# **Local Testing and Invocation of an AWS Lambda Function with SAM CLI and Docker**

## **Overview**

This project demonstrates how to create, build, and test an AWS Lambda function locally using AWS SAM CLI and Docker. We will simulate an AWS environment locally and invoke a simple Lambda function that responds with "Hello from Lambda!".

This guide is designed for beginners and includes all steps, from creating an AWS account to running the function locally. By the end, anyone should be able to replicate the setup.

## **Prerequisites**

Before starting, ensure you have the following installed and configured:

### **1. AWS Account**

* Sign up at [https://aws.amazon.com](https://aws.amazon.com/).
* You don’t need to deploy to the cloud yet; this guide focuses on local testing.

### **2. Python 3.11**

* Check your Python version by running: python --version
* Python 3.11 is required because our Lambda runtime will be python3.11.

### **3. Docker Desktop**

* Download and install Docker: <https://www.docker.com/products/docker-desktop>
* Ensure Docker is running: docker info
* Docker is required because SAM uses containers to emulate Lambda runtimes locally.

### **4. AWS SAM CLI**

* Install SAM CLI: <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/install-sam-cli.html>
* Verify installation: sam --version

### **5. Python Virtual Environment (optional but recommended)**

* Create a virtual environment: python -m venv venv
* Activate it:
  + Windows: venv\Scripts\activate
  + Linux/macOS: source venv/bin/activate

## **Step 1: Project Structure**

Create a folder for your project:

AWS Lambda Apps with Amazon Q Developer  
├── hello\_world/ # Lambda function code  
│ ├── app.py # Python Lambda handler  
│ └── requirements.txt # Python dependencies (if any)  
├── template.yaml # AWS SAM template  
├── README.md # Documentation  
└── venv/ # Python virtual environment

**Explanation:**

* hello\_world/app.py: Entry point for your Lambda function (file\_name.function\_name)
* requirements.txt: Python dependencies (SAM installs these automatically)
* template.yaml: Defines Lambda function, runtime, memory, timeout, and API Gateway events

## **Step 2: Create the Lambda Function (app.py)**

Inside hello\_world/, create app.py with the following content:

import json

def lambda\_handler(event, context):  
return {  
"statusCode": 200,  
"body": "Hello from Lambda!"  
}

**Explanation:**

* lambda\_handler is the default entry point.
* Returns a simple JSON response to test invocation.

## **Step 3: Create requirements.txt (Optional)**

If your Lambda needs Python packages, list them in hello\_world/requirements.txt. Example:

requests  
boto3

**Explanation:**

* SAM automatically installs dependencies into Docker during build.
* Must be inside the Lambda folder.

## **Step 4: Create the SAM Template (template.yaml)**

At the project root, create template.yaml:

AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: Sample SAM Template for Lambda

Globals:  
Function:  
Timeout: 5  
MemorySize: 128

Resources:  
HelloWorldFunction:  
Type: AWS::Serverless::Function  
Properties:  
CodeUri: hello\_world/  
Handler: app.lambda\_handler  
Runtime: python3.11  
Architectures:  
- x86\_64  
Events:  
HelloWorld:  
Type: Api  
Properties:  
Path: /hello  
Method: get

Outputs:  
HelloWorldApi:  
Description: API Gateway endpoint URL  
Value: !Sub "https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/Prod/hello/"

**Explanation:**

* CodeUri: Path to Lambda code
* Handler: Points to app.py and lambda\_handler
* Runtime: Python version
* Events: Define triggers
* Outputs: API URL if deployed to AWS

## **Step 5: Build the Lambda Locally**

Use SAM CLI to build the Lambda:

sam build --use-container --template template.yaml

**Explanation:**

* --use-container: Builds Lambda inside Docker to mimic AWS runtime
* SAM reads template.yaml, installs dependencies, and creates .aws-sam/build
* Do not manually move requirements.txt; SAM handles it automatically

## **Step 6: Validate the SAM Template**

Ensure your template is valid:

sam validate --template .aws-sam/build/template.yaml

**Explanation:** Checks formatting and resource references

## **Step 7: Test Lambda Locally**

Invoke your Lambda function:

sam local invoke HelloWorldFunction

**Expected Output:**  
{"statusCode": 200, "body": "Hello from Lambda!"}

**Explanation:** Invokes the function inside Docker to ensure it works

## **Step 8: Start Local API Gateway**

Start a local API Gateway to access Lambda via HTTP:

sam local start-api

**Default URL:** <http://127.0.0.1:3000/hello>

Access in browser or via curl/Postman:

curl <http://127.0.0.1:3000/hello>

**Expected Response:**  
{"statusCode": 200, "body": "Hello from Lambda!"}

**Note:** A 403 on /favicon.ico is normal; it does not affect Lambda.

## **Tools, Frameworks, and Services Used**

* AWS SAM CLI: Build and test Lambda locally
* Docker Desktop: Containerized environment mimicking Lambda runtime
* Python 3.11: Lambda runtime language
* Virtual Environment (venv): Isolated Python environment for packages
* API Gateway (simulated locally): HTTP endpoint for Lambda testing

## **Summary of Important Notes**

* Do not manually move files into .aws-sam/build; SAM handles this
* Ensure Docker is running before building or invoking Lambdas
* Folder structure is critical: hello\_world/ must contain app.py and requirements.txt
* template.yaml paths must match folder names (CodeUri: hello\_world/)
* Invoke locally first with sam local invoke to validate code, then use sam local start-api

## **References**

* AWS SAM Documentation: <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>
* AWS Lambda Python Runtime: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html>
* Docker Desktop: <https://www.docker.com/products/docker-desktop>