SAM Best Practices MONSANTO

Luiz Yanai, Solutions Architect

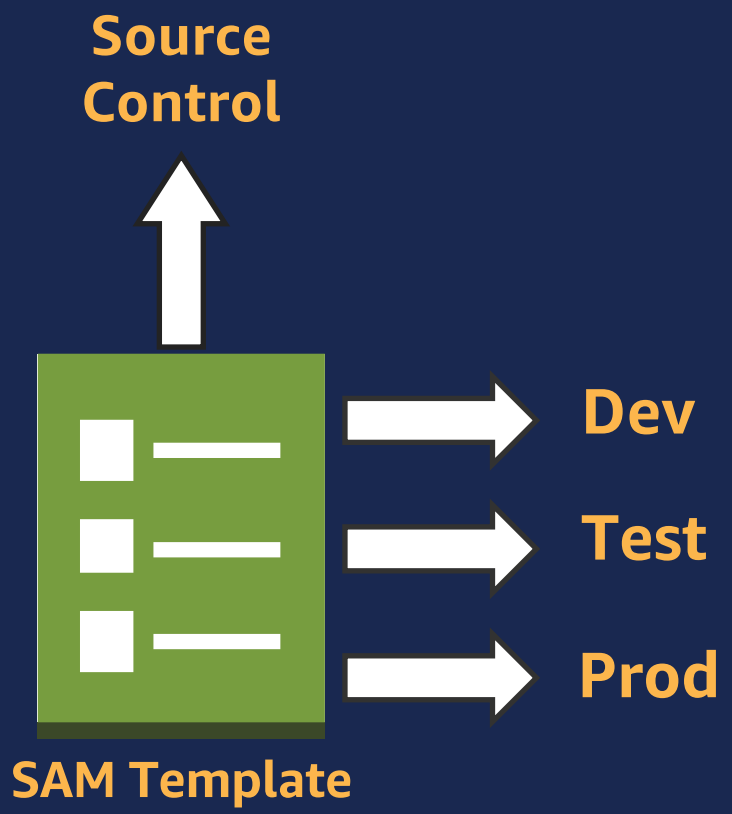
A Brief recap…

SAM template capabilities

- Mix with other AWS CloudFormation resources in the same template
  - e.g., Amazon S3, Amazon Kinesis, AWS Step Functions
- Supports use of parameters, mappings, outputs, etc.
- Supports intrinsic functions
- Can use ImportValue
  (exceptions for RestApiId, Policies, StageName attributes)
- YAML or JSON

SAM best practices

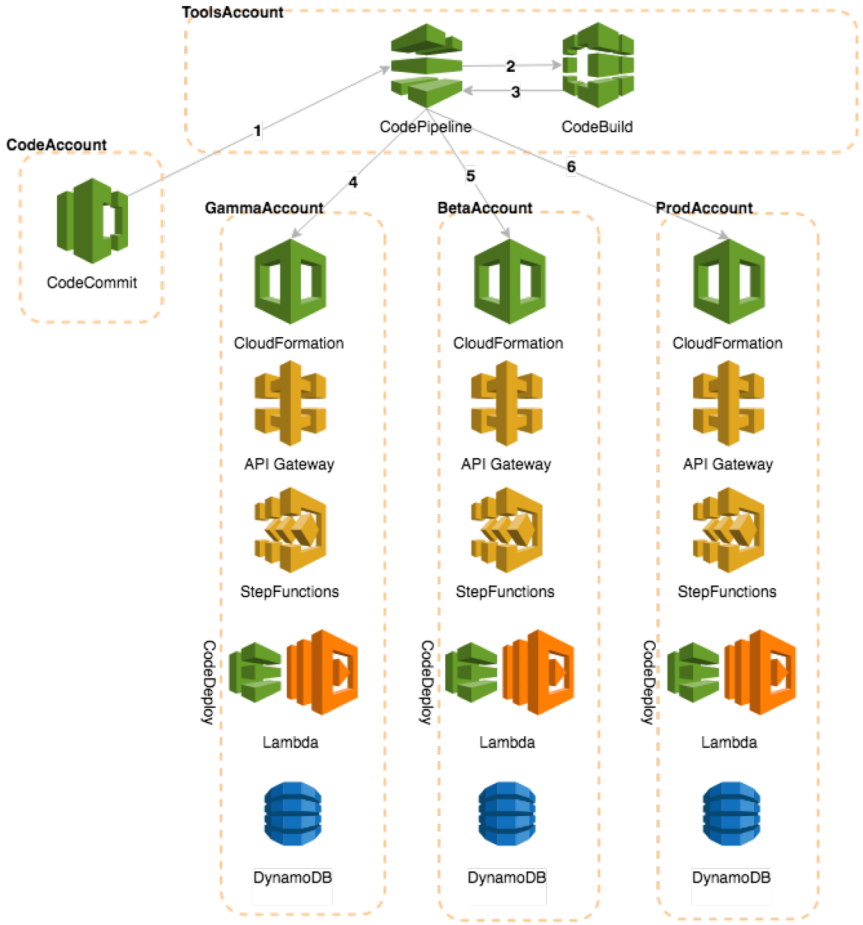
- Use Parameters and Mappings when possible to build dynamic templates based on user inputs and pseudo parameters such as `AWS::Region`
- Use the Globals section to simplify templates
- Use ExportValue & ImportValue to share resource information across stacks
- Build out multiple environments, such as for Development, Test, Production using the same template, even across accounts



SAM best practices

- Build function handlers as independent Lambda function files or binaries, unless they share code
    - Or build language-specific modules to share common code
- Manage functions with the same event source in the same repository and SAM template
- Locally lint and validate your SAM template before committing
    - Do it again in your CI/CD process

SAM best practices



https://aws.amazon.com/blogs/devops/aws-building-a-secure-cross-account-continuous-delivery-pipeline/

Lambda environment variables

- Key-value pairs dynamically passed to your function
- Available via standard environment variable APIs
  - Node.js: `process.env`
  - Python: `os.environ`
- Optionally encrypt via AWS Key Management Service (AWS KMS)
  - Specify IAM roles that have access to the keys to decrypt the information
- Useful for creating environments per stage

Amazon API Gateway stage variables

- Stage variables act like environment variables
- Use stage variables to store configuration values
- Stage variables are available in the `$context` object
- Values are accessible from most fields in API Gateway
  - Lambda function ARN
  - HTTP endpoint
  - Custom authorizer function name
  - Parameter mappings

Lambda and API Gateway variables + SAM

```
Parameters:
  Environment:
    Description: Environment of this stack
    Type: String
    Default: Dev
    AllowedValues:
      - Dev
      - Test
      - Prod
Mappings:
  IncludeNewFeatures:
    Dev:
      NewFeatureFlag: true
    Test:
      NewFeatureFlag: true
    Prod:
      NewFeatureFlag: false
```

```
# Lambda
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    ...
  Environment:
    Variables:
     ENVIRONMENT: !Ref: Environment
     NEW_FEATURES: !FindInMap [IncludeNewFeatures,
!Ref Environment, NewFeatureFlag]
…
#API Gateway
MyApiGatewayApi:
  Type: AWS::Serverless::Api
   Properties:
     ...
   Variables:
    ENVIRONMENT: !Ref: Environment
```

Lambda and API Gateway variables + SAM

```
Parameters:
  Environment:
    Description: Environment of this stack
    Type: String
    Default: Dev
    AllowedValues:
      - Dev
      - Test
      - Prod
Mappings:
  IncludeNewFeatures:
    Dev:
      NewFeatureFlag: true
    Test:
      NewFeatureFlag: true
    Prod:
      NewFeatureFlag: false
```

```
# Lambda
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    ...
  Environment:
   Variables:
    ENVIRONMENT: !Ref: Environment
    NEW_FEATURES: !FindInMap [IncludeNewFeatures,
!Ref Environment, NewFeatureFlag]
…
#API Gateway
MyApiGatewayApi:
  Type: AWS::Serverless::Api
   Properties:
     ...
   Variables:
    ENVIRONMENT: !Ref: Environment
```

SAM + safe deployments

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler

      Runtime: nodejs8.10

      AutoPublishAlias: !Ref ENVIRONMENT
      DeploymentPreference:
        Type: Linear10PercentEvery10Minutes
        Alarms:
          # A list of alarms that you want to monitor
          - !Ref AliasErrorMetricGreaterThanZeroAlarm
          - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
        Hooks:
          # Validation functions run before & after traffic shifting
          PreTraffic: !Ref PreTrafficLambdaFunction
          PostTraffic: !Ref PostTrafficLambdaFunction
```

SAM + safe deployments

```
MyLambdaFunction:
  Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler

      Runtime: nodejs8.10

      AutoPublishAlias: !Ref ENVIRONMENT
      DeploymentPreference:
        Type: Linear10PercentEvery10Minutes
        Alarms:
          # A list of alarms that you want to monitor
          - !Ref AliasErrorMetricGreaterThanZeroAlarm
          - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
        Hooks:
          # Validation functions run before & after traffic shifting
          PreTraffic: !Ref PreTrafficLambdaFunction
          PostTraffic: !Ref PostTrafficLambdaFunction
```

Deployments with AWS SAM

- Lambda alias traffic shifting enables canaries and blue/green deployments
- Automatic rollback based on CloudWatch metrics/alarms
- Pre/post-traffic triggers can integrate with other services (or call other Lambda functions)

Deployment preference type

Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10PercentEvery10Minutes
Linear10PercentEvery1Minute
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

SAM globals

```
Globals:
  Function:
    Runtime: nodejs8.10
    Handler: index.handler
    Timeout: 10
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
      Properties:
        ...
  MyLongRunningFunction:
    Type: AWS::Serverless::Function
      Properties:
        Timeout: 30
        ...
```

Customer scenario…

Lambda Role

- Build secure functions using the least privilege principle to define policies
- Use SAM supported policy templates

SQSPollerPolicy
LambdaInvokePolicy
CloudWatchPutMetricPolicy
DynamoDBCrudPolicy
SQSSendMessagePolicy
SNSPublishMessagePolicy

VPCAccessPolicy
SNSCrudPolicy
KMSDecryptPolicy
SSMParameterReadPolicy
AWSSecretsManagerGetSecretValuePolicy
…

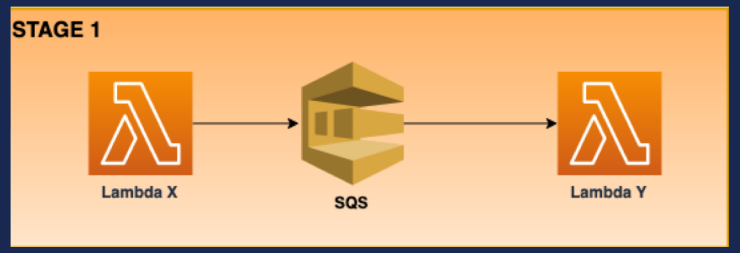
https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-policy-template-list.html

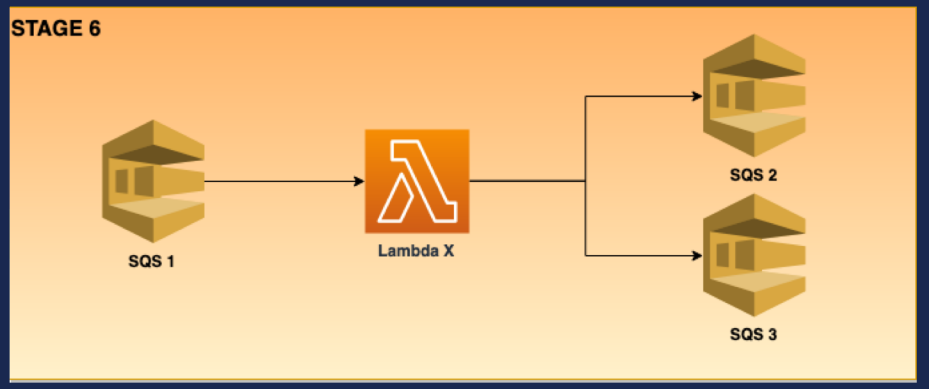
SAM policy templates

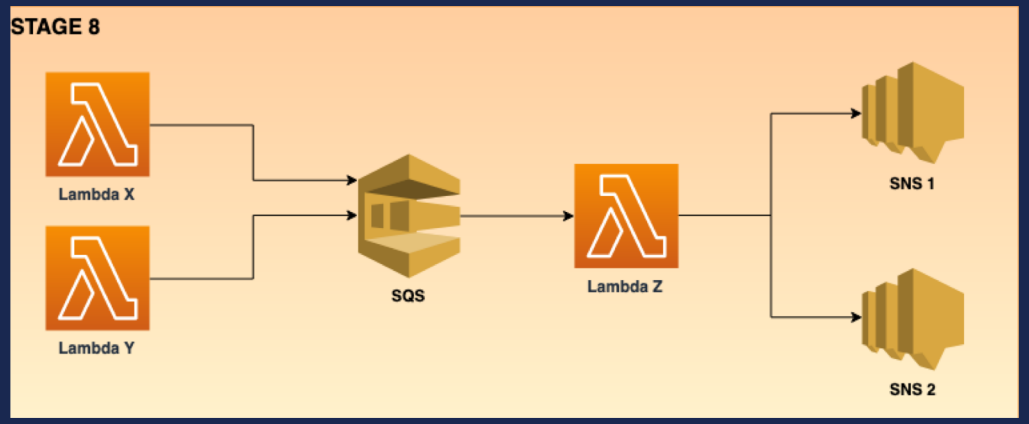
```
MyLambdaFunction:
  Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10

      Policies:
        # CRUD permissions for one table
        - DynamoDBCrudPolicy:
            TableName: !Ref MyTable

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

SQS

- Manage functions with the same event source in the same repository and SAM template

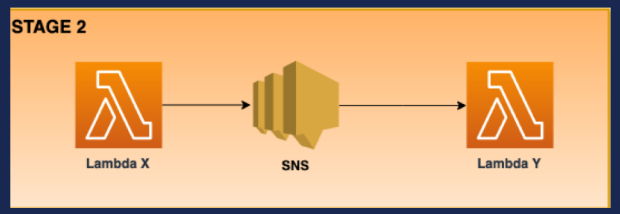


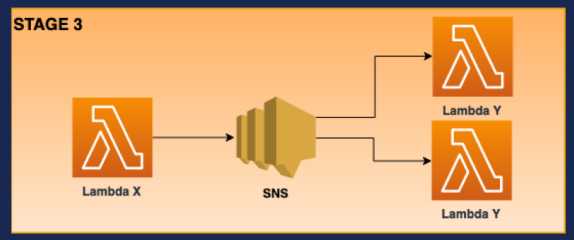


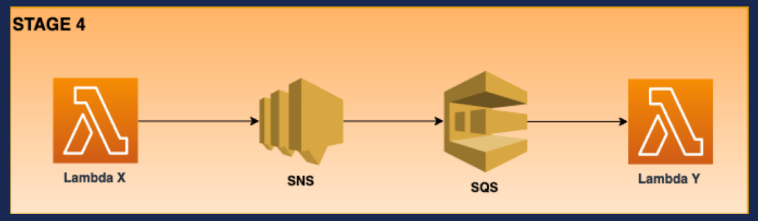


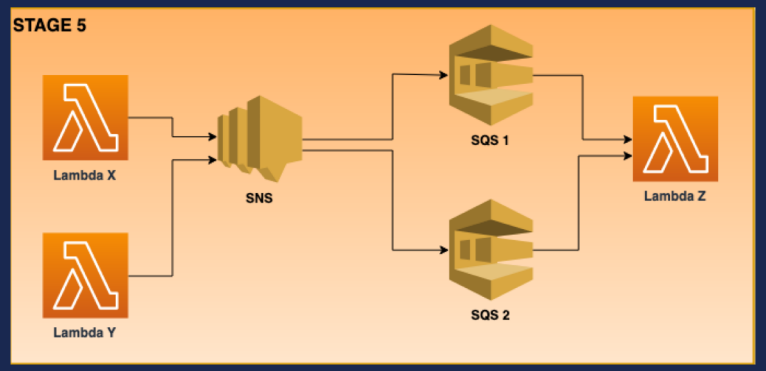
SNS

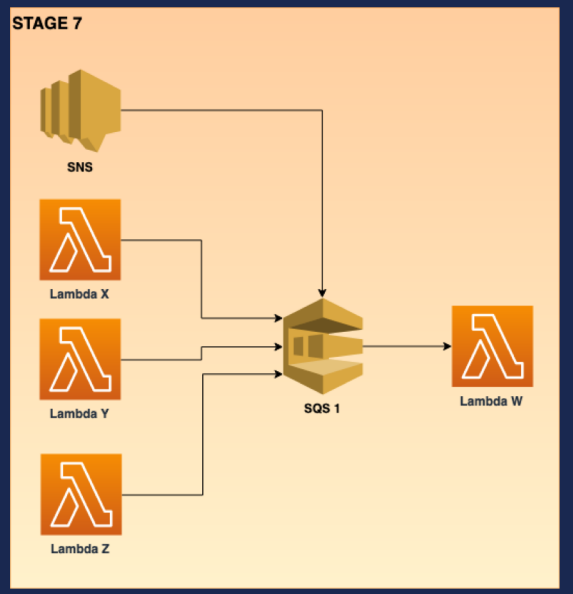
- Manage functions with the same event source in the same repository and SAM template











https://aws.amazon.com/blogs/compute/introducing-amazon-eventbridge-schema-registry-and-discovery-in-preview/

API Gateway

- Manage functions with the same event source in the same repository and SAM template
- Mix with other AWS CloudFormation resources in the same template
- Use SAM resource AWS::Serverless::Api (OpenApi definition)

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-resource-api.html

https://github.com/awslabs/serverless-application-model/blob/develop/examples/2016-10-31/api_lambda_token_auth/template.yaml

AWS SAM CLI

- CLI tool for local building, validating, testing of serverless apps
- Works with Lambda functions and “proxy-style” APIs
- Response object and function logs available on your local machine
- Mimic Lambda’s execution environment with open- source docker-lambda images
  - Emulates timeout, memory limits, runtimes

https://github.com/awslabs/aws-sam-cli

Thank you!

Luiz Yanai *lyanai@amazon.com*