SAM Documentation

Release 0.1.0

AWS

May 03, 2022

CONTENTS

CHAPTER

ONE

WHAT'S NEW?

- `Globals` section
- Support for Traffic Shifting Lambda deployments
- Refer to resources automatically created by SAM
- `FAQ` section

1.1 Globals Section

Contents

Resources in a SAM template tend to have shared configuration such as Runtime, Memory, VPC Settings, Environment Variables, Cors, etc. Instead of duplicating this information in every resource, you can write them once in the `Globals` section and let all resources inherit it.

Example:

```
Globals:
  Function:
    Runtime: nodejs6.10
    Timeout: 180
    Handler: index.handler
    Environment:
      Variables:
        TABLE_NAME: data-table
```

(continues on next page)

(continued from previous page)

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          MESSAGE: "Hello From SAM"

  ThumbnailFunction:
    Type: AWS::Serverless::Function
    Properties:
      Events:
        Thumbnail:
          Type: Api
          Properties:
            Path: /thumbnail
            Method: POST
```

In the above example, both `HelloWorldFunction` and `ThumbnailFunction` will use nodejs6.10 runtime, 180 seconds timeout and index.handler Handler. `HelloWorldFunction` adds MESSAGE environment variable in addition to the inherited TABLE_NAME. `ThumbnailFunction` inherits all the Globals properties and adds an API Event source.

1.1.1 Supported Resources and Properties

Currently, the following resources and properties are being supported:

```
Globals:
  Function:
    # Properties of AWS::Serverless::Function
    Handler:
    Runtime:
    CodeUri:
    DeadLetterQueue:
    Description:
    MemorySize:
    Timeout:
    VpcConfig:
    Environment:
    Tags:
    Tracing:
    KmsKeyArn:
    Layers:
    AutoPublishAlias:
    DeploymentPreference:
    PermissionsBoundary:
    ReservedConcurrentExecutions:
    EventInvokeConfig:
    Architectures:
    EphemeralStorage:

  Api:
```

(continues on next page)

(continued from previous page)

```
    # Properties of AWS::Serverless::Api
    # Also works with Implicit APIs
    Auth:
    Name:
    DefinitionUri:
    CacheClusterEnabled:
    CacheClusterSize:
    Variables:
    EndpointConfiguration:
    MethodSettings:
    BinaryMediaTypes:
    MinimumCompressionSize:
    Cors:
    GatewayResponses:
    AccessLogSetting:
    CanarySetting:
    TracingEnabled:
    OpenApiVersion:
    Domain:

  HttpApi:
    # Properties of AWS::Serverless::HttpApi
    # Also works with Implicit APIs
    Auth:
    CorsConfiguration:
    AccessLogSettings:
    Tags:
    DefaultRouteSettings:
    RouteSettings:
    Domain:

  SimpleTable:
    # Properties of AWS::Serverless::SimpleTable
    SSESpecification:
```

Implicit APIs

APIs created by SAM when you have an API declared in the `Events` section are called "Implicit APIs". You can use Globals to override all properties of Implicit APIs as well.

Unsupported Properties

Following properties are **not** supported in Globals section. We made the explicit call to not support them because it either made the template hard to understand or opened scope for potential security issues.

AWS::Serverless::Function:

- Role
- Policies
- FunctionName

- Events

AWS::Serverless::Api:

- StageName
- DefinitionBody

AWS::Serverless::HttpApi:

- StageName
- DefinitionBody
- DefinitionUri

1.1.2 Overridable

Properties declared in the Globals section can be overriden by the resource. For example, you can add new Variables to environment variable map or override globally declared variables. But the resource **cannot** remove a property specified in globals environment variables map. More generally, Globals declare properties shared by all your resources. Some resources can provide new values for globally declared properties but cannot completely remove them. If some resources use a property but others do not, then you must not declare them in the Globals section.

Here is how overriding works for various data types:

Primitive Values are replaced

String, Number, Boolean etc

Value specified in the resource will **replace** Global value

Example:

Runtime of `MyFunction` will be set to python3.6

```
Globals:
  Function:
    Runtime: nodejs4.3

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Runtime: python3.6
```

Maps are merged

Maps are also known as dictionaries or collections of key/value pairs

Map entries in the resource will be **merged** with global map entries. In case of duplicates the resource entry will override the global entry.

Example:

```yaml
Globals:
  Function:
    Environment:
      Variables:
        STAGE: Production
        TABLE_NAME: global-table

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          TABLE_NAME: resource-table
          NEW_VAR: hello
```

In the above example the environment variables of `MyFunction` will be set to:

```
{
  "STAGE": "Production",
  "TABLE_NAME": "resource-table",
  "NEW_VAR": "hello"
}
```

Lists are additive

Lists are also known as arrays

Global entries will be **prepended** to the list in the resource.

Example:

```yaml
Globals:
  Function:
    VpcConfig:
      SecurityGroupIds:
        - sg-123
        - sg-456

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      VpcConfig:
        SecurityGroupIds:
          - sg-first
```

In the above example the Security Group Ids of `MyFunction`'s VPC Config will be set to:

```
[ "sg-123", "sg-456", "sg-first" ]
```

1.2 Safe Lambda deployments

Contents

Pushing to production can be nerve-wracking even if you have 100% unit test coverage and a state-of-art full CD system. It is a good practice to expose your new code to a small percentage of production traffic, run tests, watch for alarms and dial up traffic as you gain more confidence. The goal is to minimize production impact as much as possible.

To enable traffic shifting deployments for Lambda functions, we will use Lambda Aliases, which can balance incoming traffic between two different versions of your function, based on preassigned weights. Before deployment, the alias sends 100% of invokes to the version used in production. During deployment, we will upload the code to Lambda, publish a new version, send a small percentage of traffic to the new version, monitor, and validate before shifting 100% of traffic to the new version. You can do this manually by calling Lambda APIs or let AWS CodeDeploy automate it for you. CodeDeploy will shift traffic, monitor alarms, run validation logic and even trigger an automatic rollback if something goes wrong.

SAM comes built-in with CodeDeploy support. You can enable automated traffic shifting Lambda deployments by adding the following lines to your `AWS::Serverless::Function` resource property or in the Globals section.

```
AutoPublishAlias: live
DeploymentPreference:
  Type: Linear10PercentEvery10Minutes
```

The rest of this document dives deep into how this snippet works, available configurations, and debugging techniques when deployments don't work as expected.

1.2.1 Instant traffic shifting using Lambda Aliases

Every Lambda function can have any number of Versions and Aliases associated with them. Versions are immutable snapshots of a function including code & configuration. If you are familiar with git, they are similar to commits. In general, it is a good practice to publish a new version every time you update your function code. When you invoke a specific version (using the function name + version number combination) you are guaranteed to get the same code & configuration irrespective of the state of the function. This protects you against accidentally updating production code.

To effectively use the versions, you should create an Alias which is literally a pointer to a version. Aliases have a name and an ARN similar to the function and are accepted by the Invoke APIs. If you invoke an Alias, Lambda will in turn invoke the version that the Alias is pointing to.

In production, you will first update your function code, publish a new version, invoke the version directly to run tests against it, and, after you are satisfied, flip the Alias to point to the new version. Traffic will instantly shift from using your old version to using the new version.

SAM provides a simple primitive to do this for you. Add the following property to your `AWS::Serverless::Function` resource:

```
AutoPublishAlias: <alias-name>
```

This will:

- Create an Alias with `<alias-name>`
- Create & publish a Lambda version with the latest code & configuration derived from the `CodeUri` property. Optionally it is possible to specify property *AutoPublishCodeSha256* that will override the hash computed for Lambda `CodeUri` property.
- Point the Alias to the latest published version
- Point all event sources to the Alias & not to the function
- When the `CodeUri` property of `AWS::Serverless::Function` changes, SAM will automatically publish a new version & point the alias to the new version

In other words, your traffic will shift "instantly" to your new code.

> NOTE: `AutoPublishAlias` will publish a new version only when the `CodeUri` changes. Updates to other configuration (ex: MemorySize, Timeout) etc will *not* publish a new version. Hence your Alias will continue to point to old version that uses the old configurations.

1.2.2 Traffic shifting using CodeDeploy

For production deployments, you may want more controlled traffic shifting from an old version to a new version which monitors alarms and triggers a rollback if necessary. CodeDeploy is an AWS service which can do this for you. It uses Lambda Alias' ability to route a percentage of traffic to two different Lambda Versions. To use this feature, set the `DeploymentPreference` property of `AWS::Serverless::Function` resource:

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs12.x
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before & after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
      # Provide a custom role for CodeDeploy traffic shifting here, if you don't supply␣
↪one
      # SAM will create one for you with default permissions
      Role: !Ref IAMRoleForCodeDeploy # Parameter example, you can pass an IAM ARN

AliasErrorMetricGreaterThanZeroAlarm:
  Type: "AWS::CloudWatch::Alarm"
```

(continues on next page)

(continued from previous page)

```
  Properties:
    AlarmDescription: Lambda Function Error > 0
    ComparisonOperator: GreaterThanThreshold
    Dimensions:
      - Name: Resource
        Value: !Sub "${MyLambdaFunction}:live"
      - Name: FunctionName
        Value: !Ref MyLambdaFunction
    EvaluationPeriods: 2
    MetricName: Errors
    Namespace: AWS/Lambda
    Period: 60
    Statistic: Sum
    Threshold: 0

LatestVersionErrorMetricGreaterThanZeroAlarm:
  Type: "AWS::CloudWatch::Alarm"
  Properties:
    AlarmDescription: Lambda Function Error > 0
    ComparisonOperator: GreaterThanThreshold
    Dimensions:
      - Name: Resource
        Value: !Sub "${MyLambdaFunction}:live"
      - Name: FunctionName
        Value: !Ref MyLambdaFunction
      - Name: ExecutedVersion
        Value: !GetAtt MyLambdaFunction.Version.Version
    EvaluationPeriods: 2
    MetricName: Errors
    Namespace: AWS/Lambda
    Period: 60
    Statistic: Sum
    Threshold: 0

PreTrafficLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: preTrafficHook.handler
    Policies:
      - Version: "2012-10-17"
        Statement:
        - Effect: "Allow"
          Action:
            - "codedeploy:PutLifecycleEventHookExecutionStatus"
          Resource:
            !Sub 'arn:${AWS::Partition}:codedeploy:${AWS::Region}:${AWS::AccountId}
↪:deploymentgroup:${ServerlessDeploymentApplication}/*'
      - Version: "2012-10-17"
        Statement:
        - Effect: "Allow"
          Action:
            - "lambda:InvokeFunction"
```

(continues on next page)

(continued from previous page)

```
        Resource: !GetAtt MyLambdaFunction.Arn
    Runtime: nodejs12.x
    FunctionName: 'CodeDeployHook_preTrafficHook'
    DeploymentPreference:
      Enabled: False
      Role: ""
    Environment:
      Variables:
        CurrentVersion: !Ref MyLambdaFunction.Version
```

When you update your function code and deploy the SAM template using CloudFormation, the following happens:

- CloudFormation publishes a new Lambda Version from the new code
- Since a deployment preference is set, CodeDeploy takes over the job of actually shifting traffic from old version to new version.
- Before traffic shifting starts, CodeDeploy will invoke the **PreTraffic Hook** Lambda function. This Lambda function must call back to CodeDeploy with an explicit status of Success or Failure, via the PutLifecycleEventHookExecutionStatus API. On Failure, CodeDeploy will abort and report a failure back to CloudFormation. On Success, CodeDeploy will proceed with the specified traffic shifting. Here is a sample Lambda Hook function.
- `Type: Linear10PercentEvery10Minutes` instructs CodeDeploy to start with 10% traffic on new version and add 10% every 10 minutes. It will complete traffic shifting in 100 minutes.
- During traffic shifting, if any of the CloudWatch Alarms go to *Alarm* state, CodeDeploy will immediately flip the Alias back to old version and report a failure to CloudFormation.
- After traffic shifting completes, CodeDeploy will invoke the **PostTraffic Hook** Lambda function. This is similar to PreTraffic Hook where the function must callback to CodeDeploy to report a Success or a Failure. PostTraffic hook is a great place to run integration tests or other validation actions.
- If everything went well, the Alias will be pointing to the new Lambda Version.
- If you supply the "Role" argument to the DeploymentPreference, it will prevent SAM from creating a role and instead use the provided CodeDeploy role for traffic shifting

NOTE: Verify that your AWS SDK version supports PutLifecycleEventHookExecutionStatus. For example, Python requires SDK version 1.4.8 or newer.

Traffic Shifting Configurations

In the above example `Linear10PercentEvery10Minutes` is one of several preselected traffic shifting configurations available in CodeDeploy. You can pick the configuration that best suits your application. See docs for the complete list:

- Canary10Percent30Minutes
- Canary10Percent5Minutes
- Canary10Percent10Minutes
- Canary10Percent15Minutes
- AllAtOnce
- Linear10PercentEvery10Minutes
- Linear10PercentEvery1Minute

- Linear10PercentEvery2Minutes
- Linear10PercentEvery3Minutes

They work as follows:

- **LinearXPercentYMinutes**: Traffic to new version will linearly increase in steps of X percentage every Y minutes.

  Ex: `Linear10PercentEvery10Minutes` will add 10 percentage of traffic every 10 minute to complete in 100 minutes.

- **CanaryXPercentYMinutes**: X percent of traffic will be routed to new version for Y minutes. After Y minutes, 100 percent of traffic will be sent to new version. Some people call this as Blue/Green deployment.

  Ex: `Canary10Percent15Minutes` will send 10 percent traffic to new version and 15 minutes later complete deployment by sending all traffic to new version.

- **AllAtOnce**: This is an instant shifting of 100% of traffic to new version. This is useful if you want to run run pre/post hooks but don't want a gradual deployment. If you have a pipeline, you can set Beta/Gamma stages to deploy instantly because the speed of deployments matter more than safety here.
- **Custom**: Aside from Above mentioned Configurations, Custom Codedeploy configuration are also supported. (Example. Type: CustomCodeDeployConfiguration)

PreTraffic & PostTraffic Hooks

CodeDeploy allows you to run an arbitrary Lambda function before traffic shifting actually starts (PreTraffic Hook) and after it completes (PostTraffic Hook). With either hook, you have the opportunity to run logic that determines whether the deployment must succeed or fail. For example, with PreTraffic hook you could run integration tests against the newly created Lambda version (but not serving traffic). With PostTraffic hook, you could run end-to-end validation checks.

Hooks are extremely powerful because:

- **Not limited by Lambda function duration**: CodeDeploy invokes the hook function asynchronously. The function will receive a `deploymentId` and `lifecycleEventHookExecutionId` that should be used with a call to the CodeDeploy API to report success or failure. Therefore you can build a workflow that runs for several minutes or hours before completing the hook by calling the CodeDeploy API.
- **New Version is created before PreTraffic Hook runs**: Before PreTraffic hook runs, the Lambda Version containing the new code has been created but this version is not serving any traffic yet. Therefore, in your hook function, you can directly invoke the Version to run integration tests or even pre-warm the Lambda containers before exposing it to production traffic.

  > NOTE: The event payload delivered to the Hook function will not contain the Lambda ARN to be tested. We recommend adding an Environment variable to the Hook function that maintains the current Version of the Lambda requiring safe deployments

```
Environment:
  Variables:
    CurrentVersion: !Ref MySafeLambdaFunction.Version
```

- **Hooks are executed per-function**: Each Lambda function gets its own PreTraffic and PostTraffic hook (technically speaking hooks are executed once per DeploymentGroup, but in this case the DeploymentGroup contains only one Lambda Function). So you can customize the hooks logic to the function that is being deployed.

  > NOTE: If the Hook functions are created by the same SAM template that is deployed, then make sure to turn off traffic shifting deployments for the hook functions. Also, the Role SAM generates

for a Lambda Execution Role does not include all permissions needed for Pre and Post hook functions, since it will not contain the necessary permissions to call the CodeDeploy APIs or Invoke your new Lambda function for testing. Instead, use the Policies attribute to provide the CodeDeploy and Lambda permissions needed. The example also shows a Policy that provides access to the CodeDeploy resource that SAM automatically generates. Finally, use the `FunctionName` property to control the exact name of the Lambda function CloudFormation creates. Otherwise, CloudFormation will create your Lambda function with the Stack name and a unique ID added as part of the name.

```
FunctionName: 'CodeDeployHook_preTrafficHook'
DeploymentPreference:
    Enabled: False
Policies:
    - Version: "2012-10-17"
      Statement:
      - Effect: "Allow"
        Action:
          - "codedeploy:PutLifecycleEventHookExecutionStatus"
        Resource: "*"
    - Version: "2012-10-17"
      Statement:
      - Effect: "Allow"
        Action:
          - "lambda:InvokeFunction"
        Resource: !GetAtt MyLambdaFunction.Arn
```

Checkout the lambda_safe_deployments folder for an example for how to create SAM template that contains a hook function.

Internals

Internally, SAM will create the following resources in your CloudFormation stack to make all of this work:

- One `AWS::CodeDeploy::Application` per stack, that is referencable via ${ServerlessDeploymentApplication}
- One `AWS::CodeDeploy::DeploymentGroup` per `AWS::Serverless::Function` resource. Each Lambda Function in your SAM template belongs to its own Deployment Group.
- Adds `UpdatePolicy` on `AWS::Lambda::Alias` resource that is connected to the function's Deployment Group resource.
- One `AWS::IAM::Role` called "CodeDeployServiceRole", if no custom role is provided

CodeDeploy assumes that there are no dependencies between Deployment Groups and hence will deploy them in parallel. Since every Lambda function is to its own CodeDeploy DeploymentGroup, they will be deployed in parallel. The CodeDeploy service will assume the new CodeDeployServiceRole to Invoke any Pre/Post hook functions and perform the traffic shifting and Alias updates.

NOTE: The CodeDeployServiceRole only allows InvokeFunction on functions with names prefixed with `CodeDeployHook_`. For example, you should name your Hook functions as such: `CodeDeployHook_PreTrafficHook`.

Production errors preventing deployments

In some situations, an issue that is happening in production may prevent you from deploying a fix. This may happen when a deployment happens when traffic is too low to register enough errors to trigger a roll back, or where someone is sending malicious traffic through to a lambda and you haven't accounted for the scenario where they do.

When this happens, the alarm for errors in the current lambda version is in an error state, which will cause code deploy to roll back any attempted deploys straight away.

To release code in this situation, you need to

- Go into the CodeDeploy console
- Select the application you want to deploy to
- Select the corresponding Deployment Group
- Select "Edit"
- Select "Advanced - optional"
- Select "ignore alarm configuration"
- Save the changes

Run your deployment as usual

Then once deployment has successfully run, return to the CodeDeploy console, and follow the above steps but this time deselect "ignore alarm configuration".

1.3 Policy Templates

When you define a Serverless Function, SAM automatically creates the IAM Role required to run the function. Let's say your function needs to access couple of DynamoDB tables, you need to give your function explicit permissions to access the tables. You can do this by adding AWS Managed Policies to Serverless Function resource definition in your SAM template.

For Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Policies:
      # Give DynamoDB Full Access to your Lambda Function
      - AmazonDynamoDBFullAccess
    ...

MyTable:
  Type: AWS::Serverless::SimpleTable
```

Behind the scenes, `AmazonDynamoDBFullAccess` will give your function access to **all** DynamoDB APIs against **all** DynamoDB tables in **all** regions. This is excessively permissive when all that your function does is Read & Write values from the `MyTable` created in the stack.

SAM provides a tighter and more secure version of AWS Managed Policies called **Policy Templates**. This are a set of readily available policies that can be scoped to a specific resource in the same region where your stack exists. Let's modify the above example to use a policy template called `DynamoDBCrudPolicy`:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Policies:

      # Give just CRUD permissions to one table
      - DynamoDBCrudPolicy:
          TableName: !Ref MyTable

    ...

MyTable:
  Type: AWS::Serverless::SimpleTable
```

1.3.1 How to Use

Policy Templates are specified in `Policies` property of AWS::Serverless::Function resource. You can mix policy templates with AWS Managed Policies, custom managed policies or inline policy statements. Behind the scenes SAM will expand the policy template to an inline policy statement based on the definition listed in policy_templates.json file.

Every policy template requires zero or more parameters, which are the resource that this policy is scoped to. Your template will fail to deploy if the value for a required parameter is not specified. You can consult the policy_templates.json file for name of the policy templates, parameter names as well as the actual policy statement it represents.

If you want a quick reference of all policies, checkout the all_policy_templates.yaml SAM template in examples folder.

> NOTE: If a policy template does not require a parameter, you should still specify the value to be an empty dictionary like this:

```
Policies:
  - CloudWatchPutMetricPolicy: {}
```

1.4 SAM Internals

Explore the topics in this section to learn more about the internals of how SAM works.

1.4.1 CloudFormation Resources Generated By SAM

        - *S3*
        - *SNS*
        - *Kinesis*
        - *MQ*
        - *MSK*
        - *SQS*
        - *DynamoDb*
        - *Schedule*
        - *CloudWatchEvent (superseded by EventBridgeRule, see below)*
        - *EventBridgeRule*
    - *AWS::Serverless::Api*

When you create a Serverless Function or a Serverless API, SAM will create additional AWS resources to wire everything up. For example, when you create a `AWS::Serverless::Function`, SAM will create a Lambda Function resource along with an IAM Role resource to give appropriate permissions for your function. This document describes all such generated resources, how they are named, and how to refer to them in your SAM template.

AWS::Serverless::Function

Given a Function defined as follows:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
```

Following resources will be generated:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Function	MyFunction
AWS::IAM::Role	MyFunction**Role**

With AutoPublishAlias Property

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    AutoPublishAlias: live
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Version	MyFunction**Version***SHA* (10 digits of SHA256 of CodeUri)
AWS::Lambda::Alias	MyFunction**Alias***live*

With DeploymentPreference Property

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    AutoPublishAlias: live
    DeploymentPreference:
      Type: Linear10PercentEvery10Minutes
      Role: "arn"
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::CodeDeploy::Application	ServerlessDeploymentApplication (only one per stack)
AWS::CodeDeploy::DeploymentGroup	MyFunction**DeploymentGroup**
AWS::IAM::Role	CodeDeployServiceRole

NOTE: `AWS::IAM::Role` resources are only generated if no Role parameter is supplied for DeploymentPreference

With Events

A common theme with all Events is SAM will generate a `AWS::Lambda::Permission` resource to give event source permission to invoke the function. Other generated resources depend on the specific event type.

API

This is called an "Implicit API". There can be many functions in the template that define these APIs. Behind the scenes, SAM will collect all implicit APIs from all Functions in the template, generate a Swagger, and create an implicit `AWS::Serverless::Api` using this Swagger. This API defaults to a StageName called "Prod" that cannot be configured.

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      ThumbnailApi:
        Type: Api
        Properties:
          Path: /thumbnail
```

(continues on next page)

(continued from previous page)

```
        Method: GET
  ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::ApiGateway::RestApi	*ServerlessRestApi*
AWS::ApiGateway::Stage	*ServerlessRestApi***Prod**Stage
AWS::ApiGateway::Deployment	*ServerlessRestApi*Deployment*SHA* (10 Digits of SHA256 of Swagger)
AWS::Lambda::Permission	MyFunction**ThumbnailApi**Permission**Prod** (Prod is the default Stage Name for implicit APIs)

NOTE: `ServerlessRestApi*` resources are generated one per stack.

HTTP API

This is called an "Implicit HTTP API". There can be many functions in the template that define these APIs. Behind the scenes, SAM will collect all implicit HTTP APIs from all Functions in the template, generate an OpenApi doc, and create an implicit `AWS::Serverless::HttpApi` using this OpenApi. This API defaults to a StageName called "$default" that cannot be configured.

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      ThumbnailApi:
        Type: HttpApi
        Properties:
          Path: /thumbnail
          Method: GET
  ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::ApiGatewayV2::Api	*ServerlessHttpApi*
AWS::ApiGatewayV2::Stage	*ServerlessHttpApiApiGatewayDefaultStage*
AWS::Lambda::Permission	MyFunction**ThumbnailApi**Permission

NOTE: `ServerlessHttpApi*` resources are generated one per stack.

Cognito

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      CognitoTrigger:
        Type: Cognito
        Properties:
          UserPool: !Ref MyUserPool
          Trigger: PreSignUp
    ...

MyUserPool:
  Type: AWS::Cognito::UserPool
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permissions	*MyFunction*CognitoPermission
AWS::Cognito::UserPool	Existing MyUserPool resource is modified to append `LambdaConfig` property where the Lambda function trigger is defined

NOTE: You **must** refer to a Cognito UserPool defined in the same template. This is for two reasons:

1. SAM needs to add a `LambdaConfig` property to the UserPool resource by reading and modifying the resource definition

2. Lambda triggers are specified as a property on the UserPool resource. Since CloudFormation cannot modify a resource created outside of the stack, this bucket needs to be defined within the template.

S3

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      S3Trigger:
        Type: S3
        Properties:
          Bucket: !Ref MyBucket
          Events: s3:ObjectCreated:*
    ...

MyBucket:
  Type: AWS::S3::Bucket
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**S3Trigger**Permission
AWS::S3::Bucket	Existing MyBucket resource is modified to append `NotificationConfiguration` property where the Lambda function trigger is defined

NOTE: You **must** refer to an S3 Bucket defined in the same template. This is for two reasons:

1. SAM needs to add a `NotificationConfiguration` property to the bucket resource by reading and modifying the resource definition

2. Lambda triggers are specified as a property on the bucket resource. Since CloudFormation cannot modify a resource created outside of the stack, this bucket needs to be defined within the template.

SNS

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: SNS
        Properties:
          Topic: arn:aws:sns:us-east-1:123456789012:my_topic
          SqsSubscription:
            QueuePolicyLogicalId: CustomQueuePolicyLogicalId
            QueueArn: !GetAtt MyCustomQueue.Arn
            QueueUrl: !Ref MyCustomQueue
            BatchSize: 5
            Enabled: true
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**EventSourceMapping
AWS::SNS::Subscription	MyFunction**MyTrigger**
AWS::SQS::Queue	MyFunction**MyTrigger**Queue
AWS::SQS::QueuePolicy	MyFunction**MyTrigger**QueuePolicy

NOTE: `AWS::Lambda::Permission` resources are only generated if SqsSubscription is `false`. `AWS::Lambda::EventSourceMapping`, `AWS::SQS::Queue`, `AWS::SQS::QueuePolicy` resources are only generated if SqsSubscription is `true`.

`AWS::SQS::Queue` resources are only generated if SqsSubscription is `true`.

Example:

```
MyFunction:
Type: AWS::Serverless::Function
Properties:
  ...
  Events:
    MyTrigger:
      Type: SNS
      Properties:
        Topic: arn:aws:sns:us-east-1:123456789012:my_topic
        SqsSubscription: true
  ...
```

Kinesis

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: Kinesis
        Properties:
          Stream: arn:aws:kinesis:us-east-1:123456789012:stream/my-stream
          StartingPosition: TRIM_HORIZON
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**

MQ

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: MQ
        Properties:
          Broker: arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-
↪2fgh-3i45j6k178l9
          SourceAccessConfigurations:
            Type: BASIC_AUTH
```

(continues on next page)

(continued from previous page)

```
          URI: arn:aws:secretsmanager:us-west-2:123456789012:secret:my-path/my-secret-
↪name-1a2b3c
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**

MSK

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: MSK
        Properties:
          Stream: arn:aws:kafka:us-east-1:123456789012:cluster/mycluster/6cc0432b-8618-
↪4f44-bccc-e1fbd8fb7c4d-2
          StartingPosition: TRIM_HORIZON
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**

SQS

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: SQS
        Properties:
          Queue: arn:aws:sqs:us-east-1:123456789012:my-queue
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**

DynamoDb

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTrigger:
        Type: DynamoDb
        Properties:
          Stream: arn:aws:dynamodb:us-east-1:123456789012:table/TestTable/stream/2016-08-
↪11T21:21:33.291
          StartingPosition: TRIM_HORIZON
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTrigger**Permission
AWS::Lambda::EventSourceMapping	MyFunction**MyTrigger**

Schedule

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      MyTimer:
        Type: Schedule
        Properties:
          Input: rate(5 minutes)
          DeadLetterConfig:
            Type: SQS
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**MyTimer**Permission
AWS::Events::Rule	MyFunction**MyTimer**
AWS::SQS::Queue	MyFunction**MyTimer**Queue
AWS::SQS::QueuePolicy	MyFunction**MyTimer**QueuePolicy

CloudWatchEvent (superseded by EventBridgeRule, see below)

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      OnTerminate:
        Type: CloudWatchEvent
        Properties:
          Pattern:
            source:
              - aws.ec2
            detail-type:
              - EC2 Instance State-change Notification
            detail:
              state:
                - terminated
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**OnTerminate**Permission
AWS::Events::Rule	MyFunction**OnTerminate**

EventBridgeRule

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      OnTerminate:
        Type: EventBridgeRule
        Properties:
          Pattern:
            source:
              - aws.ec2
            detail-type:
              - EC2 Instance State-change Notification
            detail:
              state:
                - terminated
            DeadLetterConfig:
              Type: SQS
            RetryPolicy:
              MaximumEventAgeInSeconds: 600
```

(continues on next page)

(continued from previous page)

```
            MaximumRetryAttempts:3
    ...
```

Additional generated resources:

CloudFormation Resource Type	Logical ID
AWS::Lambda::Permission	MyFunction**OnTerminate**Permission
AWS::Events::Rule	MyFunction**OnTerminate**
AWS::SQS::Queue	MyFunction**OnTerminate**Queue
AWS::SQS::QueuePolicy	MyFunction**OnTerminate**QueuePolicy

AWS::Serverless::Api

In contrast to Implict APIs, you can explicitly define your API resource by providing an entire Swagger definition of your API.

Example:

```
MyApi:
  Type: AWS::Serverless::Api
  Properties:
    ...
    DefinitionUri: s3://bucket/swagger.json
    StageName: dev
    ...
```

Generated resources:

CloudFormation Resource Type	Logical ID
AWS::ApiGateway::RestApi	MyApi
AWS::ApiGateway::Stage	MyApi**dev**Stage
AWS::ApiGateway::Deployment	MyApiDeployment*SHA* (10 Digits of SHA256 of DefinitionUri or DefinitionBody value)

NOTE: By just specifying AWS::Serverless::Api resource, SAM will *not* add permission for API Gateway to invoke the the Lambda Function backing the APIs. You should explicitly re-define all APIs under `Events` section of the AWS::Serverless::Function resource but include a *RestApiId* property that references the AWS::Serverless::Api resource. SAM will add permission for these APIs to invoke the function.

Example:

```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      GetApi:
        Type: Api
        Properties:
          Path: /
          Method: GET
```

(continues on next page)

(continued from previous page)

```
        # This is the property that instructs SAM to just add permissions␣
↪for an explicitly defined API
        RestApiId: !Ref MyApi
```

1.5 FAQ

Frequently Asked Questions

1.5.1 How to manage multiple environments?

> *Terminology clarification: Environment and Stage can normally be used interchangeably but since AWS API Gateway relies on a concrete concept of Stages we'll use the term Environment here to avoid confusion.*

We recommend a one-to-one mapping of environment to Cloudformation Stack.

This means having a separate CloudFormation stack per environment, using a single template file with a dynamically set target stack via the `--stack-name` parameter in the `aws cloudformation deploy` command.

For example, lets say we have 3 environments (dev, test, and prod). Each of those would have their own CloudFormation stack — *dev-stack*, *test-stack*, *prod-stack*. Our CI/CD system will deploy to *dev-stack*, *test-stack*, and then *prod-stack* but will be pushing one template through all of these stacks.

This approach limits the 'blast radius' for any given deployment since all resources for each environment are scoped to a different CloudFormation Stack, so we will never be editing production resources on accident.

If we need to bring up separate stacks for different reasons (multiple region deployments, developer/branch stacks) it will be straightforward to do so with this approach since the same template can be used to bring up and manage a new stack independent of any others.

In cases where you need to manage different stages differently this can be done through a combination of Stack Parameters, Conditions, and Fn::If statements.

1.5.2 How to enable API Gateway Logs

Work is underway to make this functionality part of the SAM specification. Until then a suggested workaround is to use the `aws cli update-stage` command to enable it.

```
aws apigateway update-stage \
  --rest-api-id <api-id> \
  --stage-name <stage-name> \
  --patch-operations \
    op=replace,path=/*/*/logging/dataTrace,value=true \
    op=replace,path=/*/*/logging/loglevel,value=Info \
    op=replace,path=/*/*/metrics/enabled,value=true
```

The command above can be run as a post deployment CI step or it could be triggered by a custom resource within the same CloudFormation template.

Please note that in either case you will see metric gaps between the time CloudFormation updates API Gateway and the time this command runs.

1.5.3 How to deploy Lambda@Edge functions with SAM?

At present, SAM doesn't support Lambda@Edge as a native event. However you can follow this example to ease deployment: Lambda Edge Example.