Exercise 06.1:CodePipeline : Using CodePipeline with Lambda

- **Reference:** https://docs.aws.amazon.com/codepipeline/latest/userguide/actions-invoke-lambda-function.html

Step 1: Start your EC2 instances with tags Dev and Production
Step 2: Create a Lambda function
Name: Mylambda-codepipeline
Runtime: Nodejs(default)
Permission: New role with basic permissions
Click Create function.
Step 3: Go to IAM and search for the lambda role created with your lambda function in the above step and click attach policies.
Step 4: Choose Permissions tab, and then choose Add online policy. Choose the JSON tab, and then paste the following policy into the field to allow the lambda function to change status in codepipeline

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "logs:*"
   ],
   "Effect": "Allow",
   "Resource": "arn:aws:logs:*:*:*"
  },
  {
   "Action": [
    "codepipeline:PutJobSuccessResult",
    "codepipeline:PutJobFailureResult"
    ],
    "Effect": "Allow",
    "Resource": "*"
```

```
    }
  ]}
```

Step 5: Choose Review Policy and give a name "AccessToCodePipelinePolicy" and click Create Policy

Step 6: Go back to the lambda function and add the following code in the function

```
var assert = require('assert');
var AWS = require('aws-sdk');
var http = require('http');
exports.handler = function(event, context) {
   var codepipeline = new AWS.CodePipeline();
     // Retrieve the Job ID from the Lambda action
   var jobId = event["CodePipeline.job"].id;
      // Retrieve the value of UserParameters from the Lambda action configuration in AWS
CodePipeline, in this case a URL which will be
   // health checked by this function.
   var url = event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;
      // Notify AWS CodePipeline of a successful job
   var putJobSuccess = function(message) {
      var params = {
         jobId: jobId
      };
      codepipeline.putJobSuccessResult(params, function(err, data) {
         if(err) {
            context.fail(err);
         } else {
            context.succeed(message);
         }
      });
   };
      // Notify AWS CodePipeline of a failed job
   var putJobFailure = function(message) {
      var params = {
         jobId: jobId,
         failureDetails: {
            message: JSON.stringify(message),
            type: 'JobFailed',
            externalExecutionId: context.awsRequestId
         }
      };
      codepipeline.putJobFailureResult(params, function(err, data) {
         context.fail(message);
      });
   };
```

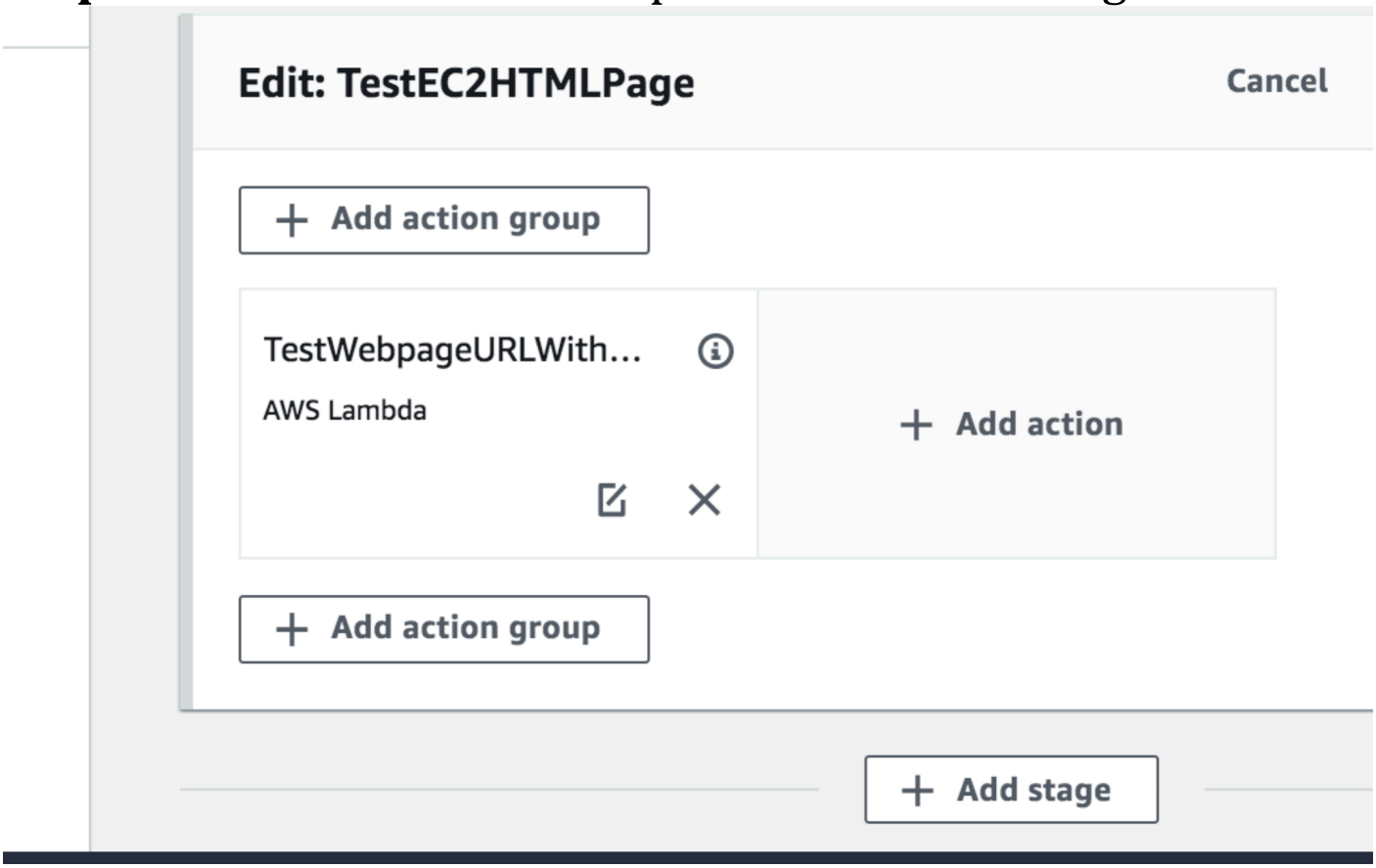
```
  // Validate the URL passed in UserParameters
  if(!url || url.indexOf('http://') === -1) {
    putJobFailure('The UserParameters field must contain a valid URL address to test, including
http:// or https://');
    return;
  }
    // Helper function to make a HTTP GET request to the page.
  // The helper will test the response and succeed or fail the job accordingly
  var getPage = function(url, callback) {
    var pageObject = {
      body: '',
      statusCode: 0,
      contains: function(search) {
        return this.body.indexOf(search) > -1;
      }
    };
    http.get(url, function(response) {
      pageObject.body = '';
      pageObject.statusCode = response.statusCode;

      response.on('data', function (chunk) {
        pageObject.body += chunk;
      });

      response.on('end', function () {
        callback(pageObject);
      });

      response.resume();
    }).on('error', function(error) {
      // Fail the job if our request failed
      putJobFailure(error);
    });
  };

  getPage(url, function(returnedPage) {
    try {
      // Check if the HTTP response has a 200 status
      assert(returnedPage.statusCode === 200);
      // Check if the page contains the text "Congratulations"
      // You can change this to check for different text, or add other tests as required
      assert(returnedPage.contains('Congratulations'));

      // Succeed the job
      putJobSuccess("Tests passed.");
    } catch (ex) {
      // If any of the assertions failed then fail the job
      putJobFailure(ex);
```

```
    }
  });
};
```

Click Save

Step 5: Go to the codepipeline and add a stage after the Deploy stage (TestEC2HTMLPage)

Step 6: Click Add Action group and add the following details
Action Name: TestWebpageURLWithLambda1
Action Provide: Lambda
InputArtifacts: Output of Build Artifacts
Function Name: Your Lambda Function
User Parameters: Give the Dev machines URL
Click Done
Step 7: Click Add Action and provide the following details



Action Name: TestWebpageURLWithLambda2
Action Provide: Lambda
InputArtifacts: Output of Build Artifacts
Function Name: Your Lambda Function
User Parameters: http://google.com

Step 8: Save the Action group and Pipeline. Click Release change
Step 9: The Test1 should be successful and Test2 should be failed