Animesh Parab T2-T21 88

WRITTEN ASSIGNMENT-2

Q.1 How to deploy Lambda function on AWS?

=>Deploying a Lambda function on AWS involves several steps. Here's a detailed guide on how to deploy a Lambda function:

Prerequisites:

An AWS account.

The AWS Command Line Interface (CLI) installed and configured with appropriate permissions.

Your Lambda function code packaged as a ZIP archive or uploaded to an Amazon S3 bucket.

Familiarity with the programming language and runtime you're using for your Lambda function (e.g., Node.js, Python, Java, etc.).

Step-by-Step Guide to Deploying a Lambda Function:

Create or Prepare Your Lambda Function Code:

Write your Lambda function code or prepare it if you haven't already. Ensure it follows the AWS Lambda function structure, including the handler function.

Package Your Code:

If your function code consists of multiple files or dependencies, package it as a ZIP archive. Make sure that the primary function handler is at the top level of the archive.

Create an Execution Role (if needed):

Lambda functions often need permissions to interact with other AWS services. Create an AWS Identity and Access Management (IAM) role that grants the necessary permissions to your Lambda function. The role should have policies attached that allow access to AWS resources, like S3, DynamoDB, or others.

Upload Code to Amazon S3 (if needed):

If your deployment package is larger than 3 MB, you will need to upload it to an Amazon S3 bucket. Make sure your Lambda function has permissions to access this bucket.

Deploy the Lambda Function:

You can deploy a Lambda function using the AWS Management Console, AWS CLI, AWS SDKs, or AWS CloudFormation. Here's how to deploy using the AWS CLI:

Open a terminal and run the following AWS CLI command to create your Lambda function:

aws lambda create-function --function-name MyFunctionName --runtime nodejs14.x --role arn:aws:iam::123456789012:role/MyRole --handler index.handler --zip-file fileb://function.zip

Replace MyFunctionName with your desired function name.

Specify the correct runtime for your function (e.g., nodejs14.x for Node.js 14).

Use the --role option with the ARN of the IAM role you created.
Specify the --handler option with the name of your handler function.
Use --zip-file to reference your deployment package.

Test Your Lambda Function:

After creating your Lambda function, you can test it using the AWS Management Console, AWS CLI, or an SDK. Make sure it works as expected.

Configure Event Sources (if needed):

If your Lambda function is triggered by events from other AWS services (e.g., S3, SNS, API Gateway), configure these event sources in the AWS Management Console.

Set Up Environment Variables (if needed):

If your function relies on environment variables, configure these in the Lambda function's configuration.

Deploy Updates (if needed):

If you make changes to your function code or configuration, you can update the Lambda function by uploading a new deployment package, and the changes will be applied.

Monitor and Troubleshoot:

Use AWS CloudWatch and other monitoring tools to track the performance and behavior of your Lambda function. You can also view logs and error messages in CloudWatch Logs to troubleshoot issues.

Scale and Manage Your Function:

AWS Lambda automatically scales your function to handle incoming requests. You can configure concurrency limits, timeout settings, and other properties to manage its behavior.

Cost Management:

Keep an eye on the costs associated with your Lambda function, as you'll be billed based on the number of requests and duration of execution. Utilize cost management tools and set up billing alerts to avoid unexpected charges.

Deploying a Lambda function on AWS is a straightforward process, but it's important to pay attention to configuration, permissions, and monitoring to ensure your function operates as expected in a production environment.

Q.2 What are the deployment options for AWS Lambda?

AWS Lambda offers several deployment options, each suited for different use cases and development workflows. Here are the primary deployment options for AWS Lambda, explained in detail:

Upload Deployment Package:

This is the simplest deployment option. You create a ZIP archive that contains your function code and dependencies. Then, you manually upload it when creating or updating your Lambda function. This approach is suitable for small functions or when you need full control over your deployment process.

Steps:

Create a ZIP archive containing your function code and dependencies.
Use the AWS Management Console, AWS CLI, or AWS SDKs to create or update your Lambda function, providing the ZIP archive as the deployment package.
S3 Bucket Deployment:
For larger deployment packages or for separating the deployment process from the Lambda function creation or update, you can store your deployment package in an Amazon S3 bucket. When you create or update a Lambda function, you specify the S3 bucket location for your deployment package.
Steps:
Upload your deployment package to an S3 bucket.
Use the AWS Management Console, AWS CLI, or AWS SDKs to create or update your Lambda function, specifying the S3 bucket location.
AWS Serverless Application Model (SAM):
SAM is an open-source framework for building serverless applications. It extends AWS CloudFormation to provide a simplified way to define the Amazon API Gateway APIs, AWS Lambda functions, and Amazon DynamoDB tables needed by your serverless application. You can define your Lambda function configurations and deployment details in a template.yaml file, which SAM uses to create and deploy the function.
Steps:
Create a template.yaml file that defines your Lambda function and its dependencies.
Use the AWS SAM CLI to package and deploy your serverless application to AWS. This CLI tool simplifies packaging and deployment, including the creation of Amazon S3 buckets for deployment.
Lambda Layers:
Lambda Layers allow you to separate your function code from its dependencies. You can create a custom Layer with your dependencies and then reference it in your

Lambda function. When updating the dependencies, you only need to update the Layer, reducing the size and complexity of the deployment package.
Steps:
Create a Lambda Layer containing your dependencies.
Reference the Layer in your Lambda function configuration.
When updating dependencies, update the Layer without changing your function code.
Container Images (AWS Lambda for Containers):
AWS Lambda added support for running functions in container images. With this option, you build a container image with your function code, dependencies, and any runtime environment you need. You can use your preferred container registry to store the image. Lambda manages the container execution, scaling, and resource allocation.
Steps:

Build a Docker container image with your function code and dependencies.
Push the image to a container registry (e.g., Amazon ECR, Docker Hub).
Create a Lambda function using the image from your container registry.
Continuous Deployment Tools:
You can integrate AWS Lambda deployment into your continuous deployment pipelines using tools like AWS CodePipeline, AWS CodeBuild, Jenkins, or any other CI/CD solution. This approach automates the deployment process, allowing you to push changes to your function code in a version-controlled manner.
Steps:
Set up a CI/CD pipeline that monitors your code repository.
Configure the pipeline to build and deploy your Lambda function when changes are detected.
Each of these deployment options has its own advantages and is suitable for different scenarios. The choice depends on factors such as the size and complexity of your function, your development workflow, and whether you require more granular control over your deployments.

Q.3 What are the 3 full deployment modes that can be used for AWS?
In the context of AWS, there are three primary full deployment modes, each offering a distinct approach to deploying and managing applications. These modes are designed to accommodate different use cases and operational requirements. Let's explore each of them in detail:
EC2-Based Deployment:
Description:
EC2-based deployment is the traditional deployment mode in AWS where you provision and manage virtual machines (EC2 instances) to run your applications. In this mode,

you have full control over the underlying infrastructure, including the choice of instance types, operating systems, and configuration. This mode is ideal for applications that require a high degree of customization or when you have legacy systems that need to run in a traditional virtualized environment.
Use Cases:
Running legacy applications or software that can't be containerized.
Applications with complex network configurations or specific hardware requirements.
When you need to manage the entire stack from the operating system up.
Key Components:
Amazon Elastic Compute Cloud (EC2) instances.
Amazon Virtual Private Cloud (VPC) for network isolation.
Elastic Load Balancers for distributing traffic.
Amazon RDS or other database services for data storage.
Benefits:
Complete control over infrastructure.

Compatibility with a wide range of software.
Ability to run non-containerized or legacy applications.

Challenges:

Manual scaling and management of EC2 instances.
More operational overhead compared to serverless or container-based solutions.
Limited automation compared to other deployment modes.

Serverless Deployment:

Description:

Serverless deployment is a modern cloud computing paradigm in which you focus solely on writing code (usually in the form of functions) and let the cloud provider manage all the underlying infrastructure. AWS Lambda is a key component of this approach, allowing you to run code in response to events without provisioning or managing servers. Serverless computing is highly scalable and event-driven.

Use Cases:

Building scalable, event-driven applications.
Microservices architecture.
Real-time data processing and analysis.
Reducing operational overhead by offloading infrastructure management.

Key Components:

AWS Lambda for running code in response to events.
Amazon API Gateway for exposing APIs.
Various AWS services for data storage and processing.
Amazon EventBridge or Amazon S3 event triggers for event-driven applications.

Benefits:

Auto-scaling and high availability.
Minimal operational overhead.
Pay-per-use pricing.
Easy integration with other AWS services.

Challenges:

Stateless execution, which may require workarounds for stateful applications.
Limited runtime options compared to EC2 instances.
Function duration and resource limits.

Container-Based Deployment:

Description:

Container-based deployment leverages containerization technology to package applications and their dependencies into a consistent and portable format. AWS offers Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS) for managing containers in a scalable, automated, and highly available manner. This deployment mode is ideal for containerized applications, microservices, and orchestrating container workloads.

Use Cases:

Microservices architecture.

Porting and running containerized applications.
Managing applications that need to scale and have dependencies isolated in containers.

Key Components:

Amazon ECS or Amazon EKS for managing containers.
Amazon ECR for container registry.
Docker for building and running containers.
Kubernetes for container orchestration (EKS).

Benefits:

Scalability and flexibility of containerization.
Portability and consistency of containers.
Advanced orchestration and management features in Kubernetes.

Challenges:

Container management complexity.
Learning curve for orchestrators like Kubernetes.
Ongoing operational overhead.

These three full deployment modes represent different approaches to deploying and managing applications in AWS. Your choice should be based on your specific requirements, including the nature of your applications, your scalability needs, and your desired level of operational control. It's not uncommon for organizations to use acombination of these deployment modes, depending on their application portfolio and use cases.

Q.4 What are the 3 components of AWS Lambda?

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS) that allows you to run code in response to events without the need to manage servers. AWS Lambda has three core components that work together to enable serverless compute capabilities:

Lambda Function:

Description: A Lambda function is the core unit of execution in AWS Lambda. It represents your code, which can be written in various programming languages such as Node.js, Python, Java, Go, and more. A Lambda function is a small, self-contained piece of code that can perform a specific task when triggered by an event. It can be as simple as a few lines of code or more complex, and it typically follows a specific structure, including a handler function that AWS Lambda invokes when an event occurs.

Use Cases: Lambda functions are used for a wide range of purposes, including data processing, automation, real-time file processing, API endpoints, and more. They are particularly well-suited for building serverless applications and microservices.

Key Characteristics:

Small, single-purpose code.
Stateless (no persistent storage of data between invocations).
Event-driven execution.
Automatic scaling and resource allocation.

Event Source:
Description: Event sources are triggers that initiate the execution of a Lambda function. These sources can be various AWS services or external systems that generate events. When an event occurs, AWS Lambda is automatically invoked, and the event data is passed to the Lambda function for processing. AWS Lambda supports a wide range of event sources, including AWS services like Amazon S3, Amazon DynamoDB, AWS SNS, and custom event sources using AWS Step Functions.
Use Cases: Event sources enable Lambda functions to respond to changes in data, incoming requests, system events, and more. This makes them suitable for building event-driven applications and automating workflows.
Key Characteristics:
Diverse sources, including AWS services and custom events.
Real-time event triggering. Integration with various AWS services.
Execution Environment:
Description: The execution environment is the runtime environment where Lambda functions run. AWS Lambda manages and provisions these environments dynamically as needed, and it abstracts the underlying infrastructure from developers. The execution environment includes the compute resources (CPU, memory) and the network configuration necessary for the function's execution. The environment automatically scales with incoming event load.
Use Cases: The execution environment is responsible for ensuring that Lambda functions can run in a scalable and highly available manner without the need for manual provisioning or management. It enables the on-demand execution of code.
Key Characteristics:
Automatic provisioning and scaling.
Abstracts infrastructure management.
Resource allocation (memory and CPU) defined per function.
Isolation between concurrent executions.
Together, these three components form the foundation of AWS Lambda. A Lambda function processes events from various event sources within an execution environment provided by AWS Lambda. The serverless nature of Lambda, where you focus on code rather than infrastructure, allows you to build applications that are highly scalable and responsive to real-time events with minimal operational overhead. This serverless model is particularly valuable for organizations looking to optimize resource utilization and reduce infrastructure management complexities.