Lambda Access Rights

In this lesson, you will modify your application code to enable the access rights of Lambda functions.

Security requirements

AWS does not trust a Lambda function to access a database or an S3 bucket just because they belong to the same account. You need to explicitly allow the use of each external resource from a Lambda function. To do that, you'll need to modify the IAM policy associated with a function.

Your function currently has two actions:

1. displaying a form
2. processing the form

The form processing action will need access to an S3 bucket, but the form display action does not need any specific security access. When two different actions need different security levels, it's usually a good time to start thinking about breaking them into different Lambda functions.

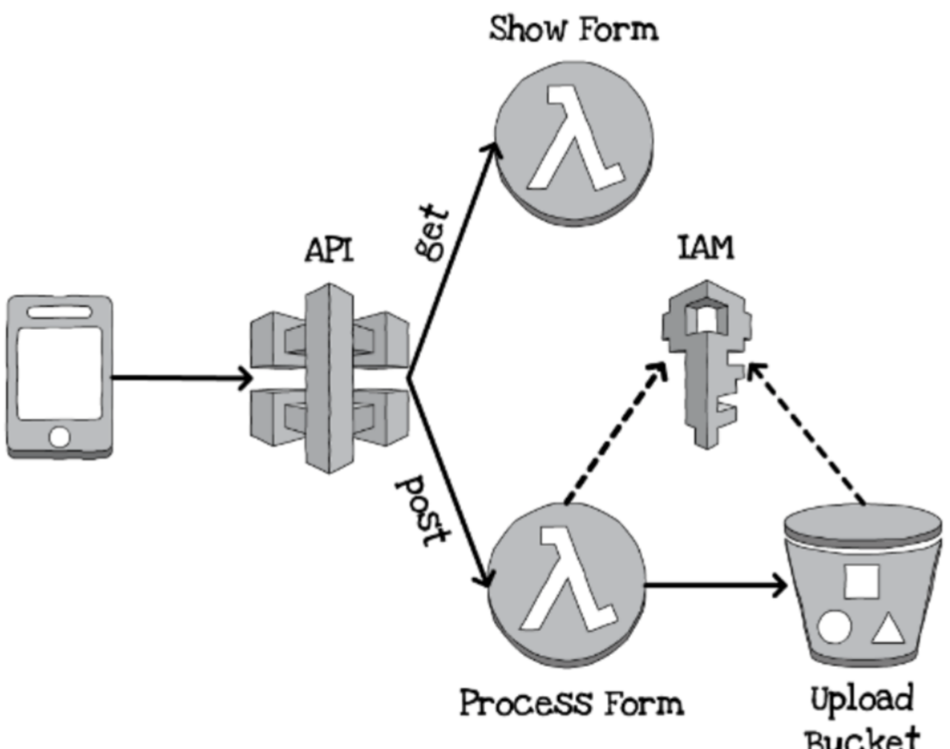
With container-based applications, an API server process usually needs a superset of all the security permissions for all the contained actions. That makes setup and deployment easier, but it also means that a small security bug can easily turn into a disaster. If intruders break through a gatekeeper process, they can easily access all back-end resources. With Lambda functions, separating actions according to their security needs becomes very easy. Each function can have specific access to only the necessary resources. If a third-party dependency introduces a security issue in one of your API endpoints, the keys to the kingdom are still locked in a safe.

In Chapter 6, you created a new API endpoint to handle POST requests and just wired it up to the same function as the GET handler. Let's split the two endpoints into two functions so you can manage security better (see the figure below).

`ShowFormFunction`

You can use this opportunity for some nice housekeeping. Let's do the following:

1. Rename the old Lambda function from `HelloWorld` to `ShowFormFunction`, so its purpose becomes clearer.
2. Rename the function code directory on your disk from `hello-world` to something more meaningful, such as `user-form`, and update the `CodeUri` property of the function accordingly.
3. Rename `app.js` inside the project code directory to something more meaningful, for example, `show-form.js`, and update the `Handler` property of the function accordingly.
4. Rename the first event mapping to something meaningful, for example, `ShowForm`.
5. Finally, instead of `/hello` as the API resource path, just use the root resource (`/`).



We'll divide processing into two functions to control security better.

The second event handler will be moved into a new Lambda function, so you can delete it from this block. The remaining configuration for the first function should look like in the following:

```yaml
ShowFormFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: user-form/
    Handler: show-form.lambdaHandler
    Runtime: nodejs12.x
    Events:
      ShowForm:
        Type: Api
        Properties:
          Path: /
          Method: get
          RestApiId: !Ref WebApi
```

Line 21 to Line 33 of code/ch7/template.yaml

You can also change the source code for the form display Lambda function. It no longer needs to handle post requests, so you can make it simpler. Update it to look like this:

```javascript
const htmlResponse = require('./html-response');
const formHtml = `
  <html>
  <head>
    <meta charset="utf-8"/>
  </head>
  <body>
    <form method="POST">
      Please enter your name:
      <input type="text" name="name"/>
      <br/>
      <input type="submit" />
    </form>
  </body>
  </html>
`;
exports.lambdaHandler = async (event, context) => {
  return htmlResponse(formHtml);
};
```

code/ch7/user-form/show-form.js

`ProcessFormFunction`

The start of the second function configuration will be very similar to the first one, apart from the HTTP method and the JavaScript file reference. Let's make it talk to `process-form.js`, which we'll create in a minute, and use the `POST` method. We add a block similar to the code in the next listing:

```yaml
ProcessFormFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: user-form/
    Handler: process-form.lambdaHandler
    Runtime: nodejs12.x
    Events:
      SubmitForm:
        Type: Api
        Properties:
          Path: /
          Method: post
          RestApiId: !Ref WebApi
```

Line 34 to Line 46 of code/ch7/template.yaml

Let's complete the second function in the next lesson.