Authorising Access With IAM Policies

In this lesson, you will learn how to configure IAM to allow storage access.

WE'LL COVER THE FOLLOWING

Configuring IAM

Passing a reference to the bucket in an environment variable will let the Lambda function know where to write, but it still won't have the permission to do so. You will need to configure IAM to allow storage access. SAM hides that complexity significantly and avoids dozens of lines of boilerplate code for each function. It has convenient policy templates for popular AWS services, including S3. In this case, you can use the `S3FullAccessPolicy`, which gives a Lambda function read and write access to all objects in a bucket.

In the `ProcessFormFunction` template, specify a `Policies` property, followed by a list of policies. (Note that YAML uses dashes to create lists, so you'll need to use a dash prefix before each element in the Policy list.) This section should be at the same indentation level as the other function properties, so `Policies` should be aligned with `Events` and `Environment`:

```
Policies:
  - S3FullAccessPolicy:
      BucketName: !Ref UploadS3Bucket
```

Line 50 to Line 52 of code/ch7/template.yaml

Here are some other interesting policy templates from SAM:

- `S3ReadPolicy` provides read-only access to a bucket.
- `LambdaInvokePolicy` allows calling a specific Lambda function

- `DynamoDBCrudPolicy` provides full access to a DynamoDB table.

SAM developers are frequently adding policy templates, so for an up-to-date list, it's best to check the SAM GitHub repository. You can find the list of all supported templates and their arguments in the all_policy_templates.yaml file.

> ### SAM policies or CloudFormation policies
>
> SAM policy templates are convenient but relatively crude. You can create many finer-grained policies directly with IAM, for example, to grant access only to specific attributes in the DynamoDB table, or only to files with specific prefixes on S3, and only for certain operations.
>
> You can specify a full IAM policy in the SAM function `Policies` property, so you can mix and match templated and non-templated policies.

Technically, policies are not really assigned to a Lambda function. They are assigned to an IAM security role, which is associated with a Lambda function. SAM automatically creates a role for each function, named by appending the word `Role` to the function name. For example, for `ShowFormFunction`, the role resource will be called `ShowFormFunctionRole`. You can use that resource in any other CloudFormation IAM template to configure the role in ways that SAM can't do out of the box. See *CloudFormation Resources Generated By SAM* for more information. If you want to manage the function roles explicitly, for example, to set up a single shared role for several functions, use the `Role` property of a function and provide the role name or ARN. SAM will not create an implicit role for your function in this case.

You're almost ready to deploy this template with two different functions. Before you do that, remember that you changed the API Gateway resource path from `/hello` to just the root. Let's update the stack outputs accordingly, so you can get the new URL easily. Remove the word `hello` from the end of the URL in the stack outputs, and perhaps rename the `HelloWorldApi` output to something more meaningful.

```
Outputs:
  UserFormApi:
    Description: "API Gateway endpoint URL"
```

```
    Value: !Sub "https://${WebApi}.execute-api.${AWS::Region}.amazonaws.com/${AppStage}/"
```

Line 53 to 56 of code/ch7/template.yaml

In order to test whether your function is saving files to S3 correctly, you'll also need to know the name of the bucket that SAM created for you. Setting another template output will help you get that information easily. It is added at the same indentation level as `UserFormApi`.

```
UploadBucket:
  Description: "S3 Bucket for user information"
  Value: !Ref UploadS3Bucket
```

Line 57 to Line 59 of code/ch7/template.yaml

Finally, you'll build, package, and deploy the project again. This might take slightly longer than usual because SAM needs to create two new functions and associated objects, remove the old functions and roles, and also create the bucket.

```
$ aws cloudformation describe-stacks --stack-name sam-test-1 --query Stacks[].Outputs
[
  [
    {
      "OutputKey": "UserFormApi",
      "OutputValue": "https://keqbc29e60.execute-api.us-east-1.amazonaws.com/api/",
      "Description": "API Gateway endpoint URL"
  },
  {
      "OutputKey": "UploadBucket",
      "OutputValue": "sam-test-1-uploads3bucket-g6evg49hxobz",
      "Description": "S3 Bucket for user information"
    }
  ]
]
```

Environment Variables

Key:	Value:
AWS_ACCESS_KEY_ID	Not Specified...
AWS_SECRET_ACCE...	Not Specified...
BUCKET_NAME	Not Specified...
AWS_REGION	Not Specified...

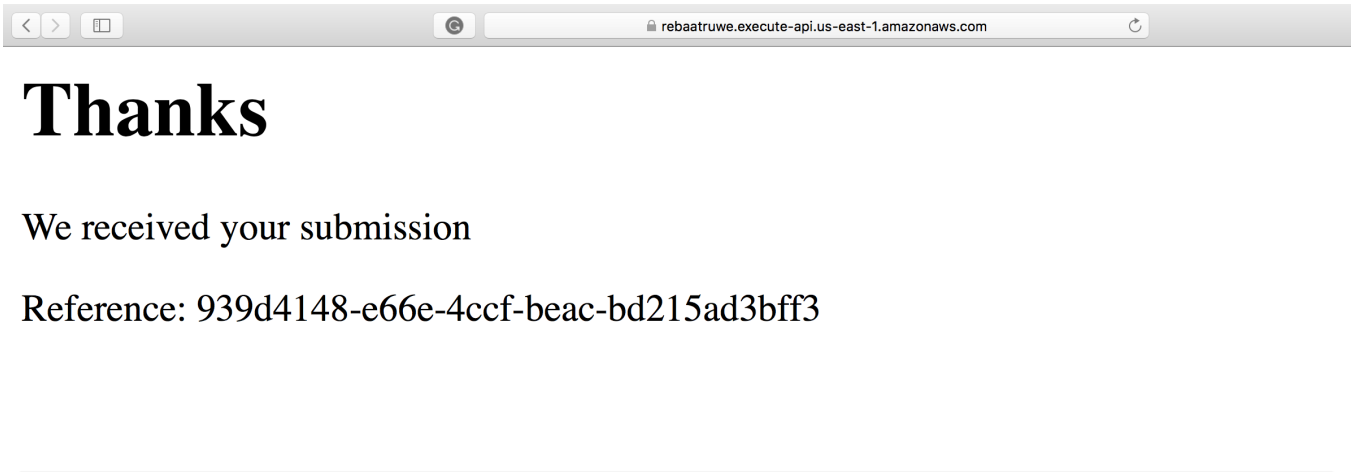
```
{
```

```
  "body": "{\"message\": \"hello world\"}",
  "resource": "/{proxy+}",
  "path": "/path/to/resource",

  "httpMethod": "POST",
  "isBase64Encoded": false,
  "queryStringParameters": {
    "foo": "bar"
  },
  "pathParameters": {
    "proxy": "/path/to/resource"
  },
  "stageVariables": {
    "baz": "qux"
  },
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, sdch",
    "Accept-Language": "en-US,en;q=0.8",
    "Cache-Control": "max-age=0",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-Country": "US",
    "Host": "1234567890.execute-api.us-east-1.amazonaws.com",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Custom User Agent String",
    "Via": "1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)",
    "X-Amz-Cf-Id": "cDehVQoZnx43VYQb9j2-nvCh-9z396Uhbp027Y2JvkCPNLmGJHqlaA==",
    "X-Forwarded-For": "127.0.0.1, 127.0.0.2",
    "X-Forwarded-Port": "443",
    "X-Forwarded-Proto": "https"
  },
  "requestContext": {
    "accountId": "123456789012",
    "resourceId": "123456",
    "stage": "prod",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "requestTime": "09/Apr/2015:12:34:56 +0000",
    "requestTimeEpoch": 1428582896000,
    "identity": {
      "cognitoIdentityPoolId": null,
      "accountId": null,
      "cognitoIdentityId": null,
      "caller": null,
      "accessKey": null,
      "sourceIp": "127.0.0.1",
      "cognitoAuthenticationType": null,
      "cognitoAuthenticationProvider": null,
      "userArn": null,
      "userAgent": "Custom User Agent String",
      "user": null
    },
    "path": "/prod/path/to/resource",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "apiId": "1234567890",
    "protocol": "HTTP/1.1"
  }
}
```

Check out the stack outputs and you should see the new URL (without `/hello`) and the bucket SAM created for us. Open the application page in a browser and try it out.

Once you submit the form, you should see a reference on the thank-you screen (see figure below). The Lambda function will also store the full request body to S3, so you can process it later. Note that in this case, you will store the full request, not just the form data.



The form processing Lambda function saved the file to S3 and provided the user with a reference.

Open the AWS S3 Web Console, look for the bucket created by CloudFormation, and check out the contents. You should see a single file, named after the reference that the form handler page printed out.

Get ready to learn how to deal with network timeouts in the next lesson.