COMSATS University Islamabad, Abbottabad Campus

Department of Computer Science

SDA

Lab Assignment

Rana Muhammad Tayyab ATIQ

FA22-BSE-098

Solution

Serverless computing offers many advantages, but it also presents several challenges. One of the main issues is **cold start latency**, which occurs when a function is invoked after being idle for some time. To mitigate this, you can use strategies like keeping functions warm through scheduled events or leveraging **Provisioned Concurrency** in AWS Lambda to maintain pre-warmed instances, ensuring faster execution. Another challenge is managing **stateful applications**, as serverless functions are inherently stateless. To handle this, external storage solutions like **DynamoDB** or **Redis** can be used to persist state across multiple function invocations. Finally, monitoring and debugging serverless applications can be more complex due to the ephemeral nature of the environment. To address this, cloud services like **CloudWatch Logs** and **CloudWatch Metrics** can be utilized for logging and performance tracking, while **AWS X-Ray** helps trace requests and identify bottlenecks. Third-party monitoring tools such as Datadog can also be employed to enhance visibility and troubleshooting. These solutions help overcome the key challenges of serverless computing, improving performance, scalability, and maintainability

Lambda Function with DynamoDB (Node.js):

```
const AWS = require('aws-sdk');

const dynamoDB = new AWS.DynamoDB.DocumentClient();


exports.handler = async (event) => {

  const sessionId = event.sessionId;

  const newState = event.newState;


  // Save state to DynamoDB

  await dynamoDB.put({

    TableName: 'UserSessions',

    Item: {

      sessionId: sessionId,

      state: newState

    }

  }).promise();
```

```
    return {
        statusCode: 200,
        body: JSON.stringify('State saved successfully'),
    };
};
```

Lambda Function with CloudWatch Logs and X-Ray (Node.js):

```
const AWS = require('aws-sdk');
const dynamoDB = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event) => {
    console.log("Received event:", JSON.stringify(event, null, 2)); // Log incoming event

    try {
        const sessionId = event.sessionId;
        const newState = event.newState;

        // Save state to DynamoDB
        await dynamoDB.put({
            TableName: 'UserSessions',
            Item: {
                sessionId: sessionId,
                state: newState
            }
        }).promise();

        console.log("State saved successfully");
        return {
            statusCode: 200,
            body: JSON.stringify('State saved successfully'),
        };
    } catch (error) {
        console.error("Error saving state:", error); // Log error
        throw error;
    }
};
```