Standard Operating Procedure (SOP): Serverless Deployment of a Python 3.12 Lambda Function using Docker on Windows (x86\_64)

This document provides a step-by-step guide for deploying your my-app-test serverless application, leveraging AWS Lambda, Docker containers, the Serverless Framework, and API Gateway from a Windows 11 development environment. It specifically incorporates solutions for common issues encountered during this process, such as image manifest incompatibilities and API Gateway deployment failures.

1. Document Purpose

This SOP aims to provide a clear, repeatable process for deploying the my-app-test serverless application. It is designed to guide users through the entire deployment lifecycle, from setting up the development environment to successfully deploying and testing the application, including specific troubleshooting steps for problems previously identified.

2. Prerequisites

Ensure all the following tools are installed and configured on your Windows 11 machine:

* AWS Account: An active AWS account with programmatic access enabled for an IAM user. Store the access keys securely (e.g., via ~/.aws/credentials).
* AWS CLI: AWS Command Line Interface installed and configured with your AWS credentials.
* Node.js & npm: Install the latest LTS version.
* Serverless Framework: Install globally via npm:

bash

npm install -g serverless

Use code with caution.

* Docker Desktop: Install and ensure it's running. It includes Docker Engine and Buildx.
* Windows Subsystem for Linux 2 (WSL2): Ensure WSL2 is enabled and configured, as Docker Desktop relies on it for Linux container functionality.
* Python 3.12: Python installed locally.

3. Project Setup

Create a project directory (e.g., D:\AWS\app2). This folder will contain your application code and configuration files.

3.1 Project Structure

D:\AWS\app2\

├── Dockerfile

├── main.py

├── requirements.txt

└── serverless.yml

3.2 Application Code (main.py)

This is your Lambda function's core logic.

python

*# D:\AWS\app2\main.py*

import json

def lambda\_handler(event, context):

"""

AWS Lambda handler function for the chatbot.

It returns a simple "Hello from Docker Lambda!" message.

"""

print(f"Received event: {json.dumps(event)}")

return {

"statusCode": 200,

"body": "Hello from Docker Lambda!"

}

Use code with caution.

3.3 Dependencies (requirements.txt)

This file lists the Python packages your Lambda function requires. If you have no external dependencies, this file can be empty.

# D:\AWS\app2\requirements.txt

# Example: requests==2.32.3

# Add other Python package dependencies here

3.4 Dockerfile

This defines the Docker container image for your Lambda function. It targets linux/amd64 to match the AWS Lambda execution environment.

dockerfile

*# D:\AWS\app2\Dockerfile*

FROM --platform=linux/amd64 public.ecr.aws/lambda/python:3.12

*# Set the working directory to the Lambda task root*

WORKDIR ${LAMBDA\_TASK\_ROOT}

*# Copy requirements.txt and install Python dependencies*

*# Upgrade pip to ensure the latest version is used*

*# --no-cache-dir minimizes image size by not storing cached packages*

*# --prefer-binary can speed up installation if pre-compiled binaries are available*

COPY requirements.txt .

RUN pip install --upgrade pip && \

pip install --no-cache-dir --prefer-binary -r requirements.txt

*# Copy your application code*

COPY main.py .

*# Set the Lambda function handler (format: <filename>.<function\_name>)*

CMD ["main.lambda\_handler"]

Use code with caution.

3.5 Serverless Configuration (serverless.yml)

This file configures the Serverless Framework for deploying your containerized Lambda function and associated AWS resources, including API Gateway.

yaml

*# D:\AWS\app2\serverless.yml*

service: my-app-test

provider:

name: aws

runtime: provided.al2 *# Required for image-based functions*

region: us-east-1 *# Specify your target AWS region*

timeout: 30 *# Lambda function timeout in seconds*

memorySize: 2048 *# Lambda function memory in MB*

ecr:

images:

app-lambda-test:

path: . *# Path to your Dockerfile for image building*

platform: linux/amd64 *# Crucial: Build image for x86\_64 architecture*

provenance: false *# FIX: Disable in-toto provenance layer for Lambda compatibility*

functions:

chatbot:

image:

name: app-lambda-test *# Reference the Docker image defined above*

architecture: x86\_64 *# Set Lambda function to run on x86\_64 (matches image)*

events:

- http: *# Defines an API Gateway endpoint trigger*

path: chat *# The URL path for the API endpoint*

method: post *# The HTTP method allowed (e.g., POST)*

environment:

*# Example: Pass an environment variable to the Lambda function from your shell environment*

GROQ\_API\_KEY: ${env:GROQ\_API\_KEY}

Use code with caution.

4. Docker Build and Push Preparation

The following steps ensure your Docker environment is correctly set up for cross-platform builds.

4.1 Initialize Docker Buildx

This ensures Docker Buildx is properly configured for building images, especially for different architectures.

* Open PowerShell as Administrator in your project directory (D:\AWS\app2>).
* Run the following commands:

powershell

docker buildx create --use

docker buildx inspect --bootstrap

Use code with caution.

* + *Troubleshooting:* If you face issues, you can remove existing builders and recreate:

powershell

docker buildx rm $(docker buildx ls -q)

docker buildx create --use

docker buildx inspect --bootstrap

Use code with caution.

5. Deployment using Serverless Framework

This section outlines the steps for deploying your service, incorporating the fix for the "image manifest not supported" error and the API Gateway deployment issue.

5.1 Address AWS & Local State (Fixing "Invalid API identifier")

The "Invalid API identifier" error indicates an inconsistency between Serverless and AWS resources. Resolve this by cleaning up these resources.

1. Attempt to remove the existing Serverless deployment:
   * From the D:\AWS\app2> directory:

bash

serverless remove

Use code with caution.

* + If successful, the API Gateway and other resources are cleanly deleted; proceed to step 5.2.
  + If the command fails with the "Invalid API identifier" error:
    1. Go to the AWS Management Console -> API Gateway service in your specified region (us-east-1).
    2. Look for an API Gateway corresponding to your my-app-test service.
    3. Manually delete any matching API Gateway.
    4. Optionally try serverless remove again.

1. Clean local Serverless artifacts: Delete the .serverless directory. This clears any cached deployment state.
   * From D:\AWS\app2>:

bash

rm -rf .serverless/ *# For PowerShell/Git Bash*

*# or for Command Prompt:*

rmdir /s /q .serverless\

Use code with caution.

5.2 Deploy the Service

With a clean state, deploy your Serverless service. Serverless will build the Docker image, push it to ECR, and deploy the Lambda function and API Gateway.

* From the D:\AWS\app2> directory:

bash

serverless deploy

Use code with caution.

* The deployment output will provide the API Gateway endpoint URL (e.g., https://xxxxxx.execute-api.us-east-1.amazonaws.com/chat) upon successful completion.

6. Verification and Testing

1. AWS Console Verification:
   * Open the AWS Management Console and navigate to the Lambda service in your specified region.
   * Find the chatbot Lambda function.
   * Check its Configuration tab:
     + Runtime settings: Confirm the Handler is main.lambda\_handler and Architecture is x86\_64.
     + Container image: Verify that the image URI and tag match what Serverless deployed.
   * Navigate to the API Gateway service. Confirm that a new API Gateway was created and is correctly configured with a /chat POST endpoint linked to your Lambda function.
2. Invoke the Function via API Gateway:
   * Use a tool like Postman, curl, or Insomnia to send a POST request to your API Gateway endpoint.
   * Example curl command (replace URL with your actual endpoint):

bash

curl -X POST https://YOUR\_API\_GATEWAY\_URL/chat

Use code with caution.

* + You should receive the response: Hello from Docker Lambda! and a 200 OK status.

1. Check CloudWatch Logs:
   * Go to the CloudWatch service in the AWS console.
   * Navigate to Log groups.
   * Find the log group for your Lambda function (e.g., /aws/lambda/my-app-test-dev-chatbot).
   * Inspect the latest log stream to see the Received event message and ensure there are no errors.

7. Cleanup

To remove all deployed resources:

* From the D:\AWS\app2> directory:

bash

serverless remove