AWS SAM
Serverless Application Model

A common and intriguing question – what is SAM and why we need this in the first place

To the First point – what is it?

AWS Serverless Application Model (AWS SAM) is an open-source framework designed to simplify the ***building, testing, and deployment of serverless applications on AWS***. While it leverages AWS CloudFormation for infrastructure as code, AWS SAM offers additional features and benefits tailored specifically for serverless applications.

What are those additional features that we will get, which are not getting in the CloudFormation?

We have majorly 04 benefits

Simplified Syntax:

AWS SAM provides a simplified syntax for defining serverless resources like AWS Lambda functions, API Gateway endpoints, DynamoDB tables, and more. This reduces the verbosity of CloudFormation templates.

Serverless-Specific Transformations:

AWS SAM templates are transformed into standard CloudFormation templates before deployment. This allows you to use all CloudFormation features while benefiting from the serverless-specific syntax and capabilities of SAM.

Local Development and Testing:

SAM CLI enables you to run and test your serverless applications locally. This includes invoking Lambda functions, simulating API Gateway endpoints, and connecting to local resources.

Support for Deployment and CI/CD:

AWS SAM integrates with CI/CD pipelines, allowing you to automate the deployment of serverless applications. It supports rolling back deployments and managing application versions and aliases.

AWS SAM
Serverless Application Model

While AWS CloudFormation is a powerful tool for defining and deploying AWS infrastructure, AWS SAM enhances and simplifies the process for serverless applications, providing additional features, a more user-friendly syntax, and tools for local development and testing.

Now – If someone asks you the same above question, but not in comparison with the CFN, why SAM .in specific – same above answer, but conveying in the different manner.

Why SAM

- **Single-deployment configuration**. SAM makes it easy to organize related components and resources, and operate on a single stack. You can use SAM to share configuration (such as memory and timeouts) between resources, and deploy all related resources together as a single, versioned entity.
- **Local debugging and testing**. Use SAM CLI to locally build, test, and debug SAM applications on a Lambda-like execution environment. It tightens the development loop by helping you find & troubleshoot issues locally that you might otherwise identify only after deploying to the cloud.
- **Deep integration with development tools**. You can use SAM with a suite of tools you love and use.
    - IDEs: PyCharm, IntelliJ, Visual Studio Code, Visual Studio, AWS Cloud9
    - Build: CodeBuild
    - Deploy: CodeDeploy, Jenkins
    - Continuous Delivery Pipelines: CodePipeline
    - Discover Serverless Apps & Patterns: AWS Serverless Application Repository
- **Built-in best practices**. You can use SAM to define and deploy your infrastructure as configuration. This makes it possible for you to use and enforce best practices through code reviews. Also, with a few lines of configuration, you can enable safe deployments through CodeDeploy, and can enable tracing using AWS X-Ray.
- **Extension of AWS CloudFormation**. Because SAM is an extension of AWS CloudFormation, you get the reliable deployment capabilities of AWS CloudFormation. You can define resources by using CloudFormation in your SAM template.

- Also, you can use the full suite of resources, intrinsic functions, and other template features that are available in CloudFormation.

In Simple terms → This is what Sam does

MEET SAM.
USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.
DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

We will see what this one depicts – while we go through our course.

AWS SAM
Serverless Application Model

👍 AWS SAM Working Model

Step 01: Write the SAM Template:

You start by writing a SAM template file (**template.yaml).** This file defines your serverless application using SAM's simplified syntax. You specify resources like Lambda functions, API Gateway endpoints, DynamoDB tables, etc.

About this template – we will study in depth, in the next coming few pages …‼

Step02: Develop Your Code:

Write the code for your serverless functions (e.g., Lambda functions). This code can be in various supported languages like Python, Node.js, Java, etc. Organize your codebase with directories and files as needed.

Step 03: Local Development and Testing:

Use the SAM CLI to develop and test your application locally. The CLI provides commands to invoke Lambda functions locally, start a local API Gateway for testing HTTP endpoints, and more.

```sh
sam local invoke FunctionName
sam local start-api
```

Step04: Package the Application:

Once you are satisfied with your local development and testing, package your application. This involves creating a deployment package (a zip file) for your Lambda functions and uploading it to an S3 bucket.

sam package **--s3-bucket your-s3-bucket --output-template-file packaged.yaml**

The sam package command packages your code and dependencies, uploads them to S3, and generates a new SAM template (**packaged.yaml**) with references to the S3 objects.

Step05: Deploy the Application:

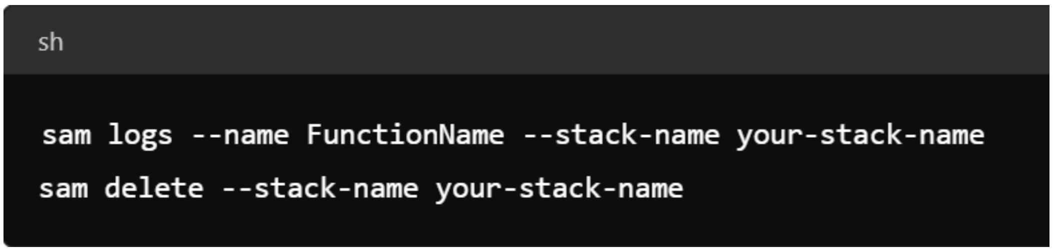
Deploy your packaged application using the SAM CLI. This step creates or updates the CloudFormation stack with the resources defined in your SAM template.

sam deploy --template-file packaged.yaml --stack-name your-stack-name --capabilities CAPABILITY_IAM

During deployment, the SAM template is transformed into a CloudFormation stack. CloudFormation provisions the resources (e.g., Lambda functions, API Gateway, DynamoDB tables) based on the template.

Step 06: Monitor and Manage:

After deployment, you can monitor and manage your serverless application using AWS Management Console, CloudWatch, and other AWS services. SAM also provides commands for managing your application stack, such as viewing logs and deleting the stack.



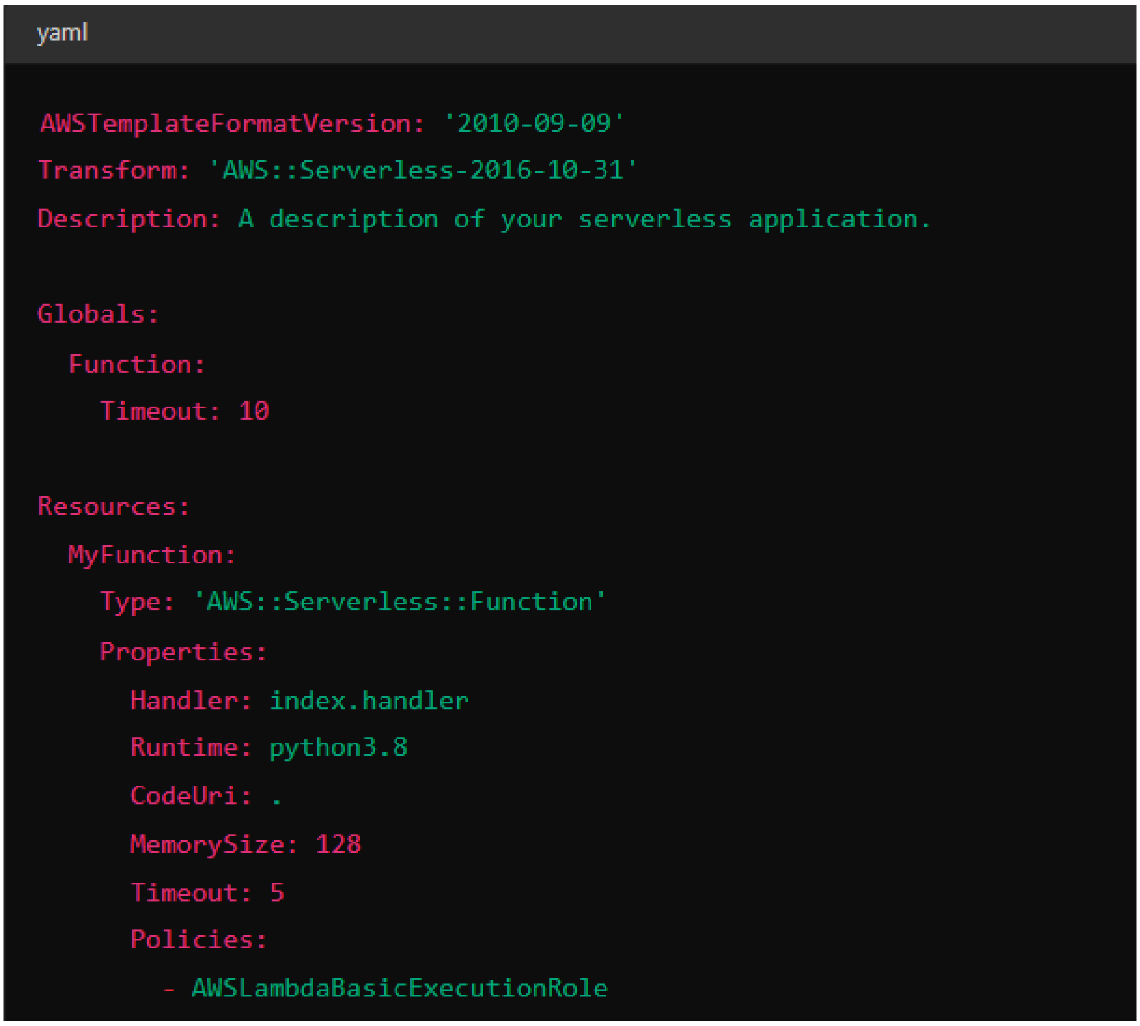
```sh
sam logs --name FunctionName --stack-name your-stack-name
sam delete --stack-name your-stack-name
```

Well done ...!!! Good job ..!!! – We know what is SAM about and it working model, now we will look into the How do we write and leverage SAM in our daily Cloud Engineer life.

AWS SAM
Serverless Application Model

To become the beginner of the Sam Usage – we need to know few things, those are

SAM Template Format (means your template.yaml)



```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: A description of your serverless application.

Globals:
  Function:
    Timeout: 10

Resources:
  MyFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: python3.8
      CodeUri: .
      MemorySize: 128
      Timeout: 5
      Policies:
        - AWSLambdaBasicExecutionRole
```

Now what each filed signifies – we will see

AWS SAM
Serverless Application Model

AWSTemplateFormatVersion:

Specifies the version of the AWS CloudFormation template format to use. The value '2010-09-09' is the latest stable version.

Transform:

Specifies that this template uses the AWS SAM transform, which extends CloudFormation with serverless-specific resources.

Description:

A human-readable description of the application.

Globals:

Defines default properties for supported resource types within the template. In this case, it sets a default timeout for all Lambda functions.

Resources:

Specifies the AWS resources that make up your serverless application.

MyFunction:

- Type: Defines the resource type. Here, it's an AWS:Serverless::Function, which represents a Lambda function.
- Properties: Defines the properties of the resource.
- Handler: Specifies the function within your code that Lambda calls to start execution.
- Runtime: Specifies the runtime environment for the Lambda function.

Keep in mind

Transform: The Transform field is essential for using AWS SAM features. It tells CloudFormation to process this template using the AWS SAM transformation.

Globals: This section allows you to set default properties for multiple resources of the same type, reducing redundancy.

Resources: This is where you define the various AWS resources your application requires, such as Lambda functions, DynamoDB tables, and API Gateway endpoints.

Now we saw the SAM Template – Now we will see, SAM Commands and what they used for

AWS SAM CLI commands are designed to help you build, test, package, deploy, and manage serverless applications.

AWS SAM
Serverless Application Model

Command 01: **sam init**

Initializes a new serverless application with a SAM template.

Use: Sets up a new SAM project with a specified runtime and project name.

```
sam init --runtime python3.8 --name my-sam-app
```

Command 02: **sam validate**

Validates an AWS SAM template to ensure it is syntactically correct.

Use: Checks the SAM template for errors before deployment.

```
sam validate --template-file template.yaml
```

Command 03: **sam build**

Builds your serverless application and prepares it for deployment.

Use: Compiles the code and dependencies of your Lambda functions.

```
sam build
```

Command 04: **sam local invoke**

Invokes a Lambda function locally for testing.

Use: Tests Lambda functions locally with a specified event.

```
sam local invoke MyFunction --event event.json
```

Command 05: **sam local start-api**

Starts a local API Gateway for testing HTTP requests.

Use: Simulates API Gateway endpoints locally.

```
sam local start-api
```

Command 06: **sam package**

Packages your application and uploads artifacts to S3.

Use: Prepares the application for deployment by uploading code to S3 and generating a packaged template.

```
sam package --template-file template.yaml --s3-bucket my-s3-bucket --output-template-file packaged.yaml
```

Command 07: **sam deploy**

Deploys the packaged application using CloudFormation.

Use: Deploys the application to AWS, creating or updating the CloudFormation stack.

```
sam deploy --template-file packaged.yaml --stack-name my-sam-stack --capabilities CAPABILITY_IAM
```

Command 07: **sam logs**

Fetches logs for a Lambda function.

Use: Retrieves logs from CloudWatch Logs for a specified Lambda function.

```
sam logs --name MyFunction
```

Command 09: **sam delete**

Deletes the deployed stack from AWS.

Use: Removes the CloudFormation stack and all associated resources

```
sam delete --stack-name my-sam-stack
```

Command 10: **sam publish**

Publishes an application to the AWS Serverless Application Repository.

Use: Shares your serverless application with others through the Serverless Application Repository.

```
sam publish --template packaged.yaml
```

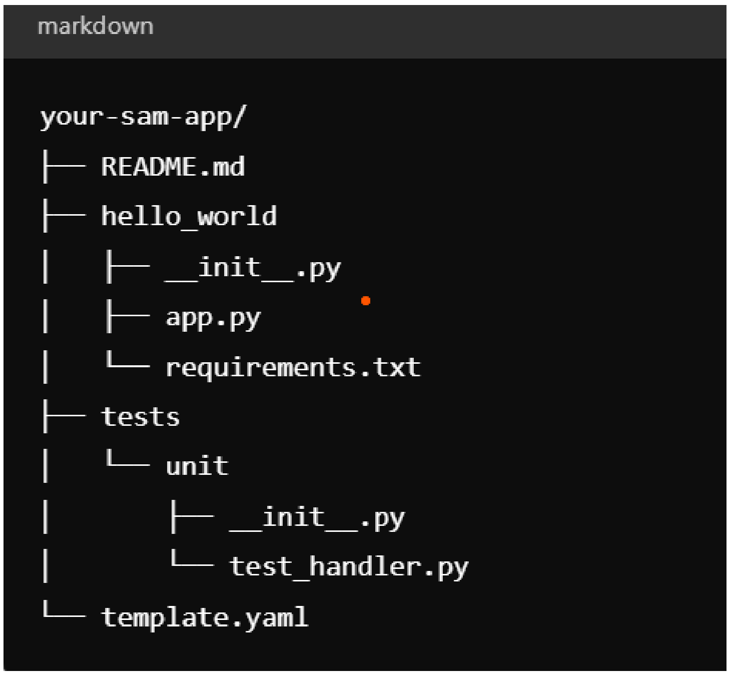
Command 11: **sam list**

Lists resources and operations related to SAM.

Use: Displays stack outputs or other relevant information.

```
sam list stack-outputs --stack-name my-sam-stack
```

Third thing that you should learnt about your app repository structure



markdown

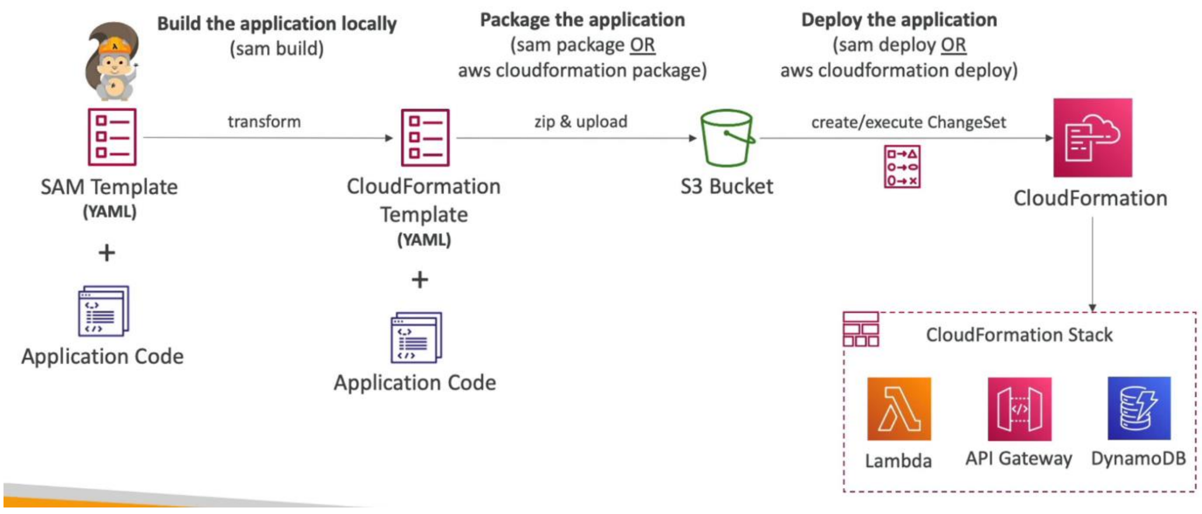
```
your-sam-app/
├── README.md
├── hello_world
│   ├── __init__.py
│   ├── app.py
│   └── requirements.txt
├── tests
│   └── unit
│       ├── __init__.py
│       └── test_handler.py
└── template.yaml
```

This is how, usual & typical app repo looks like using AWS SAM

Practical Examples, we will see once we are building the CloudFormation Templates as well.

AWS SAM
Serverless Application Model

This is how SAM works ..!!



Serverless Application Repository (SAR)

- Repository for serverless applications packaged using SAM
- Packaged applications can be shared
    - Publicly
    - With specific AWS accounts
- The packaged applications can be directly deployed (no duplicate work)
- Application can be customized using **environment variables**

AWS SAM
Serverless Application Model

SAM Questions

https://lisireddy.medium.com/aws-sam-questions-cf252cde0c0d

Wish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)