# Lesson 4

In this lesson are going to add the ability to upload videos from the browser to an S3 bucket. To do this we are going to:

1. Create a Lambda function to grant us credentials/policy to upload a files to an S3 bucket.
2. Configure API Gateway to allow our website to access this Lambda function and retrieve the necessary policy document.
3. Update the website to request the policy document and upload the file to S3.

## Create a Lambda function

You will need to create a new Lambda function in the AWS console. This Lambda function will generate a policy document to allow our users upload videos to S3. Step through the following:

* Click **Lambda** in the AWS Console
* Create a blank new function (skip blueprint)
* Name the function **get-upload-policy**
* Add two brackets to the body of the function (e.g. **{}**)
* Assign the **lambda-s3-execution-role** policy to it (the same policy created in lesson 1)
* Leave all other settings on default and save

## Create IAm User

The policy and the credentials that we are going to generate in the Lambda function need to be signed by an IAM user that has permissions to upload files to S3. Let’s create this user now.

* Open IAM console
* Click **Users** and create a new user called **upload-s3**
* Download the user’s access & secret keys. You will need these.
* Click the **upload-s3** user and click the **Permissions** tab
* Expand **Inline Policies**
* Create to create new Inline Policy and select **Custom Policy**
* Set the name of the policy as **upload-policy**
* Copy the following to the Policy Document and save (make sure to specify your upload bucket name in the policy).

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:ListBucket"

],

"Resource": [

"arn:aws:s3:::**YOUR\_UPLOAD\_BUCKET\_NAME**"

]

},

{

"Effect": "Allow",

"Action": [

"s3:PutObject"

],

"Resource": [

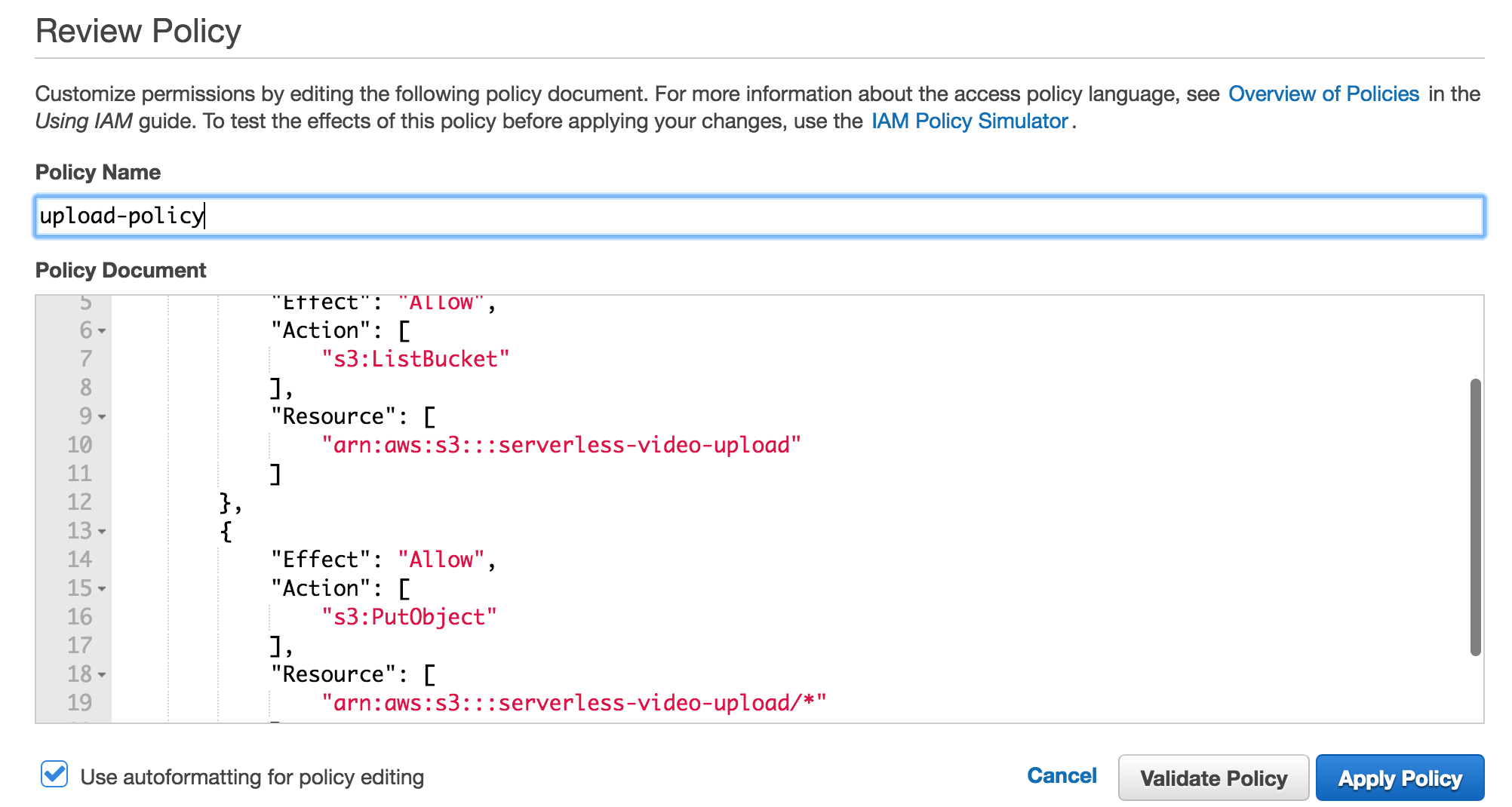
"arn:aws:s3::: **YOUR\_UPLOAD\_BUCKET\_NAME** /\*"

]

}

]

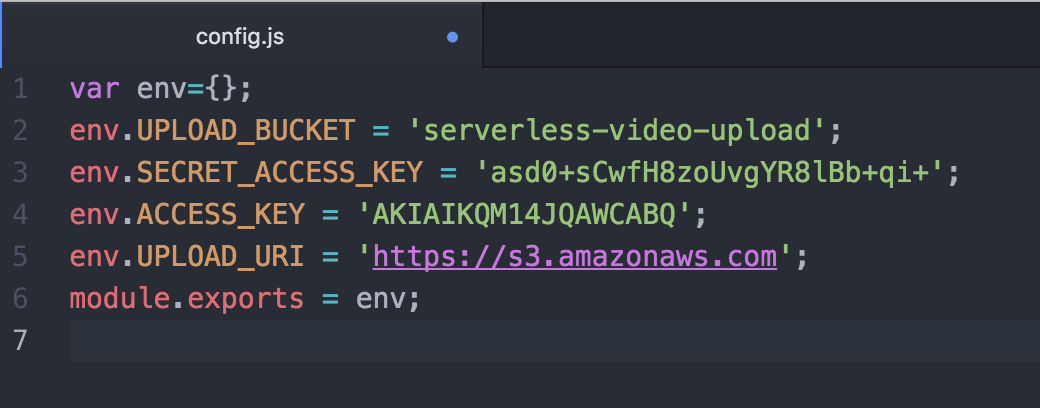
}



## Configure function

Open the Lambda function provided in Lesson 4 on your computer. It’s located in lesson-4/create-s3-upload-policy-document.

* Update **config.js** and update:
  + **UPLOAD\_BUCKET** to be the name of your upload bucket
  + **ACCESS\_KEY** and **SECRET\_ACCESS\_KEY** to match the user you created in step 2



* Open a terminal / command-prompt and navigate to the following folder:

lesson-4/create-s3-upload-policy-document

* Run the following command to create a zip file for your function

npm run predeploy

* If you are Windows, you will have to zip up your function yourself.

## Deploy Function

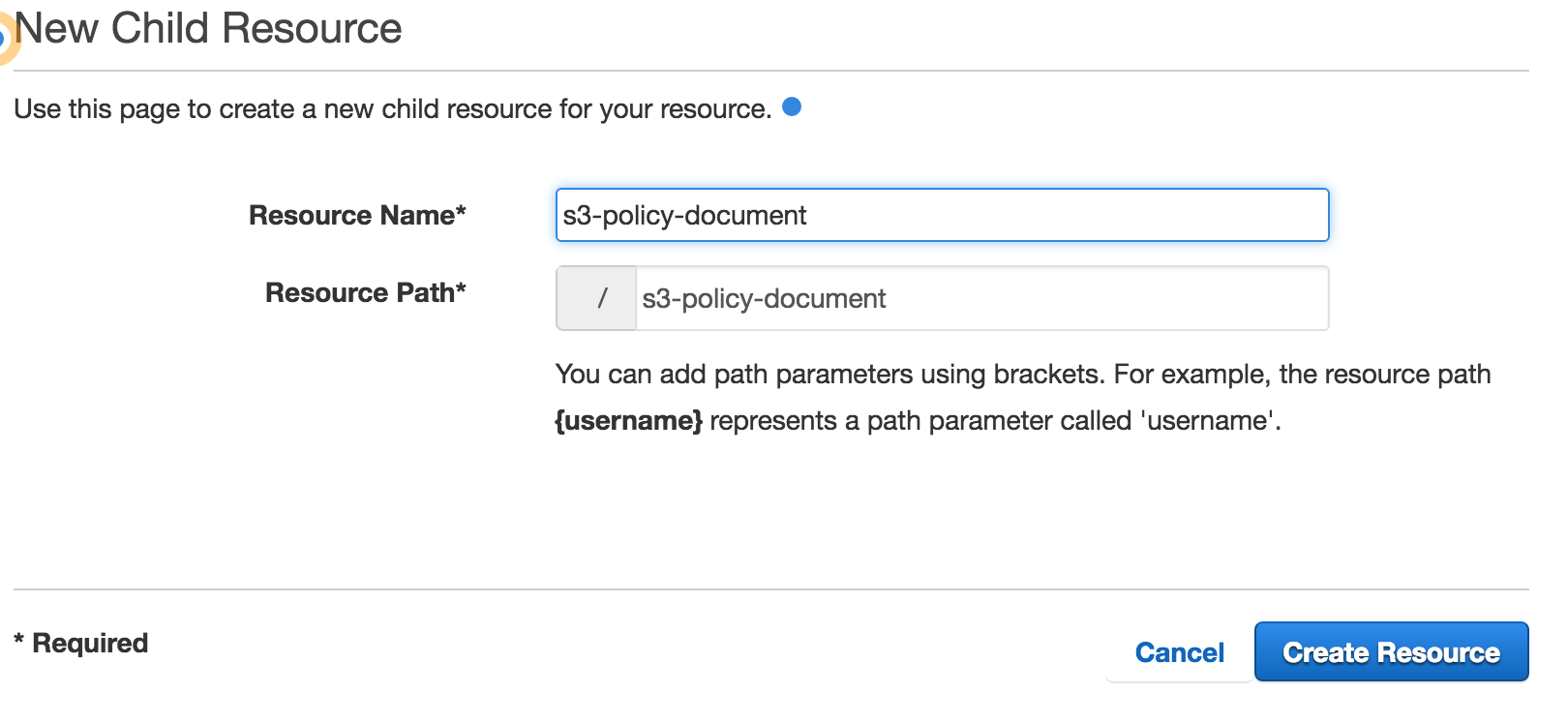
Now we need to deploy the function to AWS.

* In the AWS console click **Lambda.**
* Click **get-upload-policy** in the function list.
* Click **Upload** to upload the function and then click **Save.**

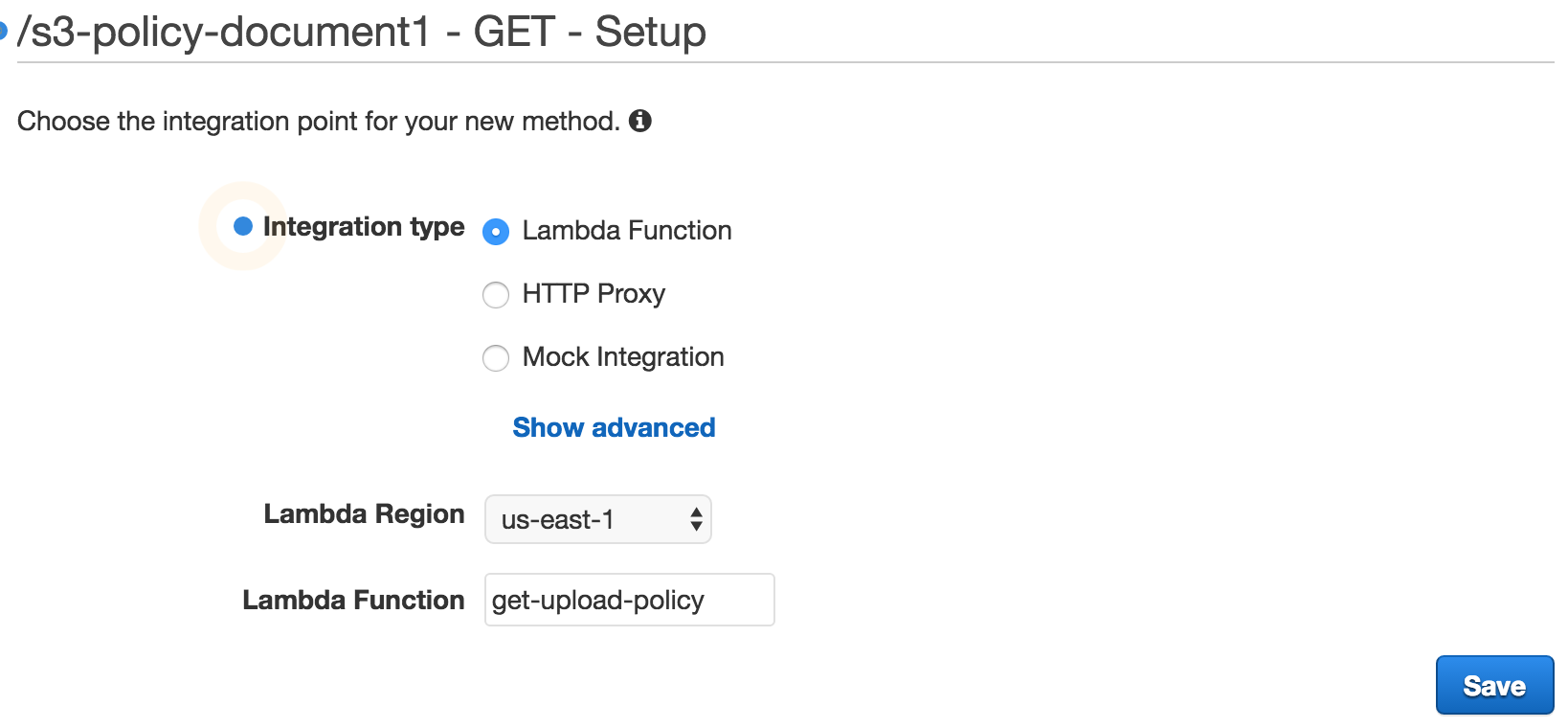
## Create Resource & Method in the API Gateway

In this step we will create a resource and a method in the API Gateway. We will use it to invoke the Lambda function we deployed in the previous step.

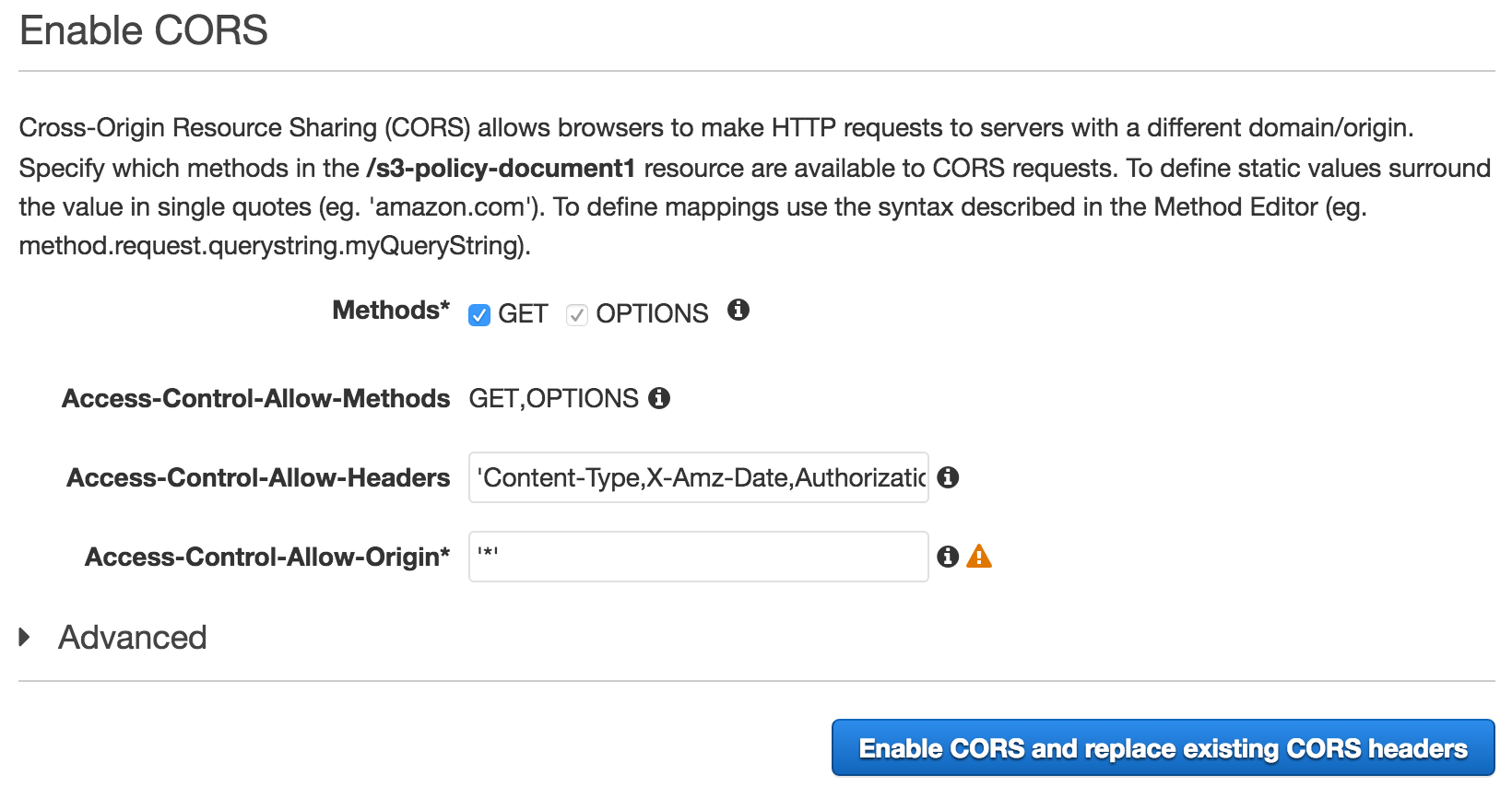
* Click **API Gateway** in the AWS console
* Select **24-hour-video**
* Click **Actions** and then click **Create Resource**
* Set the Resource Name to **s3-policy-document**



* Click **Create Resource**
* Make sure that **s3-policy-document** is selected under **Resources** and click **Actions**
* Click **Create Method**
* From the dropdown box under the resource name, select **GET** and click the tick button to save
* In the screen that immediately appears:
  + Select **Lambda Function** radio
  + Set **us-east-1** as the Lambda Region
  + Type **get-upload-policy** in Lambda Function textbox
  + Click **Save**. Click **OK** in the dialog box that appears.



* Click **Actions** and click **Enable CORS**
* Click **Enable CORS and replace existing CORS headers**

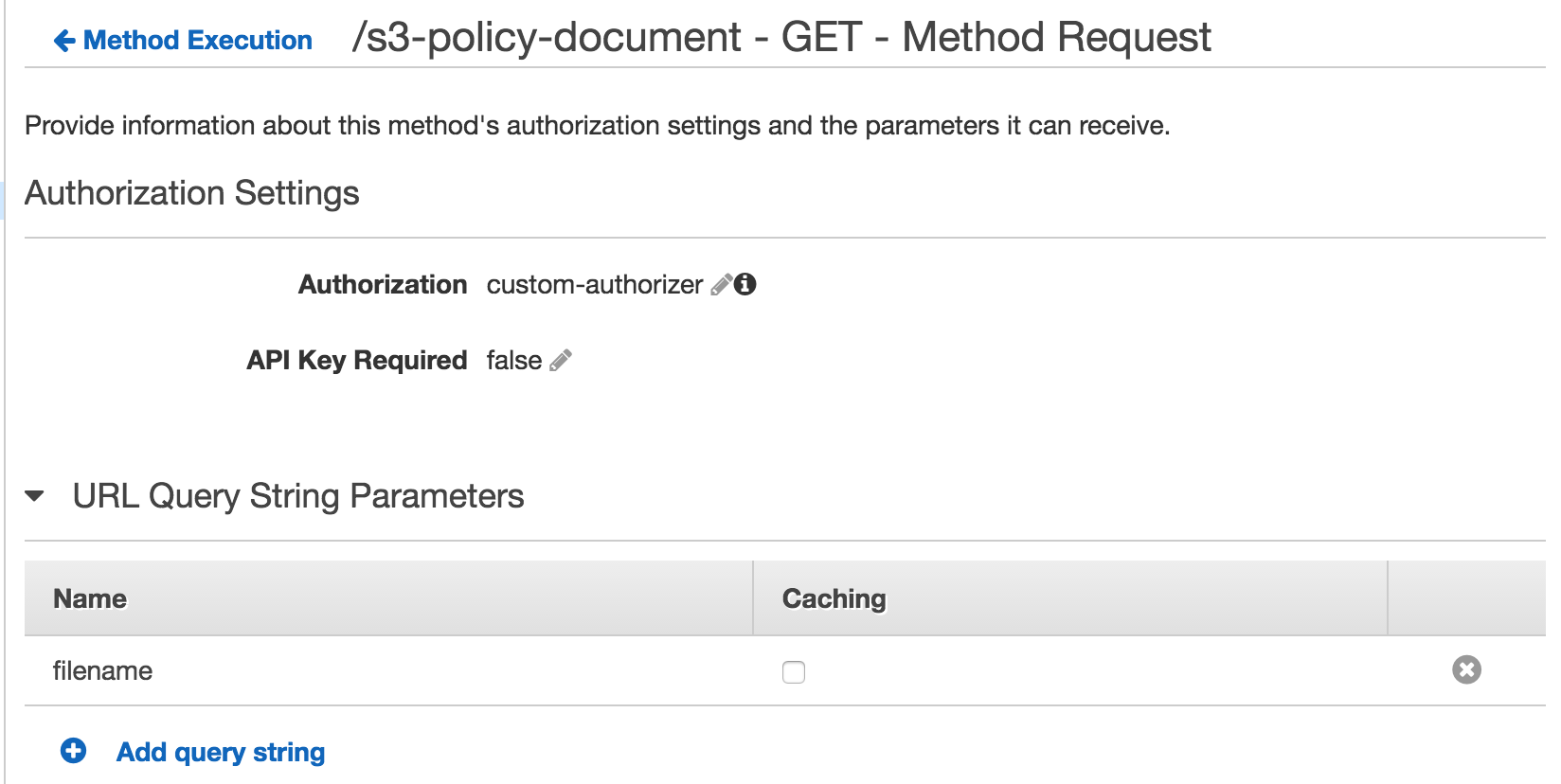


* Click **Yes, replace existing values**

## API Gateway Mapping and Security

There are two more things we need to do in the API Gateway. We are going to pass a query string parameter in our request. This parameter will contain the filename of the file which we need to upload. We need to create mapping in the API Gateway to correctly pass this information in to a Lambda function. Finally, we need to enable a custom authorizer so that only authenticated users can invoke our function.

* In the API Gateway click on **24-hour-video** under **APIs**
* Click **GET** under **/s3-policy-document** in Resources
* Click **Method Request**
* Expand **URL Query String Parameters**
* Click **Add query string**
* Type in **filename** and click the tick button to save.



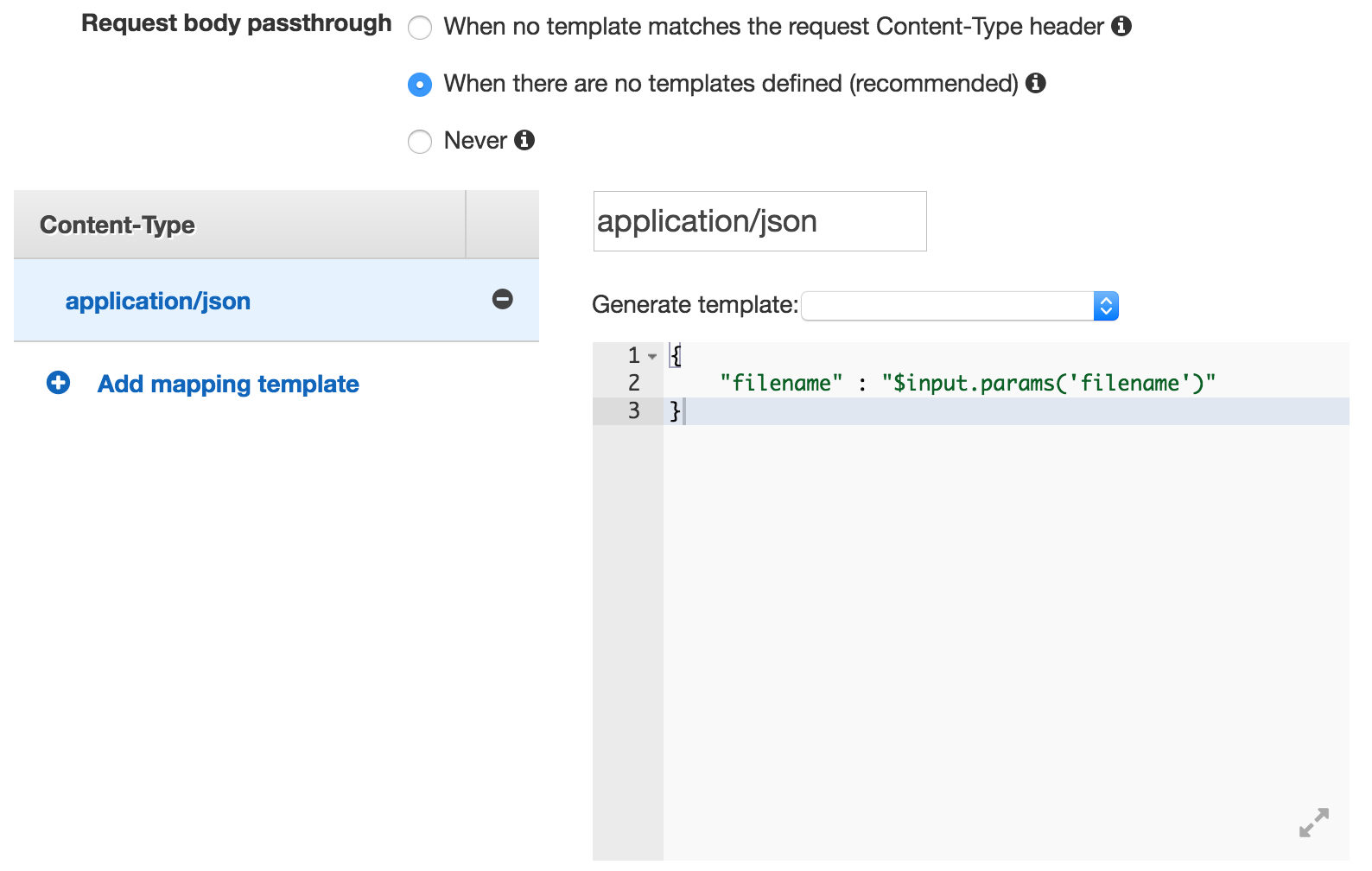
* Click **Method Execution** to go back to the main Method Execution screen
* Click **Integration Request**
* Expand **Body Mapping Templates**
* Click **Add mapping template**
* Type in **application/json** and click the tick button to save
* Click **Yes, secure this integration**
* In the template section type in the following:

{

"filename" : "$input.params('filename')"

}

* Click **Save**

****

## Testing

Now we are ready to test our upload functionality via the website.

Open a terminal / command-prompt and navigate to the following folder:

lesson-4/website

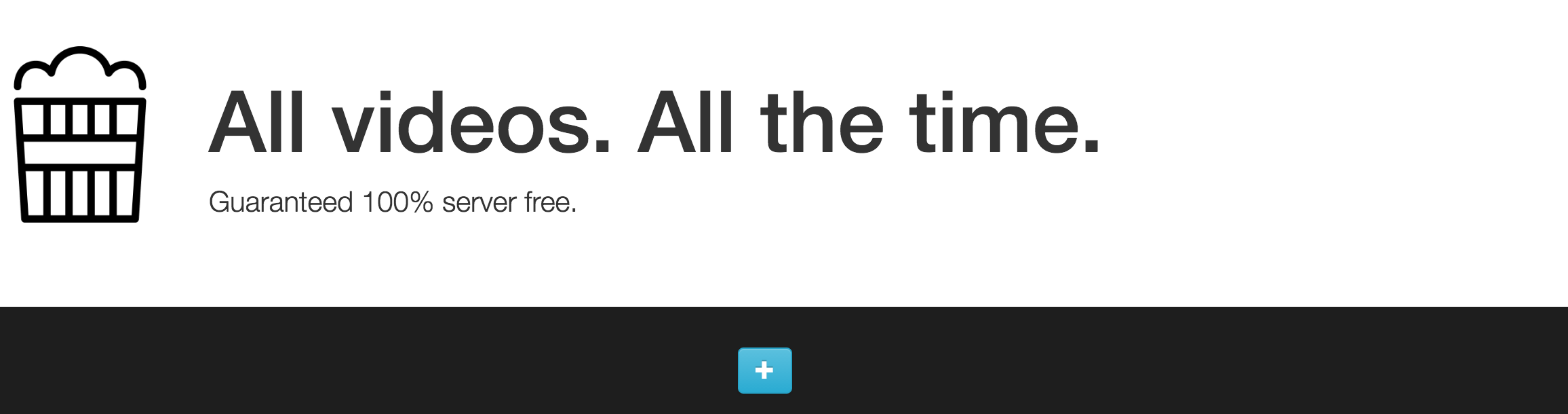
Run the following command to make sure that required npm components are installed:

npm install

Run the following command to start the website:

npm start

Open the website and sign in. Click on the **plus** button to upload a movie file. You will see a progress bar while the upload takes place.



Jump in to the AWS console and have a look at the buckets. Did the file upload to the upload S3 bucket? Are there new files in the transcoded S3 bucket?

If something didn’t work make sure to check that:

1. The **config.js** file in your website contains the right Auth0 credentials and API Gateway URL.
2. You have followed steps 1-7 exactly and copied everything exactly as specified in this lesson plan.

**We are nearly there! There’s one more lesson left and you’ll have your full YouTube clone ☺**