Deploying Serverless Application Using SAM and CodeStar

By Xiang Shen

AWS CodeStar was released in April 2017 and it is changing the way development teams deliver software in AWS. It provides a unified user interface, enabling you to easily manage all your development activities in one place.

With AWS CodeStar you can now automatically create entire environments for your application and all of its associated AWS resources. Thanks to its simplicity, you can create efficient development workflows that will be able to build, test, and release software on AWS much faster than before. Something that you may find helpful is this great blog post by Tara Walker, which lists the key benefits of CodeStar and guides you in setting up a sample web application using an AWS Lambda Node.js project template.

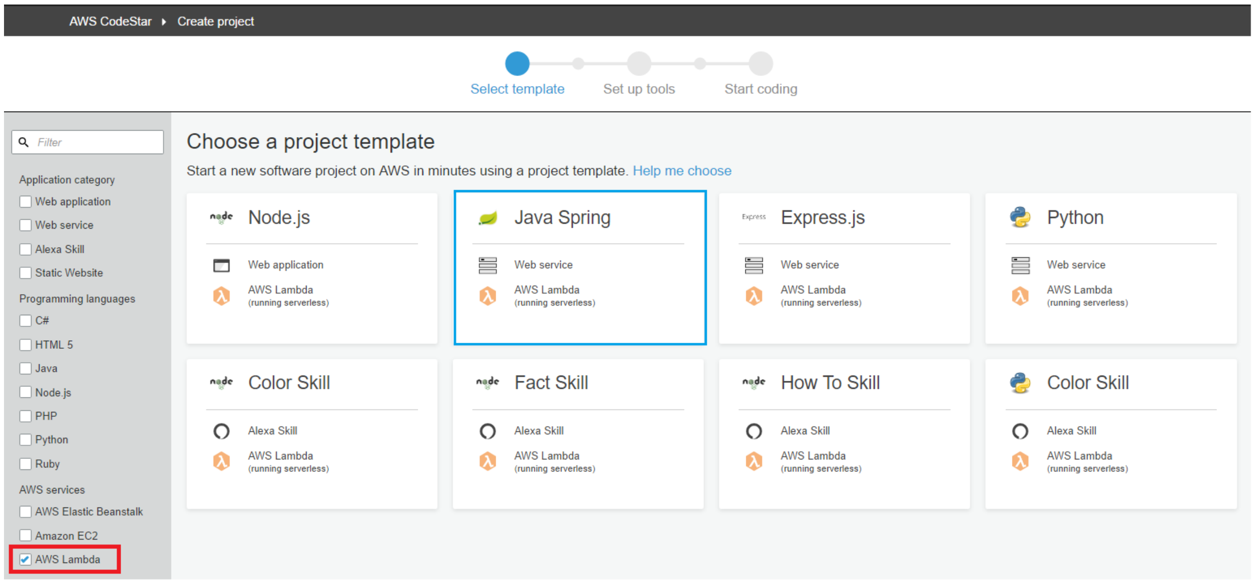
If you have ever worked with a serverless application based on AWS Lambda, you probably know it takes quite a bit of effort to configure the application and automate the packaging and deployment process especially when using multiple AWS resources.

The first time I tried the Node.js example myself, I was amazed how fast and simple the process was – in a few minutes my web application was up and running with all the necessary resources correctly configured, including IAM roles, a CodeCommit repository, and a pipeline in CodePipeline. In addition it took advantage of AWS Serverless Application Model (SAM) for packaging and deployment.

But what about the other languages? For the remainder of this blog I will demonstrate how to setup a CodeStar project using an AWS Lambda Java template. I will then use SAM with AWS CodePipeline to build and deploy the application.

Create the AWS CodeStar Java Project

The first thing we need to do is to create the project. After we login to our AWS account, go to the AWS CodeStar console, and select “Create a new project”. We can use the project filter in the left column and find all the available AWS Lambda templates, then choose the ‘Java Spring’ template as showing below:



Choose a project template

Start a new software project on AWS in minutes using a project template. Help me choose

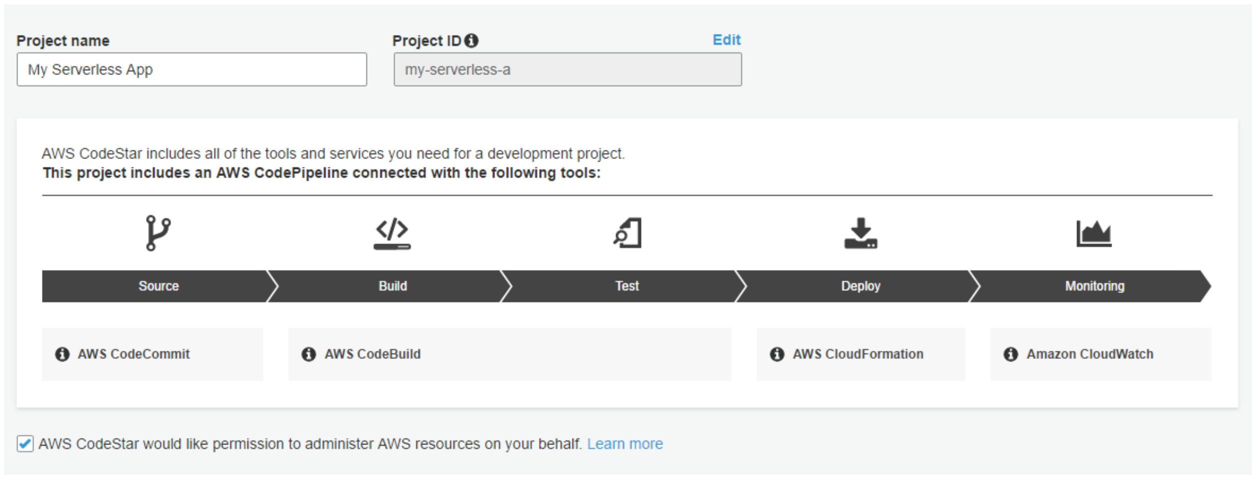
Application category
- [ ] Web application
- [ ] Web service
- [ ] Alexa Skill
- [ ] Static Website

Programming languages
- [ ] C#
- [ ] HTML 5
- [ ] Java
- [ ] Node.js
- [ ] PHP
- [ ] Python
- [ ] Ruby

AWS services
- [ ] AWS Elastic Beanstalk
- [ ] Amazon EC2
- [✓] AWS Lambda

Node.js Web application AWS Lambda (running serverless)	Java Spring Web service AWS Lambda (running serverless)	Express.js Web service AWS Lambda (running serverless)	Python Web service AWS Lambda (running serverless)
Color Skill Alexa Skill AWS Lambda (running serverless)	Fact Skill Alexa Skill AWS Lambda (running serverless)	How To Skill Alexa Skill AWS Lambda (running serverless)	Color Skill Alexa Skill AWS Lambda (running serverless)

In the following page, we need to give a name for our project and AWS CodeStar displays the tools and services it will connect for us.



Project name: My Serverless App

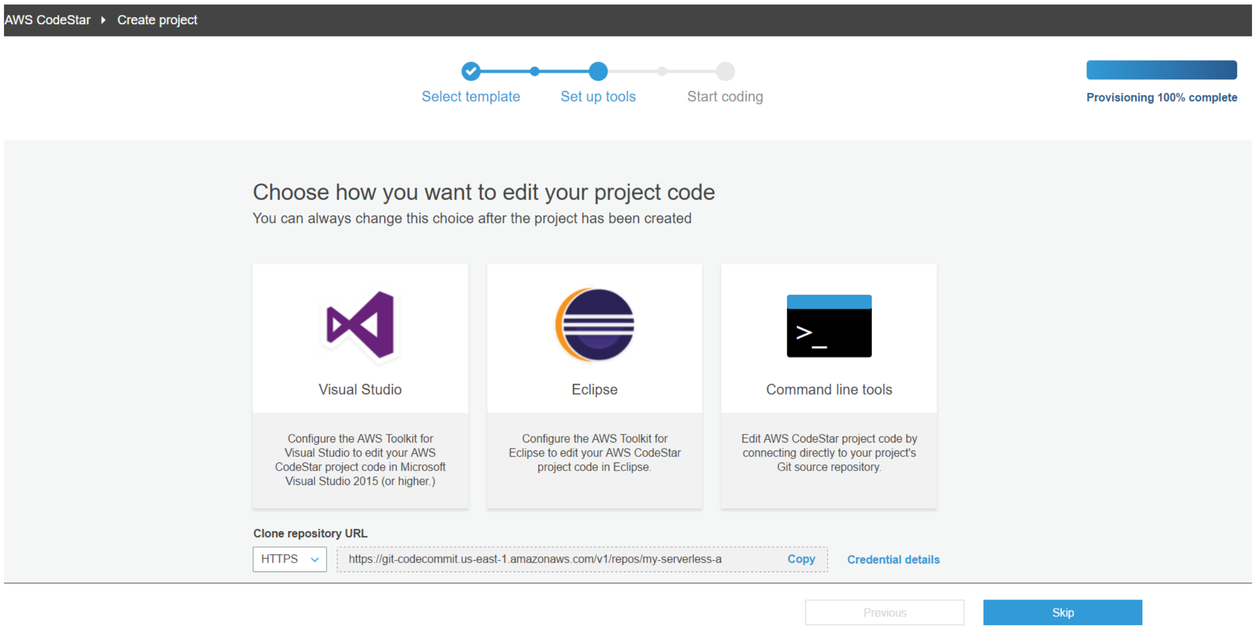
Project ID: my-serverless-a

AWS CodeStar includes all of the tools and services you need for a development project.
This project includes an AWS CodePipeline connected with the following tools:

Source	Build	Test	Deploy	Monitoring
AWS CodeCommit	AWS CodeBuild		AWS CloudFormation	Amazon CloudWatch

[✓] AWS CodeStar would like permission to administer AWS resources on your behalf. Learn more

Afterwards click the ‘Create Project’ button, AWS CodeStar will start provisioning the resources and creating the project. Meanwhile, it will also help us configure our local development environment.


AWS CodeStar ▸ Create project

Choose how you want to edit your project code

You can always change this choice after the project has been created

Configure the AWS Toolkit for Visual Studio to edit your AWS CodeStar project code in Microsoft Visual Studio 2015 (or higher.)

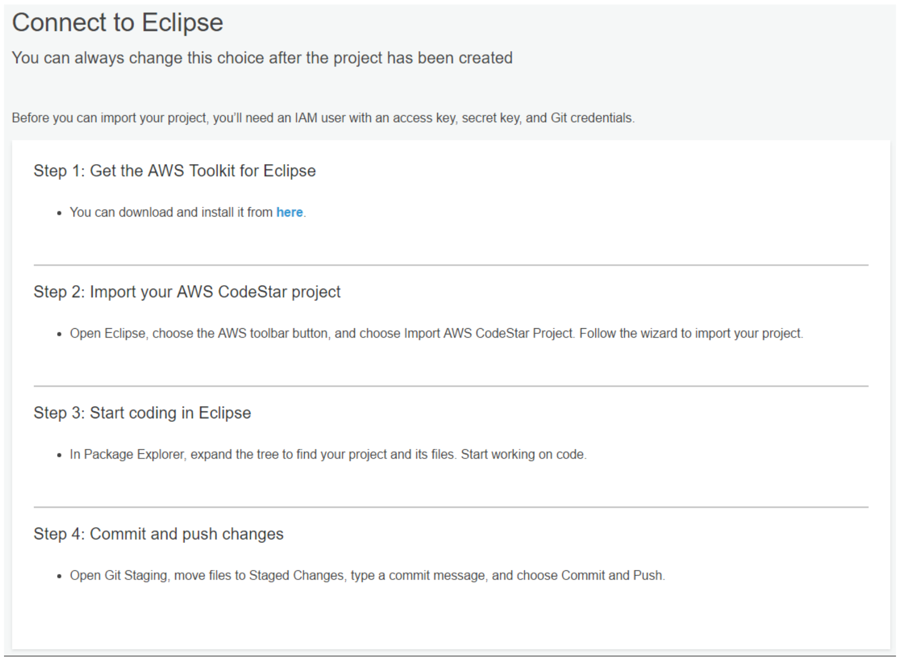
Configure the AWS Toolkit for Eclipse to edit your AWS CodeStar project code in Eclipse.

Edit AWS CodeStar project code by connecting directly to your project's Git source repository.

Clone repository URL

HTTPS https://git-codecommit.us-east-1.amazonaws.com/v1/repos/my-serverless-a

Since I am an Eclipse user, after the provision is 100% complete I click the Eclipse box and follow the steps to setup my Eclipse project.


Connect to Eclipse

You can always change this choice after the project has been created

Before you can import your project, you'll need an IAM user with an access key, secret key, and Git credentials.

Step 1: Get the AWS Toolkit for Eclipse

- You can download and install it from here.

Step 2: Import your AWS CodeStar project

- Open Eclipse, choose the AWS toolbar button, and choose Import AWS CodeStar Project. Follow the wizard to import your project.

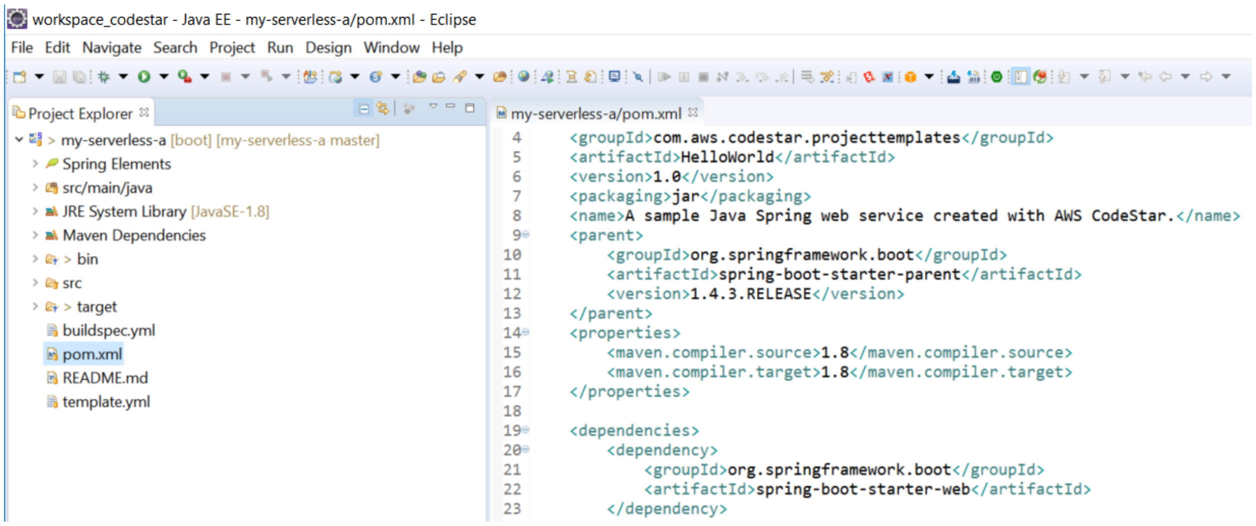
Step 3: Start coding in Eclipse

- In Package Explorer, expand the tree to find your project and its files. Start working on code.

Step 4: Commit and push changes

- Open Git Staging, move files to Staged Changes, type a commit message, and choose Commit and Push.

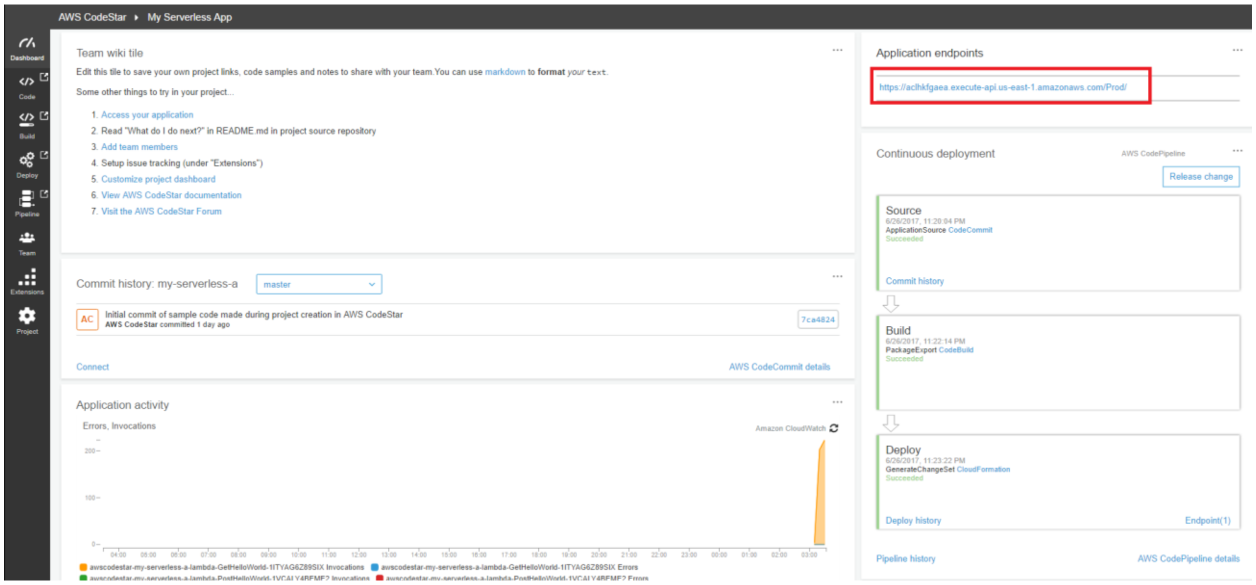
After the project is configured in Eclipse, my Eclipse IDE would look like this:


workspace_codestar - Java EE - my-serverless-a/pom.xml - Eclipse
File Edit Navigate Search Project Run Design Window Help
Project Explorer
my-serverless-a [boot] [my-serverless-a master]
Spring Elements
src/main/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
bin
src
target
buildspec.yml
pom.xml
README.md
template.yml
my-serverless-a/pom.xml

```
    <groupId>com.aws.codestar.projecttemplates</groupId>
    <artifactId>HelloWorld</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <name>A sample Java Spring web service created with AWS CodeStar.</name>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.3.RELEASE</version>
    </parent>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

And if I go back to the AWS CodeStar dashboard, I will have a screen similar to the following one.



Team wiki tile

Edit this tile to save your own project links, code samples and notes to share with your team. You can use markdown to **format** *your* `text`.

Some other things to try in your project...

1. Access your application
2. Read "What do I do next?" in README.md in project source repository
3. Add team members
4. Setup issue tracking (under "Extensions")
5. Customize project dashboard
6. View AWS CodeStar documentation
7. Visit the AWS CodeStar Forum

Commit history: my-serverless-a master

AC Initial commit of sample code made during project creation in AWS CodeStar
AWS CodeStar committed 1 day ago 7ca4824

Connect AWS CodeCommit details

Application activity

Errors, Invocations Amazon CloudWatch

🟧 awscodestar-my-serverless-a-lambda-GetHelloWorld-1ITYAG6Z89SIX Invocations 🟦 awscodestar-my-serverless-a-lambda-GetHelloWorld-1ITYAG6Z89SIX Errors
🟩 awscodestar-my-serverless-a-lambda-PostHelloWorld-1VCALY4BFME2 Invocations 🟥 awscodestar-my-serverless-a-lambda-PostHelloWorld-1VCALY4BFME2 Errors

Application endpoints

https://aclhkfgaea.execute-api.us-east-1.amazonaws.com/Prod/

Continuous deployment AWS CodePipeline

Release change

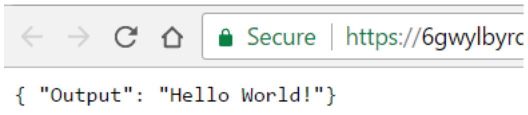
Source
6/26/2017, 11:20:04 PM
ApplicationSource CodeCommit
Succeeded
Commit history

Build
6/26/2017, 11:22:14 PM
PackageExport CodeBuild
Succeeded

Deploy
6/26/2017, 11:23:22 PM
GenerateChangeSet CloudFormation
Succeeded
Deploy history Endpoint(1)

Pipeline history AWS CodePipeline details

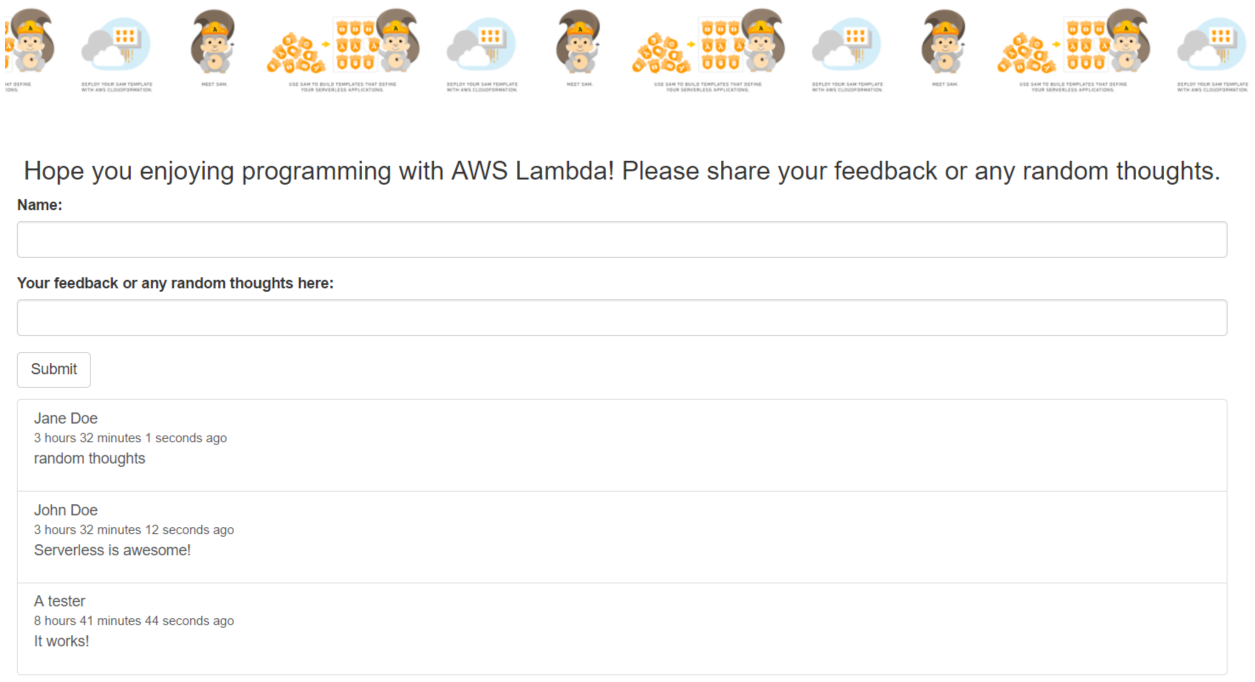
When I click the URL in the 'Application endpoint' field I get a JSON 'Hello World!'



```
{ "Output": "Hello World!"}
```

Modify the AWS CodeStar Java Project

So far it is nice and easy, within a few minutes I have everything I need to start working on my demo application. What I would like to create for my demo application is a simple web app to allow users to leave their feedback and thoughts like the one shown below:


Hope you enjoying programming with AWS Lambda! Please share your feedback or any random thoughts.

Name:

Your feedback or any random thoughts here:

Submit

Jane Doe
3 hours 32 minutes 1 seconds ago
random thoughts

John Doe
3 hours 32 minutes 12 seconds ago
Serverless is awesome!

A tester
8 hours 41 minutes 44 seconds ago
It works!

It would run on AWS Lambda and use DynamoDB to store data. I would also like to enable AWS X-Ray in my testing environment for debugging and performance analysis.

So there are a couple of things that I need to do. The first is to update the application dependencies. It is one of the best practices that an AWS Lambda application only includes the needed dependencies. Therefore, instead of using the all-inclusive AWS SDK Jar, I add the following Jars to my Maven POM file:

```
<dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-dynamodb</artifactId>
        <version>1.11.143</version>
</dependency>
<dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-xray</artifactId>
        <version>1.11.144</version>
</dependency>
<dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-xray-recorder-sdk-core</artifactId>
        <version>1.1.1</version>
</dependency>
<dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-xray-recorder-sdk-aws-sdk</artifactId>
        <version>1.1.1</version>
</dependency>
```

```
		<dependency>
			<groupId>com.amazonaws</groupId>
			<artifactId>aws-xray-recorder-sdk-aws-sdk-Instrumentor</artifactId>
			<version>1.1.1</version>
		</dependency>
```

The next step is to add my source code. An easy way is just to replace the whole 'src' folder with my own.

Since AWS CodeBuild will be used for building the application, the buildspec.yml file needs to be updated like so:

```
version: 0.1
phases:
  build:
    commands:
      - echo Entering build phase...
      - echo Build started on `date`
      - pip install --upgrade awscli               # To get the latest CLI.
      - mvn package shade:shade                    # Use Maven to generate the shade Jar
      - jar xf target/Feedback*.jar                # Preparation for SAM
      - rm -rf target src buildspec.yml pom.xml    # Remove unnecessary files
      # The output file is not really a JSON file. And if you change it here,
      # you need update the artifacts section below and the CodePipeline stage too.
      - aws cloudformation package --template template.yml --s3-bucket $S3_BUCKET --output-template
template-export.json
artifacts:
  files:
    - template-export.json
```

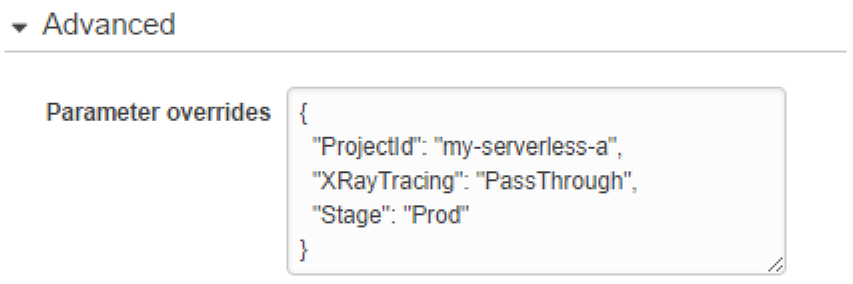
For our project a default SAM template is generated by AWS CodeStar, we need to update that file as well to include additional parameters, a Lambda function, a DynamoDB table and a couple of APIs.

```
AWSTemplateFormatVersion: 2010-09-09
Transform:
- AWS::Serverless-2016-10-31   # Serverless transformation
- AWS::CodeStar                # CodeStar transformation

Parameters:
  ProjectId:
    Type: String
    Description: AWS CodeStar projectID used to associate new resources to team members
  XRayTracing:
    Type: String
    Description: Specifies X-Ray tracing mode. Accepted values are Active and PassThrough
  Stage:
    Type: String
    Description: Specifies stage name
Resources:
  FeedbackFunc:
    Type: AWS::Serverless::Function
    Properties:
```

```yaml
      Handler: com.aws.codestar.demo.handler.FeedbackHandler
      Runtime: java8
      MemorySize: 1024              # Lambda function memory size in MB
      Timeout: 15                   # Lambda function timeout in seconds
      Tracing: !Ref XRayTracing     # Enable X-Ray tracing for the function
      Role:
        Fn::ImportValue:
          !Join ['-', [!Ref 'ProjectId', !Ref 'AWS::Region', 'LambdaTrustRole']]
      Events:         # depending on the requirements and design, each API could map to its own Lambda
        GetRoot:
          Type: Api
          Properties:
            Path: /
            Method: get
        GetEvent:
          Type: Api
          Properties:
            Path: /feedback
            Method: get
        PostEvent:
          Type: Api
          Properties:
            Path: /feedback
            Method: post
      Environment:
        Variables:
          TABLE_NAME: !Ref FeedbackTable   # Reference of the DynamoDB table
          XRAY_TRACING: !Ref XRayTracing   # Reference of the X-Ray tracing flag
  FeedbackTable:
    Type: AWS::DynamoDB::Table             # SimpleTable can be used if we don't care about the other props
    Properties:
      TableName: !Sub Feedback-${Stage}    # Define the table name. Currently SimpleTable doesn't support it but it's coming soon
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
Outputs:
  ApiUrl:                                  # Need this output so our testing action can use it
    Description: URL for application
    Value: !Sub 'https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/Prod'
    Export:
      Name: !Sub 'ApiUrl-${Stage}'
```

We also need to update the ‘GenerateChangeSet’ action in the Deploy stage of our pipeline. This enables us to provide values for our parameters and they can be used in SAM and our function.



```
{
  "ProjectId": "my-serverless-a",
  "XRayTracing": "PassThrough",
  "Stage": "Prod"
}
```

Currently, when using some AWS resources and adding more stages in the pipeline, additional permissions need to be granted to the IAM roles generated by AWS CodeStar. For demo purposes only, the following IAM custom policy is created and attached to all the AWS CodeStar worker roles. Please note that the CloudFormation stack and DynamoDB table names need to be updated for different projects and they will be used in the testing stage, which we are going to discuss in the next section.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStacks",
                "cloudformation:DescribeChangeSet",
                "cloudformation:CreateChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:ExecuteChangeSet"
            ],
            "Resource": "arn:aws:cloudformation:us-east-1:*:stack/awscodestar-my-serverless-a-lambda-
test/*"
        },
        {
            "Action": [
                "lambda:GetFunction",
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:*Item*",
                "dynamodb:Describe*",
                "dynamodb:ListTables",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111196147180:table/Feedback-Production",
                "arn:aws:dynamodb:us-east-1:111196147180:table/Feedback-Beta"
            ]
        },
        {
            "Action": [
                "codepipeline:PutJobSuccessResult",
                "codepipeline:PutJobFailureResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

After all the above steps have been completed, we can then commit the changes in our repo and push them. The code pipeline automatically created by CodeStar will pick up our changes almost immediately, build and deploy them. Again, within a few minutes our application would be updated as expected.

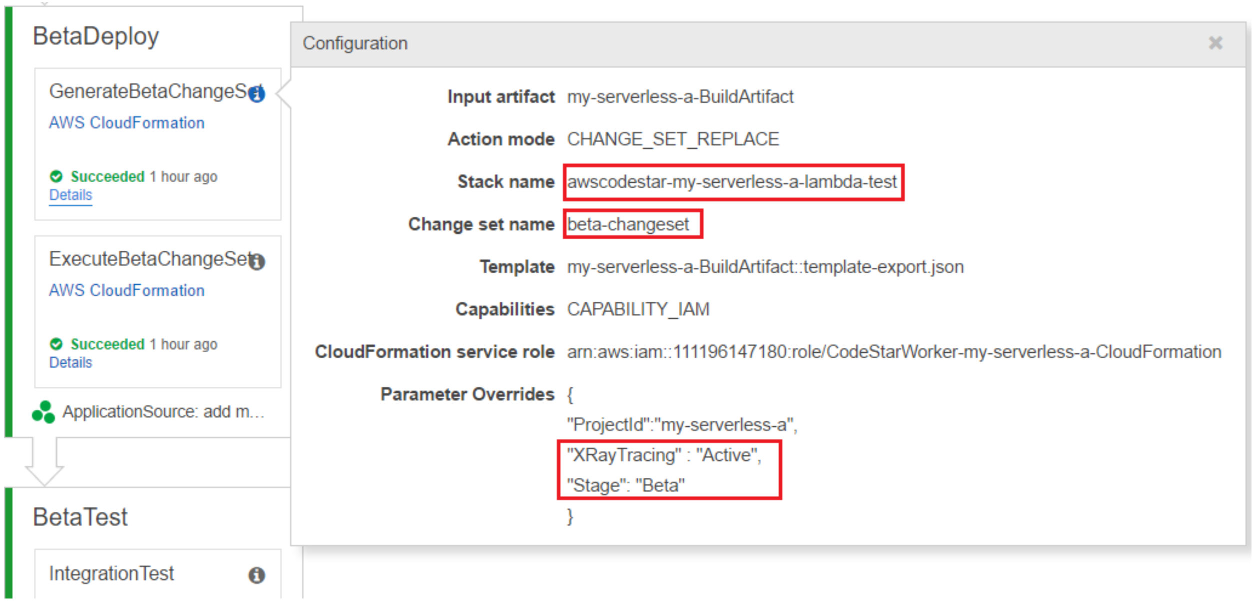
Add Testing into the Pipeline

When we build a continuous integration/deployment process, testing is crucial. Naturally, I would like to add a testing stage prior to our production deployment so we could be more confident of our code quality.

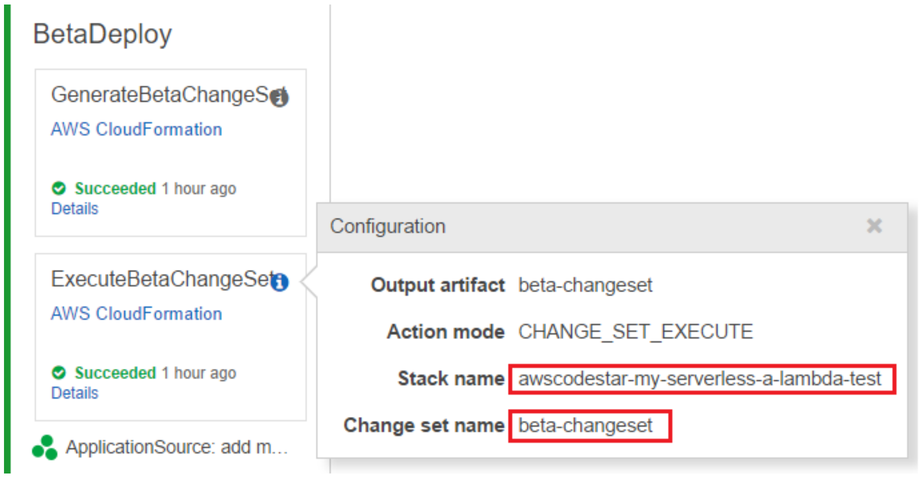
With CodeStar, we would need to create one more CloudFormation stack for a testing stage. The good news is it’s really easy when we follow the steps below:

- Create a new CodePipeline stage which will be deploying our code to a testing environment.
- Add the ‘GenerateChangeSet’ and ‘ExecuteChangeSet’ actions in the stage. We can use the default 'Deploy' stage actions as examples.
- Change the stack name so that the Change Set creates a new stack and doesn't overwrite the existing stack from the 'Deploy' stage.
- Update the Change Set name and parameters as needed.

As an example, you can see the changes in a stage called 'BetaDeploy' in my pipeline.



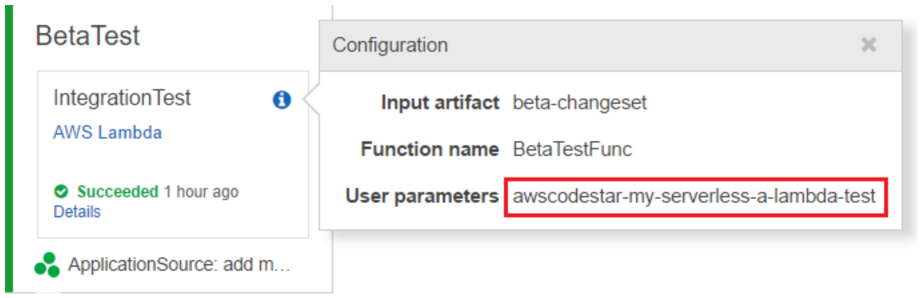
Input artifact	my-serverless-a-BuildArtifact
Action mode	CHANGE_SET_REPLACE
Stack name	awscodestar-my-serverless-a-lambda-test
Change set name	beta-changeset
Template	my-serverless-a-BuildArtifact::template-export.json
Capabilities	CAPABILITY_IAM
CloudFormation service role	arn:aws:iam::111196147180:role/CodeStarWorker-my-serverless-a-CloudFormation
Parameter Overrides	{ "ProjectId":"my-serverless-a", "XRayTracing" : "Active", "Stage": "Beta" }



Output artifact	beta-changeset
Action mode	CHANGE_SET_EXECUTE
Stack name	awscodestar-my-serverless-a-lambda-test
Change set name	beta-changeset

That covers the deployment of our code to a testing environment, which would include its own copy of Lambda function, separate DynamoDB table and API endpoint in API gateway. In terms of testing, we can add one more action in the same stage or create a new testing stage. I chose a new stage for a little bit more flexibility.

For the testing action, we can use some 3rd tools that is integrated with CodePipeline. For demo purposes, I just manually created an AWS Lambda function. It will get the URL from the previous stage and do some validation. My testing stage looks like the following:



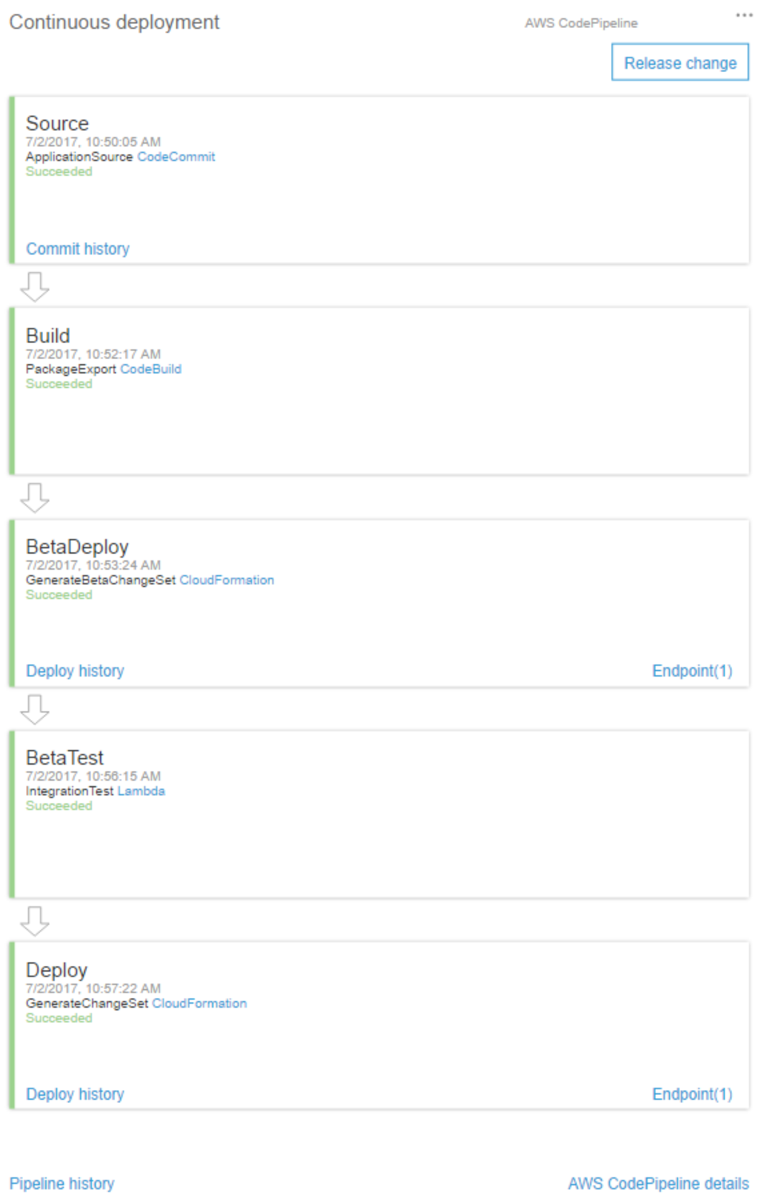
Input artifact beta-changeset
Function name BetaTestFunc
User parameters awscodestar-my-serverless-a-lambda-test

Please note that we need to pass the name of the testing CloudFormation stack as a user parameter so that our testing script will be able to get the application URL automatically.

If you prefer to include the testing Lambda function in the deployment process, you can follow the techniques described in the blog: Continuous Deployment for Serverless Applications.

A quick recap here - at this moment, we have a beautiful five-stage pipeline which will automatically build, test and deploy our changes once they are committed and pushed to our CodeCommit repository.

Yay!


Continuous deployment

AWS CodePipeline

Release change

Source
7/2/2017, 10:50:05 AM
ApplicationSource CodeCommit
Succeeded

Commit history

Build
7/2/2017, 10:52:17 AM
PackageExport CodeBuild
Succeeded

BetaDeploy
7/2/2017, 10:53:24 AM
GenerateBetaChangeSet CloudFormation
Succeeded

Deploy history Endpoint(1)

BetaTest
7/2/2017, 10:56:15 AM
IntegrationTest Lambda
Succeeded

Deploy
7/2/2017, 10:57:22 AM
GenerateChangeSet CloudFormation
Succeeded

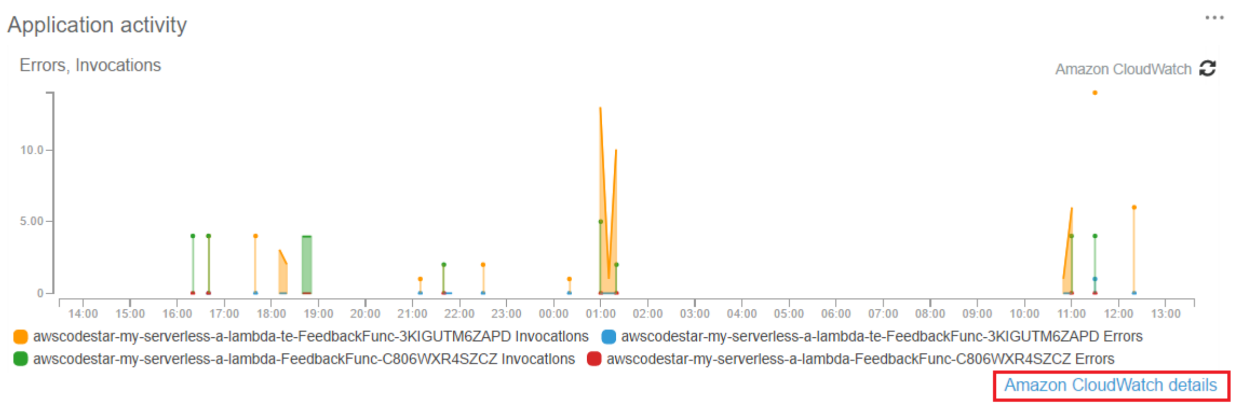
Deploy history Endpoint(1)

Pipeline history AWS CodePipeline details

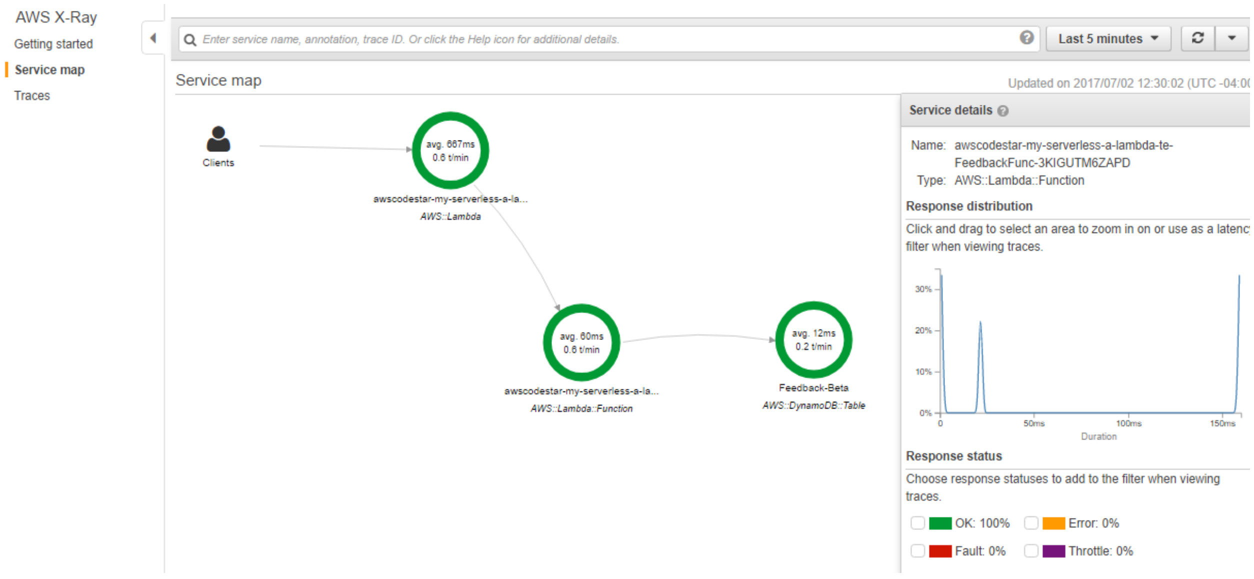
Monitoring and Performance Analysis

Even though our application is really small, we still want to monitor it and have a good understanding of its performance characteristics, especially being able to identify any performance bottlenecks.

AWS CodeStar nicely provides a tile for application activities from CloudWatch, which visualized invocations and errors of our Lambda function. If we follow the CloudWatch link in the tile, we can see more details in the CloudWatch console.



I also mentioned that AWS X-Ray is enabled for our testing stage. After we play with the testing site a little bit and then hop over to the X-Ray Console. We should be able to see a service map for our app:



Service details

Name: awscodestar-my-serverless-a-lambda-te-FeedbackFunc-3KIGUTM6ZAPD
Type: AWS::Lambda::Function

Response distribution

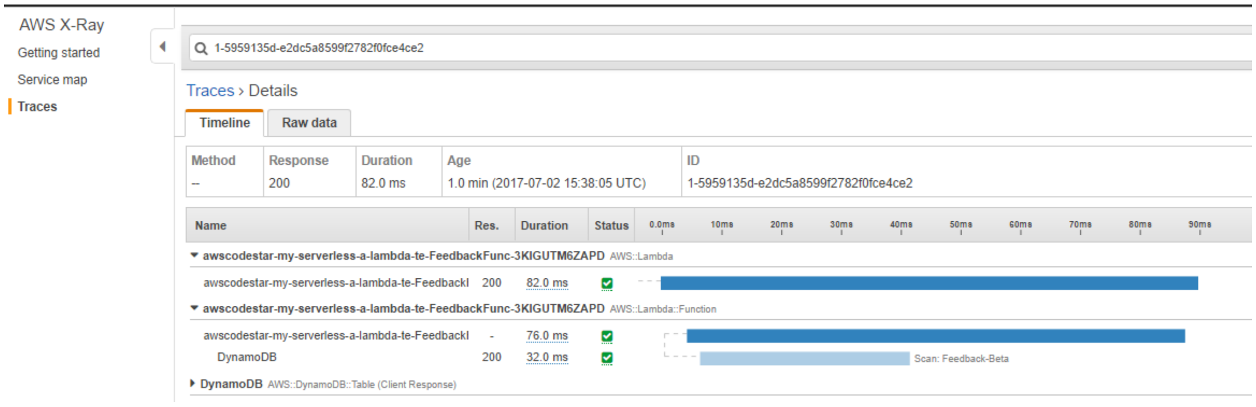
Click and drag to select an area to zoom in on or use as a latenc filter when viewing traces.

Response status

Choose response statuses to add to the filter when viewing traces.

OK: 100%　Error: 0%
Fault: 0%　Throttle: 0%

We can also drill down to each component and look at the detailed traces, for instance:



Method	Response	Duration	Age	ID
--	200	82.0 ms	1.0 min (2017-07-02 15:38:05 UTC)	1-5959135d-e2dc5a8599f2782f0fce4ce2

Name	Res.	Duration	Status
awscodestar-my-serverless-a-lambda-te-FeedbackFunc-3KIGUTM6ZAPD AWS::Lambda			
awscodestar-my-serverless-a-lambda-te-Feedbackl	200	82.0 ms	✓
awscodestar-my-serverless-a-lambda-te-FeedbackFunc-3KIGUTM6ZAPD AWS::Lambda::Function			
awscodestar-my-serverless-a-lambda-te-Feedbackl	-	76.0 ms	✓
DynamoDB	200	32.0 ms	✓
DynamoDB AWS::DynamoDB::Table (Client Response)			

AWS X-Ray is a very useful tool for performance analysis and you can read more about it in blogs: AWS X-Ray – See Inside of Your Distributed Application, AWS Lambda Support for AWS X-Ray and our documentation.

Summary

As you can see from this blog, AWS CodeStar, SAM and the other tools will significantly speed up your development process and let you focus on your application.

You can get started using the AWS CodeStar service for developing new software projects on AWS today. Learn more by reviewing the AWS CodeStar product page, the AWS CodeStar user guide documentation, and the Serverless Application Model guide.

I'm always excited to use these services and tools because they really save me time and effort for my projects. I look forward to more features and capabilities and hope you can enjoy them as well.

If you are interested, you can download the code for the demo app at <place holder>. And if you are interested in watching a live demo, you can find a similar one that I did at our DC summit.

TAGS: CodeStar, CI/CD, SAM, X-Ray, Java