4.2. Student Handout

AWS Serverless Application Model (SAM) - Student Handout

Introduction to AWS SAM

AWS Serverless Application Model (SAM) is an open-source framework designed to simplify the process of building, deploying, and managing serverless applications on AWS. It allows developers to focus on writing code while AWS manages the infrastructure.

What is AWS SAM?

AWS SAM is a framework that provides a simplified way to define serverless applications using a YAML configuration file. It specifies the AWS resources needed for the application, such as Lambda functions, API Gateway, and DynamoDB tables.

Examples:

1. **Lambda Function Definition**: Define a Lambda function in a SAM template to process image uploads.
2. **API Gateway Setup**: Use SAM to configure an API Gateway that routes HTTP requests to a Lambda function.
3. **DynamoDB Table Creation**: Specify a DynamoDB table in a SAM template to store user data.

Benefits of AWS SAM

1. **Simplified Development**: Define your application in a YAML file, and SAM handles the infrastructure setup.
2. **Local Testing**: Test serverless applications locally before deploying to AWS.
3. **Faster Deployment**: Automate packaging and deployment processes for quicker application launches.

4. **Cost-Effective**: Pay only for the resources you use, eliminating the need for idle server maintenance.

Examples:

1. **YAML Configuration**: Use a single YAML file to define multiple AWS resources for a microservices application.
2. **Local API Testing**: Test an API locally using SAM CLI before deploying it to AWS.
3. **Automated Deployment**: Deploy a serverless application with a single command using SAM CLI.

Key Concepts in AWS SAM

AWS Lambda

AWS Lambda is a compute service that runs code in response to events without the need for server management.

Examples:

1. **Event-Driven Processing**: Use Lambda to process S3 bucket events, such as file uploads.
2. **Scheduled Tasks**: Schedule Lambda functions to run at specific intervals using CloudWatch Events.
3. **Data Transformation**: Use Lambda to transform data before storing it in a database.

API Gateway

API Gateway is a service for creating and managing APIs, acting as a front door for applications.

Examples:

1. **REST API Creation**: Create a RESTful API to interact with a Lambda function.
2. **WebSocket API**: Set up a WebSocket API for real-time communication with clients.
3. **Rate Limiting**: Implement rate limiting on API endpoints to control traffic.

DynamoDB

DynamoDB is a fully managed NoSQL database service for storing application data.

Examples:

1. **User Data Storage**: Store user profiles and preferences in a DynamoDB table.
2. **Session Management**: Use DynamoDB to manage user sessions in a web application.
3. **Data Caching**: Implement a caching layer using DynamoDB for frequently accessed data.

SAM Templates

SAM templates are YAML files that define the resources and configurations for serverless applications.

Examples:

1. **Resource Definition**: Define multiple Lambda functions and API Gateway endpoints in a single SAM template.
2. **Environment Variables**: Use SAM templates to set environment variables for Lambda functions.
3. **IAM Roles**: Specify IAM roles and permissions for AWS resources in a SAM template.

Installing and Configuring SAM CLI

Steps to Install SAM CLI:

1. **Install Homebrew (macOS/Linux)**:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. **Install SAM CLI**:

```
brew tap aws/tap

brew install aws-sam-cli
```

3. **Verify Installation**:

```
sam --version
```

Configuring AWS Credentials for SAM CLI:

1. **Install AWS CLI**: Follow the instructions here.
2. **Configure AWS CLI**:

```
aws configure
```

Examples:

1. **Homebrew Installation**: Use Homebrew to install SAM CLI on a macOS system.
2. **AWS CLI Configuration**: Configure AWS CLI with access keys for deploying applications.
3. **Version Check**: Verify SAM CLI installation by checking the version.

Creating and Deploying Serverless Applications with SAM CLI

Step 1: Initialize a New SAM Application

```
sam init
```

Step 2: Build the Application

```
cd sam-app

sam build
```

Step 3: Deploy the Application

```
sam deploy --guided
```

Examples:

1. **Python Runtime**: Initialize a SAM application using the Python runtime.
2. **Hello World Template**: Use the Hello World template to create a basic serverless application.
3. **Guided Deployment**: Deploy an application using the guided deployment option in SAM CLI.

Debugging and Testing Serverless Applications Locally

Simulating AWS Lambda Executions Locally

```
sam local invoke "HelloWorldFunction"
```

Simulating API Gateway Locally

```
sam local start-api
```

Examples:

1. **Local Lambda Invocation**: Test a Lambda function locally using SAM CLI.
2. **Local API Server**: Start a local server to test API Gateway configurations.
3. **HTTP Request Testing**: Send HTTP requests to a local API server for testing.

Hands-On: Building, Deploying, and Testing a Serverless Application

Step 1: Create a New SAM Application

```
sam init
```

Step 2: Build the Application

```
cd sam-app

sam build
```

Step 3: Test the Application Locally

```
sam local start-api
```

Step 4: Deploy the Application

```
sam deploy --guided
```

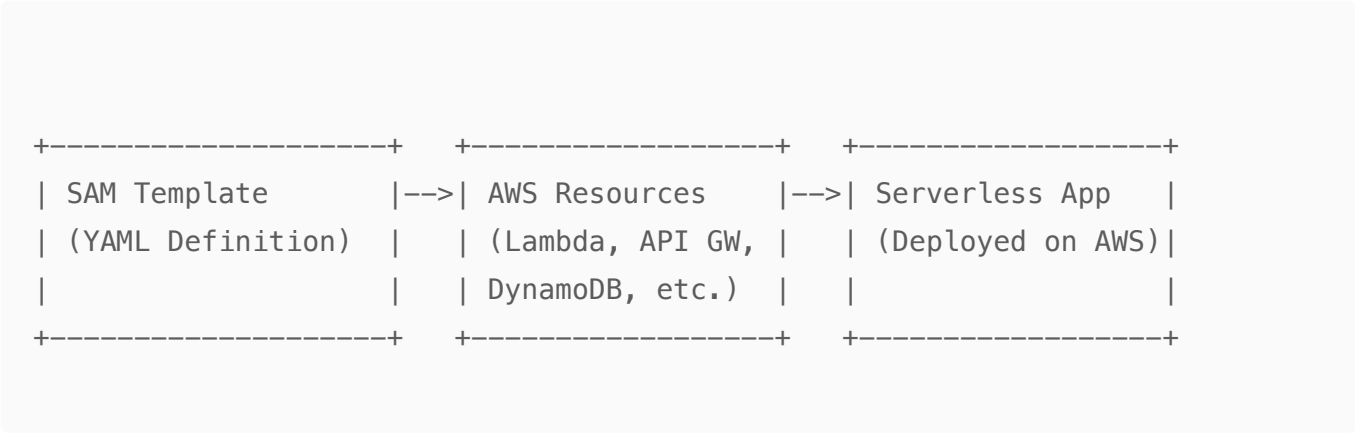
Examples:

1. **Project Initialization**: Initialize a new SAM project with a specific runtime and template.
2. **Local API Testing**: Test the "Hello World" API locally using a web browser.
3. **AWS Deployment**: Deploy the application to AWS using SAM CLI.

Conclusion

This session covered the basics of AWS SAM, including key concepts like Lambda, API Gateway, DynamoDB, and SAM templates. We also explored the process of installing and configuring SAM CLI, creating and deploying serverless applications, and testing them locally.

Diagram: AWS SAM Workflow



Feel free to explore further and start building your own serverless applications using AWS SAM!