AWS Day 5 – DevOps, Serverless, and Security Best Practices

---

1. Introduction to DevOps in AWS

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). The goal is to shorten the system development life cycle and deliver high-quality software continuously.

DevOps Tools in AWS:

- **AWS CodeCommit** – Source control (Git).
- **AWS CodeBuild** – Build and test code automatically.
- **AWS CodeDeploy** – Automate application deployments.
- **AWS CodePipeline** – CI/CD pipeline orchestration.

Why use DevOps in AWS?

- Automates the software release process.
- Improves development speed and efficiency.
- Integrates with existing tools (GitHub, Jenkins).
- Scales easily with infrastructure.

---

2. CodePipeline, CodeBuild, and CodeDeploy (Overview)

AWS CodePipeline

- Automates the build, test, and deploy phases of the release process.
- Supports source providers like GitHub, CodeCommit, and Bitbucket.

Example Flow:

1. Code pushed to repository.
2. Trigger CodePipeline.
3. Build and test using CodeBuild.
4. Deploy using CodeDeploy.

AWS CodeBuild

- Fully managed continuous integration service.
- Compiles source code, runs tests, and produces software packages.

Buildspec.yml – Defines the build commands and settings.

AWS CodeDeploy

- Automates deployment to EC2, Lambda, or on-premise.
- Supports in-place and blue/green deployments.

---

3. AWS Lambda: Triggers & Use Cases

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers.

Key Concepts:

- **Function:** Your code to be executed.
- **Trigger:** Event that invokes the function (API Gateway, S3, DynamoDB, etc.).
- **Runtime:** Supported languages (Node.js, Python, Java, etc.).

Common Use Cases:

- File processing (triggered by S3 uploads).
- API backends (combined with API Gateway).
- Scheduled tasks (using EventBridge or CloudWatch).
- Real-time notifications (triggered by DynamoDB Streams).

---

4. API Gateway + Lambda Integration

Amazon API Gateway helps you create, publish, and manage RESTful APIs.

Integration with Lambda:

1. Create a REST API in API Gateway.
2. Define a resource and method (e.g., `/users`, POST).
3. Integrate the method with a Lambda function.
4. Deploy the API to a stage (e.g., `prod`).
5. Invoke the API endpoint to trigger Lambda.

Benefits:

- Fully managed and scalable.
- Handles authentication, rate-limiting, and monitoring.
- Seamless connection to Lambda and other AWS services.

---

5. Security Best Practices

Shared Responsibility Model

- **AWS Responsibility:** Security *of* the cloud (hardware, networking, etc.).
- **Customer Responsibility:** Security *in* the cloud (data, access control, configurations).

Key Management Service (KMS)

- Encrypts data using customer-managed keys.
- Works with S3, EBS, RDS, Lambda, and more.
- Supports automatic key rotation and IAM integration.

---

6. Lab: Create a Simple Serverless App with Lambda

Objective:

Build a simple function that responds to an HTTP request.

Step-by-Step: Create Lambda Function and Trigger via API Gateway

Step 1: Create a Lambda Function

1. Go to AWS Console → Lambda → Create Function.
2. Choose **Author from scratch**.
3. Function name: `HelloWorldFunction`.
4. Runtime: Python 3.9 (or Node.js).
5. Click **Create Function**.
6. In the **Function code**, enter:

```
def lambda_handler(event, context):

    return {

        'statusCode': 200,

        'body': 'Hello from Lambda!'

    }
```

7. Click **Deploy**.

Step 2: Create an API Gateway

1. Go to **API Gateway** → Create API → REST API.
2. Choose **Build** under REST API (not HTTP).
3. API Name: `HelloWorldAPI`.
4. Create a new resource: `/hello`.
5. Create a new **GET method**:
   - Integration type: Lambda Function.
   - Region: ap-south-1 (Mumbai).

    - Lambda Function: `HelloWorldFunction`.

6. Deploy the API:
    - Choose `Deploy API`.
    - Create new stage: `prod`.
7. Copy the **Invoke URL**:

arduino

```
https://<api-id>.execute-api.ap-south-1.amazonaws.com/prod/hello
```

Visit this URL in the browser. You will get:
"Hello from Lambda!"

Steps for the Instructor to Demonstrate

Step 1: Prepare the Static Website Code

Create a simple HTML file (`index.html`) with any welcome message.

```html
<!-- index.html -->
<html>
  <head><title>Demo CI/CD Page</title></head>
  <body><h1>Hello from CodePipeline!</h1></body>
</html>
```

Push this file into a **public GitHub repository**. (Can name it: `aws-cicd-demo`)

Step 2: Create an S3 Bucket to Host the Website

1. Go to **S3 > Create bucket**
2. Name: `my-cicd-demo-bucket-<your-unique-id>`
3. Uncheck **"Block all public access"**
4. Enable static website hosting:
    - Index document: `index.html`
5. Save the **endpoint URL** for later testing.

Step 3: Create a CodePipeline

1. Navigate to **CodePipeline > Create pipeline**
2. Pipeline name: `html-site-pipeline`
3. Role: Let AWS create a new one
4. **Source Stage**:

    - Provider: GitHub (connect your account)
    - Repository: Select your `aws-cicd-demo`
    - Branch: main
5. **Build Stage**:
    - Select **AWS CodeBuild**
    - Create a new project
        - Runtime: Amazon Linux, standard image
        - Buildspec: Insert your own, or use simple buildspec like below
        - Artifact: Enabled
6. **Deploy Stage**:
    - Provider: **Amazon S3**
    - Bucket: Select your S3 bucket created earlier

Step 4: Configure a Simple Buildspec File

Create a file in your GitHub repo named `buildspec.yml`

yaml
CopyEdit
```
version: 0.2

phases:
  build:
    commands:
      - echo "No actual build needed for static HTML"
artifacts:
  files:
    - index.html
```

Step 5: Push Changes and Watch Auto Deployment

1. Make a change to `index.html`, e.g., update text.
2. Commit and push to GitHub.
3. Open CodePipeline – it will:
    - Detect code change
    - Run CodeBuild (just passes artifacts)
    - Deploy updated HTML to S3
4. Visit the S3 website endpoint to see the updated result.