Lambda

# [Exercise 1](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=exercise-1)

*Create a Lambda Function: Write a Lambda function in Node.js that responds with "Hello, [Your Name]!" when invoked. Use the****event****object to allow the name to be passed in.*

*Example Input (via the AWS Lambda test event):*

***{  
 "name": "Alice"  
}***

*Example output:*

***{  
 "message": "Hello, Alice!"  
}***

*Test the function using the AWS Lambda console.*

Afbeelding met tekst, schermopname, nummer, software

Automatisch gegenereerde beschrijving

Afbeelding met tekst, software, Computerpictogram, Webpagina

Automatisch gegenereerde beschrijving

export const handler = async (event) => {  
//extract the name from the event object  
const name = event.name;

//Construct the response  
const response = {  
 message: `Hello, ${name}`  
};

//return the response  
return response;  
};

# [Exercise 2](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=exercise-2)

*Write a Lambda function in Node.js that is triggered whenever a new file is uploaded to the S3 bucket. The Lambda function should log the file name and size of the uploaded file.*

*Example Output (logged in CloudWatch):*

***File uploaded: file.txt, Size: 1024 bytes***

1. **Maak een S3 bucket aan.**
2. **Maak een nieuwe lambda functie vanuit een blueprint en koppel je S3 bucket voor de op create klikt:**  
   A screenshot of a computer

   Description automatically generated  
   A close-up of a computer screen

   Description automatically generated  
   A screenshot of a computer

   Description automatically generated  
   Zorg dat bij ‘Event types’ de juiste triggers zijn aangeduid.
3. **Klik op create function**
4. **Pas de code aan en deploy:**  
   export const handler = async (event) => {  
    try {  
    const bucketname = event.Records[0].s3.bucket.name;  
    const key = event.Records[0].s3.object.key;  
    const size = event.Records[0].s3.object.size;  
    console.log(`Bestand geüpload: ${key}, Grootte: ${size} bytes,  
    Bucket: ${bucketname}`)  
    }catch(err){  
    console.log('Mislukt')  
    }  
   };  
   A screenshot of a computer

   Description automatically generated
5. **Als je dan een file gaat uploaden in je S3 bucket, en je gaat naar de logs kijken zie je de geuploade file ertussen staan:**  
   A yellow line on a white background

   Description automatically generated

# [Exercise 3](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=exercise-3)

*Build a Simple To-Do List API Using AWS Lambda and DynamoDB. To start off create a DynamoDB table called****Todos****with a primary key called****id****(string).*

*Create a new lambda function to add a new todo item using the following example Input (via the AWS Lambda test event):*

***{  
 "id": "1",  
 "task": "Learn AWS Lambda",  
 "status": "pending"  
}***

*Tip: you use the****put****method to add new items to a dynamoDB table.*

*Add a second lambda function that retrieves and logs all****Todo****items. Tip: you use the****scan****method to find all the items in a dynamoDB table.*

1. **Maak een DynamoDb aan:**  
   A screenshot of a computer

   Description automatically generated
2. **Create een nieuwe lambda functie en voer volgende code in:**import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
   import {  
   DynamoDBDocumentClient,  
   PutCommand,  
   } from "@aws-sdk/lib-dynamodb";  
   const client = new DynamoDBClient({});  
   const dynamo = DynamoDBDocumentClient.from(client);  
   export const handler = async (event) => {  
    const item = {  
    id: event.id,  
    task: event.task,  
    status: event.status  
    }  
    await dynamo.send(new PutCommand({  
    TableName: "excercise3",  
    Item: item  
    }));  
   };
3. **Schrijf een test met volgende code:**  
   {  
    "id": "1",  
    "task": "Learn AWS Lambda",  
    "status": "pending"  
   }
4. **Deploy en test. Controleer dan of het gewerkt heeft:**
   * Ga naar je dynamoDB en klik op ‘Explore items’
   * Je vind een nieuwe ID terug:  
     A screenshot of a computer

     Description automatically generated
5. **Maak je tweede lambda functie aan met volgende code:**  
   import { DynamoDBClient, ScanCommand } from '@aws-sdk/client-dynamodb';  
   const dynamoDB = new DynamoDBClient({  });  
     
   export const handler = async (event) => {  
    try {  
    const params = {  
    TableName: 'excercise3',  
    };  
    const result = await dynamoDB.send(new ScanCommand(params));  
    return {  
    statusCode: 200,  
    body: JSON.stringify({ items: result.Items }),  
    };  
    } catch (error) {  
    console.error(error);  
    return {  
    statusCode: 500,  
    body: JSON.stringify({ message: 'Error scanning items', error:  
    error.message }),  
    };  
    }  
   };
6. **Schrijf een lege test functie met enkel:**  
   {}
7. **Deploy en test. Je ziet de logs nu in de console:  
   A computer screen shot of a computer

   Description automatically generated**

# [Exercise 4](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=exercise-4)

*We will write a lambda function that will log our intances public IP addresses when the EC2 instance finished booting (= the machine state changed to running).*

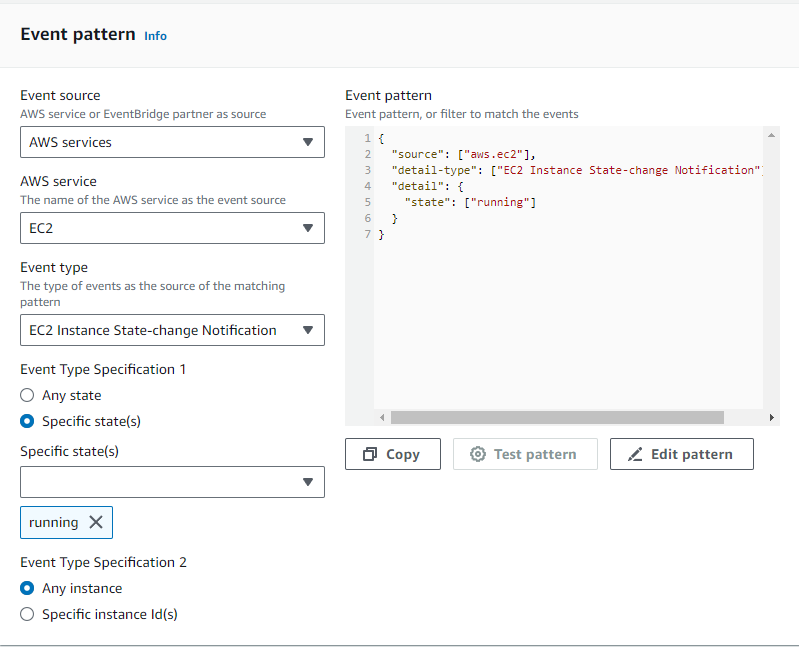
*Start of by creating a new lambda function and set up a test event based on the cloudwatch temlate. In the detail property add the following object:*

***"detail": {  
 "instance-id": "INSTANCE-ID-HERE"  
}***

*replace****INSTANCE-ID-HERE****with an ID of a running instance to test our lambda function. Now use the AWS-SDK and the****describeInstancesCommand****to fetch instance data. You can import them using the following code:*

***import { DescribeInstancesCommand, EC2Client } from "@aws-sdk/client-ec2";***

*Information on how to use this function can be found on*[***https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/ec2/command/DescribeInstancesCommand/***](https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/ec2/command/DescribeInstancesCommand/)*.*

*When you have the lambda running with the test event go to the cloudwatch service and setup a new rule under events. On the event pattern block make sure you setup the following settings:*

*As the target make sure you set the lambda function you just created. You can test your function by stopping and starting an EC2 instance and checking your lambda's cloudwatch logtrail. We will learn more about Cloudwatch later in this course.*

1. **Maak een nieuwe EC2 aan.**
2. **Create new lambda, en maak gelijk een test aan.**   
   Gebruik de template van Cloudwatch. Voeg enkel bij ‘detail’ "instance-id": "INSTANCE-ID-HERE" in. Verander het in de id van je gemaakte instance.
3. **Schrijf volgende code:**  
   import { EC2Client, DescribeInstancesCommand } from "@aws-sdk/client-ec2";  
   const client = new EC2Client({ region: "us-east-1" });   
   export const handler = async (event) => {  
    console.log("Event received:", JSON.stringify(event, null, 2));  
    const instanceId = event.detail["instance-id"];  
    if (!instanceId) {  
    console.error("Instance ID not found in event");  
    throw new Error("Instance ID is required");  
    }  
    try {  
    const command = new DescribeInstancesCommand({  
    InstanceIds: [instanceId],  
    });  
    const response = await client.send(command);  
    const publicIp = response.Reservations[0]?.Instances[0]?.PublicIpAddress;  
    if (publicIp) {  
    console.log(`Instance ID: ${instanceId}, Public IP: ${publicIp}`);  
    return { instanceId, publicIp };  
    } else {  
    console.error(`Public IP address not found for instance ID:  
    ${instanceId}`);  
    throw new Error("Public IP address not available");  
    }  
    } catch (error) {  
    console.error("Error fetching instance data:", error);  
    throw new Error("Failed to fetch instance data");  
    }  
   };
4. **Klik op deploy en test.**
5. **Ga naar cloudwatch, klik op rules, en create new rule:**  
   A screenshot of a computer

   Description automatically generated **🡺** Klik op next
6. **Kies voor ‘other’**  
   A screenshot of a computer screen

   Description automatically generated
7. **Ga naar beneden en klik volgende gegevens in:**A screenshot of a computer

   Description automatically generated
8. **Klik op Edit pattern en vul aan met volgende code:**  
   {   
    "source": ["aws.ec2"],   
    "detail-type": ["EC2 Instance State-change Notification"],   
    "detail": {   
    "state": ["running"]   
    }   
   }
9. **Klik op next, en kies voor ‘lambda function’. Duid nu jouw lambda functie aan:**  
   A screenshot of a computer

   Description automatically generated
10. **Blijf ‘next’ klikken tot je ‘create rule’ kan klikken.**
11. **Stop en start jouw EC2. Ga hierna naar je cloudwatch logs kijken:**A screenshot of a computer

    Description automatically generated

## [Extra](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=extra)

Rather than logging the IP info, lets try to post the IPs in a discord server. Set up a new discord server and go to server settings>integrations and set up a new webhook. Copy the url and save it as a variable in your lambda function. You can use the following code to post to the webhook:

**let params = {  
 embeds: [  
 {  
 "title": "EC2 instance started",  
 "color": 15258703,  
 "thumbnail": {  
 "url": "",  
 },  
 "fields": [  
 {  
 "name": `${instanceId}` ,  
 "value": publicIp,  
 "inline": true  
 }  
 ]  
 }  
 ]  
 };**

**const discordResponse = await fetch(webhookUrl, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify(params),  
 });**

Make sure you change the variables **instanceId** and **publicIp**. When using **fetch** in a lambda function you also need to add the following line to the top of your lambda function code:

**/\* global fetch \*/**

# [Exercise 5 (extra)](https://2tin-cloudessentials-resources.github.io/cloud-essentials-labs/#/6_lambda?id=exercise-5-extra)

For this lambda function we will write a web scraper that scrapes and logs the PXL dagmenu for the current day. Because we will use 3rd party modules, we will have to create a nodejs project locally and ZIP it when its finished so we can upload it to lambda. We need to do this to add the 3rd party modules to our lambda function.

Start off in a new folder called **ex5lambda** and run the command **npm init**. You can leave most of the info blank (this creates a **package.json** file). Next up run the following command in your folder:

**npm install axios cheerio**

We will use **axios** to make a http request to the PXL website. We will use **cheerio** to find the correct HTML element in the DOM tree that contains the dagmenu data.

create a new file called **index.mjs** that will hold our lambda funcion and add the following code:

**const axios = require('axios');**

**const cheerio = require('cheerio');**

**// Lambda handler**

**exports.handler = async (event) => {  
 const url = "https://www.pxl.be/Pub/Studenten/Voorzieningen-Student/Catering/Weekmenu-Campus-Elfde-Linie.html";**

**try {  
 // Fetch the content of the website  
 const { data } = await axios.get(url);  
   
 // Load the HTML into cheerio for parsing  
 const $ = cheerio.load(data);  
  
 // Find the first <h3> containing the text "dagschotel"**

**// Get the paragraph after the h3 element**

**// Log the paragraph content**

**return {  
 statusCode: 200,  
 body: JSON.stringify({  
 message: 'enter dagmenu variable here',  
 paragraph: paragraph  
 }),  
 };  
 } catch (error) {  
 console.error('Error scraping website:', error);  
 return {  
 statusCode: 500,  
 body: JSON.stringify({  
 message: 'Error scraping website',  
 error: error.message  
 }),  
 };  
 }  
};**

Lookup the cheerio module's documentation and fill in the blank code in the snippet above.

You can test your lambda function locally by creating a file **test.mjs** and add the following code to it:

**const { handler } = require('./index');**

**// Simulate calling the Lambda function**

**(async () => {  
 const event = {}; // You can define any event structure here if needed  
 const response = await handler(event);  
  
 console.log('Lambda response:', response);  
})();**

now run **npm test.js** to test your lambda function locally.

When your function works locally you can deploy it to lambda:

* Zip your project (**index.mjs**, **node\_modules**, and **package.json**).
* Upload the zip to AWS Lambda.
* Test the lambda in the AWS management console.