

# 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

AWS SAM comes in **two** parts



SAM serverless resources

<b>7</b> serverless resource types	AWS::Serverless::Function AWS::Serverless::Api AWS::Serverless::HttpApi AWS::Serverless::SimpleTable AWS::Serverless::LayerVersion AWS::Serverless::Application AWS::Serverless::StateMachine
SAM Version 2016-10-31	

Lambda Functions

## Lambda function event sources

**15**  
Function Event  
source types  
supported in SAM

- S3
- CloudWatchEvent
- SNS
- CloudWatchLogs
- Kinesis
- IoTRule
- DynamoDB
- AlexaSkill
- SQS
- Cognito
- Api
- EventBridgeRule
- HttpApi
- MSK
- Schedule

## SAM serverless resources

**7** serverless  
resource  
types

- AWS::Serverless::Function
- AWS::Serverless::Api
- AWS::Serverless::HttpApi
- AWS::Serverless::SimpleTable
- AWS::Serverless::LayerVersion
- AWS::Serverless::Application
- AWS::Serverless::StateMachine

SAM Version 2016-10-31

## SAM serverless resources – IAM policies

**70+**

Managed templates  
with more being added.

Best Practice: [start here](#)  
<https://slip.link/sam-policies>

Policy Template	Description
SQSPollPolicy	Gives permission to poll an Amazon Simple Queue Service (Amazon SQS) queue.
LambdaInvokePolicy	Gives permission to invoke an AWS Lambda function, alias, or version.
CloudWatchMetricsAlarmPolicy	Gives permission to describe CloudWatch alarm history.
CloudWatchMetricsPolicy	Gives permission to send metrics to CloudWatch.
EC2DescribePolicy	Gives permission to describe Amazon Elastic Compute Cloud (Amazon EC2) instances.
DynamoDBReadPolicy	Gives create, read, update, and delete permissions to an Amazon DynamoDB table.
DynamoDBWritePolicy	Gives read-only permission to a DynamoDB table.
DynamoDBScanPolicy	Gives write-only permission to a DynamoDB table.
DynamoDBDescribeTablePolicy	Gives permission to describe a DynamoDB table.
SESIdentityPolicy	Gives SendEmail permission to an Amazon Simple Email Service (Amazon SES) identity.
ElastiCacheReplicationPolicy	Gives POST permission to Amazon ElastiCache service.
S3ReadPolicy	Gives read-only permission to objects in an Amazon Simple Storage Service (Amazon S3) bucket.
S3WritePolicy	Gives write permission to objects in an Amazon S3 bucket.
S3DeletePolicy	Gives create, read, update, and delete permission to objects in an Amazon S3 bucket.
AmazonMachineImagePolicy	Gives permission to describe Amazon Machine Images (AMI).
CloudFormationDescribeStackPolicy	Gives permission to describe AWS CloudFormation stacks.
RekognitionDetectFacesPolicy	Gives permission to detect faces, labels, and text.
RekognitionCompareFacesPolicy	Gives permission to compare and detect faces and labels.
RekognitionSearchFacesPolicy	Gives permission to list and search faces.
RekognitionIndexFacesPolicy	Gives permission to create collection and index faces.
SQSSendMessagePolicy	Gives permission to send message to an Amazon SQS queue.
SNSPublishMessagePolicy	Gives permission to publish a message to an Amazon Simple Notification Service (Amazon SNS) topic.
CloudWatchMetricsMetricsInsightPolicy	Gives access to create, delete, describe and detach CloudWatch Metrics Insights.

AWS

API GW - Type=API

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

## SAM serverless resources – Api (REST API)



Amazon API Gateway

AWS::Serverless::Function  
**AWS::Serverless::Api**  
AWS::Serverless::HttpApi  
AWS::Serverless::SimpleTable  
AWS::Serverless::LayerVersion  
AWS::Serverless::Application  
AWS::Serverless::StateMachine

```
1 MyAPI:  
2   Type: AWS::Serverless::Api  
3   Properties:  
4     StageName: prod  
5     DefinitionUri: ./swagger.yml  
6
```

Tip: You can use OpenAPI to define your API

### API GW - HTTP API (or API GW v2)

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

## SAM serverless resources – HttpApi (HTTP API)



Amazon API Gateway

AWS::Serverless::Function  
AWS::Serverless::Api  
**AWS::Serverless::HttpApi**  
AWS::Serverless::SimpleTable  
AWS::Serverless::LayerVersion  
AWS::Serverless::Application  
AWS::Serverless::StateMachine

```
1 HttpApi:  
2   Type: AWS::Serverless::HttpApi  
3   Properties:  
4     CorsConfiguration:  
5       AllowMethods:  
6         - GET  
7         - POST  
8       AllowOrigins:  
9         - http://localhost:8080  
10        - http://slip.link
```

Tip: You can use OpenAPI to define your API

### LayerVersion

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

# SAM serverless resources - LayerVersion

AWS::Serverless::Function  
 AWS::Serverless::Api  
 AWS::Serverless::HttpApi  
 AWS::Serverless::SimpleTable  
**AWS::Serverless::LayerVersion**

```

1 MyLayer:
2   Type: AWS::Serverless::LayerVersion
3   Properties:
4     LayerName: static-data
5     Description: static data layer for app
6     ContentUri: layer/
7     CompatibleRuntimes:
8       - nodejs12.x
9     LicenseInfo: 'MIT'
10    RetentionPolicy: Retain

```

## SAM serverless resources – LayerVersion build options

**Option 1:** Set "BuildMethod" to a supported runtime or leave blank to use the default build system.

**Option 2:** Set "BuildMethod" to **makefile** to use a MakeFile directed build.

```

1 MyLayer:
2   Type: AWS::Serverless::LayerVersion
3   Properties:
4     LayerName: static-data
5     Description: static data layer for app
6     ContentUri: layer/
7     CompatibleRuntimes:
8       - nodejs12.x
9     LicenseInfo: 'MIT'
10    RetentionPolicy: Retain
11    Metadata:
12      BuildMethod: nodejs12.x | makefile

```

## AWS SAM Globals

- The problem: repetition of code

### AWS SAM Globals

```

1 MyCatFunction:
2   Type: AWS::Serverless::Function
3   Properties:
4     Handler: index.handler
5     Runtime: nodejs12.x
6     CodeUri: /cat
7     Description: Creates cat thumbnails
8     MemorySize: 1024
9     Timeout: 15
10    Policies: S3ReadPolicy
11      BucketName: !Ref MyCatBucket
12    Layers:
13      - arn:aws::5555555555:layer:aws-sdk:6
14    Environment:
15      Variables:
16        TABLE_NAME: !Ref MyAnimals
17    Events:
18      PhotoUpload:
19        Type: S3
20        Properties:
21          Bucket: !Ref MyCatBucket
22          Events: s3:ObjectCreated:*

```

```

1 MyDogFunction:
2   Type: AWS::Serverless::Function
3   Properties:
4     Handler: index.handler
5     Runtime: nodejs12.x
6     CodeUri: /dog
7     Description: Creates dog thumbnails
8     MemorySize: 1024
9     Timeout: 15
10    Policies: S3ReadPolicy
11      BucketName: !Ref MyDogBucket
12    Layers:
13      - arn:aws::5555555555:layer:aws-sdk:6
14    Environment:
15      Variables:
16        TABLE_NAME: !Ref MyAnimals
17    Events:
18      PhotoUpload:
19        Type: S3
20        Properties:
21          Bucket: !Ref MyDogBucket
22          Events: s3:ObjectCreated:*

```

- The solution: defining global variables

## AWS SAM Globals

```

1 Globals:
2   Function:
3     Runtime: nodejs12.x
4     Handler: index.handler
5     Timeout: 15
6     MemorySize: 1024
7     Layers:
8       - arn:aws::5555555555:layer:aws-sdk:6
9     Environment:
10    Variables:
11      TABLE_NAME: MyAnimals

```

```

1 MyCatFunction:
2   Type: AWS::Serverless::Function
3   Properties:
4     CodeUri: /cat
5     Description: Creates cat thumbnails
6     Policies: S3ReadPolicy
7       BucketName: !Ref MyCatBucket
8     Events:
9       PhotoUpload:
10      Type: S3
11      Properties:
12        Bucket: !Ref MyCatBucket
13        Events: s3:ObjectCreated:*

```

```

1 MyDogFunction:
2   Type: AWS::Serverless::Function
3   Properties:
4     CodeUri: /dog
5     Description: Creates dog thumbnails
6     Policies: S3ReadPolicy
7       BucketName: !Ref MyDogBucket
8     Events:
9       PhotoUpload:
10      Type: S3
11      Properties:
12        Bucket: !Ref MyDogBucket
13        Events: s3:ObjectCreated:*

```

## 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

## Parameters

- Types:
- String
- Number
- List<Number>
- CommaDelimitedList
- AWS-Specific parameter types
- AWS SSM or AWS Secrets Manager parameter types

```

1 Parameters:
2   DomainName:
3     Type: String
4     Description: Domain name for api
5   ZoneId:
6     Type: String
7     Description: Zone ID if exists. If not leave as none.
8   Default: none
9   CertArn:
10  Type: String
11  Description: Certificate ARN if exists. If not leave as none.
12  Default: none

```

```

1 Parameters:
2   DbEngine:
3     Type: AWS::SSM::Parameter::Value<String>
4     Default: /myApp/DbEngine

```

## Parameters inline

```

1 LambdaFunction:
2   Type: AWS::Serverless::Function
3   Properties:
4     CodeUri: src/
5     Runtime: nodejs12.x
6     Handler: app.lambdaHandler
7   Environment:
8     Variables:
9       DB_ENGINE: !Ref DbEngine
10      DB_VERSION: '{{resolve:ssm:/myApp/DbVersion:1}}'
11      DB_NAME: '{{resolve:secretsmanager:/myApp/DbName}}'
12      DB_USERNAME: '{{resolve:secretsmanager:/myApp/DbCreds:SecretString:Username}}'
13      DB_PASSWORD: '{{resolve:secretsmanager:/myApp/DbCreds:SecretString:Password}}'

```

## Pseudo Parameters

## Pseudo parameters

- AWS::AccountId
- AWS::NotificationARNs
- AWS::NoValue
- AWS::Partition
- AWS::Region
- AWS::StackId
- AWS::StackName
- AWS::URLSuffix

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

## Intrinsic functions

- Fn::Base64
- Fn::Cidr
- Condition functions
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or
- Fn::FindInMap
- Fn::GetAtt
- Fn::GetAZs
- Fn::ImportValue
- Fn::Join
- Fn::Select
- Fn::Split
- Fn::Sub
- Fn::Transform
- Ref

## Putting them together

## Output an endpoint for Amazon API Gateway

## Creating conditions

## Example:

## Using conditions

```
1 GeneratedZone:  
2   Type: AWS::Route53::HostedZone  
3   Condition: CreateZone  
4   Properties:  
5     Name: !Ref DomainName
```

## Part 2 - AWS SAM CLI

AWS SAM Command Line Interface (CLI)

CLI tool for local development, debugging, testing, deploying, and monitoring of serverless applications



Supports API Gateway “proxy-style” and Lambda service API testing

Response object and function logs available on your local machine

Uses open source docker-lambda images to mimic Lambda's execution environment such as timeout, memory limits, runtimes

Can tail production logs from CloudWatch logs

Can help you build in native dependencies

<https://aws.amazon.com/serverless/samples>

## 1. SAM init

## SAM init

- Creates a new serverless application
- Options
  - Runtime: nodejs, dotnet, go, python, java, ruby
  - Dependency manager: (Ex: Maven or Gradle)
  - Output directory: path to application
  - Name: Application name
  - Location: location of custom template

As a result, a new project is created:

## SAM init

Creates a serverless application  
based on user input

- Select an AWS managed template or a custom template of your own
- Select runtime
- Select dependency manager
- Name application

## Results

```
1 MyApp
2   └── README.md
3   └── events
4     └── event.json
5   └── hello-world
6     ├── app.js
7     └── package.json
8     └── tests
9       └── unit
10      └── test-handler.js
11 └── template.yaml
```

## 2. SAM build

## SAM build

- Generates deployment artifacts targeting Lambda's execution environment
- Options
  - Use container to compile for native dependencies on Lambda execution environment
  - Utilizes native build tools like Dotnet, Go, and Maven for compiling
  - Allows for file omission in final artifacts
  - Use Makefile for custom builds

The result will be:

```
1 sam build
2 Building function 'HelloWorldFunction'
3 Running NodejsNpmBuilder:NpmPack
4 Running NodejsNpmBuilder:CopyNpmrc
5 Running NodejsNpmBuilder:CopySource
6 Running NodejsNpmBuilder:NpmInstall
7 Running NodejsNpmBuilder:CleanUpNpmrc
8
9 Build Succeeded
10
11 Built Artifacts : .aws-sam/build
12 Built Template  : .aws-sam/build/template.yaml
13
14 Commands you can use next
15 =====
16 [*] Invoke Function: sam local invoke
17 [*] Deploy: sam deploy --guided
```

## Results

```
1 .aws-sam
2   └── build
3     ├── HelloWorldFunction
4     │   ├── app.js
5     │   └── node_modules
6     └── package.json
7   └── template.yaml
```

- 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

## SAM deploy

- Deploys or updates serverless application on AWS Cloud
- Deploys and manages S3 bucket for deployment artifacts
- Saves deploy options to a config file

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 processes.

```
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

```
1 $ sam local generate-event s3 put
2 {
3   "Records": [
4     {
5       "eventVersion": "2.0",
6       "s3": {
7         "bucket": {
8           "name": "example-bucket",
9           "ownerIdentity": {
10             "principalId": "EXAMPLE"
11           },
12           "arn": "arn:aws:s3:::example-bucket"
13         },
14         "object": {
15           "key": "test/key",
16           "size": 1024,
17           "eTag": "0123456789abcdef0123456789abcdef",
18           "sequencer": "0A1B2C3D4E5F678901"
19         }
20       }
21     }
22   ]
23 }
```

\*Abbreviated for display

Generates a JSON object representing the output from the given service and type.

Current available event types:

alexa-skills-kit	connect
alexa-smart-home	dynamodb
apigateway	kinesis
batch	lex
cloudformation	rekognition
cloudfront	s3
cloudwatch	sagemaker
codecommit	ses
codepipeline	sns
cognito	sqs
config	stepfunctions

```
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

**SAM local start-api**

```

1 sam local start-api
2 Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
3 You can now browse to the above endpoints to invoke your functions. You do not need to restart/reload SAM CLI
   while working on your functions, changes will be reflected instantly/automatically. You only need to restart SAM
   CLI if you update your AWS SAM template
4 2020-06-15 11:21:42 * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)

$ curl http://127.0.0.1:3000/hello

```

REST client call

```

1 Invoking app.lambdaHandler (nodejs12.x)
2 Fetching lambda:nodejs12.x Docker container image.....
3 Mounting /Users/ericd/Desktop/MyApp/.aws-sam/build/HelloWorldFunction as
   /var/task:ro,delegated inside runtime container
4 START RequestId: 03fde5d1-2ade-1f9b-f5dd-018d8f463370 Version: $LATEST
5 END RequestId: 03fde5d1-2ade-1f9b-f5dd-018d8f463370
6 REPORT RequestId: 03fde5d1-2ade-1f9b-f5dd-018d8f463370 Init Duration:
   344.13 ms Duration: 7.69 ms Billed Duration: 100 ms Memory Size: 128
   MB Max Memory Used: 39 MB
7 No Content-Type given. Defaulting to 'application/json'.
8 2020-06-15 11:24:35 127.0.0.1 - - [15/Jun/2020 11:24:35] "GET /hello
   HTTP/1.1" 200 -

```

SAM fetches proper docker container if it does not exist.

Lambda function invoked locally via local API Gateway emulator.

## Some helpful options for SAM local start-api

- env-vars
- debug-port
- debug-path
- debug-args
- log-file
- static-directory
- docker-network

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
```

## Some helpful options for SAM local invoke

- [function name]
- --event
- --env-vars
- --debug-port
- --debug-path
- --debug-args
- --log-file
- --static-directory
- --docker-network

- 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 continuously watch CloudWatch logs

## Some helpful options for SAM logs

- --filter: simple keywords or CloudWatch supported pattern
- --start-time: relative values or timestamps
- --end-time: relative values or timestamps
- --tail: continuously fetches logs

Debugging locally on VS Code

# Debugging locally with SAM CLI in VSCode

```

1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Attach to SAM CLI",
6       "type": "node",
7       "request": "attach",
8       "address": "localhost",
9       "port": 5858,
10      "localRoot": "${workspaceRoot}/{app}",
11      "remoteRoot": "/var/task",
12      "protocol": "inspector",
13      "stopOnEntry": false
14    }
15  ]
16 }

```

## NODE Example

1. In your IDE create a debug configuration
2. Add a breakpoint in your code
3. Invoke the Lambda function using *start-api*, *start-lambda*, or *invoke lambda*
4. Attach the debugger

## 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](#)