Extracted from:

# Serverless Single Page Apps

## Fast, Scalable, and Available



The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

# Serverless Single Page Apps

## Fast, Scalable, and Available


Ben Rady

*edited by Jacquelyn Carter*

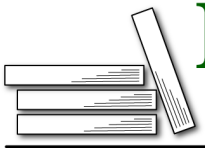
# Serverless Single Page Apps

## Fast, Scalable, and Available

Ben Rady

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz, Liz Welch (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.



Printed in the United States of America.
ISBN-13: 978-1-68050-149-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2016

## Invoking Lambda Functions

You can invoke a Lambda function from a browser in two ways. The first way is to use the AWS SDK. The second is via the Amazon API Gateway. First, we're going to see how to invoke Lambda functions from the browser, using the AWS SDK and the Cognito credentials issued to the user.

By adding a new policy to our authenticated Cognito role, we can invoke Lambda functions directly from the browser without going through a public HTTP interface. As long as users have the proper credentials that let them assume the role, they can perform any operation listed in the policy. The particular operation we want to perform is named `invoke`. To perform this operation, you need to create an instance of the `Lambda` class from the AWS library. You then need to call the `invoke` function.

### Create a Lambda Policy

To allow access to this Lambda function, you need to create a new IAM policy and add it to the authorized Cognito role, just as we did in *Authorizing DynamoDB Access, on page ?*. Unlike that policy, you won't need a `Condition` clause in this one, because you're granting access to all authenticated users.

Of course, when using the Lambda API, we have the same issues with expiring credentials that we do with DynamoDB. Wouldn't it be great if we could reuse the `sendDbRequest` function to handle all that? Actually, we can…but you'll probably want to rename it to something like `sendAwsRequest` first. If the tests still pass once you've done that, you can go ahead and write a function to call the service.

**learnjs/6200/public/app.js**

```
learnjs.popularAnswers = function(problemId) {
  return learnjs.identity.then(function() {
    var lambda = new AWS.Lambda();
    var params = {
      FunctionName: 'learnjs_popularAnswers',
      Payload: JSON.stringify({problemNumber: problemId})
    };
    return learnjs.sendAwsRequest(lambda.invoke(params), function() {
      return learnjs.popularAnswers(problemId);
    });
  });
}
```

The tests for this new function follow the same patterns as the tests for our DynamoDB functions, so they're easy to write. Just as in the previous chapter,

we're using jQuery Deferred objects to coordinate these requests. Adding this information into the views in our application takes much the same form.

One limitation of this approach is that only authenticated users will be able to make these requests. While it's possible to relax the permissions on this function and allow anyone to invoke it, there's another way to provide open access to this Lambda function so that anyone who wants to can invoke it. Next, we'll look at how to provide access to a Lambda function via Amazon's HTTP API Gateway.

## Using the Amazon API Gateway

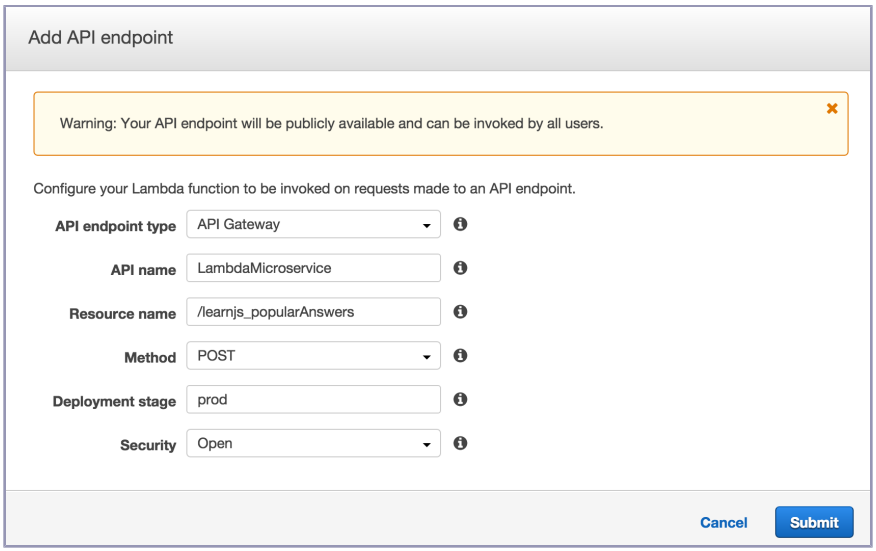
As we've seen, invoking Lambda functions via the AWS SDK with Cognito credentials can be a great way to integrate custom services into your applications, but what if you want to provide public access to a Lambda function? You can make these functions accessible via an unauthenticated HTTP request using the Amazon API Gateway.

### HTTP or HTTP?

While describing the API Gateway as "a way to invoke Lambda functions via HTTP" is correct, one thing to understand is that the AWS SDK for JavaScript *also* invokes Lambda functions via HTTP. Indeed, almost everything it does to interact with AWS is via HTTP, because that's the most stable protocol available from a web browser. It's just using a different endpoint than what's provided via the API Gateway.

The Amazon API Gateway maps *APIs* to Lambda functions through *endpoints* that you define with each function. You can create these APIs and their associated endpoints either through the Amazon API Gateway console or through the Lambda console. To create a public API for our function, we're going to use the Lambda console.

First, go to the settings page for our function in the Lambda console. Select the "API endpoints" tab, and click the "Add API endpoint" link. Once you do that, you'll be prompted to choose an endpoint type. Choose API Gateway, and you'll be presented with a configuration screen that looks like this:






Switch the security to Open, and don't mind the scary warning that shows up when you do. The whole point here is that we're trying to make this public. Next, switch the method to POST, because we want to use HTTP posts to deliver the request body. Set the deployment stage to either prod or test, depending on which version you're creating, and then pick an API name.

The API name is used to identify this API in the Gateway console. If you create lots of different applications using the same AWS account, you can use these names to differentiate APIs for different apps.

After saving these changes, you may have to wait a few seconds for them to be applied. Then you can try to make a request from the command line to ensure the API is active and publicly accessible. Get the endpoint URL from the Lambda console, and then using the curl command, make a POST request from the command line, like this:

```
$ curl -d '{"problemNumber":1}' «endpoint_url»
{"true":2,"!false":1}
```

If you get back a response like this, you know the service is working. You may get an empty JSON object if you haven't saved any answers. If you get a message that says "Missing Authentication Token," double-check your URL.

With that, we now have two methods for accessing this service: one authenticated method that uses the AWS JavaScript SDK and Cognito credentials, and a public HTTP API that anyone can access. Depending on the type of

application that you're building, you may want to use one or both of these approaches.