# Deploying Updates to Lambda with SAM and CodeDeploy

If you are running a serverless application that uses AWS Lambda, you might need to address additional considerations when you deploy updates to your Lambda functions. As an example, dependencies between service resources could cause issues for your serverless deployment if the various resources that your function needs aren't created in the correct order.

To help with serverless deployments, AWS offers the *AWS Serverless Application Model (AWS SAM),* which is an open-source framework that you can use as a scaffold for organizing your serverless resources. When you create or update an AWS SAM template, you use YAML and a shorthand syntax specific to AWS SAM to model your application. With AWS SAM, you can use code to define your application's functions and other resources, such as APIs or event-source mappings. When you deploy your application, AWS SAM expands its syntax into AWS CloudFormation syntax. By turning your serverless infrastructure into code, you can build and deploy your serverless applications more quickly and reliably.

Say that you have been working on a new version of a Lambda function for your application. The updated function has passed all your tests, and you're now ready to deploy the new version. After deployment, you want production traffic to gradually shift from the older version of the function to the new version so you can monitor how it performs.

You could deploy the new version of the function (and all its dependencies) through the console or the AWS Command Line Interface (AWS CLI), but doing so can be a very manual process. Instead of going the manual route, you could deploy the DevOps way, using AWS SAM and CodeDeploy to both deploy the new version *and* automatically shift production traffic to it.

To do so, edit the AWS SAM template to configure the necessary CodeDeploy settings. By including the `AutoPublishAlias: live` setting in the AWS SAM definition for your Lambda function, CodeDeploy deploys a new version of the Lambda function. After it deploys the function, it also creates an alias called *live* that points to the new version, which CodeDeploy will use to shift traffic between the two versions.

To specify which deployment model you want to use, add a `DeploymentPreference` section to your template's function definition. The deployment options include:

- *Canary* - This option shifts traffic in two increments spaced by an interval (in minutes). You can use predefined options to specify the percentage of traffic shifted in the first increment and the duration of the interval before CodeDeploy shifts the remaining traffic. For example, `Canary10Percent30Minutes` will send 10 percent of traffic to the new version and 30 minutes later, it will complete the deployment by sending all traffic to the new version.

- *Linear* - This option shifts traffic in equal increments spaced by an equal number of minutes. You can use predefined options to specify the percentage of traffic shifted in the increments, and the duration of the interval between each increment. For example, `Linear10PercentEvery10Minutes` will continuously shift 10 percent of traffic to the new version every 10 minutes, until 100 percent of the traffic gets sent to the new version.
- *All-at-once* - This option shifts all traffic from the old version to the new version.

You can also use the `DeploymentPreference` section to configure additional deployment resources. Examples include Amazon CloudWatch alarms that monitor for deployment errors, or hooks that test whether the Lambda function is performing as expected before or after any traffic shifting occurs.

Here's an example AWS SAM template that defines the deployment for a Lambda function:

```
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
```

With your CodeDeploy settings in the AWS SAM template, you can deploy your new version (for example, by using the AWS SAM CLI). To see the application traffic shift from the old version and the new one as you specified in real time, you can use either the CodeDeploy console or the Lambda console. You could also perform tests or view logs to check whether the new deployment is working as expected. And if you need to update your Lambda function in the future, you can apply DevOps practices and use AWS SAM and CodeDeploy to redeploy and monitor your updates in an automated, repeatable, and reliable way.

**Resources**

- AWS SAM Documentation

- AWS CodeDeploy tutorial: Deploy an updated Lambda function with CodeDeploy and the AWS Serverless Application Model
- Implementing safe AWS Lambda deployments with AWS CodeDeploy