Anti-Buddies Handoff Document

Our group accomplished quite a bit this semester, and we just wanted to document where it was that we left off so that the next group to take over this project has an easier time starting to continue the project. One of the big things we worked on was the CSS and making it all the same/similar between pages. You can look at the different CSS files to get an idea of what styles we have, or we also have a CSS style guide you could take a look at. There are little things on each page that need tidying up, mostly just aligning buttons better.

# Getting Started

Tools you’ll need:

* SQL Server Management Studio
* GitHub Desktop (technically only Git, but Github works well)
* Google Chrome for debugging
* Competent code editor
  + Brandon used Visual Studio Code for pretty much everything and it works well.
  + Alex used Brackets which is very similar in capability to VS Code
  + You’ll be editing html, css, javascript, Go (commonly referred to as GoLang when trying to Google things about it) if you continue using it for the Lambda functions, T-SQL (SQL Server Management Studio covers editing SQL script, but it might be a good idea to find a VS Code plugin that makes SQL easier to write/edit).
* Things we never used:
  + Anything Mac; none of us owned a Mac which made it next to impossible to develop the Swift version of this, so we’ve made no changes to that and we find that the project makes sense as a web app.
* Log into AWS (Amazon Web Services). We’ve been using Ben Oliverson’s account as the main, with his permission.
  + Files that are “live” are in the S3 bucket. I found it easiest to just search for the S3 bucket on top instead of going through it all.
  + The other part of AWS you’ll need is the functions. To get to those you can search for Lambda.
* For Git we used the most recent published version and cloned it into our own. Our repository is:
  + <https://github.com/alexandramcmorris/CS4450AntiBuddies> --our repo for the entire project
  + <https://github.com/boliverson/antibuddiesGoLang> --repo for the lambda functions
  + <https://github.com/macarthuret/WeberStateAntibuddies/blob/master/SQLScript.sql> --repo for the sql scripts
* Go
  + See Lambda Function section below

## Edit a Quiz

To add a new question to the main editable quiz, go to admin-quiz.html, select the quiz you’d like to edit (there is currently only the one), and you’ll see all the questions in the quiz currently (actually, it’s currently set to only display 4 questions).

### Add a new Question

Then select “+ New Question” where you’ll be taken to an interface to add a new question. Type a section (currently this can be set as anything), select a difficulty (currently the page populates with Beginner level, so if you want it to appear there immediately, select Beginner), type the question, the answers (currently there are 4), and select which answer will be considered correct, and enter any extra text you wish to be shown as an explanation of the correct answer. Hit submit!

### Delete a Question

Click the “Delete” button beside the question you want to be deleted!

### Allocating an Admin

We have one test admin; username: a password: a

In order to change a user to an admin you will have to update the user through a SQL statement. This is the code we used to set admin:

UPDATE Users

SET isAdmin = 1

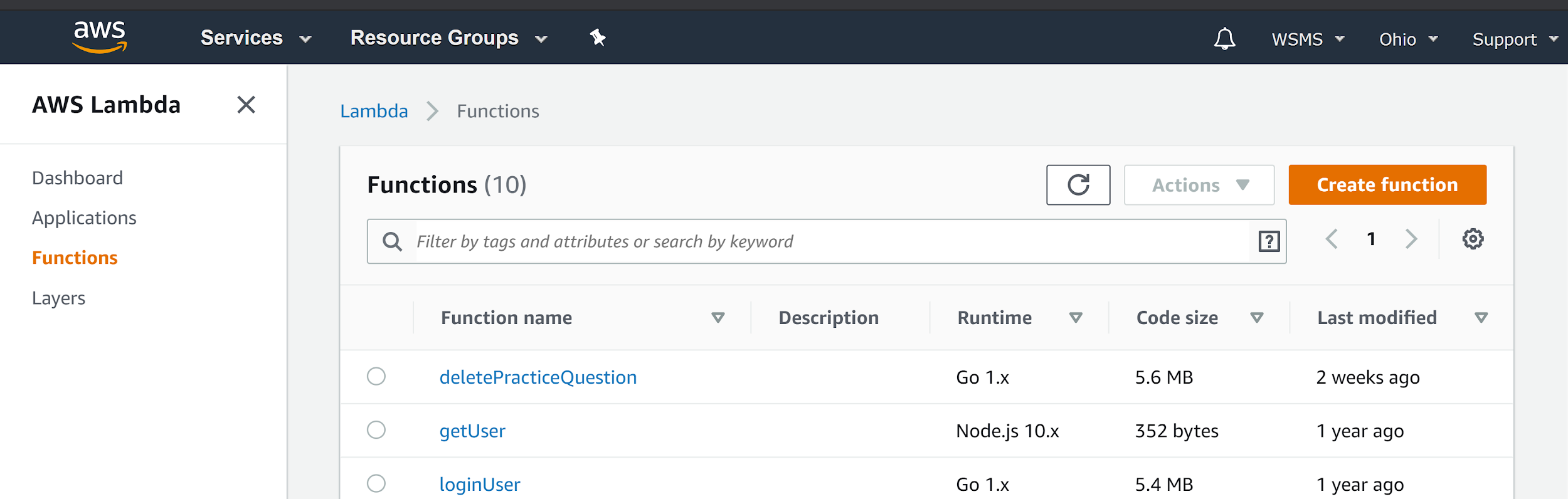
WHERE user\_id = 2;

# AWS Lambda Functions

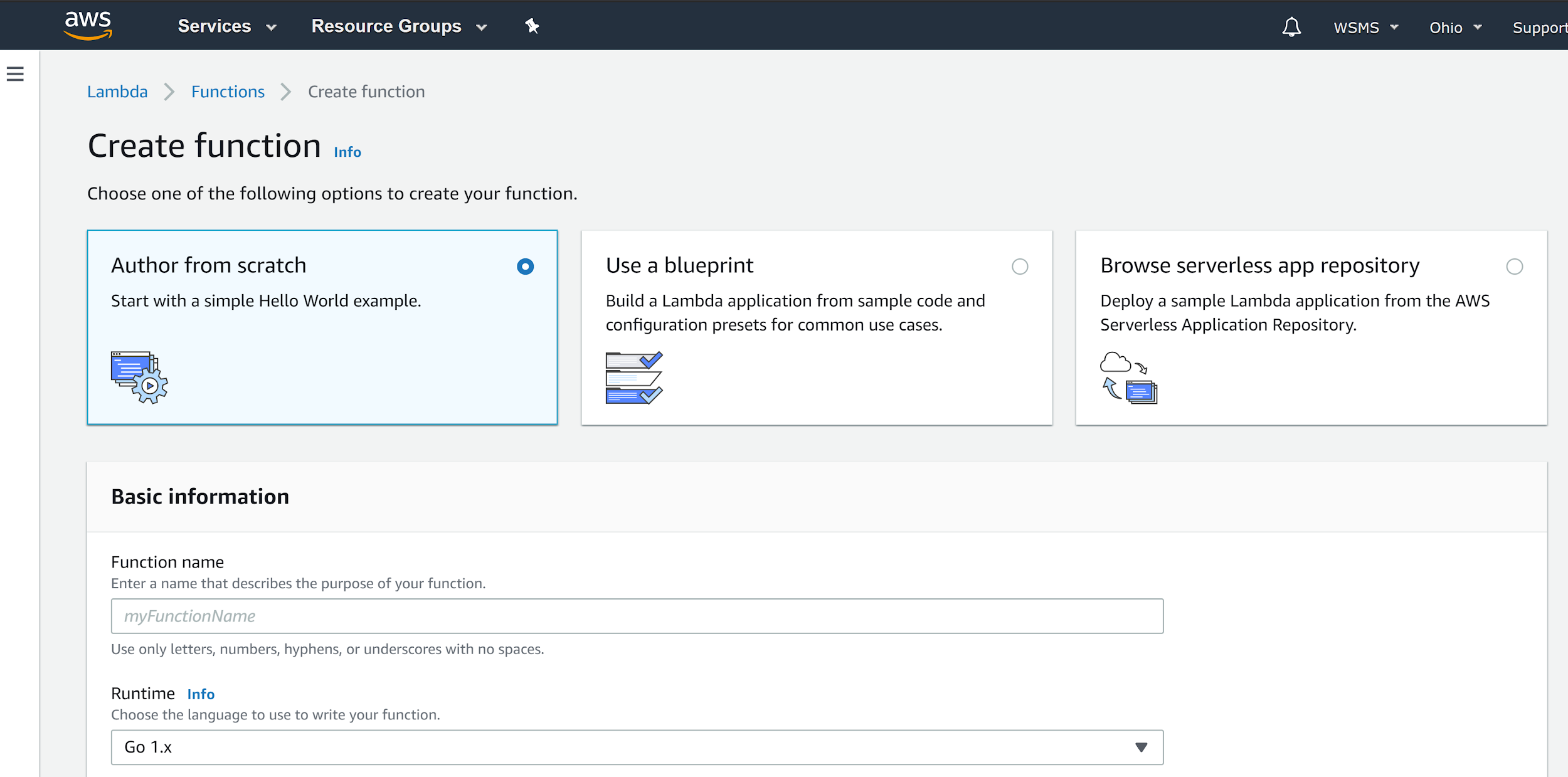
<https://github.com/boliverson/antibuddiesGoLang> --repo for the lambda functions

If you choose to keep writing Lambda functions in Go, here’s what you’ll need to get up and running:

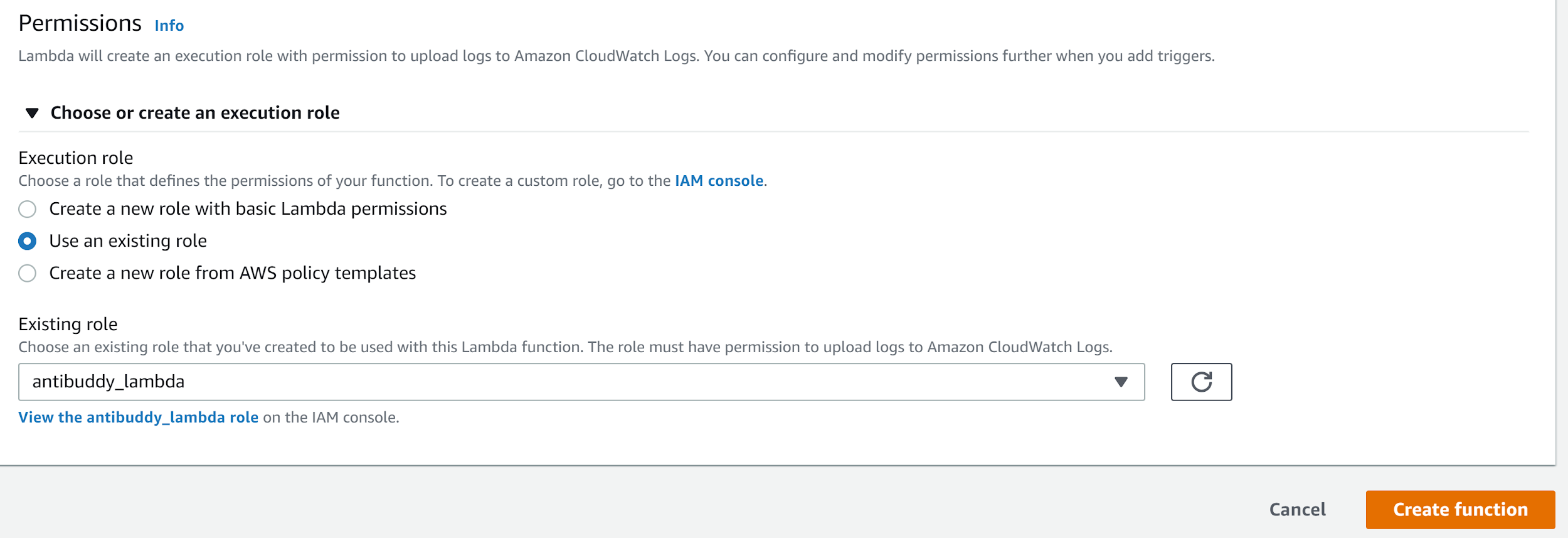
1. Download and install Go from <https://golang.org/>
2. Make sure your IDE of choice can accommodate Go on a basic level. It won’t need to be able to compile it, as you’ll do that manually using the following instructions on deployment of a Lambda function. If you find a way to do it less manually using an IDE, go ahead, but we never got around to finding one.
3. You can use some of the Lambda functions we and previous groups have created as a starting point, and there is also much more information about the specifics of how to write a Go Lambda function through:
   1. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-golang.html>
4. The following page outlines how to to actually deploy a Lambda function written in Go:
   1. <https://docs.aws.amazon.com/lambda/latest/dg/golang-package.html>
   2. This page is pretty comprehensive, you just need to be relatively comfortable using Windows command prompt or Powershell.
5. Once you have the .zip file of the function as outlined above:



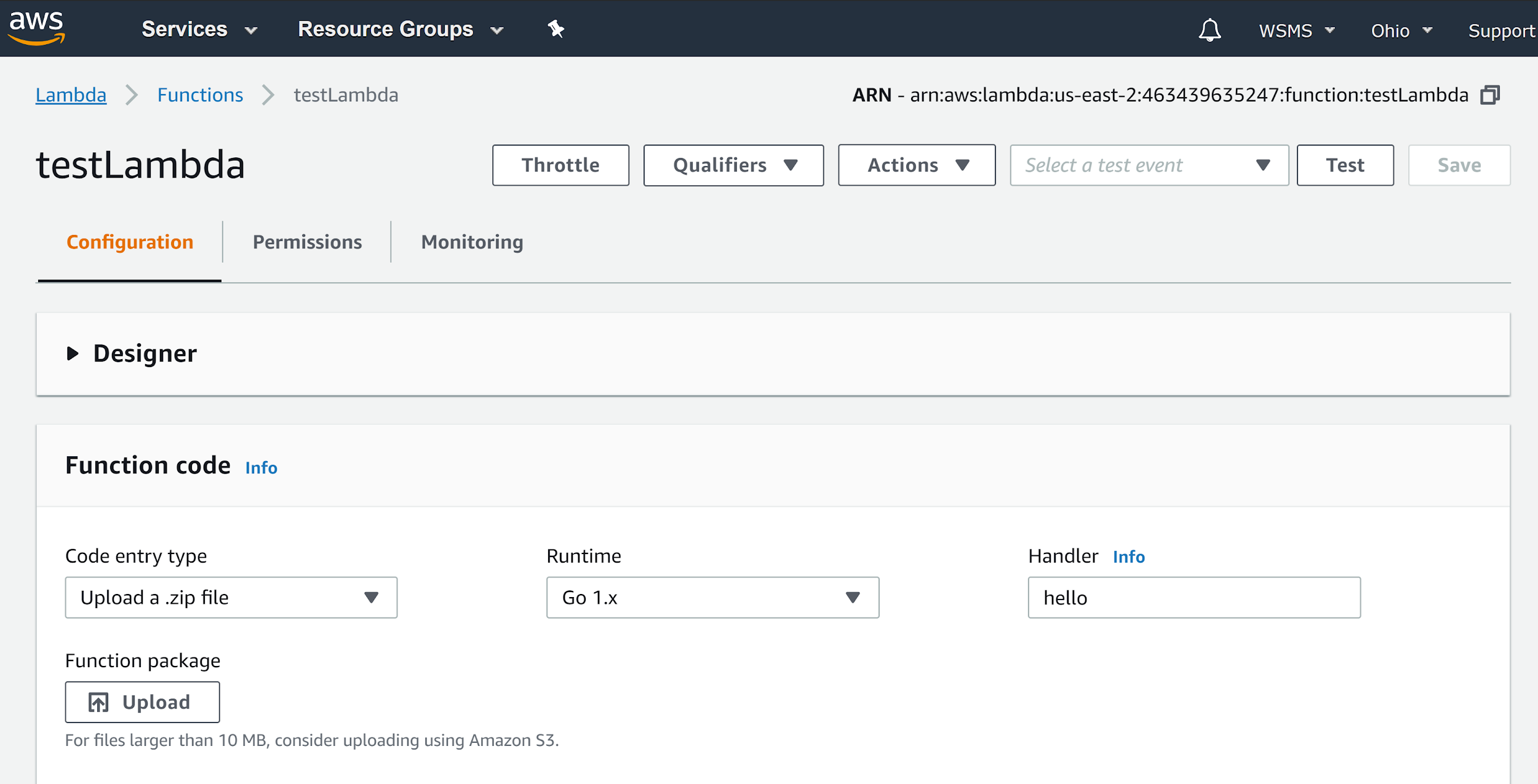
Select “Create function”.



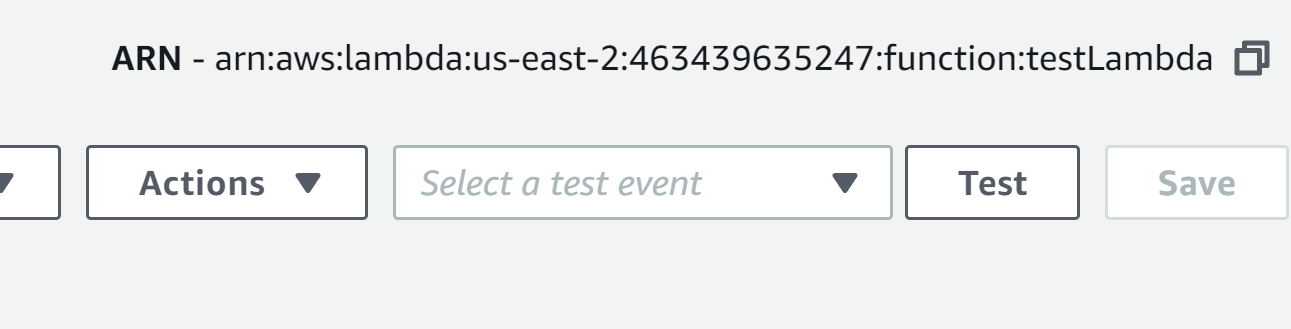
We only authored from scratch, but it may be worth looking into “Use a blueprint”. Name the function, and make sure the Runtime is set to Go.1.x (assuming that’s what you’re writing it in).



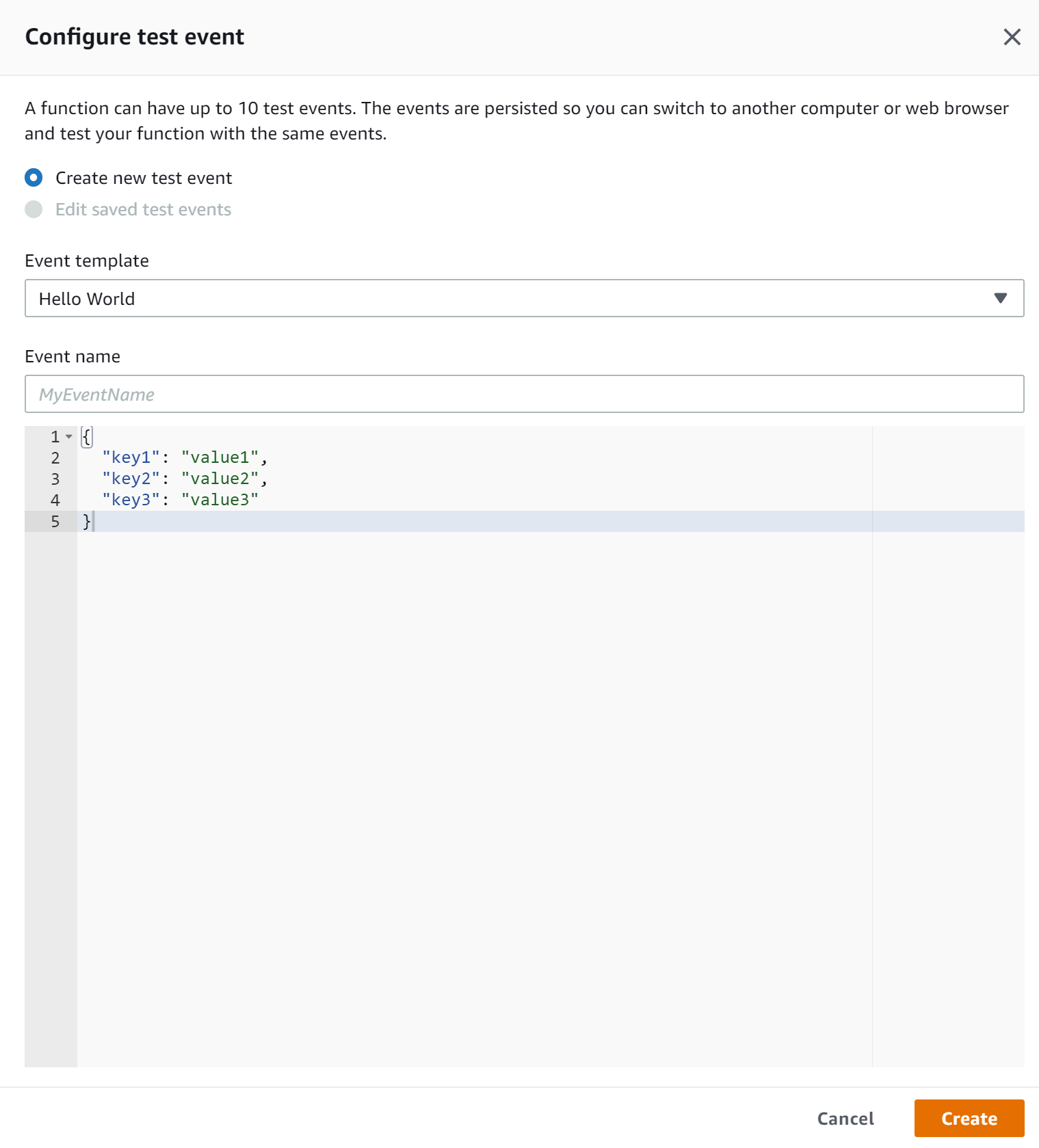
Expand the drop-down for Permissions and select “Use an existing role”, and set it to “antibuddy\_lambda”. We haven’t scrutinized this role, so we don’t guarantee that that is definitely what you ought to use, but using it will at least ensure consistency.



Click “Upload” and select the .zip of your Lambda function. Make sure the Handler is named as specified in the above AWS documentation.



Use the Test Feature to send a particular request to the Lambda function, just as if say Javascript were making a real request. To make a test, click the drop down and “Configure Test Events”.



Create your test event and make sure to hit Create, or Save if you’re editing an existing test event. Then hit test to send the selected test event to the Lambda function.

Below is an example of a simple invocation of a Lambda Function written in Go from AWS. All invokes are located in the login.js file in their respective JavaScript function

Each function takes a question parameter where the invocation type and log type should be the same for every

var questionParams = {

FunctionName : "deletePracticeQuestion", //this is the function name titled in the go file

InvocationType : "RequestResponse",

LogType : "None",

Payload : '{"questionID":"'+questionID+'"}', //payload is the parameters that are needed to get results from the function

};

lambda.invoke(questionParams, function(error, data) {

if (error) {

prompt(error, error.stack);

} else {

//if everything went well this will be where you can access any returned variables.

}

});

}

# Known Issues

* When logging in there is no security behind admins pages. For example, if you use the http anyone can have access to the admin pages.
* On the login page (index.html) you can’t log in by pressing enter.
* On quiz.html we reverted back to the original that was found at the beginning of the semester for ease of understanding. It functions right but is not connected to the database. Implementing that will be very similar to how to load the questions in admin-quiz.html using functions: getPracticeQuestions, getAnswers, placeAnswers, and adminQuizStart.
  + For reference we’ve left quiz.js as a preview of what we had done minus the db connection.

Things left to do for the admin quiz per Justin Rhees request:

* He would like the questions to allow for the option of adding pictures to answers and their questions.
  + There’s nothing about storing images in the database.
  + Ideally, there should be a high-quality image viewer, allowing users to be able to zoom in, etc.
* Allow for any number of answers for a question
  + Will require GUI modifications/additions
  + Make sure all logic can accommodate this.
* There is no GUI to allow the admin to actually change the question and answers, although we implemented a backend for it, including a Lambda function and corresponding javascript function.
* Allow editing access to the appropriate admins for any given quiz.
  + Currently only the Immunohematology quizzes can be edited through the GUI.
* Abstract out ‘question difficulty’ to be an instructor defined quiz name, as quizzes of different difficulty should just be separate quizzes for the same course.
* The Lambda functions generally run queries and scripts directly, rather than through stored procedures, functions, triggers, etc.
  + Adding such scripts to the database and simply calling them in the middle layer may be easier, cleaner, and more standard for industry.
* Add back buttons
  + We’re not familiar with the best way to do this without basically having to redirect each page manually, or using the browser history back function which probably isn’t the most advisable solution.