AWS SAM vs Serverless

AWS SAM - Introduction

- The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications.
- It provides shorthand syntax to express functions, APIs, databases, and event source mappings.
- With just a few lines per resource, you can define the application you want and model it using YAML.
- During deployment, SAM transforms and expands the SAM syntax into AWS CloudFormation syntax, enabling you to build serverless applications faster.

AWS SAM - Benefits

- **Single Deployment Configuration:** Use SAM to organize related components, share configuration such as memory and timeouts between resources, and deploy all related resources together as a single, versioned entity.
- **Local Testing and Debugging:** Use SAM CLI to step-through and debug your code. It provides a Lambda-like execution environment locally and helps you catch issues upfront.
- **Built-In Best Practices:** Deploy your infrastructure as config to leverage best practices such as code reviews. Enable gradual deployments through AWS CodeDeploy and tracing using AWS X-Ray with just a few lines of SAM config.

AWS SAM - Benefits

- **Built on AWS CloudFormation:** AWS SAM is an extension of AWS CloudFormation, so you get the reliable deployment capabilities of CloudFormation. You can also define resources using CloudFormation in your SAM template.
- **Integration with Development Tools:** SAM integrates with a suite of AWS serverless tools.

Disadvantages of AWS SAM

- The API Gateway configuration gets extensively based on the Swagger. It is irrelevant to the users since they don't need to get stuck with the Swagger.
- The event source and the feature set might get limited since it was just released.
- It is a serverless web application that is still new and might have some bugs that the current users should get worried about.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  CreateThumbnail:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Timeout: 60
      Policies: AWSLambdaExecute
      Events:
        CreateThumbnailEvent:
          Type: S3
          Properties:
            Bucket: !Ref SrcBucket
            Events: s3:ObjectCreated:*

  SrcBucket:
    Type: AWS::S3::Bucket
```

Serverless Framework

- The Serverless Framework is provider-agnostic, so you can use it to deploy serverless applications to AWS, Microsoft Azure, Google Cloud Platform, or many other providers.
- This reduces lock-in and enables a multi-cloud strategy while giving you a consistent experience across clouds.
- The Serverless Framework assists with additional aspects of the serverless application lifecycle, including building your function package, invoking your functions for testing, and reviewing your application logs.
- The Serverless Framework provides a configuration DSL which is designed for serverless applications.
- It also enables infrastructure as code while removing a lot of the boilerplate required for deploying serverless applications, including permissions, event subscriptions, logging, etc.

Serverless Framework - Benefits

- **No server management:** There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.
- **Flexible scaling:** Your application can be scaled automatically or by adjusting its capacity through toggling the units of consumption (e.g. throughput, memory) rather than units of individual servers.
- **Pay for value:** Pay for consistent throughput or execution duration rather than by server unit.
- **Automated high availability:** Serverless provides built-in availability and fault tolerance.

Serverless Framework - Installation

- **Installing Node.js:** Serverless is a Node.js CLI tool so the first thing you need to do is to install Node.js on your machine.
- **To verify that Node.js is installed:**
  - node --version
- **Installing the Serverless Framework:**
  - npm install -g serverless
- **To see which version of serverless you have installed run:**
  - serverless --version

Serverless Commands

- serverless info - Displays information about the deployed service.
- sls deploy - Deploys your entire service via CloudFormation.
- sls package - Packages your entire infrastructure into the .serverless directory by default and make it ready for deployment.
- sls remove - It will remove the deployed service, defined in your current working directory, from the provider.
- sls invoke - Invoke deployed function with command invoke and --function or shorthand -f

https://www.serverless.com/framework/docs/providers/aws/cli-reference/

```yaml
service: candidate-service

frameworkVersion: ">=1.1.0 <2.0.0"

provider:
  name: aws
  runtime: nodejs4.3
  stage: dev
  region: us-east-1

functions:
  candidateSubmission:
    handler: api/candidate.submit
    memorySize: 128
    description: Submit candidate information and starts interview process.
    events:
      - http:
          path: candidates
          method: post
```

We can specify the function related configuration with the API Gateway configuration in the serverless.yml file.

```
provider:
  name: aws
  runtime: nodejs4.3
  stage: dev
  region: us-east-1
  environment:
    CANDIDATE_TABLE: ${self:service}-${opt:stage, self:provider.stage}
    CANDIDATE_EMAIL_TABLE: "candidate-email-${opt:stage, self:provider.stage}"
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:Query
        - dynamodb:Scan
        - dynamodb:GetItem
        - dynamodb:PutItem
      Resource: "*"
```

We can also specify environment variables, IAM related configuration too in the serverless.yml file. The above image describes that.

```
resources:
  Resources:
    CandidatesDynamoDbTable:
      Type: 'AWS::DynamoDB::Table'
      DeletionPolicy: Retain
      Properties:
        AttributeDefinitions:
          -
            AttributeName: "id"
            AttributeType: "S"
        KeySchema:
          -
            AttributeName: "id"
            KeyType: "HASH"
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
        StreamSpecification:
          StreamViewType: "NEW_AND_OLD_IMAGES"
        TableName: ${self:provider.environment.CANDIDATE_TABLE}
```

In the above configuration in serverless.yml file, we can also see that DynamoDB configuration is mentioned in regards to the keys, attributes, read & write capacity and stream related configuration.