AWS Lambda Container Image Deployment with Amazon ECR

By

Esmeralda C. Cabrera Ventura

Project Overview

AWS Lambda traditionally required functions to be packaged as ZIP archives, which could limit flexibility for teams that prefer container-based development workflows. With AWS Lambda's support for OCI-compliant container images, developers can now combine serverless execution with Docker-based packaging, gaining better control over dependencies, runtime behavior, and build reproducibility.

This project demonstrates an end-to-end containerized serverless deployment by building a Python web application as a Docker image, storing it in Amazon Elastic Container Registry (ECR), and deploying it as an AWS Lambda function.

Situation / Problem Statement

Modern development teams often standardize on containers to ensure consistent builds across environments. However, they still want to benefit from:

- Event-driven execution
- Automatic scaling
- Pay-per-use pricing
- Minimal infrastructure management

The challenge is to bridge containerized development with a serverless runtime, without managing long-running servers or orchestration platforms.

Arrived Solution

The solution uses AWS Lambda container image support:

1. A Python application is packaged as a Docker image using an Alpine Linux base.
2. The image is built on an EC2 instance acting as a temporary build host.
3. The container image is stored in a private Amazon ECR repository.
4. AWS Lambda pulls the image directly from ECR and executes it as a function.
5. The deployment is validated using a test invocation.

This approach preserves container tooling while eliminating the need to manage EC2 instances in production.

Architecture Design

High-Level Architecture Flow

Developer / Build Host (EC2)

↓

Docker Build (OCI Image)

↓

Amazon ECR (Private Repository)

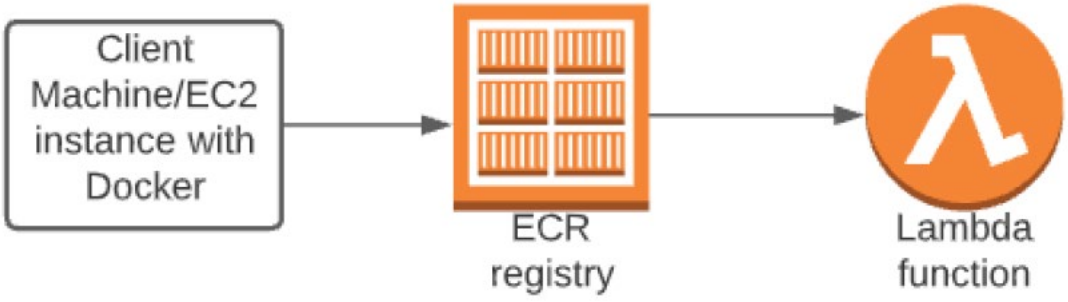
↓

AWS Lambda (Container Image Runtime)

↓

Function Invocation (Test Event)

Diagram Architecture



AWS Services Used

- **EC2 (Ubuntu 22.04)** – temporary build environment for Docker
- **Docker** – containerization of the application
- **Amazon ECR** – private container image registry
- **AWS Lambda** – serverless execution of the container image
- **IAM Roles** – secure permissions for EC2 and Lambda

This design cleanly separates build responsibilities from runtime execution, which mirrors real-world CI/CD patterns.

Step-by-Step Implementation Guide

Step 1: Docker Image Creation

Objective: Package the application as an OCI-compliant container image.

- Launched an Ubuntu 22.04 EC2 instance in the default VPC.
- Attached the IAM role LabInstanceProfile (or an equivalent role with ECR access).
- Uploaded the provided OCI.zip file to the instance using scp.
- Installed Docker and supporting tools using the provided installation script.
- Installed and configured AWS CLI:
    - Region: us-east-1
    - Output format: json
- Built the Docker image using the provided Dockerfile:
  docker build -t lambda_ecr .
- Verified that the image was successfully created using the command:
  docker images

Result:
A locally built Docker image containing the Python application, ready for registry upload.

Screenshots:

1. Docker Image Creation

2. Docker Images List

Step 2: Create Amazon ECR Repository and Upload Image

Objective: Store the container image in a managed, secure registry.

1. Navigate to Amazon ECR in the AWS Console.
2. Create a private repository named lambda_ecr.
3. Open View push commands for the repository.
4. Authenticate Docker to ECR using AWS CLI.
5. Tag the local Docker image with the ECR repository URI.
6. Push the image to ECR.

To simplify commands, a shell variable was used to store the AWS account ID:

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

Result:
The Docker image is successfully stored in Amazon ECR and available for Lambda consumption.

Screenshots:

1. Amazon ECR Repository Creation
2. ECR View Push Commands
3. AWS Account Number Stored in ${ACCOUNT_ID}
4. ECR Login Succeeded.
5. Tagging of Docker Image
6. Docker Image Pushed.
7. ECR Image Successfully Uploaded to Repo

Step 3: Create AWS Lambda Function from Container Image

Objective: Deploy and validate the container image using Lambda.

- Navigated to AWS Lambda in the AWS Console.
- Selected Create function.
- Chose Container image as the deployment option.
- Browsed and selected the image from the ECR repository.
- Assigned the existing IAM role LabRole.
- Created the function and waited for provisioning to complete.
- Ran the default Hello World test event.

Result:
The Lambda function executed successfully using the container image pulled from ECR.

Screenshots:

1. Lambda Container Image Selection
2. Lambda Execution Role Selection
3. Lambda Function Creation
4. Lambda Test Successful

Validation and Outcome

- The container image was built and stored successfully.
- Lambda pulled and executed the image without errors.
- The test invocation confirmed correct runtime behavior.

This confirms that container-based workloads can be executed serverlessly using AWS Lambda without managing application servers.

Why This Project Matters

This project reflects a modern cloud-native deployment pattern used in production environments where teams want:

- Predictable, containerized builds
- Serverless scaling and cost efficiency
- Minimal operational overhead

It demonstrates practical knowledge of AWS serverless + container integration, a highly relevant skill set for cloud engineering roles.