# Cloud Computing - Trabajo Práctico 3

## Eliminar el container dynamodb

En caso que exista un container con el nombre dynamodb el siguiente comando lo eliminará para poder reutilizar el nombre

```
docker container rm dynamodb
```

## Crear una red en docker

El siguiente comando crea una red en docker para que puedan comunicarse entre los containers

```
docker network create awslocal
```

## Ejecutar la base de datos en modo compartido

Iniciar el container de DynamoDB en modo compartido para poder acceder a las tablas tanto desde el shell como desde una aplicación externa. El container funcionará además dentro de la red creada.

```
docker run -p 8000:8000 --network awslocal --name dynamodb
dwmkerr/dynamodb -sharedDb
```

## Abrir el shell JavaScript de DynamoDB

```
http://localhost:8000/shell
```

## Crear la tabla Envio

Crear la tabla Envio con la siguiente estructura:

| **Envio** |
| --- |
| id: S (HASH)<br>fechaAlta: S<br>destino: S<br>email: S<br>pendiente: S |

Y el siguiente índice global:

| **EnviosPendientesIndex** |
| --- |
| id: S (HASH)<br>pendiente: S (RANGE) |

## Proyecto nodejs para el backend

Crear un directorio nuevo para el proyecto y dentro de él ejecutar el siguiente comando, aceptando las opciones por defecto

```
npm init
```

Agregar la librería aws-sdk para poder trabajar con DynamoDB desde el código.

```
npm install --save aws-sdk
```

Crear el archivo index.js con un editor de texto y agregar el siguiente código para definir e invocar la función `handler`.

```
var AWS = require('aws-sdk');

var handler = function() {
  var dynamodb = new AWS.DynamoDB({
    apiVersion: '2012-08-10',
    endpoint: 'http://localhost:8000',
    region: 'us-west-2',
    credentials: {
      accessKeyId: '2345',
      secretAccessKey: '2345'
    }
  });
  var docClient = new AWS.DynamoDB.DocumentClient({
    apiVersion: '2012-08-10',
    service: dynamodb
  });

  // codigo de la funcion
}

handler(); // llamada para testing
```

Comprobar que la conexión es exitosa agregando a la función una llamada a `dynamodb.listTables` y mostrar el resultado en la consola con `console.log()`.

Guardar y ejecutar el archivo `index.js` desde el directorio del proyecto con el siguiente comando

```
node index.js
```

## Convertir a función Lambda

Para convertir la funcion handler en una función lambda, realizar los siguientes cambios en el código de `index.js`

```
var AWS = require('aws-sdk');

var handler = async (event) => {
  var dynamodb = new AWS.DynamoDB({
    apiVersion: '2012-08-10',
    endpoint: 'http://dynamodb:8000',
    region: 'us-west-2',
    credentials: {
      accessKeyId: '2345',
      secretAccessKey: '2345'
    }
  });
  var docClient = new AWS.DynamoDB.DocumentClient({
    apiVersion: '2012-08-10',
    service: dynamodb
  });

  // codigo de la funcion
};

exports.handler = handler;
```

Crear el archivo `template.yaml` con la siguiente definición del API:

- `POST /envios`
  crea un nuevo envio
- `GET /envios/{idEnvio}`
  retorna un envio por id
- `GET /envios/pendientes`
  retorna un listado de envíos pendientes
- `POST /envios/{idEnvio}/entregado`
  marca un envío como entregado quitando el atributo `pendiente`

Levantar el API con SAM Local utilizando el siguiente comando:

```
sam local start-api --docker-network awslocal
```

El container de SAM Local funcionará dentro de la red `awslocal` creada anteriormente.

Comprobar el llamado a la función con Postman o un browser a la dirección

```
http://localhost:3000/envios/pendientes
```

La salida estará en la consola de sam local ya que se utilizó `console.log()` para mostrar el resultado.

A continuación, devolver el resultado utilizando la función `callback` para que pueda visualizarse correctamente en la respuesta HTTP.

## Implementar API de Envios en Lambda

Utilizando los objetos `event.httpMethod` y `event.pathParameters` implementar la funcionalidad correspondiente a los métodos declarados en la API en `template.yaml`.

La estructura del JSON que deben recibir y devolver las llamadas salen de la tabla Envio.

Por ejemplo, al obtener un envío por ID:

```
{
    "id": "1234",
    "fechaAlta": "2018-09-10T18:23:00Z",
    "destino": "MDZ",
    "email": "andsk@smgail.com",
    "pendiente": "2018-09-10T18:23:00Z"
}
```

Al crear un envío, la `fechaAlta` y `pendiente` se agregan en el backend, por lo que el POST sólo envía `destino` e `email`:

```
{
    "destino": "MDZ",
    "email": "andsk@smgail.com",
}
```