AWS Assignment 1

Objective:

This assignment will help everyone understand and implement key AWS services, including IAM, EC2, VPC, Subnets, and Nginx. The goal is to set up a basic web server in a secure VPC environment.

Assignment Tasks:

1. IAM (Identity and Access Management)

- Create an IAM user with programmatic access and assign it to a custom IAM group with **EC2 and VPC Full Access**.
- Create a policy that allows the user to start, stop, and terminate EC2 instances but restricts access to other AWS services.

2. VPC (Virtual Private Cloud) and Subnets

- Create a **custom VPC** with CIDR block **192.168.0.0/16**.
- Inside the VPC, create two **subnets**:
    - **Public Subnet**: 192.168.1.0/24
    - **Private Subnet**: 192.168.2.0/24
- Set up an **Internet Gateway** and attach it to the VPC.
- Configure a **route table** to allow internet access only for the public subnet.

3. EC2 (Elastic Compute Cloud) Instance

- Launch an **EC2 instance** in the **public subnet** using Amazon Linux 2 or Ubuntu.
- Attach a **security group** that allows inbound SSH (port 22) and HTTP (port 80) access.

4. Install & Configure Nginx Web Server

- Connect to the EC2 instance via SSH.

Submission Requirements:

1. **Screenshots** of:
    - IAM User & Policy
    - VPC, Subnets, and Route Table Configuration
    - Running EC2 Instance with Public IP

    - Webpage running on Nginx
2. **Commands Used** (in a text file or PDF).
3. **Explanation** (brief write-up on what was learned).

Evaluation Criteria:

✔ Correct implementation of IAM, VPC, Subnet, EC2
✔ Successful Nginx installation and webpage hosting
✔ Clear documentation and screenshots

AWS Assignment 2

Below is an example assignment that combines AWS Amplify, API Gateway, Lambda, SNS, and DynamoDB to build a simple serverless web application. This assignment is designed to help all gain hands-on experience with AWS serverless technologies.

Objective

Build a serverless web application that allows users to submit data through a frontend application. The submitted data is processed by an API (via API Gateway and Lambda), stored in DynamoDB, and a notification is sent using SNS upon each successful data entry.

Assignment Tasks

1. AWS Amplify – Frontend Setup & Deployment

- **Create a New Amplify App:**
    - Initialize an Amplify project (using the Amplify CLI or Amplify Console).
    - Connect your Amplify project to a Git repository containing a simple static web app (e.g., built with React, Angular, or basic html css js).

- **Frontend Application:**
    - Develop a basic user interface with a form that collects sample user data (e.g., name, email, message).
    - Add functionality to call a REST API endpoint (to be created in Task 2) when the form is submitted.
- **Deployment:**
    - Deploy the frontend using Amplify Hosting.
    - Verify that the app is accessible via the Amplify-provided URL.

2. API Gateway – REST API Setup

- **Create a REST API:**
    - In the API Gateway console, create a new REST API.
    - Define a resource (e.g., `/submit`) with a POST method.
- **Integration with Lambda:**
    - Configure the POST method to trigger a Lambda function (created in Task 3).
    - Enable CORS on the API so that the Amplify-hosted frontend can call it.

3. AWS Lambda – Function Development

- **Create a Lambda Function:**
    - Develop a Lambda function in your preferred runtime (Node.js, Python, etc.).
    - The function should perform the following:
        - Parse the incoming JSON payload from API Gateway.
        - Insert the received data into a DynamoDB table (see Task 4).
        - Publish a notification to an SNS topic (see Task 5) confirming data receipt.
        - Return a suitable response (e.g., a success message and the stored data).
- **Permissions:**
    - Ensure the Lambda execution role has permissions to interact with DynamoDB and SNS.

4. DynamoDB – Data Storage

- **Create a DynamoDB Table:**
    - In the DynamoDB console, create a table (e.g., `UserSubmissions`) with an appropriate primary key (for example, `submissionId` as a UUID or timestamp).
- **Integrate with Lambda:**
    - Within your Lambda function, use the AWS SDK to insert new records into this table.
- **Data Validation:**
    - Test the table by manually inserting a sample record or using the Lambda function.

5. SNS – Notification Setup

- **Create an SNS Topic:**
  - In the SNS console, create a new topic (e.g., `SubmissionNotifications`).
  - Optionally, subscribe an email endpoint to the topic for real-time notifications.
- **Integrate with Lambda:**
  - Update your Lambda function to publish a message to the SNS topic after successfully inserting a record into DynamoDB.
  - The notification message should contain details about the new submission.

6. Testing & Validation

- **End-to-End Testing:**
  - Use the Amplify-hosted frontend to submit data.
  - Verify that:
    - The API Gateway correctly triggers the Lambda function.
    - The Lambda function stores the data in DynamoDB.
    - An SNS notification is published (and received if using email subscriptions).
    - A proper response is returned to the frontend, and the user sees confirmation.
- **Debugging:**
  - Check CloudWatch logs for Lambda and API Gateway if issues arise during testing.

Submission Requirements

1. **Documentation:**
   - A report describing your architecture and each component's role.
   - A diagram illustrating the flow from Amplify to API Gateway, Lambda, SNS, and DynamoDB.
2. **Screenshots:**
   - Amplify Console showing the deployed frontend.
   - API Gateway configuration.
   - Lambda function code and CloudWatch logs.
   - DynamoDB table with sample data.
   - SNS topic configuration and any notification received (if applicable).
3. **Source Code:**
   - Provide a link to the Git repository containing your frontend application and Lambda function code.
4. **Commands & Configurations:**
   - A text file or PDF listing important CLI commands (if used) and configuration settings.

Evaluation Criteria

- Correct integration of AWS Amplify, API Gateway, Lambda, SNS, and DynamoDB.
- Successful end-to-end functionality of the application.
- Clarity and completeness of documentation and code.
- Implementation of best practices for security (e.g., proper IAM roles and policies) and error handling.