

Task 1: API Implementation

Source code and `README file` see in the zip file:

GermanPhoneNumberProjects

How to Build and Run docker file

1. Open a command prompt (all docker commands pasted in cmd)
2. Pull the docker image from the docker hub
`docker pull sshamnatvm/phonefilter`
3. Build the docker image
`docker build --rm -t sshamnatvm/phonefilter .`
4. Run the docker image
`docker run -p 5000:5000 sshamnatvm/phonefilter`
5. Open Swagger UI
<http://localhost:5000/swagger/index.html>

Swagger UI running from code and docker image

<http://localhost:5000/swagger/index.html>

Task 2: Testing

Added unit test as well

- Install Postman/Use web Postman site
 - Run the docker image created in previous
 - Start Postman.
 - Create a new request
1. For adding data to the testing environment use below POST method
 - Set the HTTP method to POST. (This post method is only for testing purposes, not visible from Swagger UI)
<http://localhost:5000/testPost>
 - Select the Body tab.
 - Select raw.
 - Set the type to JSON.
 - In the request body enters the below line and Click Send button:

```
{
```

```
  "phoneId": 10,
```

```
  "phoneResults": "+4915201365263, +4915201365264, +4915201365269"
```

```
}
```

- Enter the below line and Click Send button (this is for the second entry)

```
{
```

```
"phoneId": 11,
```

```
"phoneResults": "+4915201365276, +4915201365289, +4915201365299"
```

```
}
```

2. For getting all phone details and id use the below instruction
 - Set the HTTP method to GET and use the below link and click Send button.
 - <http://localhost:5000/getAllPhoneDetails>
3. Get Single detail and id use the below instruction
 - Set the HTTP method to GET and use the below link and click Send button.
 - <http://localhost:5000/getPhoneDetail/11>
 - <http://localhost:5000/getPhoneDetail/10>
4. For deleting phone number use the below instruction
 - Set the HTTP method to DELETE and use the below link and click Send button.
 - <http://localhost:5000/deletePhoneDetail/10>

Task 3: Deployment using AWS

- Navigate to the AWS Console and go to Lambda.
- Click Create function, then Author from scratch, give it a name and select .NET 6 for the runtime, and then click Create function.

Basic information

Function name
Enter a name that describes the purpose of your function.

minimal-api

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

.NET 6 (C#/PowerShell)

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

☒ x86_64
☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► Change default execution role

► Advanced settings

Cancel Create function

- Now under Runtime settings click Edit and change the handler to the name of the assembly, "LambdaAPI" if you're following along.

Runtime settings [Info](#)

Runtime
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

.NET 6 (C#/PowerShell)

Handler [Info](#)

LambdaAPI

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

☒ x86_64
☐ arm64

Info You can change either the function's runtime or the instruction set architecture in one update. To update both, you must repeat the update process.

Cancel Save

- Lastly for the Lambda, we need to package the assemblies and upload them to AWS.
- Go back to your project directory and run below command from project directory:

```
dotnet publish -c Release --self-contained false -r linux-x64 -o publish
```

- Zip up the contents of the /publish directory.
- Then back on the AWS Console, under Code source click Upload from and upload your .zip file.
- Now navigate to API Gateway and create a new HTTP API. Configure the integrations to use the Lambda we just created and give it a name.

Create and configure integrations

Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

Integrations [Info](#)

Lambda Remove

AWS Region: us-east-1 Lambda function: minimal-api Version: 2.0 [Learn more.](#)

Add integration

API name
An HTTP API must have a name. This name is cosmetic and does not have to be unique; you will use the API's ID (generated later) to programmatically refer to this API.

lambda-api

Cancel Review and Create Next

- Set up the routes manually. Test all 4 API routes we have configured.

Configure routes [Info](#)

API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method	Resource path	Integration target	
GET	/	minimal-api	Remove
GET	/	minimal-api	Remove
GET	/	minimal-api	Remove

Add route

Cancel Previous Next

- Leave the stages configured as they are and click Next, click Create
- It should be deployed and functioning.
- Take the provided Invoke URL from amazon and try it out.