations** of these network interfaces.
- **Use a NAT device to give a function internet access or use VPC endpoints to connect to services outside of your VPC.**

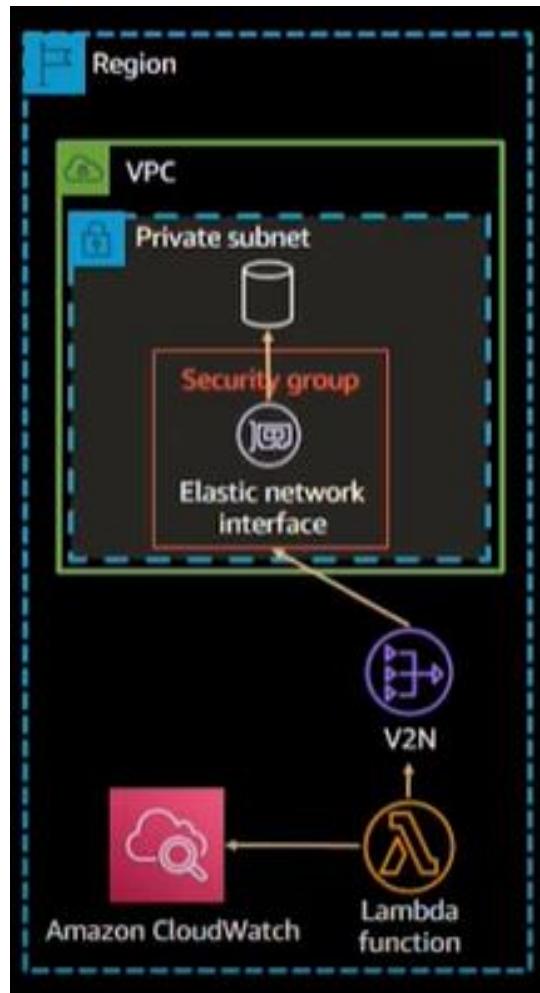
AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Deployment Options



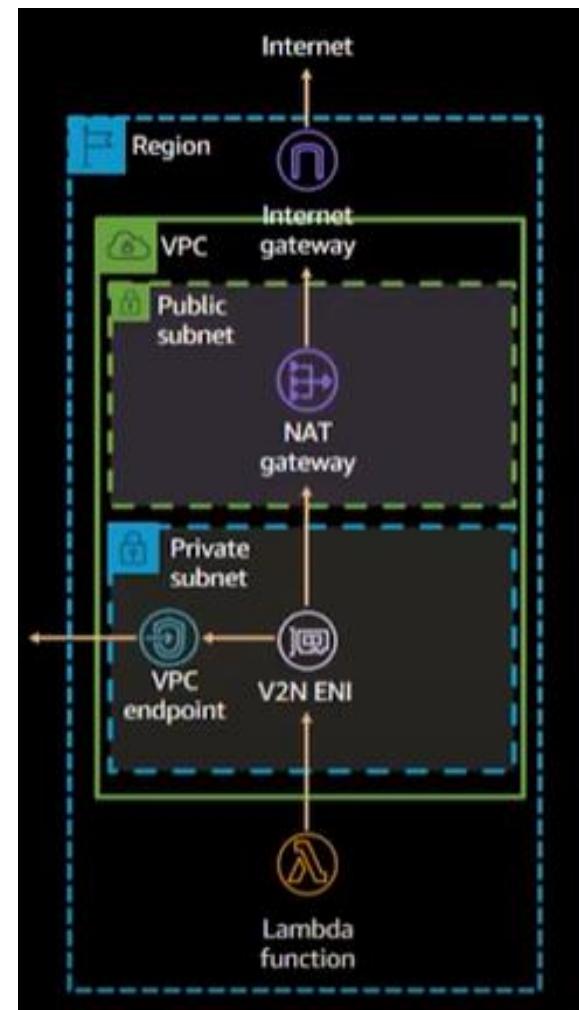
Scenario 1

(Access to Public Internet)



Scenario 2

(Access to Private Subnet VPC)



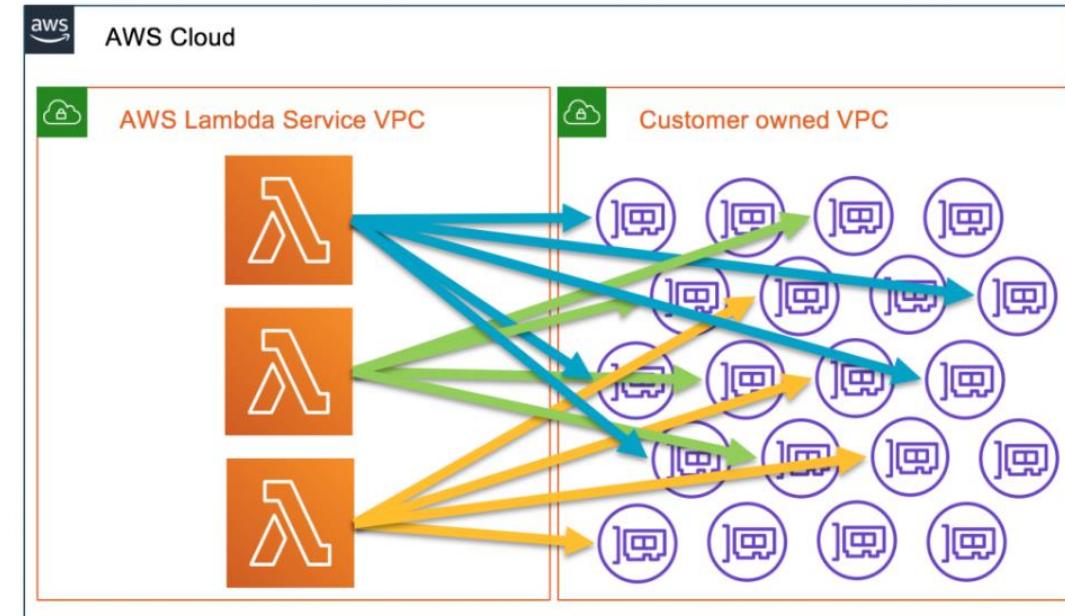
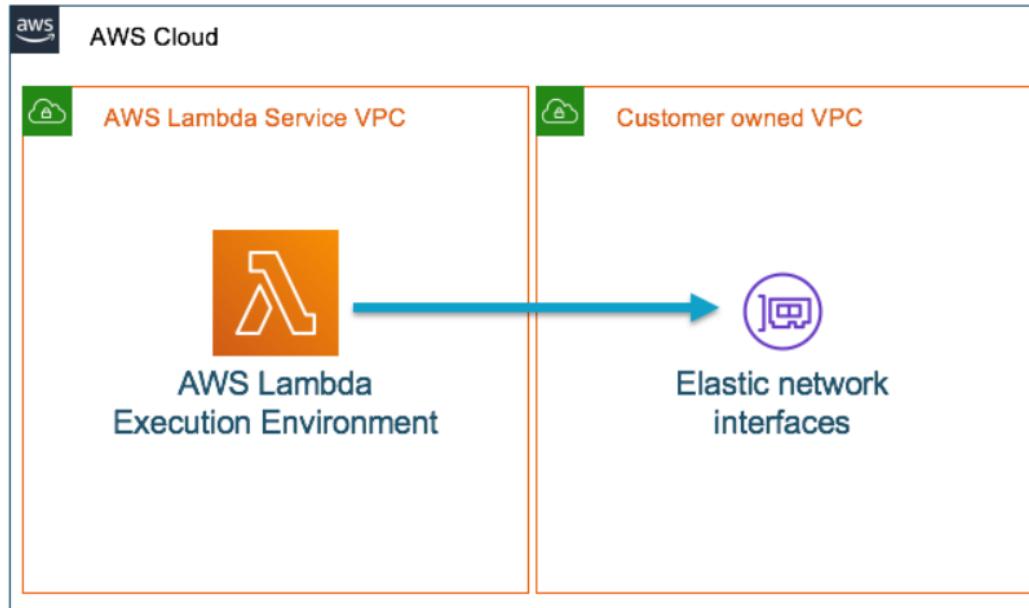
Scenario 3

(Access to Private Subnet and Public Internet/
Access AWS Services Privately)

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Deployment and access to Private Subnet Resources

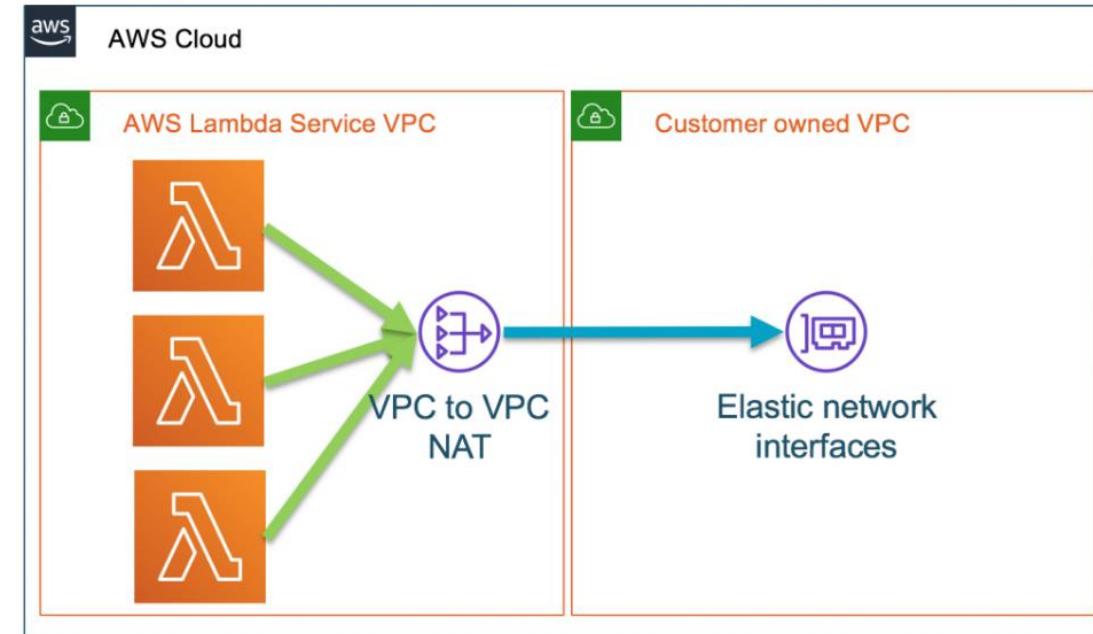
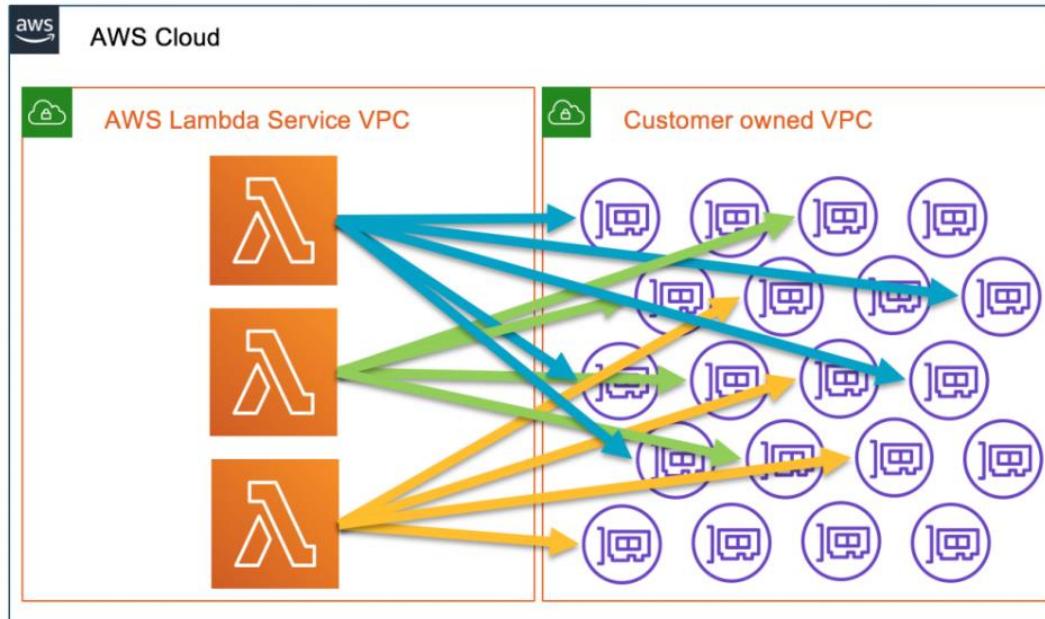
How does Lambda access AWS Services in Customer VPC in Private Subnet – Option 2



AWS Lambda and Python – Beginner to Advanced

AWS Lambda Deployment and access to Private Subnet Resources

How does Lambda access AWS Services in Customer VPC in Private Subnet – Option 2



Using **Hyperplane** (the Network Function Virtualization platform) ENI, a managed network resource that the Lambda service controls, allowing multiple execution environments to securely access resources inside of VPCs in your account.

Managed under the hood by AWS but need to be aware of configuration settings –

Whether Lambda accesses Public or Private Resources

PS: The content is created by Rahul Tandel and copyrighted

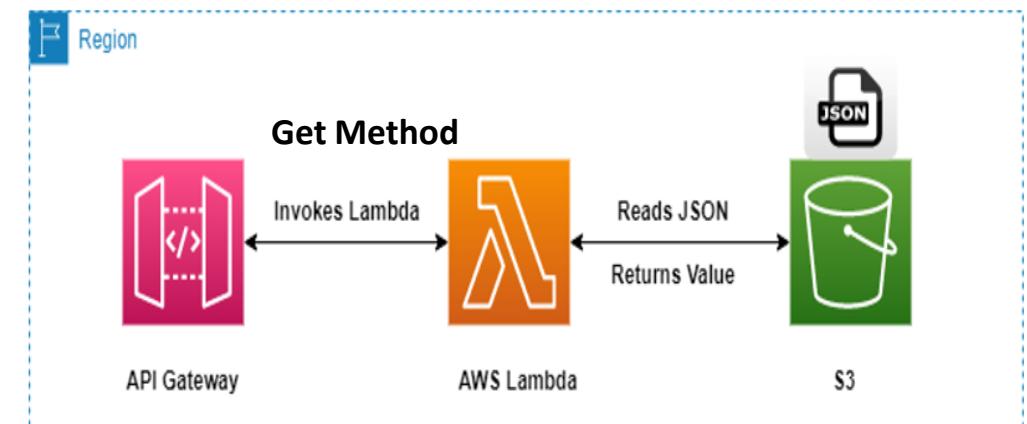
AWS Lambda and Python – Beginner to Advanced

AWS Lambda Monitoring – Use CloudWatch Metrics, Logs and Dashboard

- Lambda service **automatically monitors Lambda functions** and sends function **Metrics and Logs** to CloudWatch.
- On processing an **event**, Lambda sends **metrics about the invocation** to CloudWatch and generate logs.
- **Graphs and dashboards** can be built with these metrics on the CloudWatch console
- **Set alarms to respond to changes** in utilization, performance, or error rates.
- Lambda sends metric data to **CloudWatch in 1-minute intervals (min)**.

Sample Event

```
{ "bucket": "firstbucket01092022", "key": "bucket1.json"}
```

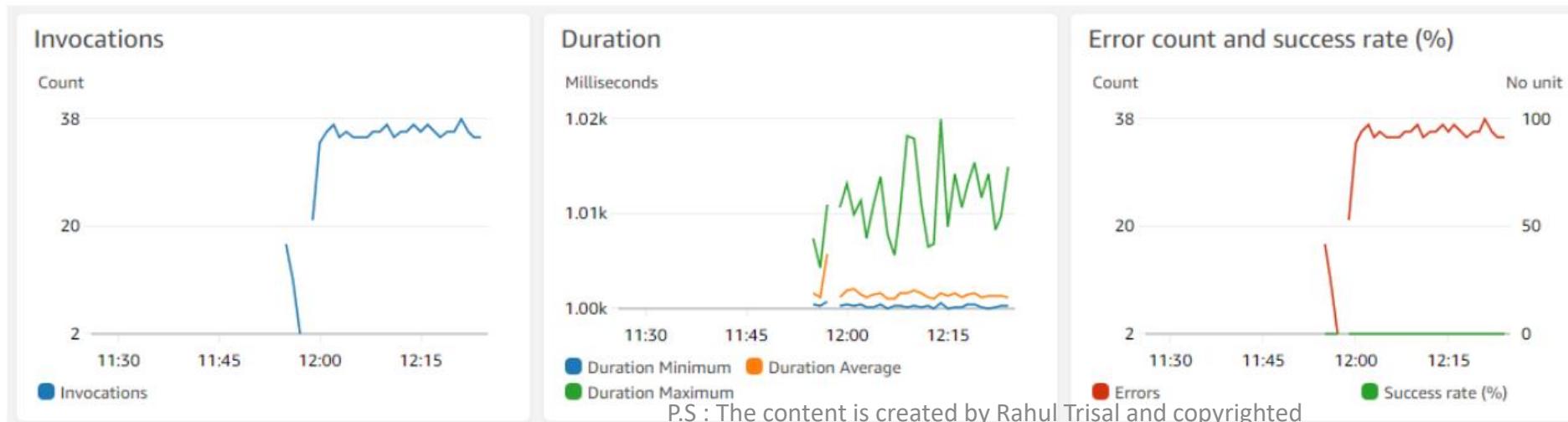


AWS Lambda and Python – Beginner to Advanced

AWS Lambda Metrics in CloudWatch

There are **eight important Lambda metrics to monitor** to understand the performance of your workload:

- 1. Invocations:** The **number of times that your function code is invoked**, including successful invocations and invocations that result in a function error.
- 2. Duration:** The **amount of time** that your function code spends processing an event.
- 3. Errors:** This logs the number of **errors thrown by a function**.



AWS Lambda and Python – Beginner to Advanced

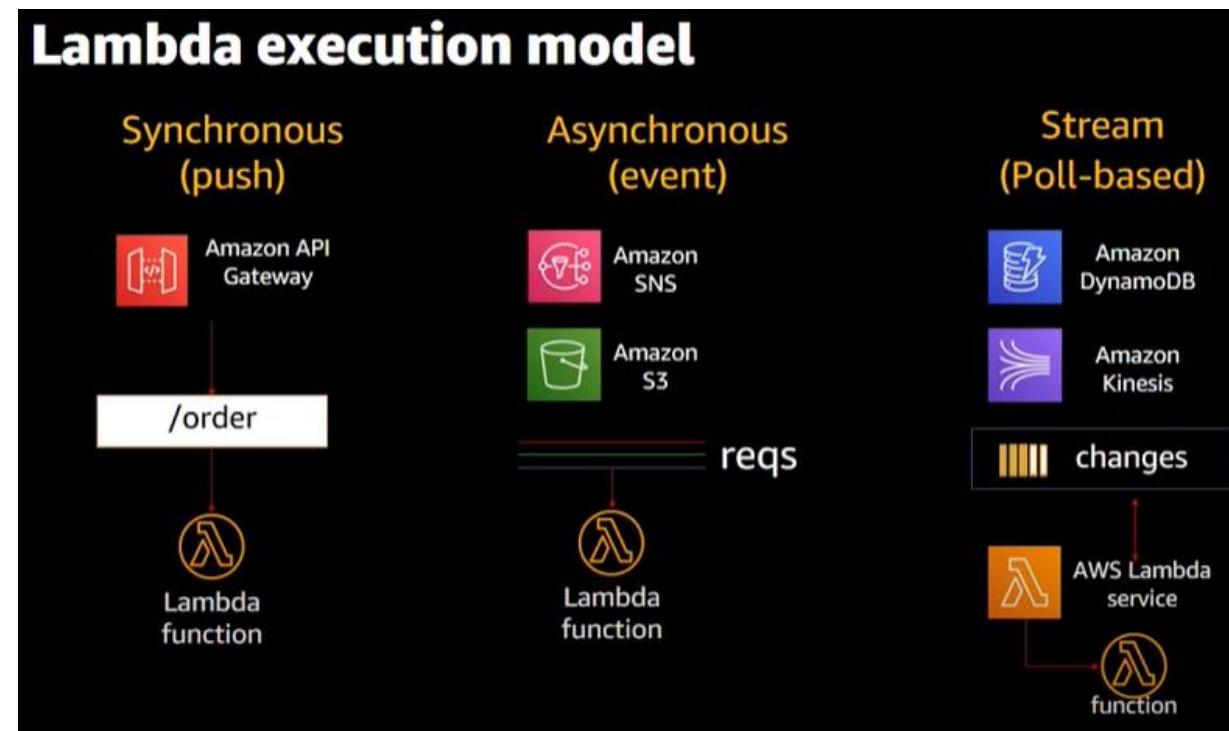
CloudWatch Metrics

4. **Throttles:** Set alarms on this metric for any **non-zero value** since this only occurs if the number of invocations exceeds concurrency in your account.
5. **ConcurrentExecutions:** monitor this value to ensure that your functions are not running close to the total **concurrency limit** for your AWS account.
6. **UnreservedConcurrentExecutions:** Similar to the previous metric but excludes functions using reserved concurrency.
7. **DeadLetterErrors:** An error is triggered if Lambda cannot write to the designated dead-letter queue, **set alarm on any non-zero values** for this metric.

AWS Lambda and Python – Beginner to Advanced

CloudWatch Metrics

8. **IteratorAge**: For Lambda functions that poll streaming sources, such as Kinesis or DynamoDB streams, the value indicates **when events are being produced faster than they are being consumed by Lambda**.
IteratorAge is the difference between the current time and when the last record of the *GetRecords* call was written to the stream.



P.S : The content is created by Rahul Trisal and copyrighted

AWS Lambda and Python – Beginner to Advanced

AWS Lambda CloudWatch Logs

- Lambda function comes with a **CloudWatch Logs log group** ([/aws/lambda/function name](#)) and a **log stream** for each instance of your function.
- The Lambda runtime environment sends details about **each invocation** to the **log stream**, and **relays logs and other output** from your function's code.
- To **output logs** from your function code, use the **print method**, or any **logging library** that writes to `stdout` or `stderr`.

AWS Lambda and Python – Beginner to Advanced

AWS Lambda CloudWatch Logs

```
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
```

```
START RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function', 'AWS_LAMBDA_LOG_STREAM_NAME':
'2020/01/31/[${LATEST}]3893xmpl7fac4485b47bb75b671a283c', 'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95 Duration: 15.74 ms Billed Duration: 16 ms
Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms
XRAY Traceld: 1-5e34a614-10bdxmplf1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85 Sampled: true
```

Report Log

1. **RequestId**
2. **Duration**
3. **Billed Duration**
4. **Memory Size**
5. **Max Memory Used**
6. **Init Duration**
7. **XRAY Traceld**
8. **SegmentId**

AWS Lambda and Python – Beginner to Advanced

Lambda function – Use Cloud Watch Logs and CloudWatch Metrics

```
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
```

```
START RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
'AWS_LAMBDA_LOG_STREAM_NAME':
'2020/01/31[$LATEST]3893xmpl7fac4485b47bb75b671a283c',
'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507fcf-xmpl-4697-b07a-ac58fc914c95 Duration: 15.74 ms Billed
Duration: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49
ms
XRAY Traceld: 1-5e34a614-10bdxmplf1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85
Sampled: true
```

Report Log

- **RequestId** – The unique request ID for the invocation.
- **Duration** – The amount of time that your function's handler method spent processing the event.
- **Billed Duration** – The amount of time billed for the invocation.
- **Memory Size** – The amount of memory allocated to the function.
- **Max Memory Used** – The amount of memory used by the function.
- **Init Duration** – For the first request served, the amount of time it took the runtime to load the function and run code outside of the handler method.
- **XRAY Traceld** – For traced requests, the [AWS X-Ray trace ID](#).
- **SegmentId** – For traced requests, the X-Ray segment ID.
- **Sampled** – For traced requests, the sampling result.

AWS Lambda and Python – Beginner to Advanced

AWS Lambda Versions and Aliases

AWS Lambda Version

- Lambda **versions** are used for **deployment of functions**.
- For example, you can publish a **new version of a function in Dev, QA and Prod**
- Lambda creates a new version of your function each time we publish the function.

A function version includes the following information:

- The **function code** and all associated dependencies.
- The **Lambda runtime** that invokes the function.
- All the **function settings**, including the environment variables.
- A **unique Amazon Resource Name (ARN)** to identify the specific version of the function.

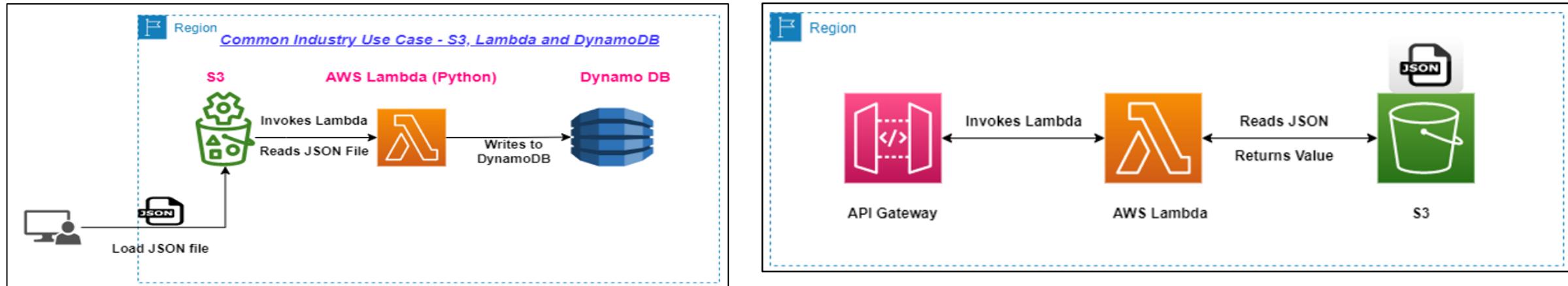
AWS Lambda and Python – Beginner to Advanced

AWS Lambda Aliases

- A Lambda alias is like a **pointer to a specific function version**.
- Users can **access the function version using the alias Amazon Resource Name (ARN)**.
- Each **alias has a unique ARN**. An alias can point only to a function version, not to another alias.
- You can **update an alias to point to a new version of the function**.

AWS Lambda and Python – Beginner to Advanced

Use Case for Aliases



- Event sources such as Amazon Simple Storage Service (Amazon S3) invoke your Lambda function. These **event sources maintain a mapping that identifies the function to invoke when events occur**. If you specify a Lambda **function alias in the mapping configuration**, you don't need to **update the mapping when the function version changes**.
- In a **resource policy**, you can grant permissions for event sources to use your Lambda function. If you specify an **alias ARN in the policy**, you don't need to update the policy when the function version changes.

AWS Lambda and Python – Beginner to Advanced

AWS Lambda – Environment Variables

- Environment variables can be used to adjust your **function's behavior without updating code**.
- Most common use case is storing and **retrieving hostname and other connection details for the database in test and prod environments**
- **Environment variable** for your function can be **defined by using a key and a value**.
- The Lambda runtime makes environment variables available to your code.
- Your function uses the **name of the key to retrieve the value of environment variable**.

The screenshot shows the 'Edit environment variables' page within the AWS Lambda console. The URL in the top left corner is 'Lambda > Functions > logs > Edit environment variables'. The main title is 'Edit environment variables'. Below it, a section titled 'Environment variables' contains a descriptive text: 'You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code.' A 'Learn more' link is provided. A table lists two environment variables:

Key	Value	Action
environment	development	Remove
dbname	rds.amazon.com	Remove

At the bottom left is a button labeled 'Add environment variable'.

AWS Lambda and Python – Beginner to Advanced

Example scenario for environment variables

- Environment variables to **customize function behavior** in test environment and production environment.
- We can create two functions with the **same code but different configurations**.
- One function connects to a **test database**, and the other connects to a **production database**.
- Environment variables can be used for function to **retrieve hostname and other connection details for the database**.

AWS Lambda and Python – Beginner to Advanced

Section 8

*Serverless Use Case-using API Gateway, AWS Lambda
and S3 (Balance Status Application)*

AWS CDK v2 – Pre-Requisites

1. Signup for AWS Account
2. IDE – Visual Code Studio - <https://code.visualstudio.com/>
3. Install AWS CLI - <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

Check using open Start --> cmd --> Run following command > *aws --version*; output : aws-cli/2.7.24...

4. Install the AWS CDK Toolkit for VS Code :
<https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/setup.html#install>

Install the AWS CDK

Install the AWS CDK Toolkit globally using the following Node Package Manager command.

```
npm install -g aws-cdk
```

Run the following command to verify correct installation and print the version number of the AWS CDK.

```
cdk --version      Output : 2.59.0 (build b24095d)
```

In case of error : Run following command >> “Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass” and then execute other commands
P.S : The content is created by Rahul Trisal and copyrighted

AWS CDK v2 – Pre-Requisites

5. Install Node.js - <https://nodejs.org/en/> (Check by running following command >> **node --version**, output should be > v10.3.0)

```
PS C:\Users\ADMIN> node --version  
v18.12.1
```

To download for other Programming Languages – Use this link : <https://cdkworkshop.com/15-prerequisites.html>

6. AWS Account User - Configure Credentials to access AWS services from Visual Studio -
<https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/establish-credentials.html>

- Create an IAM User
- Configure Credentials
- Test using following command If configured properly –> aws s3 ls (should return all s3 buckets)

```
$ aws configure  
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE  
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRficyEXAMPLEKEY  
Default region name [None]: us-west-2  
Default output format [None]: json
```

```
C:\Users\ADMIN>aws s3 ls
```

AWS CDK – Pre-Requisites

The screenshot shows the left sidebar of the AWS CDK Workshop website. At the top is a logo icon of three yellow blocks in a hexagon shape next to the text "AWS CDK Workshop". Below the logo is a search bar labeled "Search". Under the search bar is a dropdown menu set to "English". The sidebar contains a list of "Prerequisites" with the following items: AWS CLI, AWS Account and User, Node.js, IDE for your programming language, AWS CDK Toolkit, Python, .NET, Java, and Go.

- English
- Prerequisites
 - AWS CLI
 - AWS Account and User
 - Node.js
 - IDE for your programming language
 - AWS CDK Toolkit
 - Python
 - .NET
 - Java
 - Go

The TypeScript Workshop

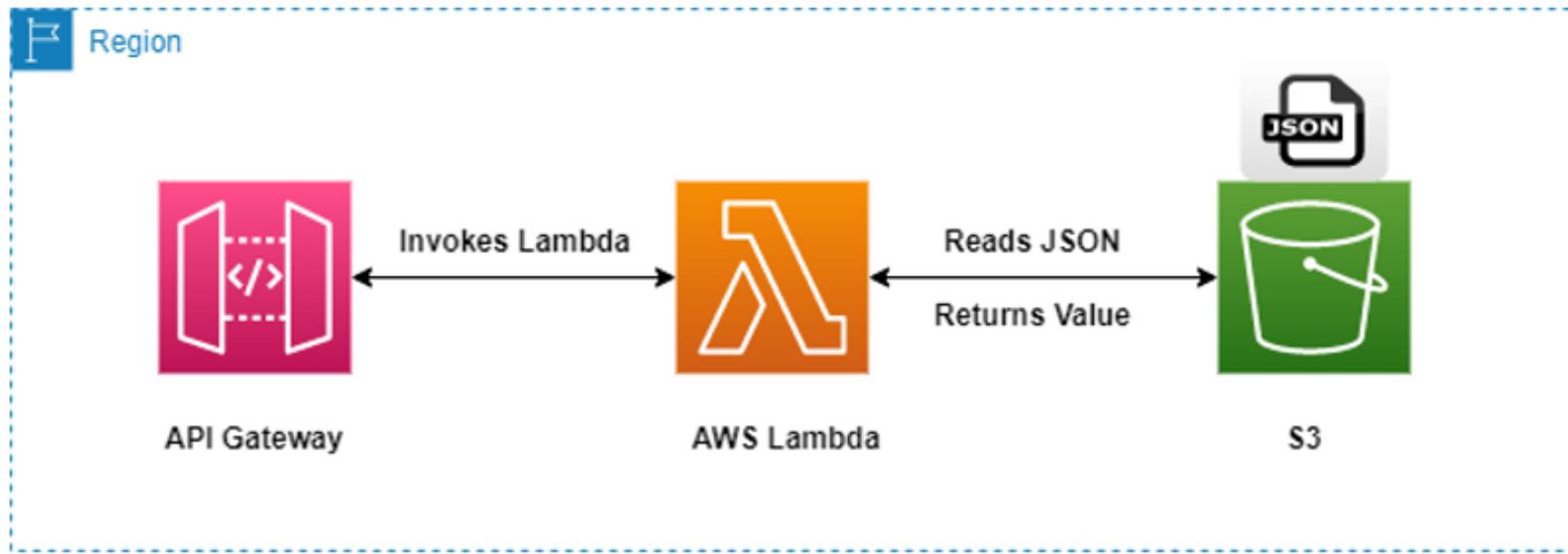
This version of the workshop will guide you through a getting started experience in TypeScript.

A disclaimer about cost: Some of the steps in this workshop will create resources that may bill your account. If you do not complete the workshop, you may still have AWS resources that are unknowingly charging your account. To ensure your account is clean after starting this workshop, check out the [cleanup section](#) at the end of the TypeScript Workshop.

 [Edit this page](#)

AWS Cloud Development Kit (CDK) v2 – Serverless Use Case

Serverless Use Case - using API Gateway, AWS Lambda and S3



- *S3*
- *IAM Role*
- *AWS Lambda*
- *API Gateway*

AWS CDK v2 – Serverless Use Case

Serverless Use Case-using API Gateway, AWS Lambda and S3 (Balance Status Application)

1. S3

- *bucketName* – ‘balanceStatus-0125’

2. IAM Role

- *roleName*
- *assumedBy*
- *description*
- *IAM Policy attached to Role* - AmazonS3FullAccess

3. AWS Lambda

- *handler* - `lambda_function.lambda_handler`
- *role*
- *code*
- *runtime*
- *LambdaCode file* – `lambda_function.py` and **Method – lambda_handler**

AWS CDK v2 – Serverless Use Case

Serverless Use Case - using API Gateway, AWS Lambda and S3

4. API Gateway

- *handler*
 - *restApiName*
 - *proxy*
 - *deploy*
-
- *Resource - balanceStatus*
 - *Method – GET*
 - *Construct - LambdaRestAPI*

Summary of Steps to create any AWS Resource using CDK v2

Step 1. – Open the new folder in Visual Studio Code Editor and open Terminal

Step 2. – Create the app: Create Infra & Services Folder - ***mkdir infra, mkdir services and cd infra***

Step 3. – Initialize the CDK with ***cdk init app --language typescript***

Step 4. – Import the module for aws service being created - [Link](#)

Step 5. – Define Scope, Logical ID and Props – ***(this, 'logical id', {props})***

Step 6. – Build the app (Optional) with ***npm run build***

Step 7. – Bootstrap (One Time) with ***cdk bootstrap***

Step 8. – Synthesize an AWS CloudFormation template for the app with ***cdk synth***

Step 9. – Deploying the stack with ***cdk deploy***

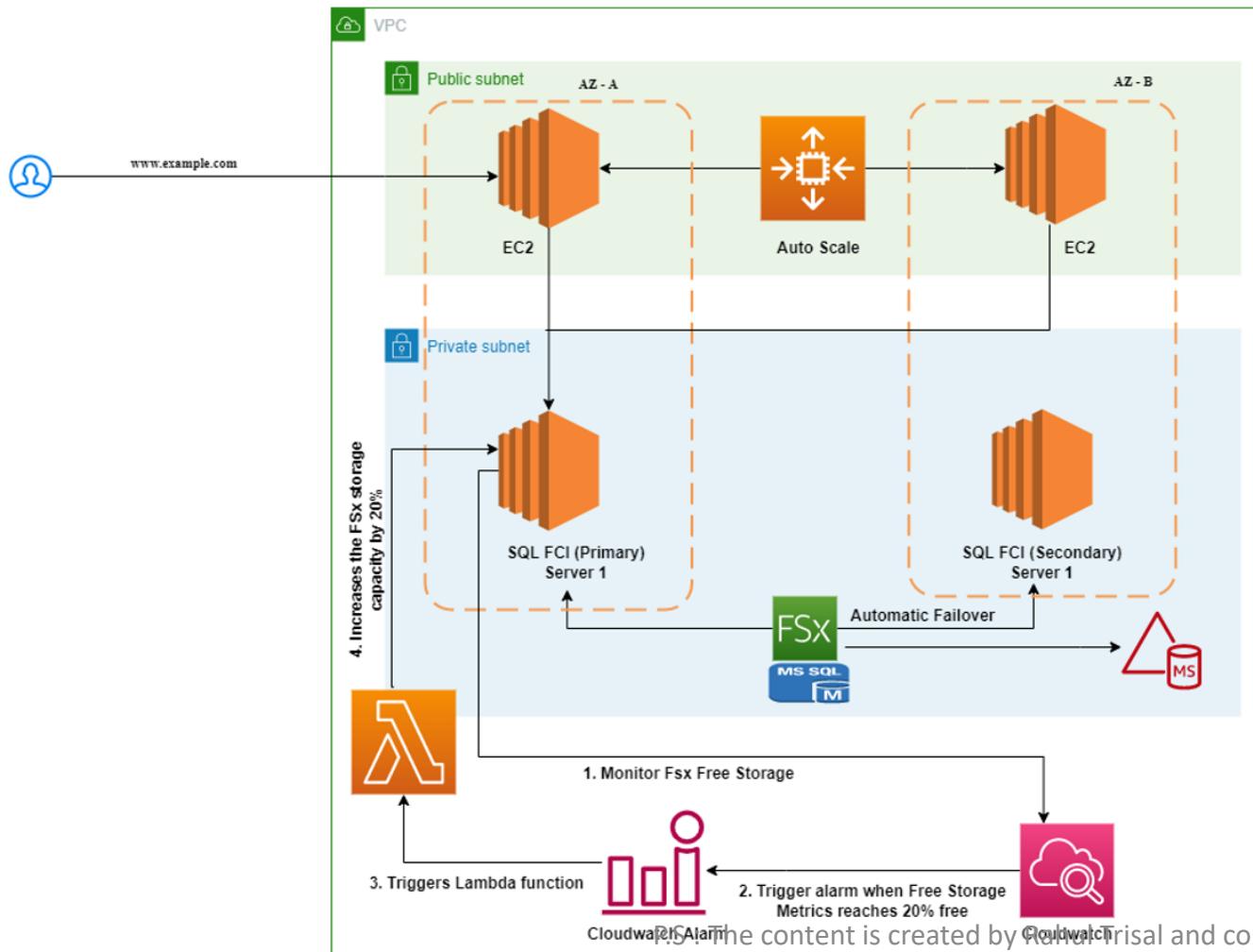
AWS Lambda and Python – Beginner to Advanced

Section on

*Real World Serverless Use Case 3 - Monitor & increase
free storage for SQL Server FCI Cluster*

AWS Lambda and Python – Beginner to Advanced

Monitor & increase free storage for SQL Server FCI Cluster using AWS Lambda- MS AD, EC2, FSx, CloudWatch and CloudWatch Alarm



- Microsoft AD
- EC2
- FSx
- CloudWatch Metrics
- CloudWatch Alarm

Python Basics – 1 (Print Function, Variables, .Format, User Input)

1. Print Function – print the message to screen or any interface; Syntax : print()

```
>> print("Hello Rahul")  
>> print(30.5)
```

2. Variable - Containers for storing data values string, float or integers and no need to declare; Syntax : x = 3, greeting = "hello" etc.

```
>> demo = "hi"  
>> x = 30
```

3. User Input - Allow user to provide an input; Syntax - input(" ")

```
>> schoolName = input("Please enter your School Name")  
>> grade = input("Please enter your Grade")  
>> print("My schoolName is : {} and grade is {}".format(schoolName, grade))
```

4. Print variables in Strings---- 'format()' method - .format(a,b)

```
>> grade = 4,  
>> section = 'A'  
>> print("My class is : {} and section is {}".format(grade, section))
```

Python Basics – 2 (Data Types Introduction)

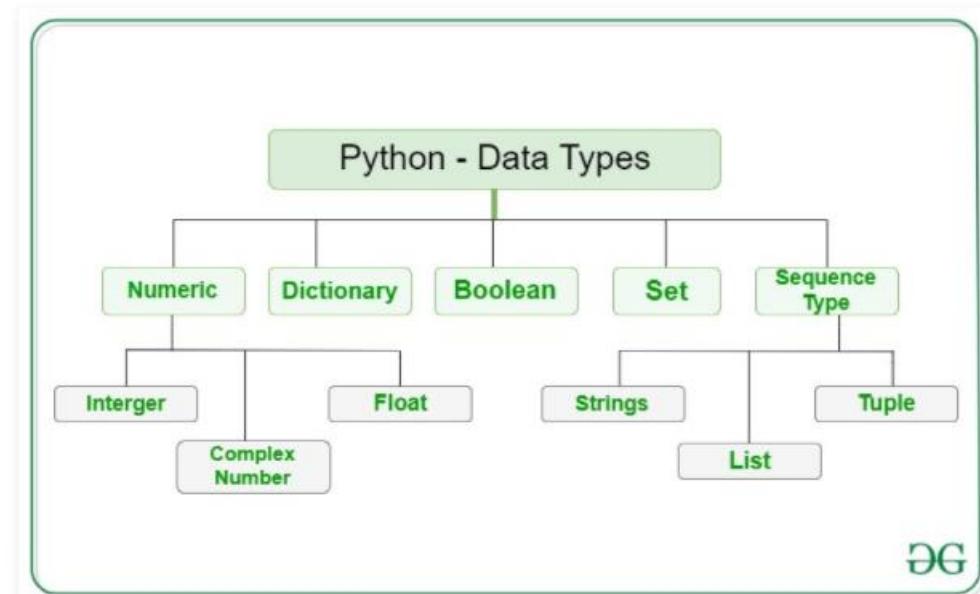
5. Data Types Introduction

Data types are the **classification or categorization of data items**. It represents the **kind of value that tells what operations can be performed on a particular data**.

- Numeric (Integer, Complex Number, Float)
- Dictionary – **Most Important**; Key-value pairs; Dict = {1: 'Rahul', 2: 'John', 3: 'Joy'}
- Boolean (True or False)
- Set - Sets are used to store multiple items in a single variable; fruits = {"apple", "banana", "cherry"}
- Sequence Type (String, List and Tuple)
- **Strings** is a sequence of characters internally stored as binary ("Aaron")
 - ASCII value of the letter 'A' is 65.
- **List** - Lists in Python can be created by just placing the sequence inside the square brackets[]
 - ["Rahul", "John", "Joy"]

6. Determine Data Type – Syntax - type(variable)

```
>> type(variable)
```



AWS Lambda with Python – Basics - 3

12. Function

Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.

Syntax:

```
def function_name(argument/parameters):  
    return expression or value
```

Example

```
# A simple Python function to check whether x is even or odd
```

```
def evenOdd(x):  
    if (x % 2 == 0):  
        print("even")  
    else:  
        print("odd")
```

```
# Driver code to call the function
```

```
evenOdd(2)  
evenOdd(3)
```

Python Basics – 2 (Dictionary)

10. Data Types – Dictionary

- curly brackets
- key: pair values
- Its mutable but keys immutable
- Nested Dictionary

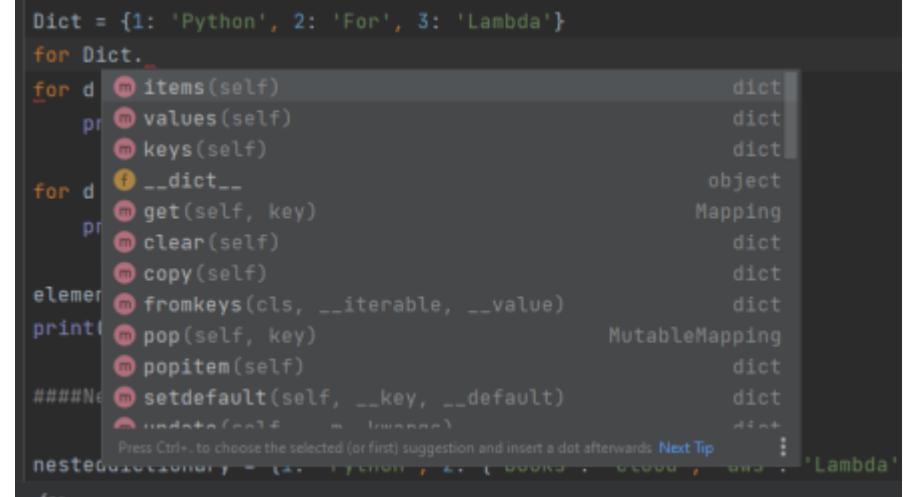
.....
`d = {1: 'Python', 2: 'For', 3: 'Lambda'}`

Nested Dictionary

.....
`nd = {1:'Python', 2:{'books': 'arch' , 'aws':'Lambda'}}`

- Items
- Keys
- Values

- **Elements in Dictionary – Getting specific values from Key Names**
- **Elements in Dictionary – Getting specific values from Key Names in a Nested Dictionary**
- **Adding an element to dictionary - d[3] = "red"**
- **Dictionary Methods**



A screenshot of a Python code editor showing a dictionary definition and a list of its methods. The code is:

```
Dict = {1: 'Python', 2: 'For', 3: 'Lambda'}
for Dict...
for d m items(self)
    pr m values(self)
    m keys(self)
for d f __dict__
    pr m get(self, key)
    m clear(self)
    m copy(self)
element m fromkeys(cls, __iterable, __value)
print(m pop(self, key))
    m popitem(self)
#####Ne m setdefault(self, __key, __default)
    m update(dict, **kwargs)
nesteddictionary = {1: 'Python', 2: {'books': 'arch', 'aws': 'Lambda'}}
```

The methods listed are: items, values, keys, __dict__, get, clear, copy, fromkeys, pop, popitem, setdefault, and update. A tooltip at the bottom right says: "Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip".

Python Basics – 2 (Loops and Slicing)

7. Loop (for loop)

For loops are used for sequential traversal. For example: traversing a list or string or array etc.

```
>> data = "length"
```

Syntax :

```
>> for k in data :
```

print(k) -----> k, colon, spacing (4 or 5)

8. String length function - returns the length of a string: Syntax – len()

```
a = "Hello"
```

```
print(len(a))
```

9. String Slicing in Python

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end. **Syntax:** var[start: stop: step]

- data = "john"
- print(data[0:2:1])

0	1	2	3	4	5	6
A	S	T	R	I	N	G

Python Basics – 3 (List) with Loop and if/else statement

11. Data Types – List

- Lists in Python can be created by just placing the sequence inside the **square brackets []**
 - A single list may contain Data Types like Integers, Strings, as well as Objects.
 - List in Python are ordered and have a definite count. The elements in a **list are indexed with 0 being the first index.**
 - slice(start, stop, step)
 - Reverse [:: -1]
-
- l = [1, 4, 'For', 6, 'Anisha']
 - **Nested List =>** nestedList = nestedList = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
-
- By positional value
 - list = [2,3,4] or [2]
 - k = list[0:2:1]
 - print(k)
 - For Loop and if statement

Python Basics – 2 (Loops and Slicing)

7. Loop (for loop)

For loops are used for sequential traversal. For example: traversing a list or string or array etc.

```
>> data = "length"
```

Syntax :

```
>> for k in data :
```

```
    print(k) -----> k, colon, spacing (4 or 5)
```

8. String length function - returns the length of a string: Syntax – len()

```
a = "Hello"
```

```
print(len(a))
```

9. String Slicing in Python

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end. **Syntax:** var[start: stop: step]

- `data = "john"`
- `print(data[0:2:1])`

0	1	2	3	4	5	6
A	S	T	R	I	N	G

Python Basics – 1 (Pre-Req– Install PyCharm)

1. Install PyCharm(Community edition) – Free

<https://www.jetbrains.com/pycharm/download/#section=windows>

2. Get a free tier AWS account – Free

https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all

AWS Lambda with Python – Basics - 3

12. Function

Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.

Syntax:

```
def function_name(argument/parameters):  
    return expression or value
```

Example

```
# A simple Python function to check whether x is even or odd
```

```
def evenOdd(x):  
    if (x % 2 == 0):  
        print("even")  
    else:  
        print("odd")
```

```
# Driver code to call the function
```

```
evenOdd(2)  
evenOdd(3)
```

Python Basics – 2 (Loops and Slicing)

7. Loop (for loop)

For loops are used for sequential traversal. For example: traversing a list or string or array etc.

```
>> data = "length"
```

Syntax :

```
>> for k in data :
```

print(k) -----> k, colon, spacing (4 or 5)

8. String length function - returns the length of a string: Syntax – len()

```
a = "Hello"
```

```
print(len(a))
```

9. String Slicing in Python

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end. **Syntax:** var[start: stop: step]

- `data = "john"`
- `print(data[0:2:1])`

0	1	2	3	4	5	6
A	S	T	R	I	N	G

Thank You

Create S3, EC2 and DynamoDB – Introduction -1

- AWS' [**Boto3**](#) library is used commonly to **integrate Python applications with various AWS services**.
- The two most commonly used features of boto3 are **Clients and Resources**.
- **Event driven** – S3 File Upload, S3 Event notification triggers the Lambda.

Client

- Clients provide a **low-level interface to the AWS service**.
- All AWS service operations supported by clients

Syntax

```
import boto3
```

```
s3 = boto3.client("s3")
```

```
.....
```

Resource

- Resources are a higher-level abstraction compared to clients.
- Few AWS service operations not supported

Syntax

```
import boto3
```

```
s3 = boto3.resource("s3")
```

Create S3, EC2 and DynamoDB – Introduction -1

Lambda function handler in Python

The **Lambda function *handler*** is the method in your function code that processes events. When your **function is invoked**, Lambda runs the **handler method**.

```
def handler_name(event, context):  
    ...  
    return some_value
```

Naming

The Lambda function handler name specified at the time that you create a Lambda function is derived from:

The name of the file in which the Lambda handler function is located.

The name of the Python handler function.

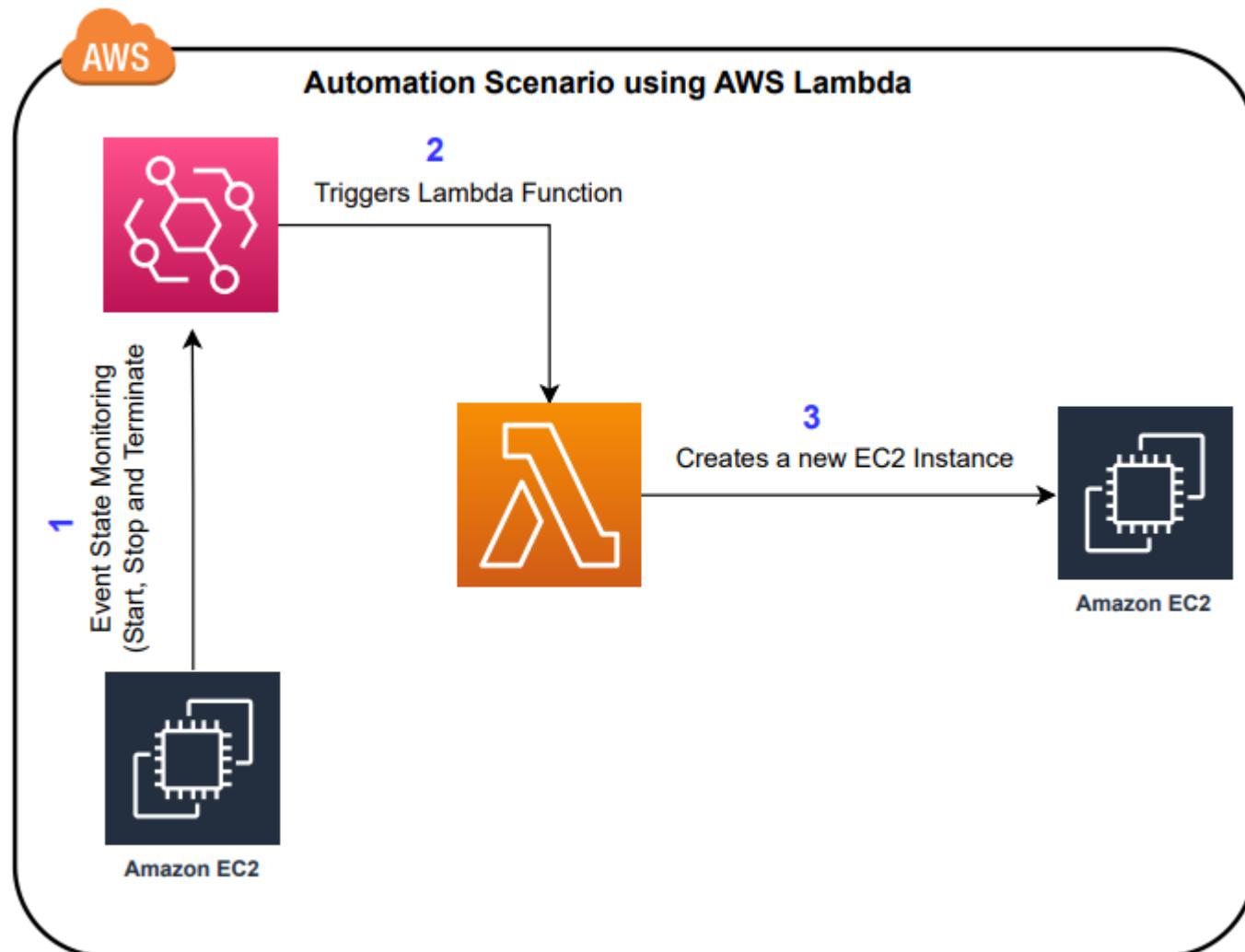
A function handler can be any name; however, the default name in the Lambda console is `lambda_function.lambda_handler`. This function handler name reflects the function name (`lambda_handler`) and the file where the handler code is stored (`lambda_function.py`).

Runtime settings Info			Edit
Runtime	Handler Info	Architecture Info	
Python 3.9	<code>lambda_function.lambda_handler</code>	x86_64	

P.S : The content is created by Rahul Trisal and copyrighted

Create EC2 using Lambda and Python Boto3 SDK

Use Case Scenario



AWS Lambda Authorizer with Cognito - Steps

1. Create backend service which client needs to access such as Lambda - **Backend Lambda - apigatewaylambdas31025**
2. Create the API Gateway
3. Resource - **students**
4. Method - **GET**
5. Deploy to a Stage - **dev**
6. Test the API GW + Lambda
7. **Create the Amazon Cognito User Pool**
 - Configure app client
 - Configure implicit grant
8. **Create Users**
9. **Generate id_token and access token from call back url**
10. **Validate using access token**

AWS Lambda Authorizer – Services to be created

1. Backend Lambda - apigatewaylambdas31025
2. APIGW - Create API GW
3. Resource - students
4. Method - GET
5. Stage - Dev
6. Lambda Authorizer Function – lambdaauthtest (1. Log event, 2. Validate Token, 3. Return the Policy)
7. Authorization Token – authorizationtoken
8. Policy Document –

```
authorizationpolicy = { "principalId": "rahul", "policyDocument": { "Version": "2012-10-17", "Statement": [ {"Action": "execute-api:Invoke", "Resource": ["arn:aws:execute-api:us-east-1:196715057542:felje4y7nj/Dev/GET/student"], "Effect": "auth"}] }}
```