AWS SAM

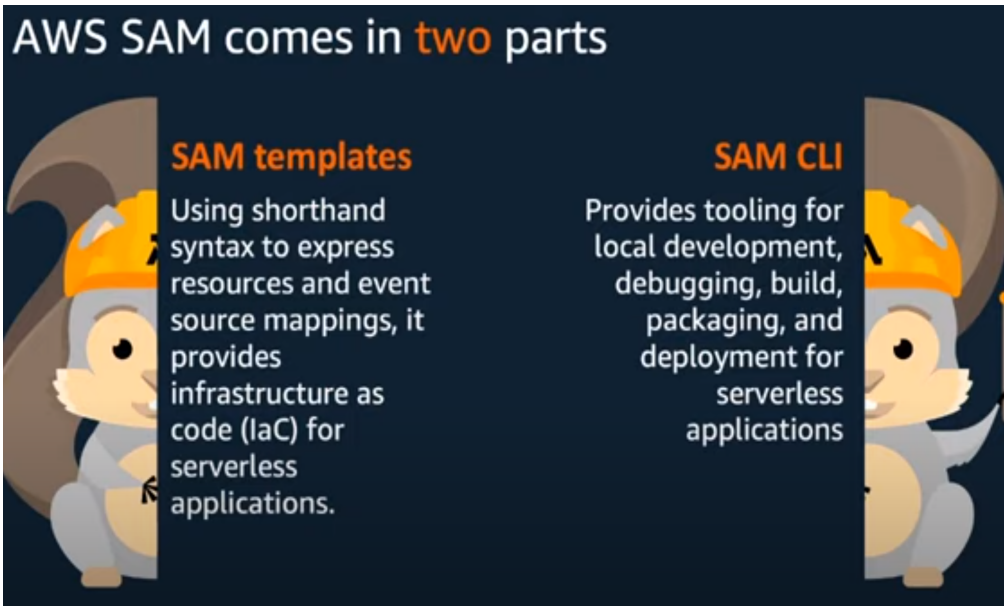
Based on Mastering the AWS Serverless Application Model (AWS SAM) - AWS Online Tech Talks

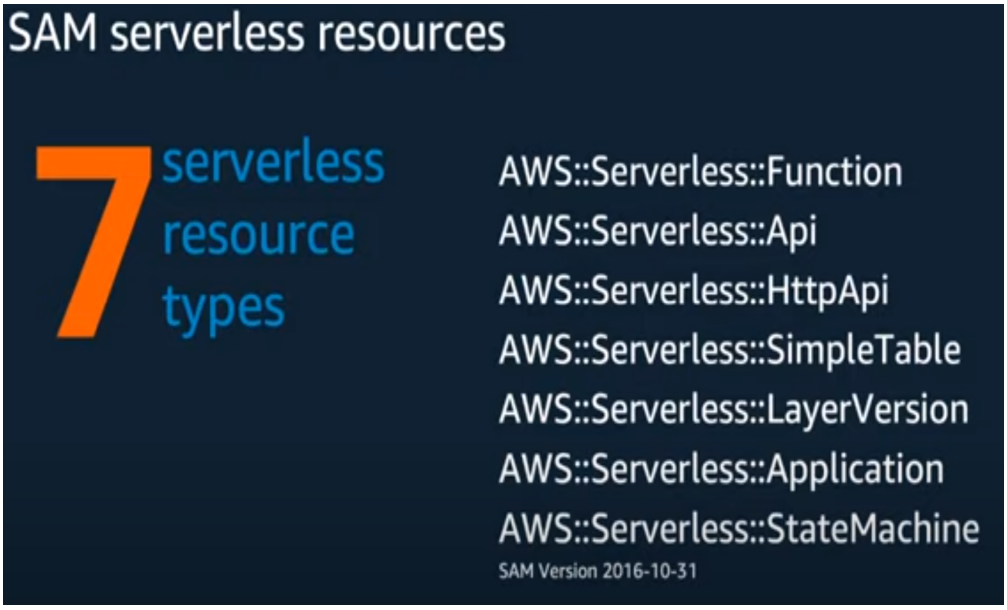
Note:

- If SAM command is not working:

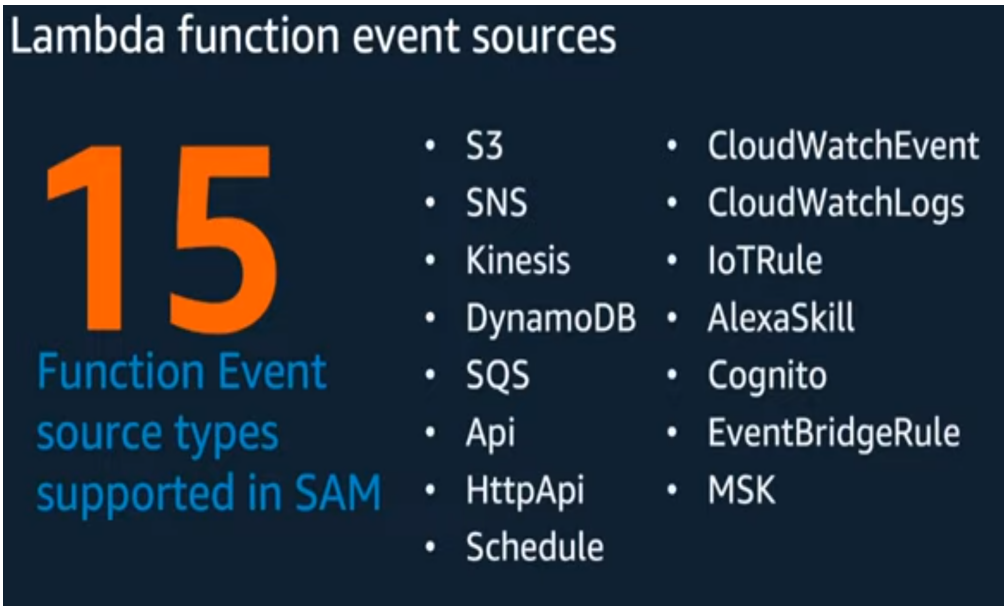
```
alias sam="sam.cmd"
sam --version
```

Part 1 - AWS SAM Templates



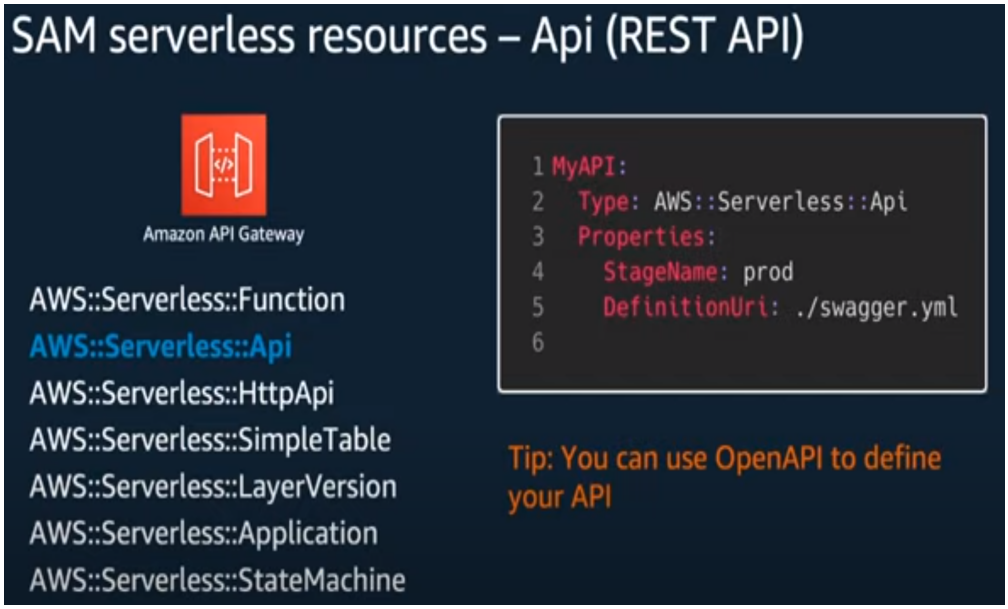


Lambda Functions



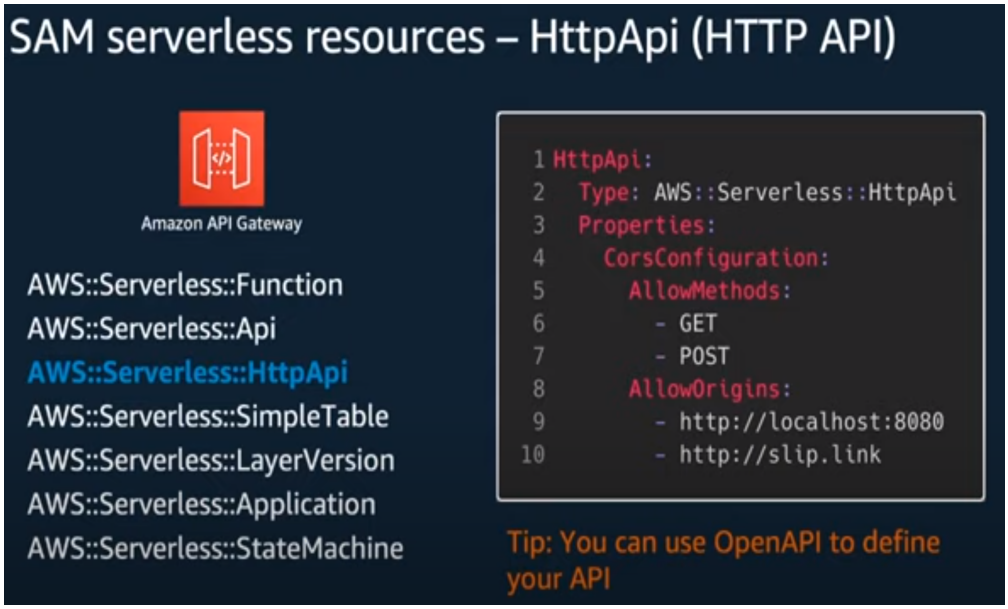
API GW - Type=API

- You can use OpenAPI (swaggers) to define the API



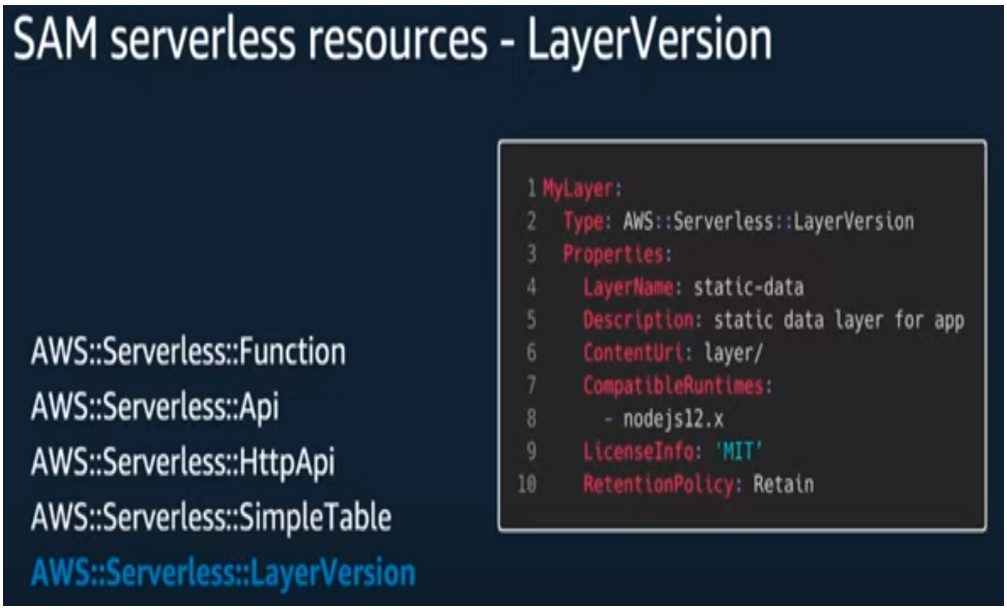
API GW - HTTP API (or API GW v2)

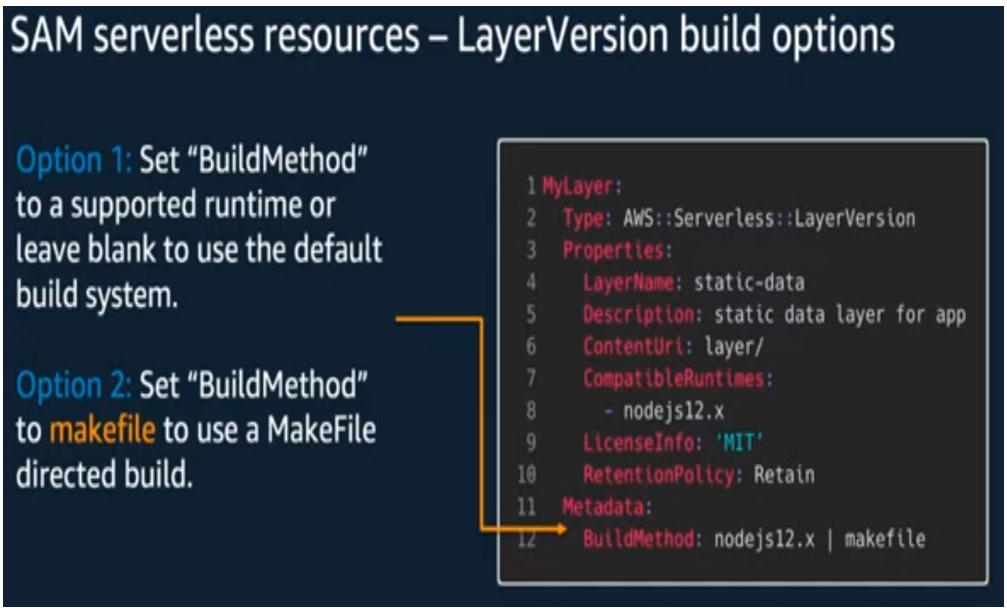
- **It is meant to be faster, lower cost and easiesr to use**



LayerVersion

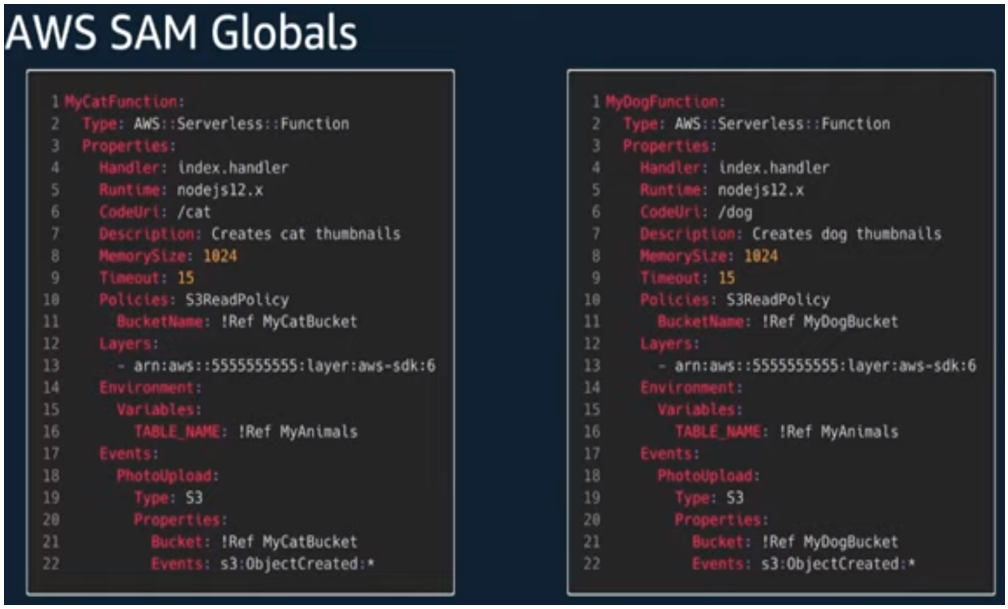
- Layers allow you to **share code accross multiple Lambda functions**



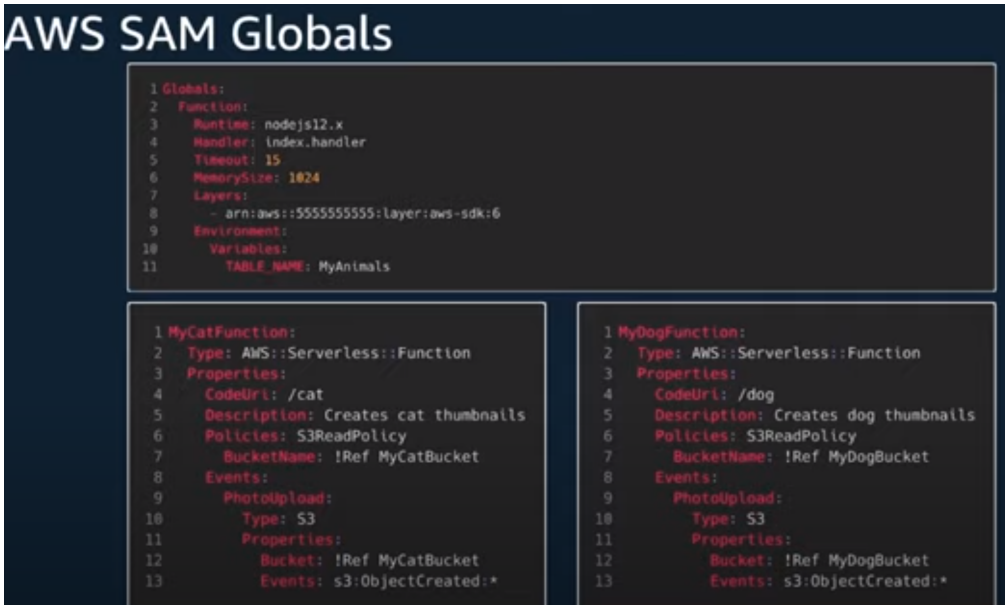


AWS SAM Globals

- The problem: repetition of code



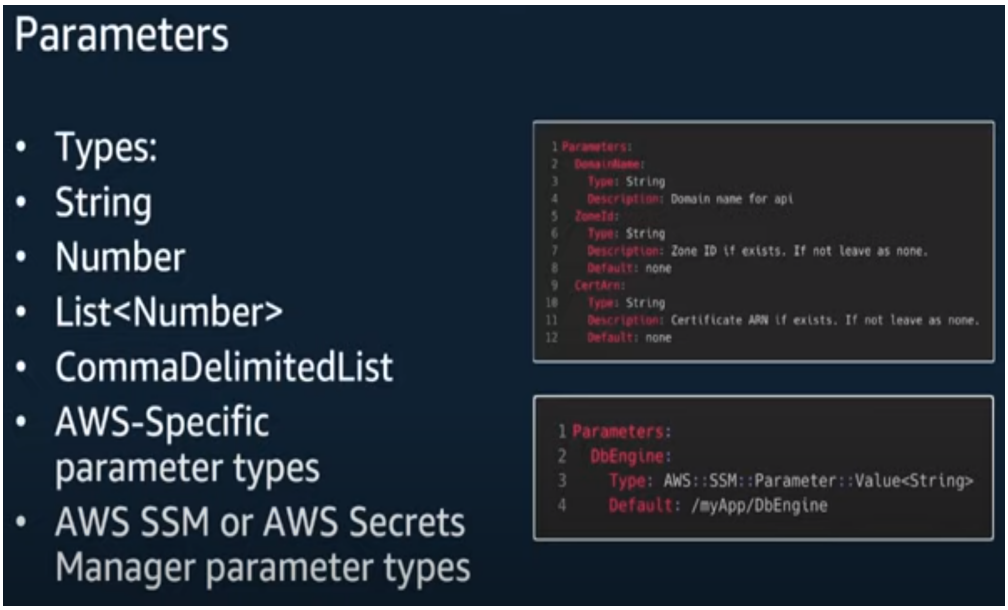
- The solution: defining global variables

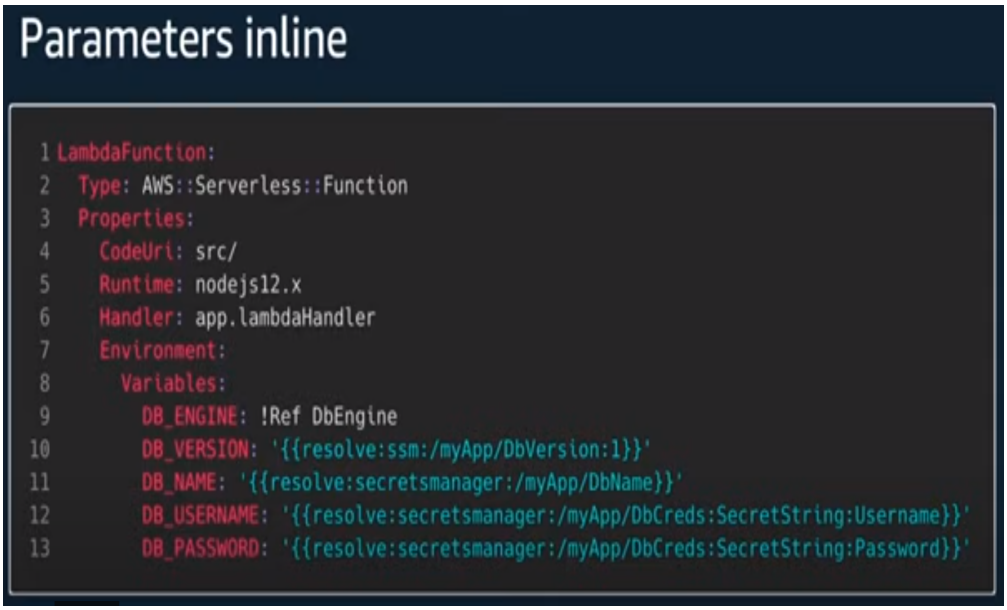


Buidling Reusable Templates

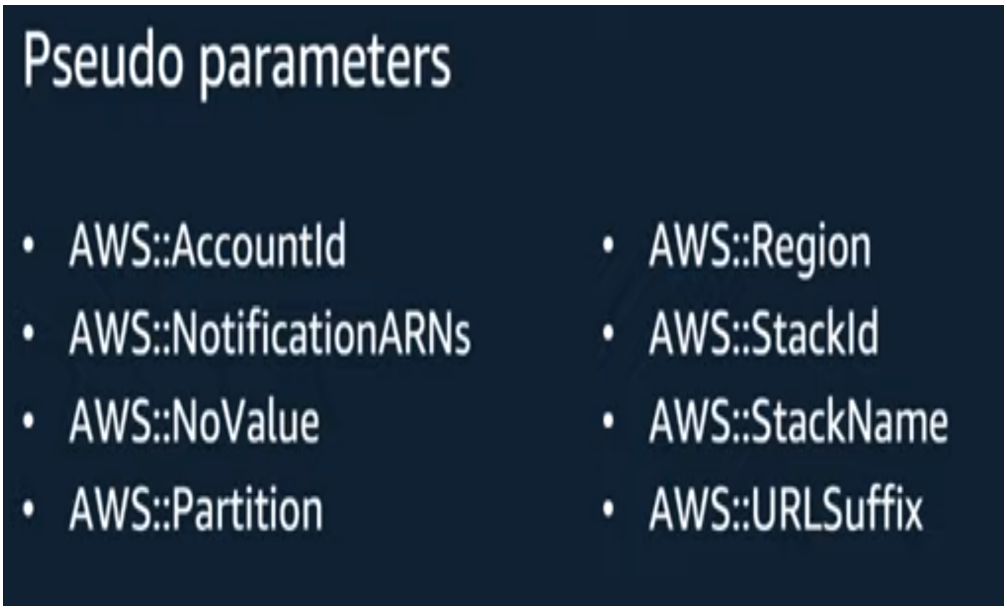
- As a best practice, each environment should have its own account
- Using the same template to build the same infrastructure ensures consistency accross all the accounts
- There are several ways to achieve it:

1. Using **Parameters**
   - The parameters can be stored on the Template itself or on **SSM** or on **Secrets Manager** services

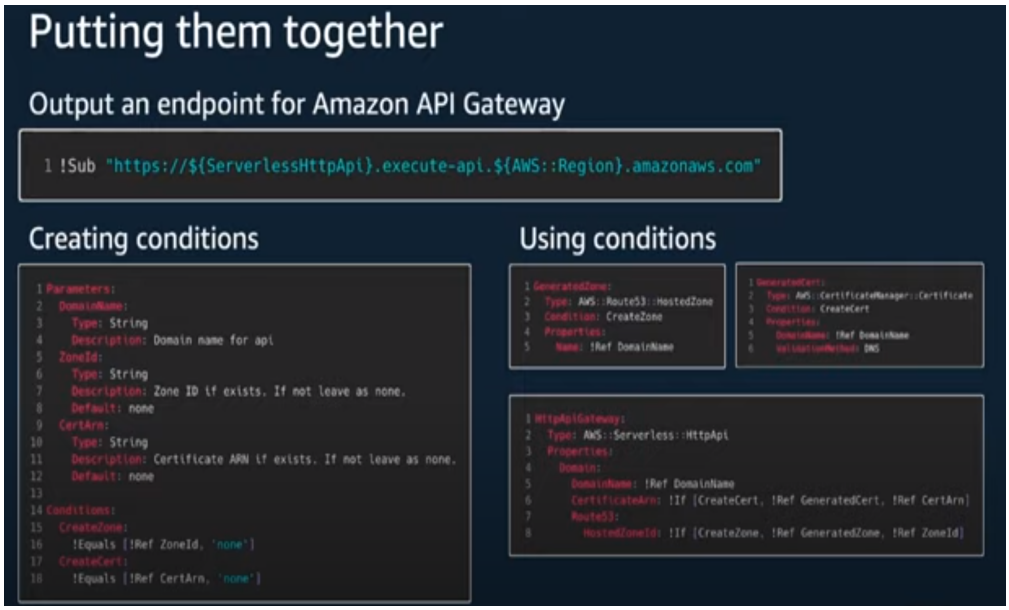




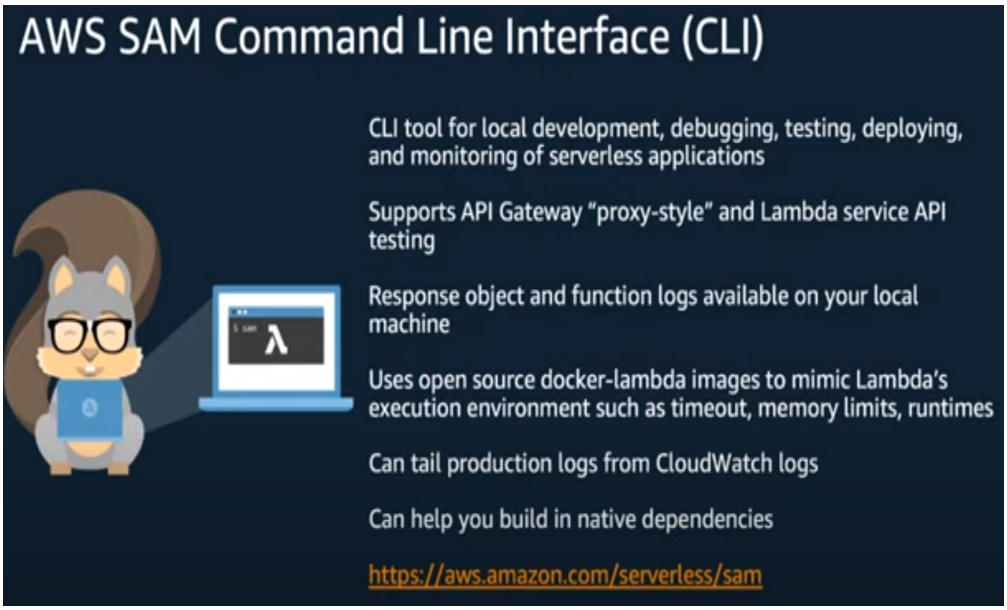
Pseudo Parameters



We can use even Intrinsic Functions (similar to the ones used by CloudFormation)

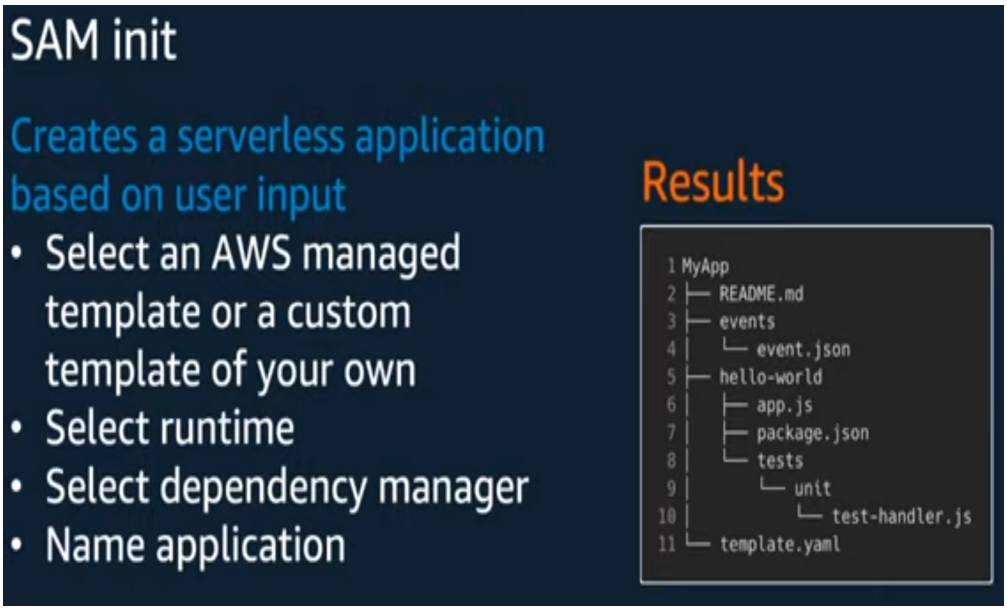
Example: 

Part 2 - AWS SAM CLI

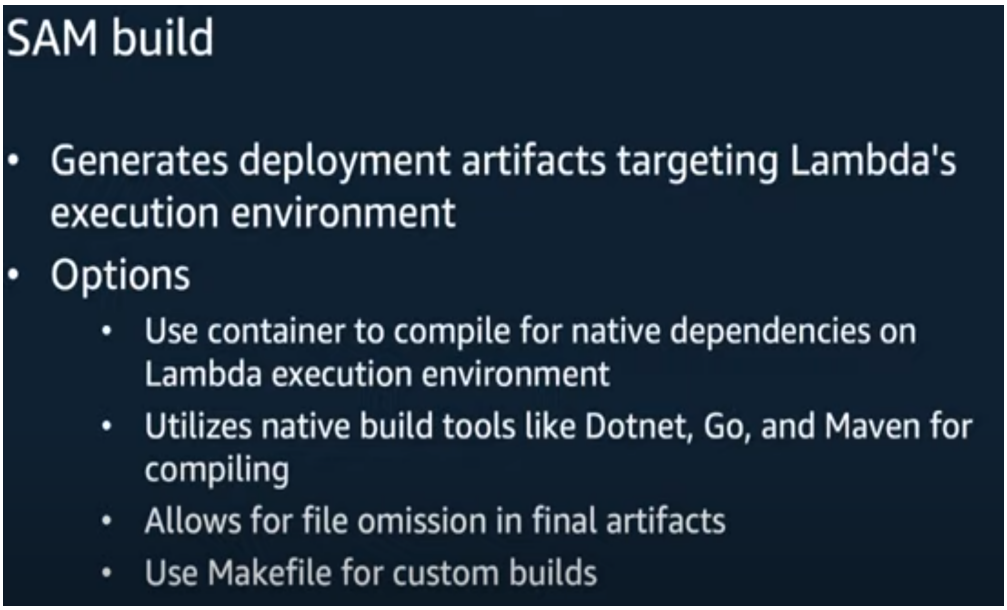


1. SAM init

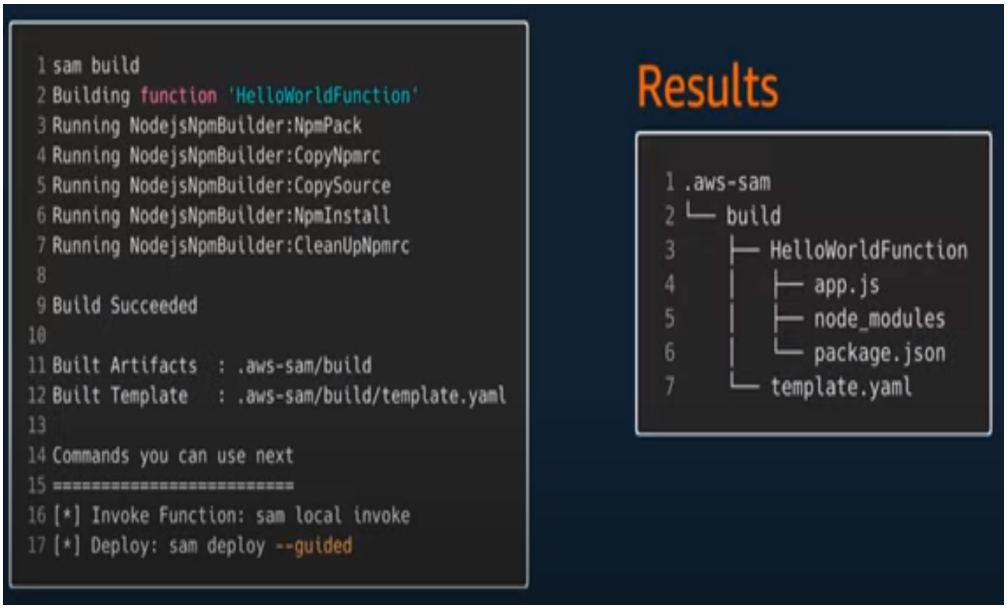
As a result, a new project is created:



2. SAM build



The result will be:



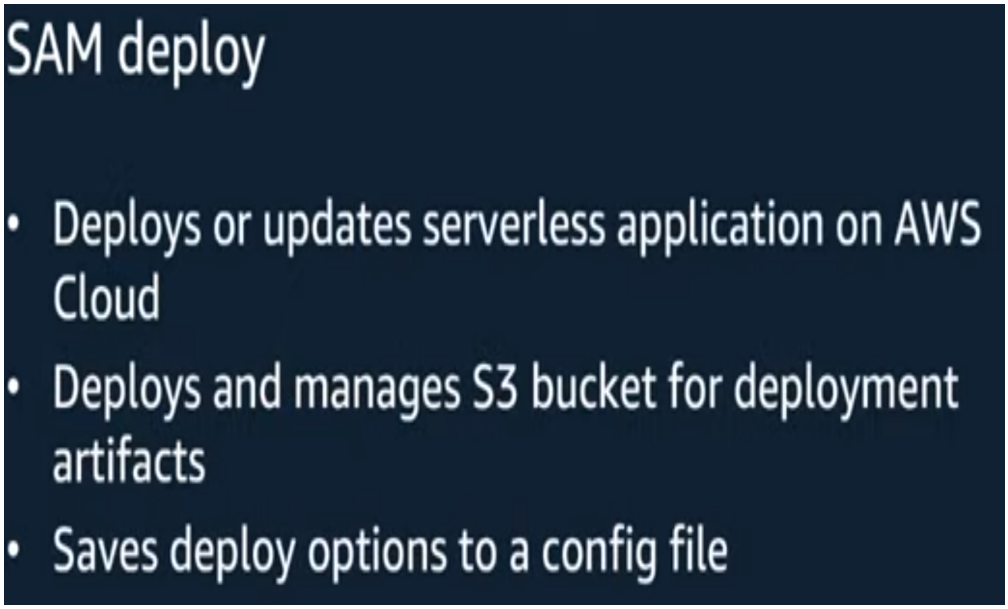
- Once the build is done and **artifacts** are created there are some commands you can use next:

- Validate SAM template: **sam validate**
- Invoke Function: **sam local invoke**
- Test Function in the Cloud: **sam sync --stack-name {{stack-name}} --watch**
- Deploy: **sam deploy --guided**

```
sam local invoke InitStateFunction
```

3. SAM deploy

- It will upload the artifacts into an S3 bucket and then trigger a cloudFormation build



Additional commands:

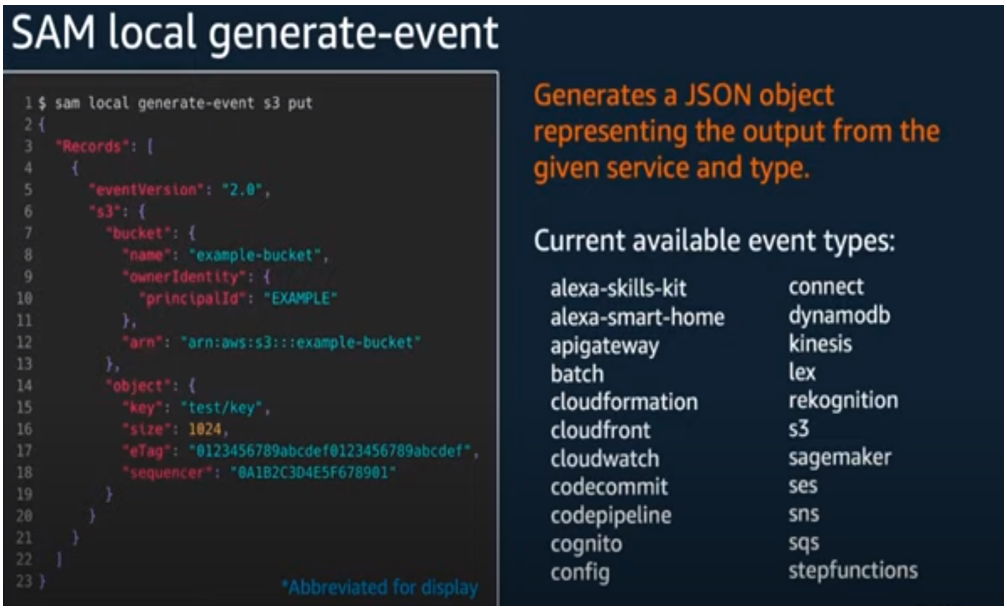
SAM package

- this command can be used for creating a zip file that is uploaded to an S3 bucket.
- It can be useful if there is a CI/CD pipeline that is involved in the build and deployment proccesses.

```
sam package --s3-bucket ej-apps --output-template-file out.yaml
```

SAM local generate-event

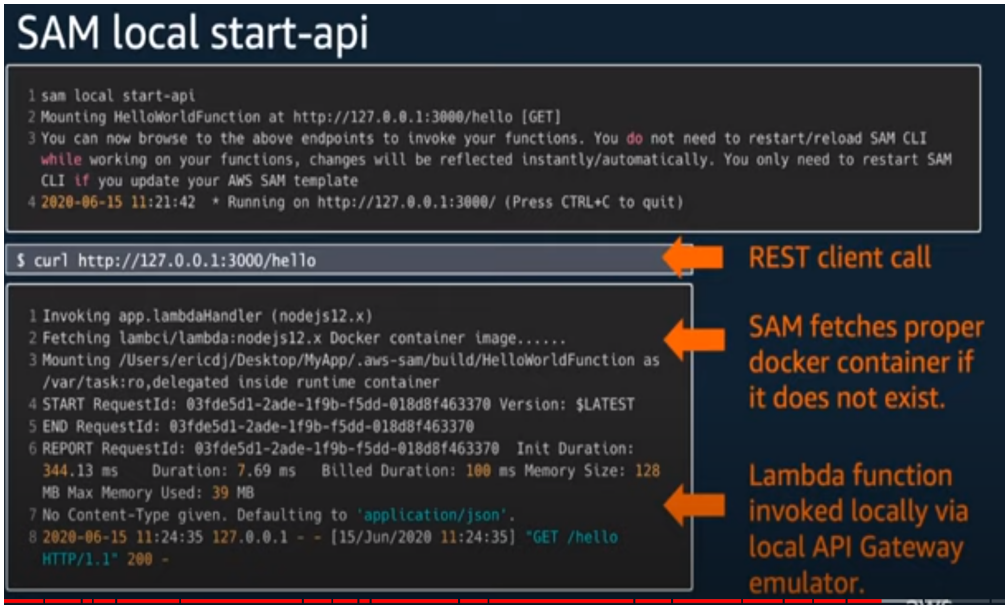
- This command generates a JSON object representing the output from a given service and type



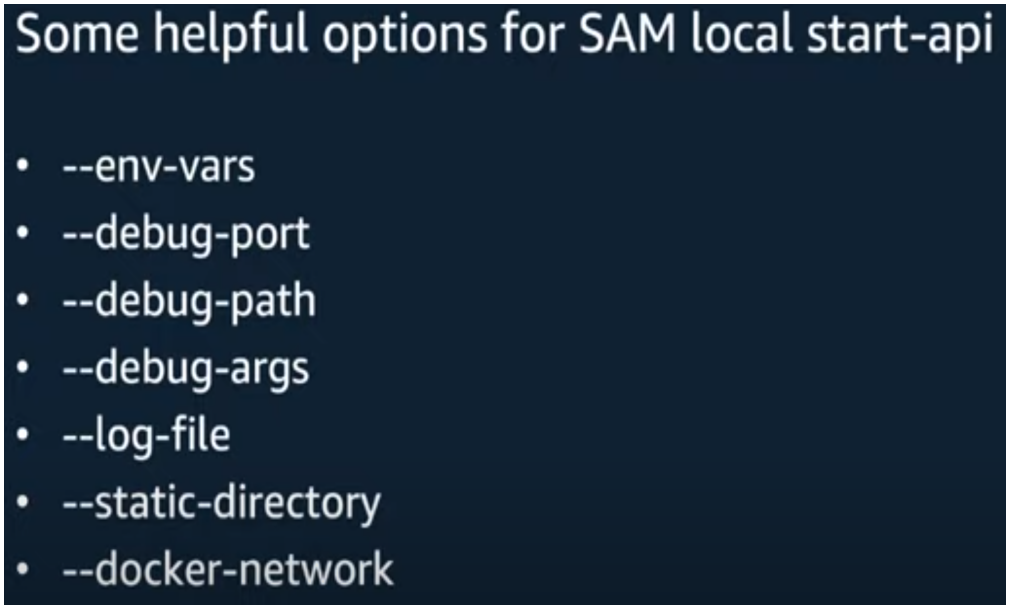

```
sam local generate-event apigateway http-api-proxy
sam local generate-event s3 put
```

SAM local start-api

- Emulate a REST API call to a Lambda function



Useful options

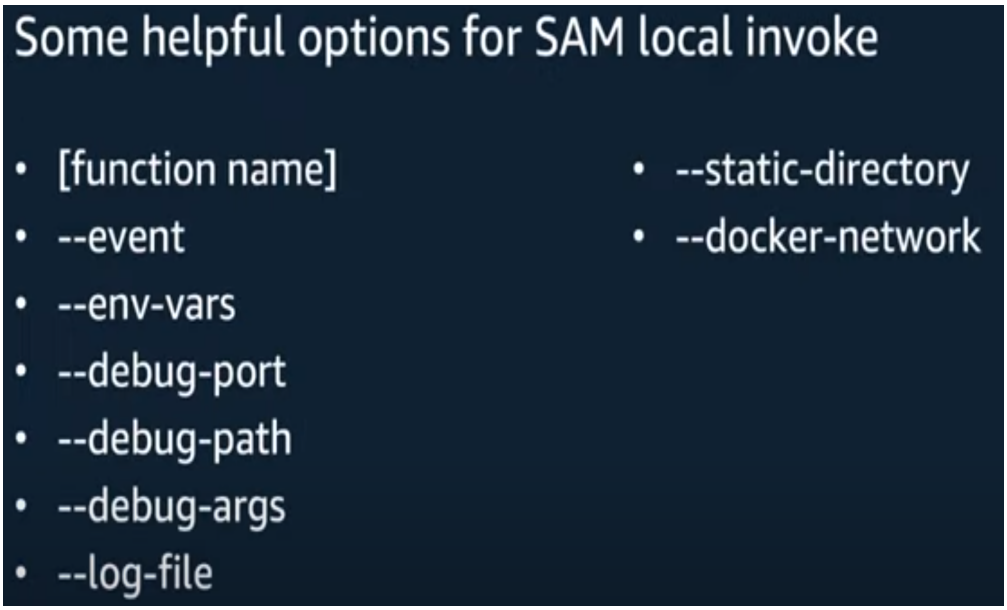


SAM local start-lambda

- this is less helpful, unless you are building a service that calls the Lambda function directly.
- you can also use the AWS CLI (aws lambda invoke --function-name "" --endpoint-url "" --no-verify-ssl out.txt)

SAM local invoke lambda

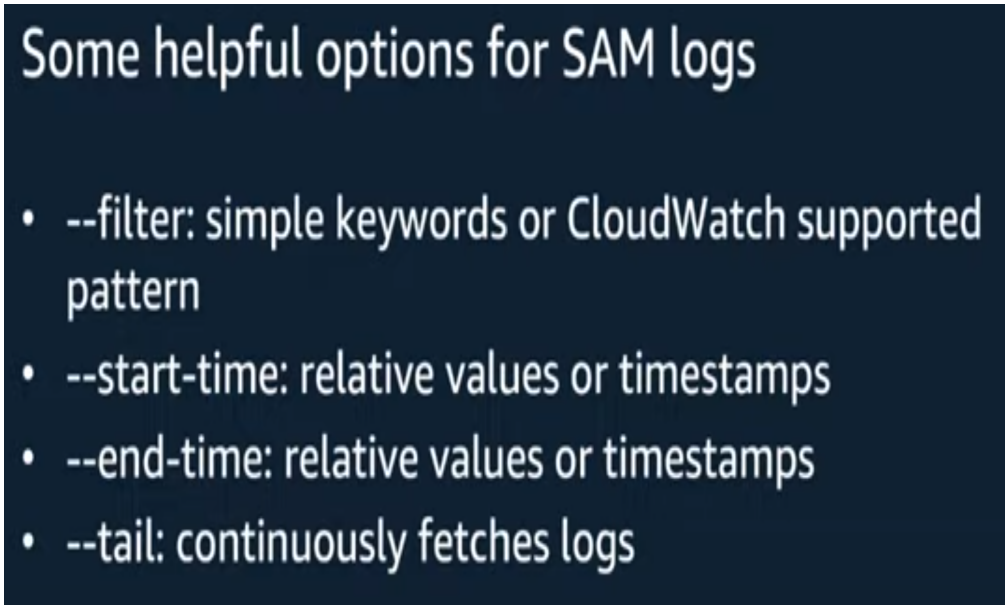
```
sam local invoke HelloWorld -e events/event.json
```



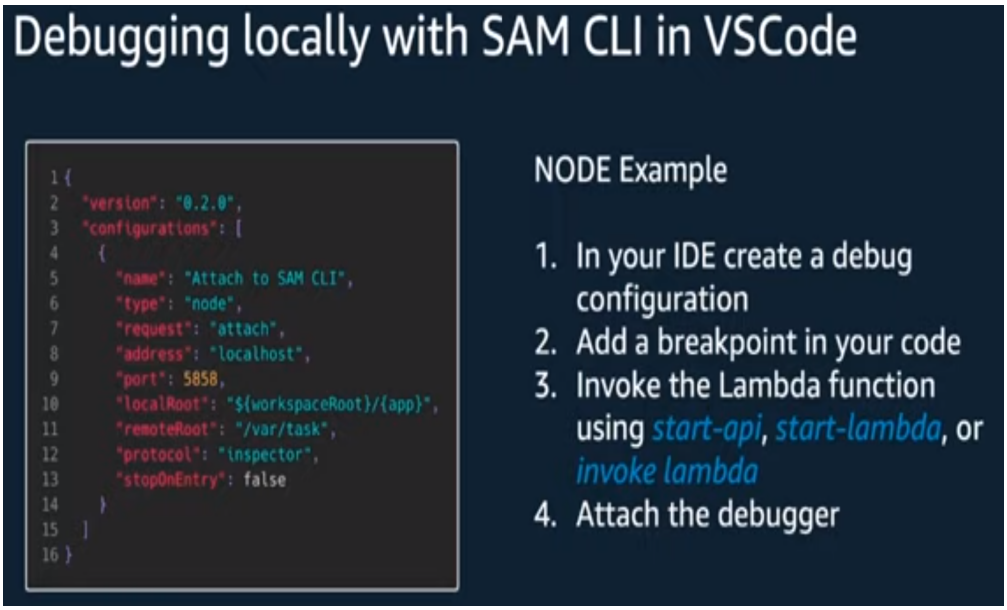
- SAM local is "build" aware so pay attention which template.yaml is being used. The manifest under ./aws-sam/build/template.yaml always takes precedence

SAM logs

- This command can replace the need to continuosly watch CloudWatch logs



Debugging locally on VS Code



Additional Materials

Lambda function for CRUD DynamoDB actions + API GW

(Lambda function from CRUD DynamoDB)
[https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html]

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ScanCommand,
  PutCommand,
  GetCommand,
  DeleteCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

const dynamo = DynamoDBDocumentClient.from(client);

const tableName = "http-crud-tutorial-items";

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

  try {
    switch (event.routeKey) {
      case "DELETE /items/{id}":
        await dynamo.send(
          new DeleteCommand({
            TableName: tableName,
```

```
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = `Deleted item ${event.pathParameters.id}`;
        break;
      case "GET /items/{id}":
        body = await dynamo.send(
          new GetCommand({
            TableName: tableName,
            Key: {
              id: event.pathParameters.id,
            },
          })
        );
        body = body.Item;
        break;
      case "GET /items":
        body = await dynamo.send(
          new ScanCommand({ TableName: tableName })
        );
        body = body.Items;
        break;
      case "PUT /items":
        let requestJSON = JSON.parse(event.body);
        await dynamo.send(
          new PutCommand({
            TableName: tableName,
            Item: {
              id: requestJSON.id,
              price: requestJSON.price,
              name: requestJSON.name,
            },
          })
        );
        body = `Put item ${requestJSON.id}`;
        break;
      default:
        throw new Error(`Unsupported route: "${event.routeKey}"`);
    }
  } catch (err) {
    statusCode = 400;
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

- This lambda function should have 4 routes defined under API GW

```
GET /items/{id}
GET /items
PUT /items
DELETE /items/{id}
```

- Testing the Integration

```
curl -X "PUT" -H "Content-Type: application/json" -d "{\"id\": \"123\", \"price\": 12345, \"name\": \"myitem\"}" https://abcdef123.execute-api.us-west-2.amazonaws.com/items

curl https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123

curl -X "DELETE" https://abcdef123.execute-api.us-west-2.amazonaws.com/items/123
```

All the above can be automated with the SAM template under