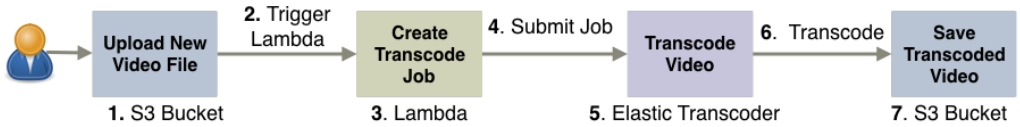
Workshop Workbook

Building a Serverless Video Sharing Website

LESSON 1

In lesson 1, we are going to create the engine of our YouTube clone. Make sure you can log into the AWS console, and follow the instructions given below.

This is the system we will end up with at the end of this lesson



NOTE: PLEASE CREATE ALL YOUR RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

1. SET YOUR REGION TO US. EAST (N. VIRIGINA)

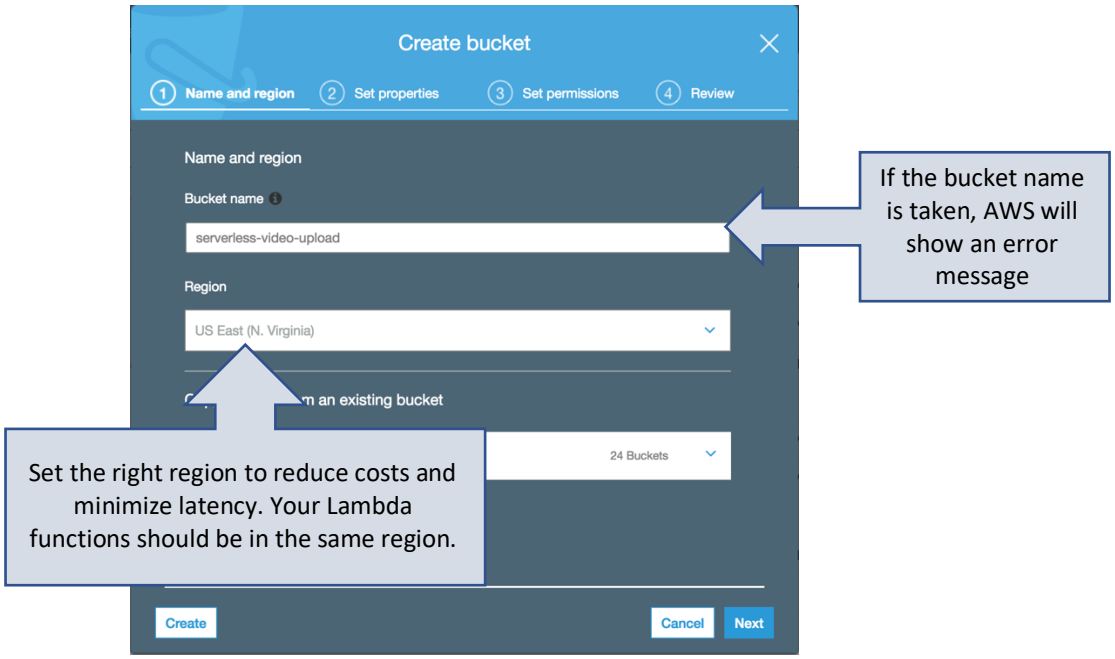
Before we kick-off the build, log in to the AWS console, and set your region to US East (N. Virginia).

Please make sure that all resources & services you create are in the same region from here on.

2. CREATE 2 S3 BUCKETS

Let's begin by creating two buckets in S3. The first bucket will serve as the upload bucket for new videos. The second bucket will contain transcoded videos put there by the Elastic Transcoder.

- To create a bucket, in the AWS console click on **S3**, and then click ***Create Bucket***.
- Enter a **Bucket Name** (e.g. *serverless-video-upload*), and choose the **region**: **US East (N. Virginia)**.
    - *Note: S3 bucket names are global. The above is just an example (and will almost certainly not be available.) Please make your bucket name unique.*
- Click **Create** to save your bucket.
- Repeat the process again to create another bucket (e.g. *serverless-video-transcoded*).
- Make a note of the bucket names, as you will be using them throughout this workshop.



3. MODIFY BUCKET POLICY

We need to make our transcoded videos publicly accessible.

- In S3 click on the **second** bucket you have created (this will be the serverless-video-transcoded bucket).
  - *Remember, the bucket name you chose above will need to be unique.*
- Click on the **Permissions** tab
- Click **Bucket Policy**
- Enter the following to the bucket policy (you can copy below text form step3-bucket-policy.txt):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AddPerm",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<YOUR-BUCKET-NAME>/*"
        }
    ]
}
```

Make sure to substitute <YOUR-BUCKET-NAME> with the actual name of your serverless-video-transcoded bucket.

- You will get a warning about the bucket being public. For the sake of this lab, this is ok.
- Click **Save**

Bucket policy editor ARN: arn:aws:s3:::[redacted]

Type to add a new policy or edit an existing policy in the text area below.

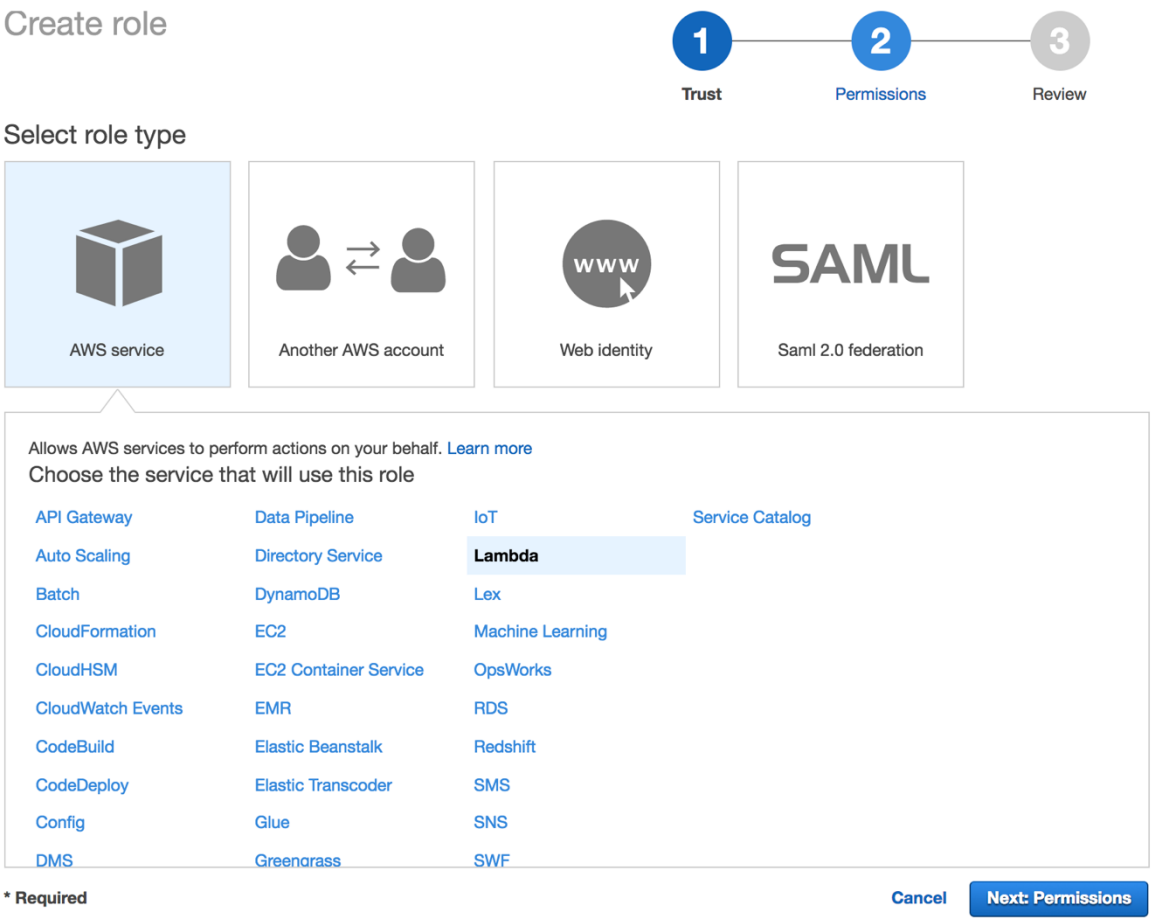
Delete Cancel Save

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AddPerm",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::serverless-video-transcoded/*"
        }
    ]
}
```

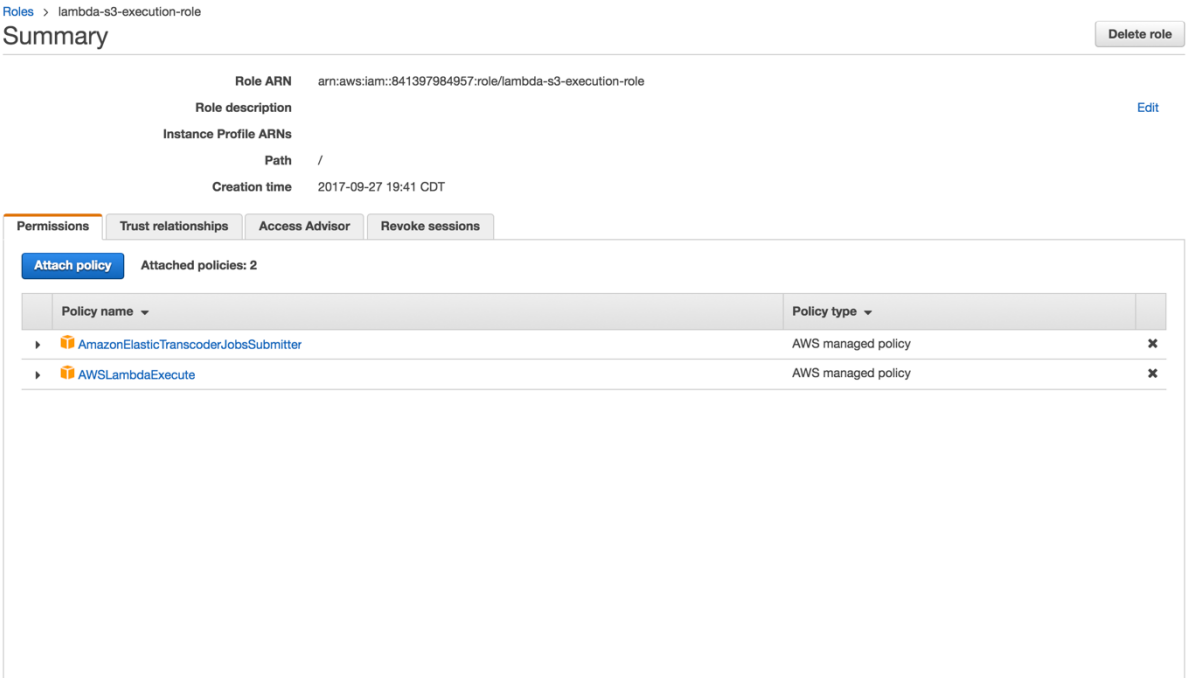
4. CREATE AN IAM ROLE FOR YOUR FIRST LAMBDA FUNCTION

Now we need to create an IAM role for our future Lambda functions. This role will allow functions to interact with S3 and the Elastic Transcoder.

- In the AWS console's **Services** tab, click **IAM** under **Security, Identity & Compliance,** and then click **Roles** from the left navigation menu.
- Click **Create Role**
- In the **Trust** step, choose **AWS Service** and **Lambda**, and then click **Next: Permissions**



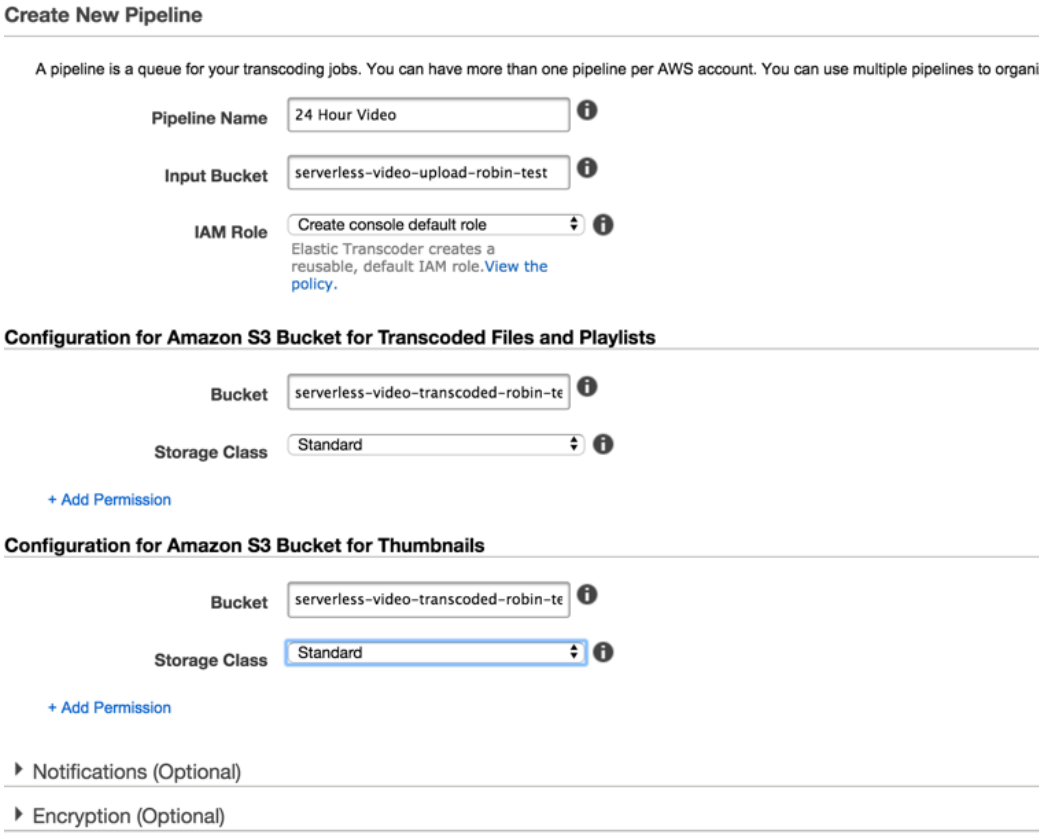
- In the Permissions step, search for and check the boxes next to:
    - **AWSLambdaExecute**
    - **AmazonElasticTranscoderJobsSubmitter**
    - **Note: Make sure the names you select match exactly what is shown here.**
- Click **Next: Review** to attach both policies to the role.
- In the Review step, name the role **lambda-s3-execution-role**, and then click **Create role** to save.
- You will be taken back to the role summary page. Click **lambda-s3-execution-role** again to see the two attached policies:



5. CONFIGURE ELASTIC TRANSCODER

Now we need to set up an Elastic Transcoder pipeline to perform video transcoding to different formats and bitrates.

- In the AWS console's **Services** tab, click on ***Elastic Transcoder*** under **Media Services**, and then click ***Create a New Pipeline***.
- Give your pipeline a **name**, such as *24 Hour Video*, and specify the **input bucket**, which in our case is the first bucket, (e.g. *serverless-video-upload*).
- Leave the IAM role as it is. Elastic Transcoder creates a default IAM role automatically.
- Under **Configuration for Amazon S3 Bucket for Transcoded Files and Playlists** specify the transcoded videos bucket, which in our case was *serverless-video-transcoded*.
- Set the **Storage Class** to **Standard**.
- We are not generating thumbnails but we should still select a bucket and a storage class. Use the second bucket, (*serverless-video-transcoded*) again, and once again set the **Storage Class** to **Standard**.
- Click ***Create Pipeline*** to save.
- Make note of the **Pipeline ID**. You'll need it soon.



6. CREATE LAMBDA FUNCTION

It is finally time to create the first Lambda function, although we are not going to provide an implementation for it just yet.

- In the AWS console's **Services** tab, click **Lambda** under **Compute**, and then click ***Create function***.
- Click **Author from scratch**.
- On the **Basic information page, Name** the function *transcode-video*.
- Under **Role**, select **Choose an existing role** and then **lambda-s3-execution-role**.
- Click **Create function**.
- Once the function is created, in the **Basic settings** section, set the **Timeout** to 0 minutes, 30 seconds.
- At the top of the page, click **Save**.

7. PREPARE & DEPLOY LAMBDA

Finally, we can have a look at the actual Lambda function and deploy it to AWS.

- **Install npm packages**
  In the terminal / command-prompt, change to the directory of the function:

cd lab-1/lambda/video-transcoder

Install npm packages by typing:

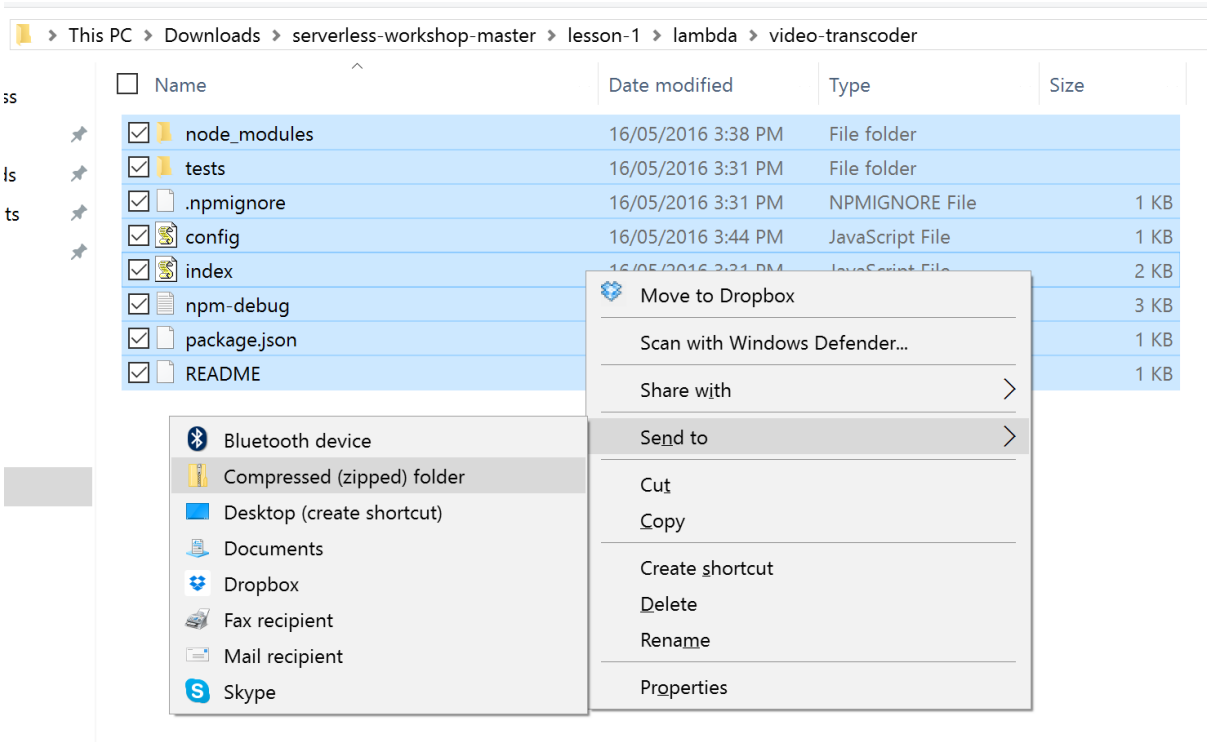
npm install

- **Zip Lambda function**

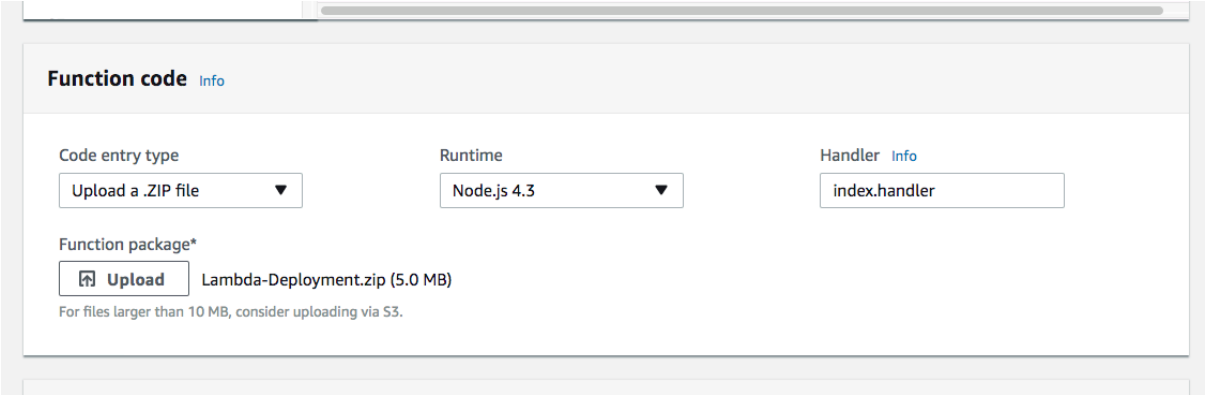
For OS X / Linux Users
Now create create a ZIP file of the function, by typing:

npm run predeploy

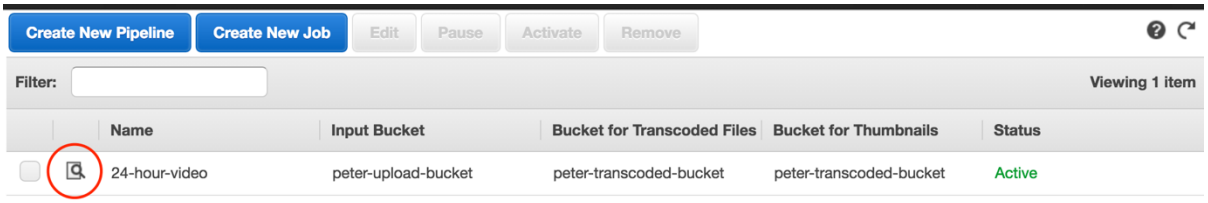
For Windows
You will need to **zip up all the files** in the **lab-1/lambda/video-transcoder** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the video-transcoder folder. Zip up the files inside of it**).



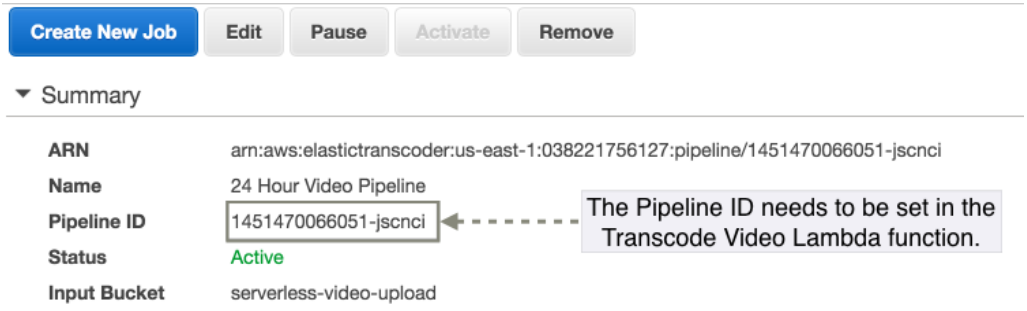
- Back in the AWS control panel, in the configuration for the *transcode-video* Lambda function:
- Under the **Function code** section, change the **Runtime** to **Node.js 4.3**.
- Set the **Code entry type** to **Upload a .ZIP file**.
- Click **Upload**:



- Select the .ZIP file of the Lambda function you created earlier.
- Scroll down to the **Environment variables**.
    - Add an environment variable with *Key* **ELASTIC_TRANSCODER_REGION** and set its *Value* to **us-east-1 (must be lower case)**
    - Add another environment variable with *Key* **ELASTIC_TRANSCODER_PIPELINE_ID** and set its *Value* to be to your Elastic Transcoder pipeline ID from step 5. (you can find it in the Elastic Transcoder console by clicking on the details icon):

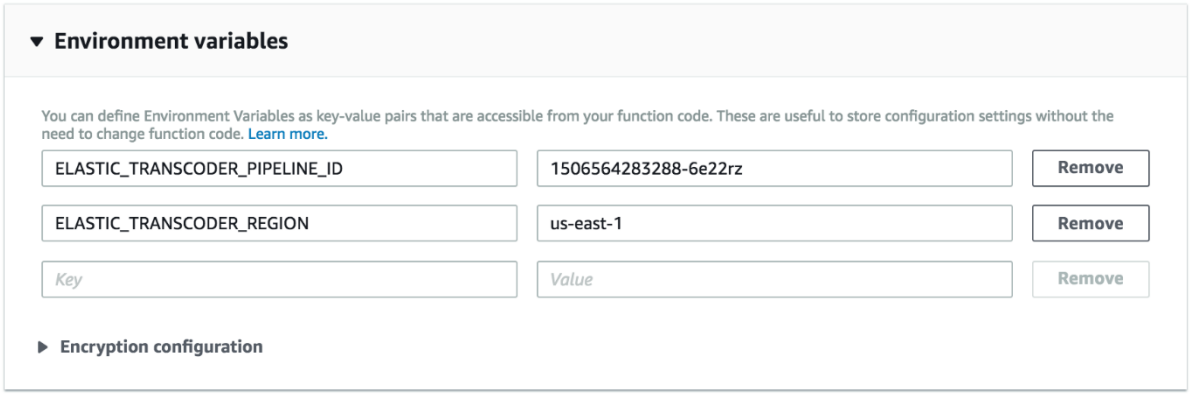


This is the Pipeline ID you need to copy into the environment variable above.



You need to get your Pipeline ID and add it to the function

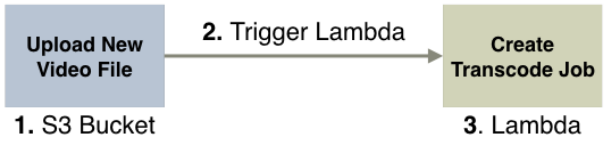
Your environment variables should look a bit like this, but the pipeline id of your elastic transcoder pipeline will be different from the one shown.



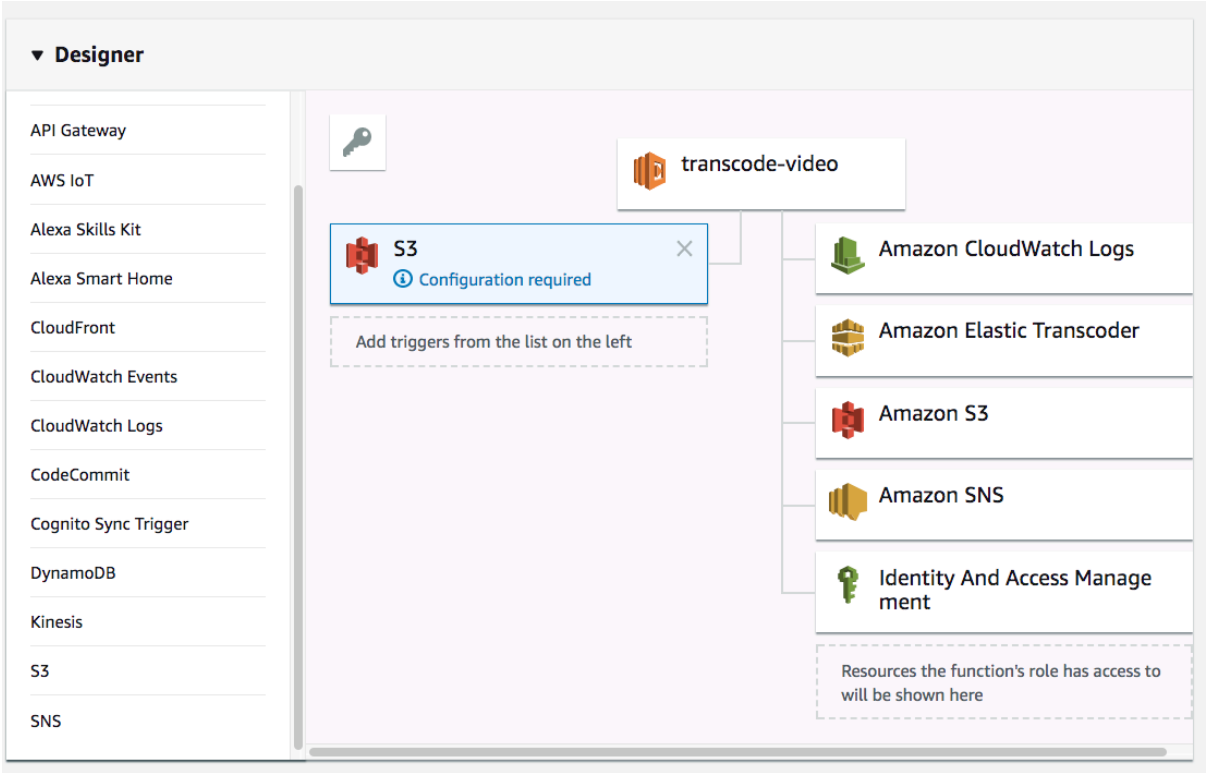
- Click the **Save** button at the top of the page to upload the function and set the environment variables.

8. CONNECT S3 TO LAMBDA

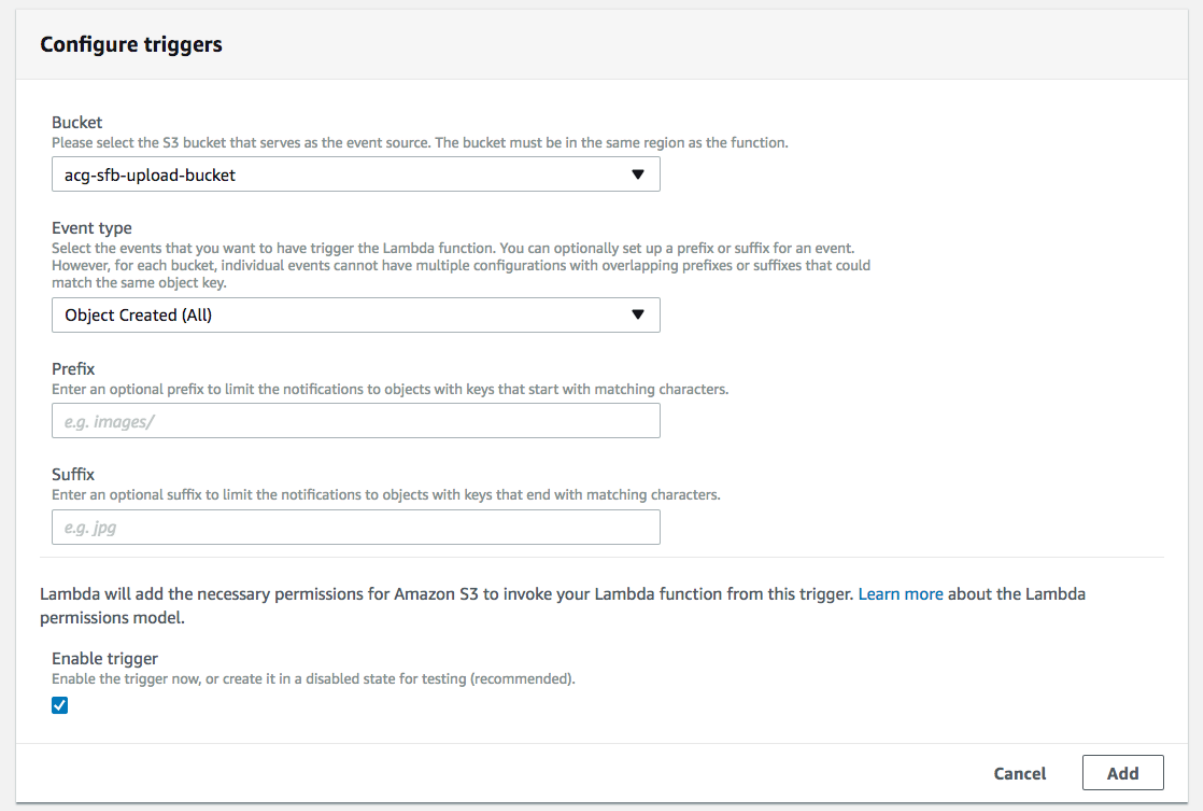
The last step before we can test the function in AWS is to connect S3 to Lambda. S3 will invoke our lambda function when a new video is uploaded:



- On the same page, scroll up to the **Designer** section
- Click on **S3** in the **Add triggers** list on the left

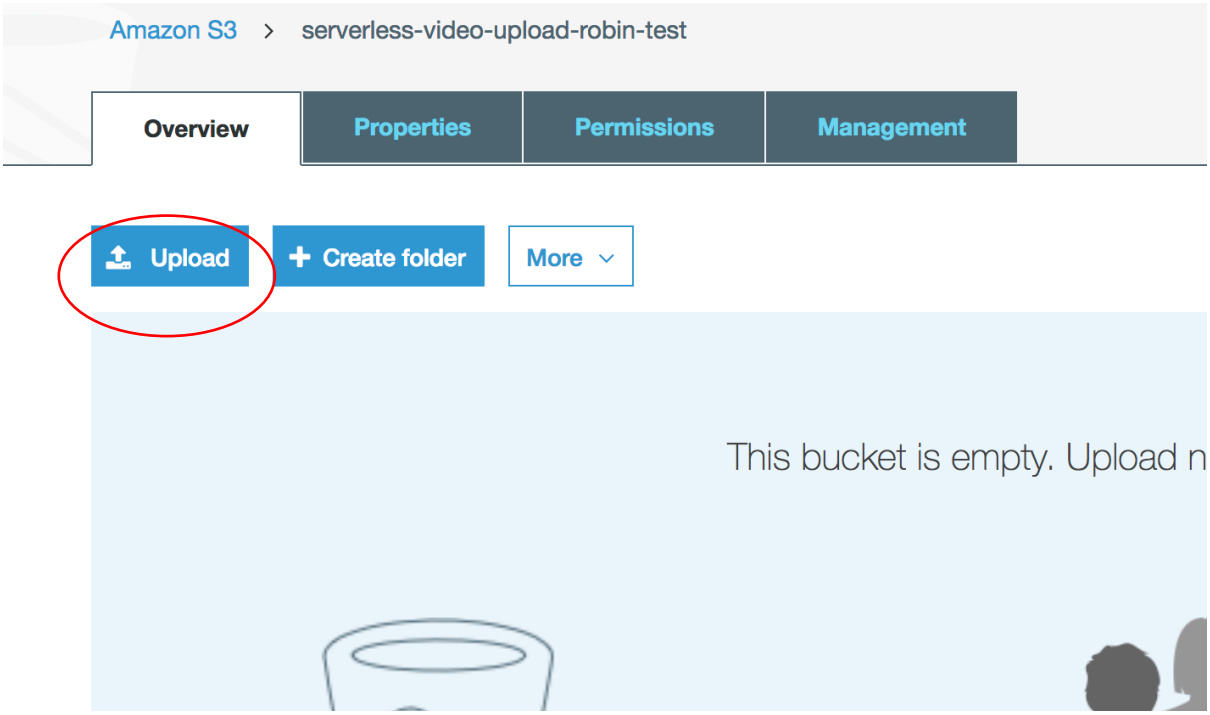


- Scroll down to the **Configure triggers** section
- Select the upload bucket (e.g. *serverless-video-upload*).
- In the event type dropdown, select ***Object Created (All)*.**
- Press **Add**
- Now click on the **Save** button up the top and AWS will link your s3 bucket and lambda function.

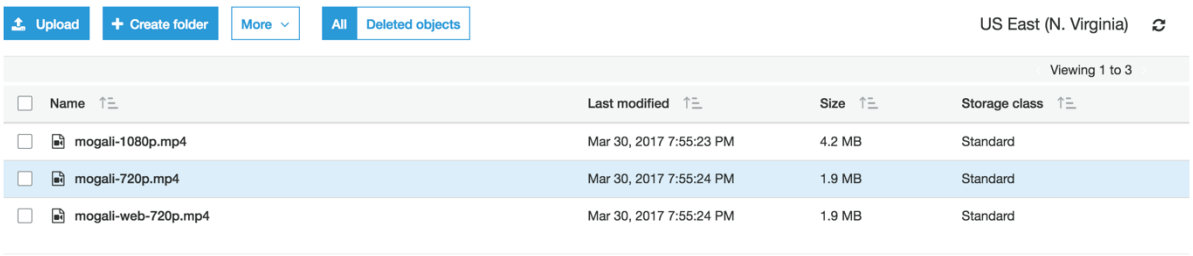


9. TESTING IN AWS

To test the function in AWS, upload a video to the upload bucket.
In the root directory of the serverless-workshop, there's a sample-videos.zip containing videos you can use to test the transcoder.
To do this, go to **S3**, navigate to the *upload* bucket, and then select ***Upload***:



- Click **Add Files**, select a video file (an .avi, .mp4, or .mov), and click **Upload**. The file you selected should appear in the *upload* bucket.
- Navigate to the *transcoded* bucket, and after a short period of time (long enough to grab a cup of coffee, not long enough for a proper nap), you should see three new videos. These files will appear in a folder rather than in the root of the bucket:



Congratulations – you now have your very own serverless video transcoding pipeline!

Important: Make sure that the files appear in the transcoded bucket before moving on to the next lesson. If they don't appear after a few minutes, double check each of the steps above.

Get Your Hands Dirty

At the moment, *24-Hour Video* is functional but it has a number of limitations that have been left for you to solve as an exercise. See you if you can implement a solution for the following problems:

1. A file with more than one period in its name (for example, *Lecture 1.1 – Programming Paradigms.mp4*) is going to produce transcoded files with truncated names. Implement a fix it so that filenames with multiple periods work.

2. Currently, any file uploaded to the upload bucket will trigger the workflow. The Elastic Transcoder, however, will fail if it's given invalid input (for example, a file that is not a video). Modify the first Lambda function to check the extension of the uploaded file and only submit avi, mp4, or mov files to Elastic Transcoder. Any invalid files should be deleted from the bucket.
3. The files in the upload bucket are going to remain there until you delete them. Come up with a way to clean up the bucket automatically after 24 hours. You might want to have a look at the Lifecycle options in S3 for ideas.
4. The current system creates three transcoded videos that are very similar. The main difference between them is the resolution and bitrate. To make the system more varied, add support for HLS and webm formats.

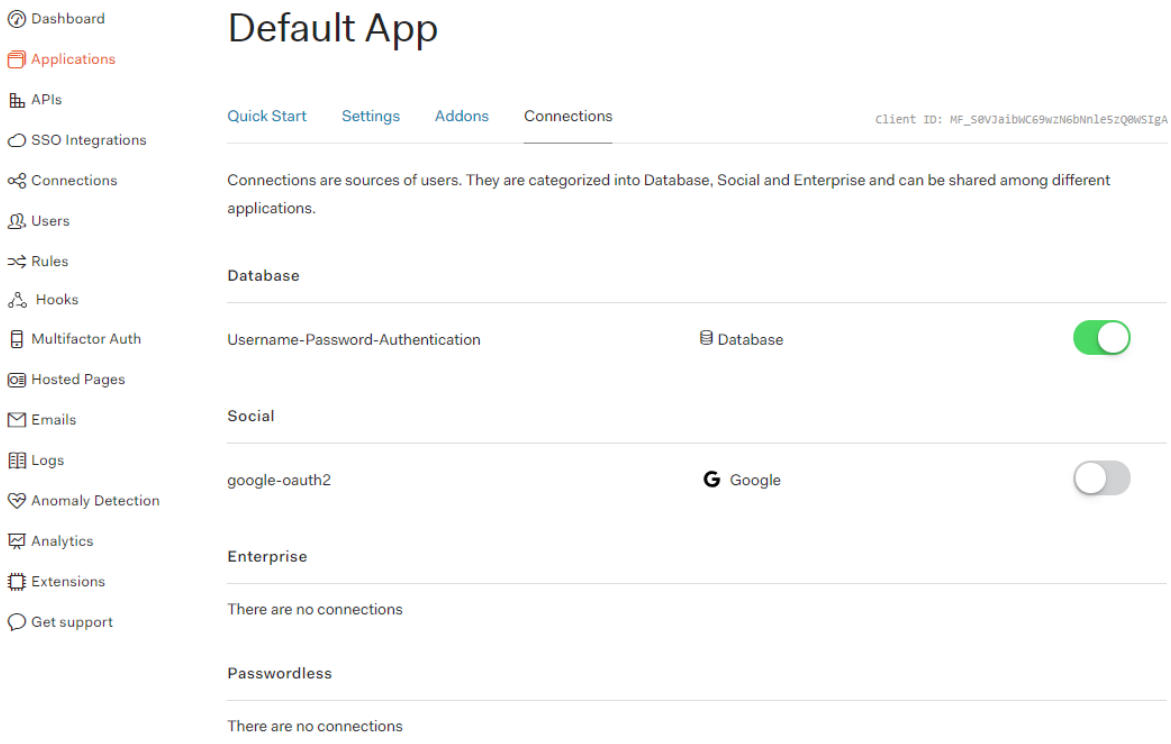
LESSON 2

In this lesson, we'll create a website for our video hosting platform. We'll integrate the website with Auth0, as the means of authenticating users.
In true serverless style, this will be a static website, meaning that it can be hosted on S3, or any CDN. It is comprised entirely of static HTML, JS, and CSS and does not need to be served by a traditional web server.

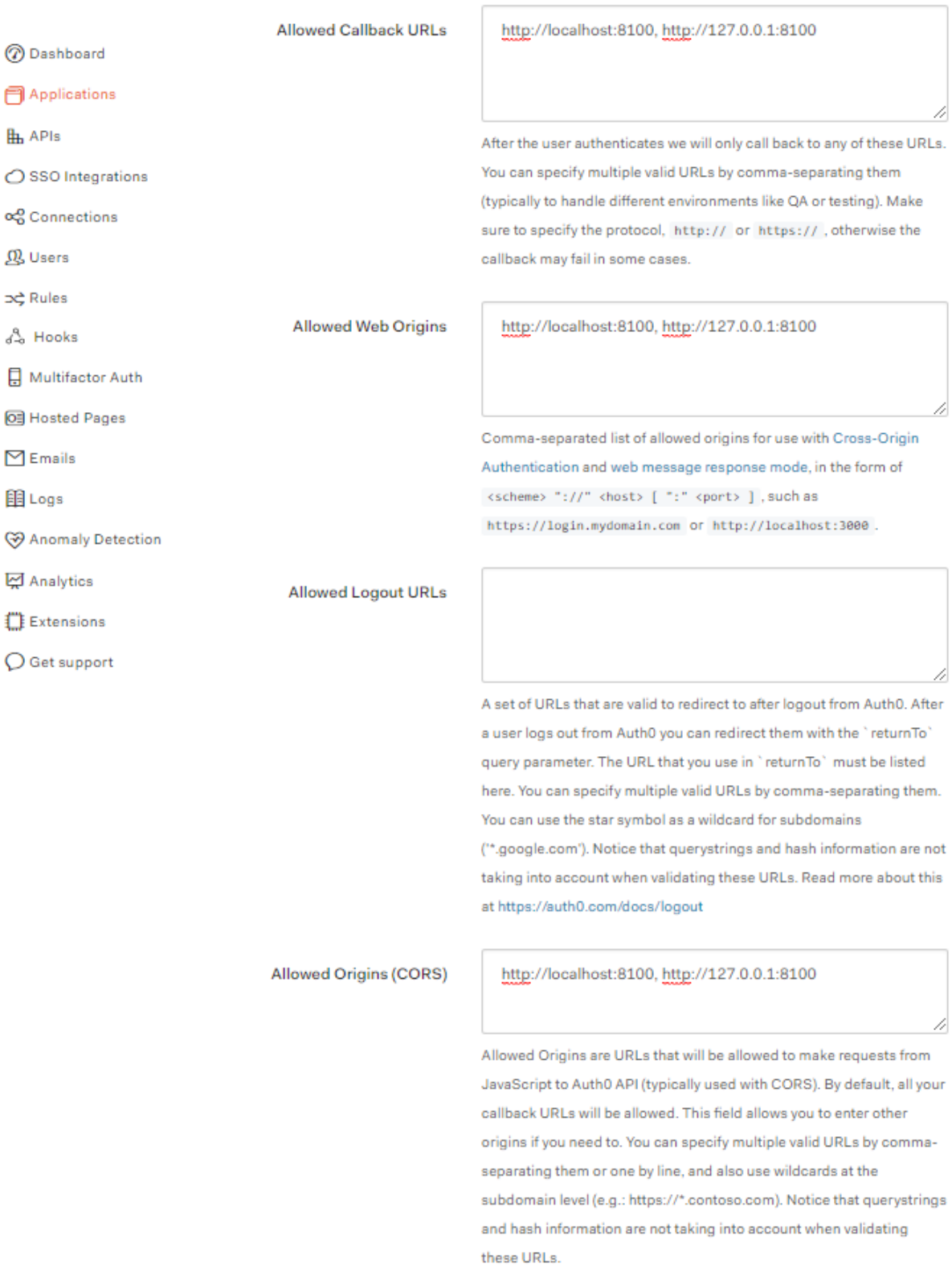
10. CREATE AUTH0 ACCOUNT

You'll need to create a free auth0 account. Visit https://auth0.com and follow the sign up steps.

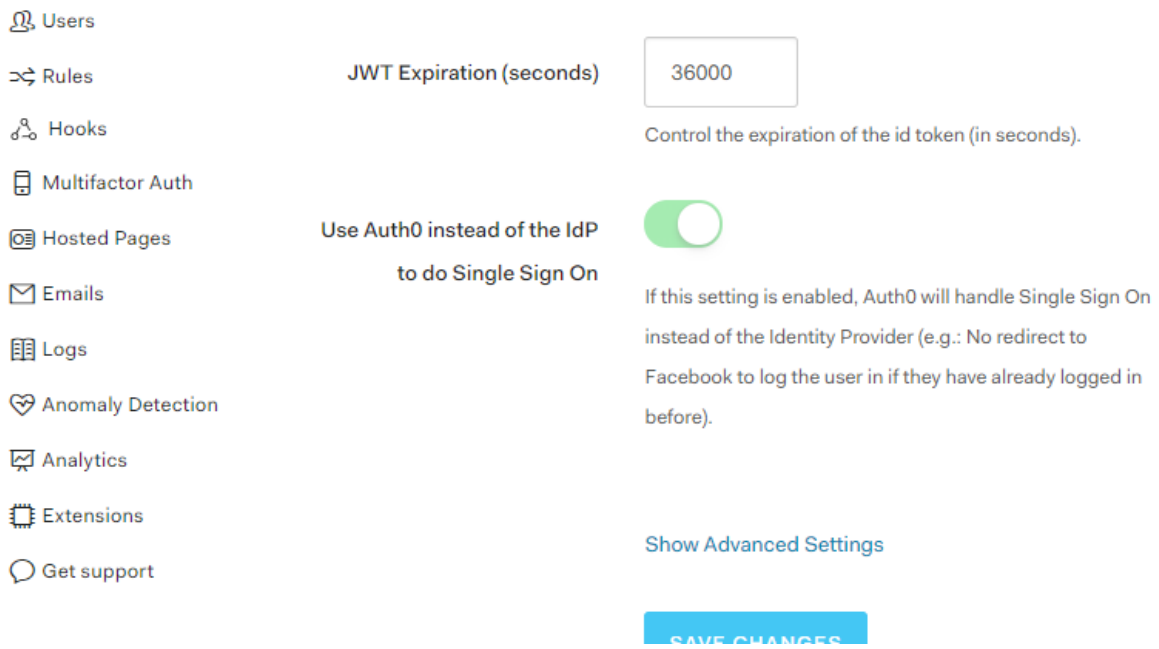
- You'll be asked to enter a **tenant domain**. Enter a name that is unique to you, e.g. janesmith-24hrvideo.auth0.com
- Enter **"US"** as your **region**.
- Click **Next** and fill out the information on the next page.
- Click **Create Account**.
- Go to **Applications** in the left navigation menu, and click on the **Default App**.
- Go to **Connections** in the **Default App** menu, and make sure that only **Username-Password-Authentication** is enabled.



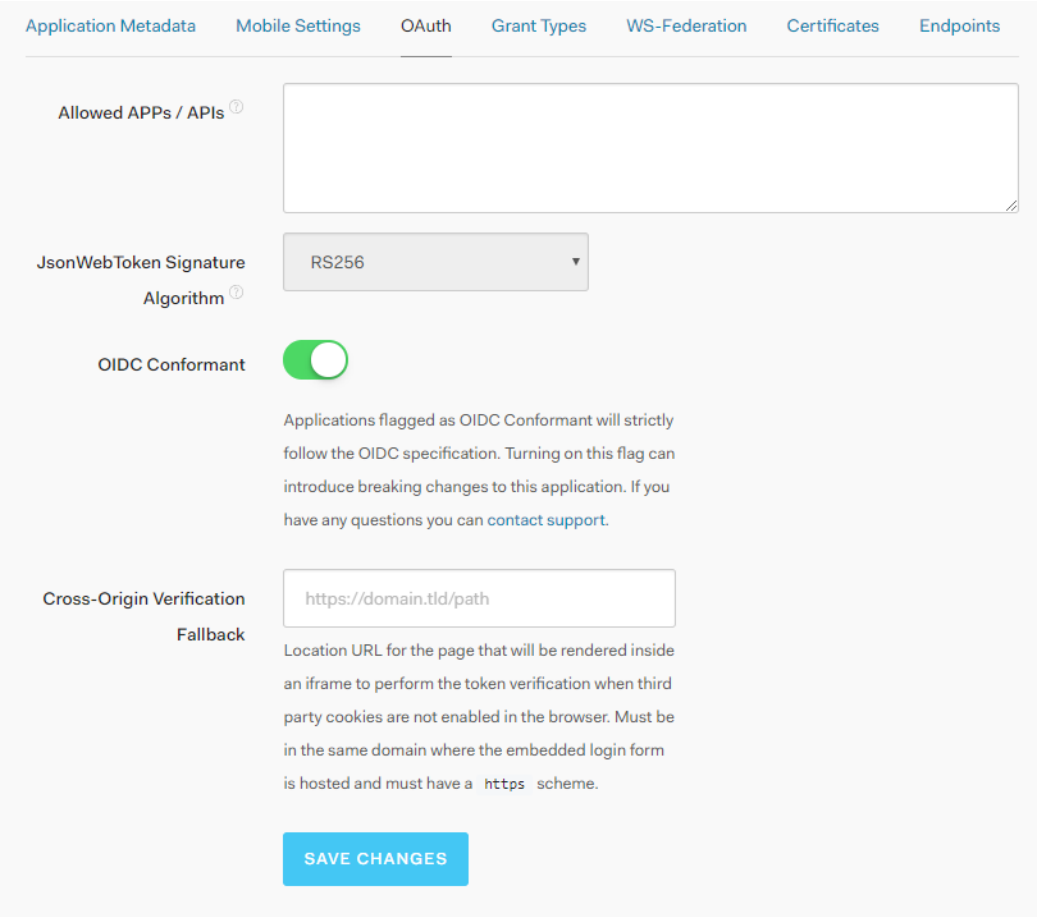
- Go to **Settings** in the **Default App** menu.
- Scroll down until you find the textbox called **Allowed Callback URLs**.
  Enter the following value in the textbox: **http://localhost:8100, http://127.0.0.1:8100**
  - Enter the same value into the following fields: **Allowed Web Origins, Allowed Origins (CORS)**



- Scroll down to the bottom, and click the **Show Advanced Settings** link.

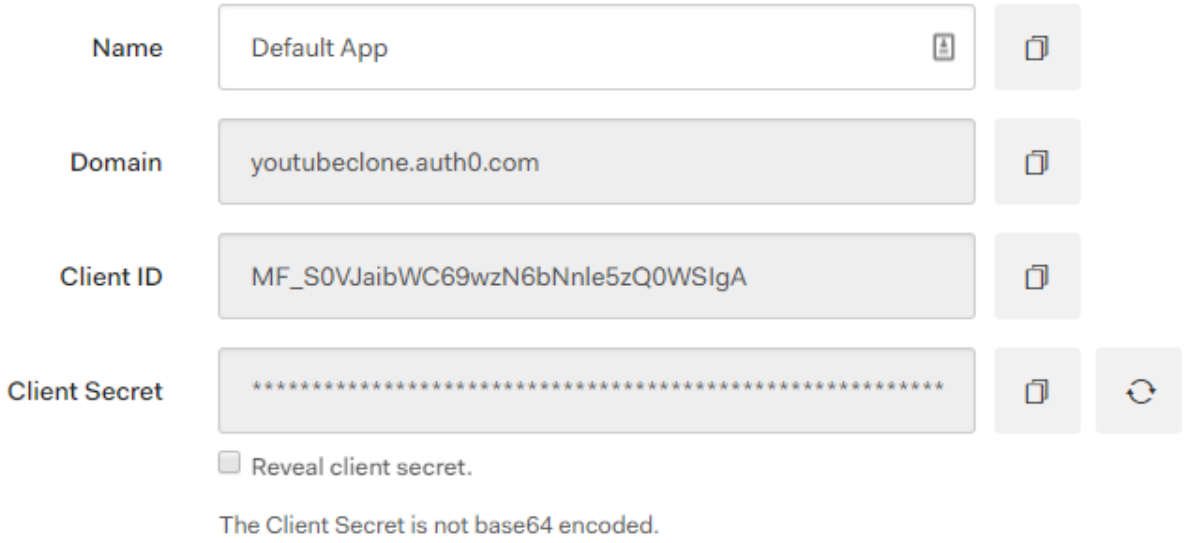


- Under **Advanced Settings**, choose the **OAuth** menu, and ensure **JsonWebToken Signature Algorithm** is set to **RS256**.
- In the same section, make sure the **OIDC Conformant** option is green.



- Scroll down and click the **Save Changes** button.

- We now need to retrieve some values from Auth0 that will be needed throughout this workshop. Scroll up to the top of the same **Settings** page, and find the **Domain**, **Client ID** and **Client Secret**. Copy these into your favourite text editor so you have them at the ready.



11. SETUP WEBSITE LOCALLY

This web site would normally be deployed via a CDN, but for the purposes of this workshop we're going to host it locally on your computer.

- Edit the following file in your favourite text editor:

  lab-2/website/js/config.js

  Enter your **auth0 domain** and **client ID**, in quotes (you made a note of these in the last step), and save the file.

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

  lab-2/website

- Run the following command, to bring down dependencies from npm (this may take a few minutes):

  npm install

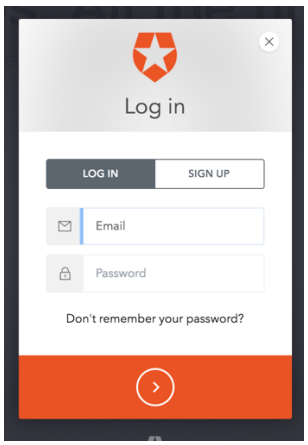
- Run the following command, to start a local web server at port 8100:

```
npm start
```

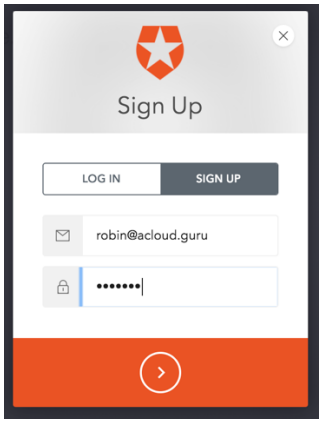
12. GIVE IT A SPIN!

- Open a web browser and navigate to:
  http://localhost:8100

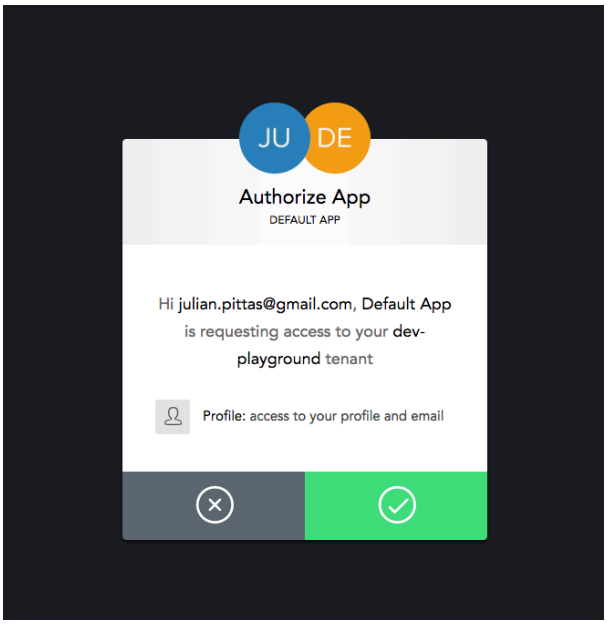
  You should see the 24 hour video web site. There's not much here yet… that's OK! We're going to iteratively build the site during the workshop.
- Notice the **Sign In** button in the top-right? Click on it to launch the authentication popup:
  This popup is rendered entirely by Auth0 – a huge timesaver if you need authentication in your platform!
  - You'll need to create an account, so click on the **Sign Up** tab:



- Enter an email address and password for your new account. This will be saved to your custom Auth0 database of users. **Remember this password**, because you'll need to sign in/out multiple times throughout this workshop



- Click the big orange button at the bottom to create your account. A popup will launch to complete the sign up.



- Click on the **green tick**
  Did you receive an error? Make sure you Always Allow Popups for this site.

- You'll be automatically signed in after your account is created. Look in the top right-hand corner of the web-site. You should see your name & a profile image / avatar (auth0 will use gravatar.com to find an image for your email address), plus a **Sign Out** button.
- When you're done with this lab, exit the "npm start" command in your terminal by pressing **<Control>-c**.

Congratulations – you now have a serverless web site with full user sign-up and authentication capabilities!

Get Your Hands Dirty

- Now use Auth0 to hookup a 3rd party social provider, such as Facebook or Twitter. Note: You will need to create an app with each provider that you hookup. Auth0's website has instructions.
- Auth0 supports running node.js rules on each user login. Add a rule to:
  - Force email verification (there is a pre-built Auth0 rule for this)
  - Only allow users from a specific white-list
- Auth0 supports the creation of delegation tokens to grant user's direct access inside your AWS account via IAM. It's worth understanding that this is possible and how it works. Read through the Auth0 documentation on this approach, and if you are game setup your web site to get an AWS delegation token.
  https://auth0.com/docs/integrations/aws

LESSON 3

In this lesson, we will create a User Profile Lambda function. This function will talk to Auth0 and retrieve information about the user. We will also set up an API Gateway. The API Gateway will allow our website to invoke the function.
Lastly, we will create a custom authorizer. A custom authorizer is a special Lambda function that the API Gateway executes to decide whether to allow or reject a request. We will use this custom authorizer to make sure that only authenticated users have access to the User Profile Lambda function.

NOTE: PLEASE CREATE ALL YOUR RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

13. SET UP THE USER PROFILE LAMBDA FUNCTION

Let's get our User Profile Lambda function organized first.

- **Install npm packages**
  In the terminal / command-prompt, change to the directory of the function:

  ```
  cd lab-3/lambda/user-profile
  ```

  Install npm packages by typing:

  ```
  npm install
  ```

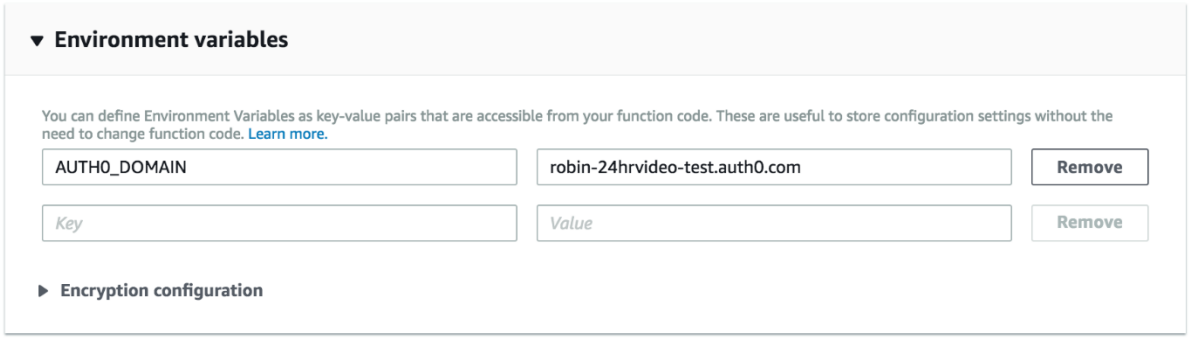
- **Zip Lambda function**

  For OS X / Linux Users
  Now create create a ZIP file of the function, by typing:

  ```
  npm run predeploy
  ```

  For Windows
  You will need to **zip up all the files** in the **lab-3/lambda/user-profile** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the user-profile folder. Zip up the files inside of it**).

- In the AWS console's **Services**, click **Lambda** under **Compute**, and then click the ***Create function*** *button.*
- Click the **Author from scratch** button.
- **Name** the function **user-profile.**
- Set the **Runtime** to **Node.js 4.3**.
- Under **Role** select **Choose an existing** role and ***lambda-s3-execution-role***.
- Click **Create function**.
- Once the function is created, scroll down to the **Basic settings** section, set the **Timeout** to 0 minutes, 30 seconds.
- For **Code entry type,** select **Upload a .ZIP file.**
- Click **Upload**, and choose the zip file you just created: **/lab-3/lambda/user-profile/Lambda-Deployment.zip**
- Expand the **Environment variables** section, and create an environment variable with the key **AUTH0_DOMAIN** and set its value to the Auth0 domain from the last lesson.

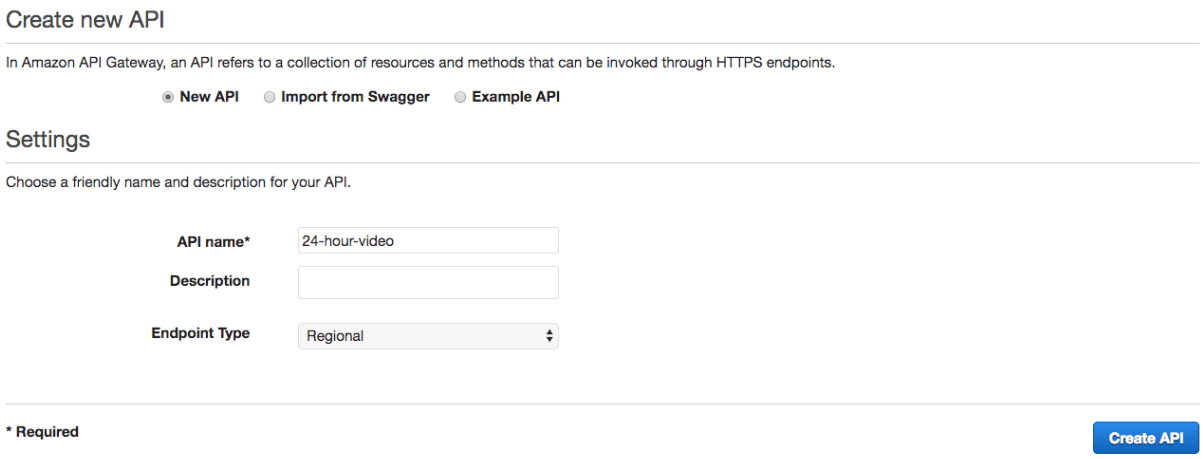


- Click *the **Save** button at the top of the page.*

14. CREATE THE API GATEWAY

The API Gateway needs to be set up to accept requests from our website. We need to create a resource, add support for a GET method, and enable Cross-Origin Resource Sharing (CORS). In the AWS console follow these steps:

- In the AWS Console's **Services** tab, click on **API Gateway** under **Networking & Content Delivery**.
- If this is your first API Gateway, click the **Get Started** button. Otherwise, click the **Create API** button.
- Select the **New API** radio button.
- Type in a name for your API, such as **24-Hour-Video** and optionally, a description.
- Set **Endpoint Type** to **Regional**
- Click **Create API** to create your API:

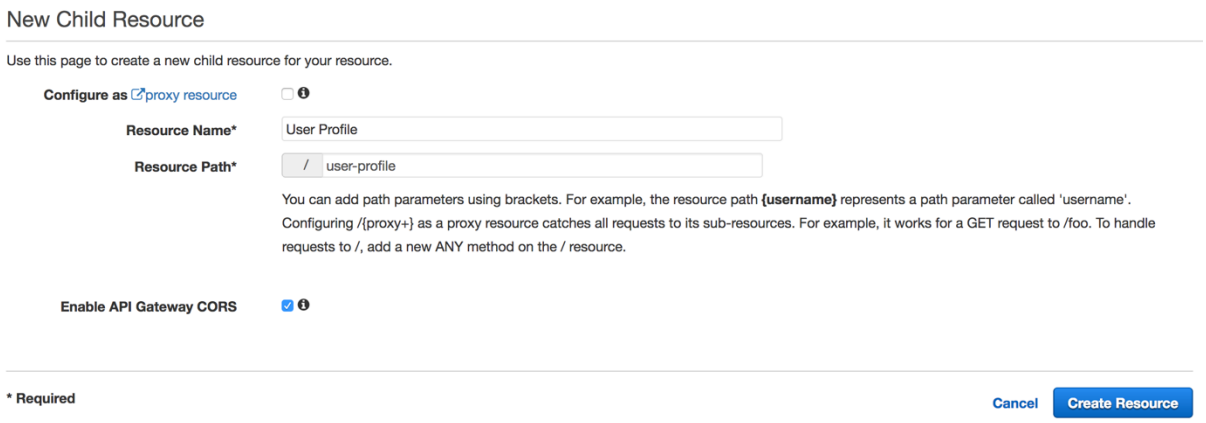


15. CREATE RESOURCE AND METHOD

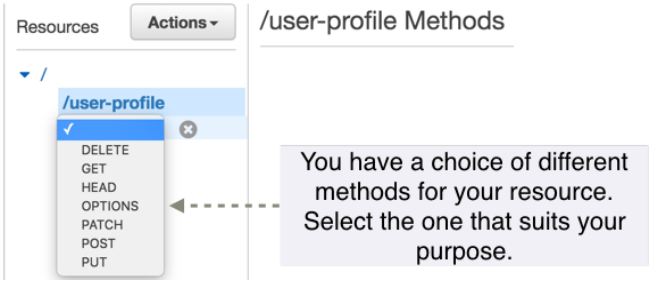
APIs in the Gateway are built around *Resources*. We are going to create a Resource called *user-profile* and combine it with a GET method.

- In the **Resources** tab in the left navigation menu, select the **Actions** dropdown and click **Create Resource**.
- Type "User Profile" in the **Resource Name**. The **Resource Path** should be automatically filled in.
    - Note: Since the website will connect to the resource path */user-profile*, this value must match exactly.
- Tick the **Enable API Gateway CORS** box.

- Click the **Create Resource** button.

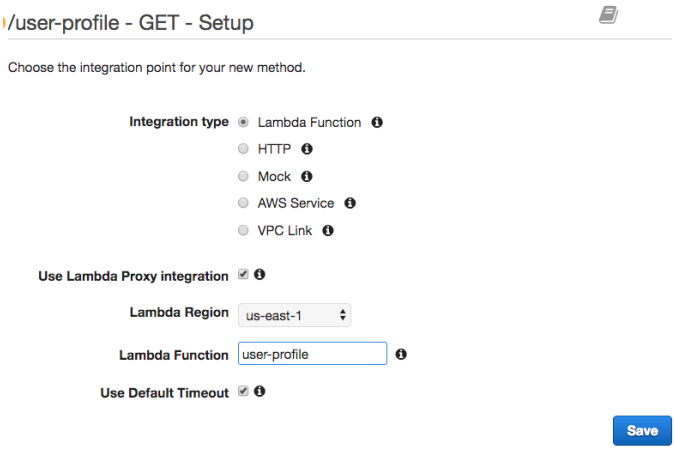


- The left-hand side list should now show **/user-profile**. Click it and then select the **Actions** dropdown again, and click the **Create Method** button to see a small dropdown under **/user-profile**.
- From that dropdown select **GET** and click the button with the **tick/check mark** on it to confirm.



Having created the GET method, we need to configure the *Integration Point*. This is the screen you should be looking at right now. (If you are not on it, click **GET** under **/user-profile).** An Integration Point specifies which Lambda function (or HTTP endpoint) the API Gateway should invoke.

- Click the **Lambda Function** radio button.
- Tick the checkbox labeled **Use Lambda Proxy Integration**.
- Select your region (**us-east-1**) from the **Lambda Region** dropdown.
- Type ***user-profile*** in the **Lambda Function** text box.



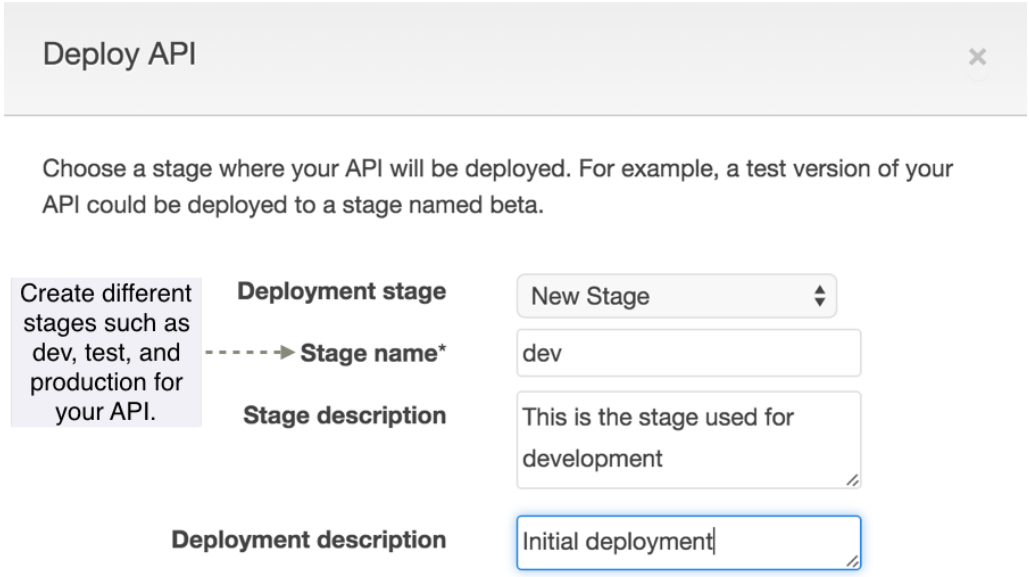
- Click **Save**.

- Click **OK** if you are asked if it's ok to add permission to the Lambda function.

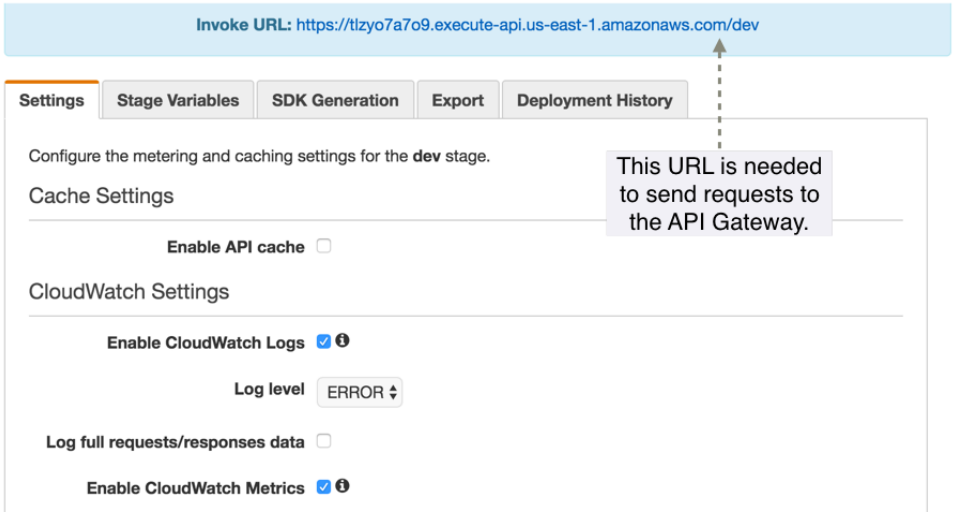
16. DEPLOY

Next, we need to deploy the API and get a URL to invoke it from the website.

- You should still be on your new API within API Gateway.
- Select the **Actions** dropdown again.
- Click **Deploy API**.
- In the **Deploy API** popup select **[New Stage]**.
- Type **dev** as the Stage Name.
- Click **Deploy** to deploy the API.



- The next page you will see will show the **Invoke URL** and a number of settings.
- Copy the **Invoke URL** as you will need it in the next step.



17. UPDATE THE WEBSITE

We need to update the website to invoke the right API Gateway URL.

- Copy the config.js file containing your account specific settings, from the last lesson:
  Copy lab-2/website/js/config.js to lab-3/website/js/config.js

- Now edit the config.js file you just copied (lab-3/website/js/config.js) to add the following line after the **auth0** section:
  apiBaseUrl: 'API GATEWAY INVOKE URL FROM STEP 4'

```
var configConstants = {
    auth0: {
        domain: 'serverless.auth0.com',
        clientId: 'ab1Qdr91xU3KTGQ01e598bwee8MQr'
    },
    apiBaseUrl: 'https://tlzyo7a7o9.execute-api.us-east-1.amazonaws.com/dev'
};
```

Don't forget to save **config.js** when you are done.

18. A NEW ROLE

API Gateway supports custom request authorizers. These are Lambda functions that the API Gateways uses to authorize requests. Custom authorizers can validate a token and return an IAM policy to authorize the request. However, before we begin using custom authorizers we are going to create a different role for it.

- In the AWS console's **Services** tab, click **IAM** under **Security, Identity & Compliance** and then click **Roles** from the left navigation menu.
- Click **Create role**.
- From the **Select role type** step, select **Lambda**, and click the **Next: Permissions** button.
- From the list of policies check the box next to **AWSLambdaBasicExecutionRole**.
- Click **Next: Review**.
- Name the role **api-gateway-lambda-exec-role**.
- Click **Create role** to save and exit.

19. CUSTOM AUTHORIZER

Having created a new IAM role we can begin work on the custom authorizer now.

- **Install npm packages**
  In the terminal / command-prompt, change to the directory of the function:

  cd lab-3/lambda/custom-authorizer

  Install npm packages by typing:

  npm install

- **Zip Lambda function**

  For OS X / Linux Users

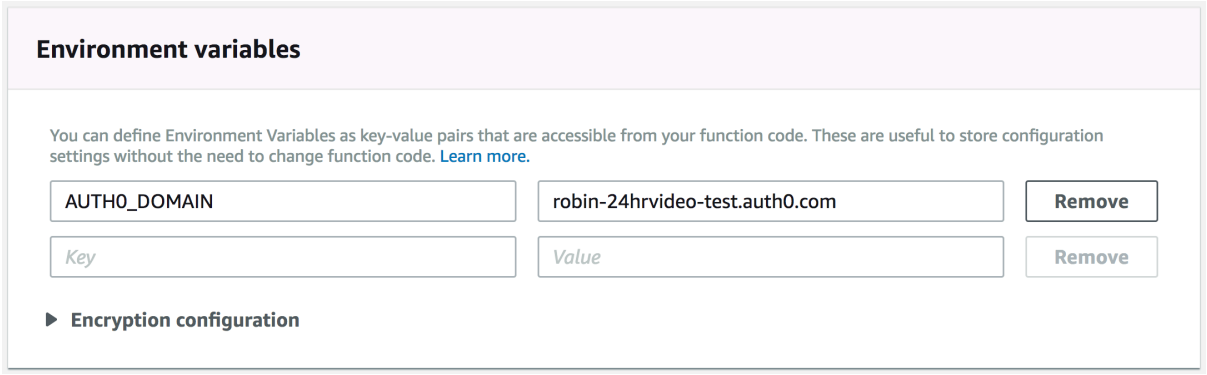
Now create create a ZIP file of the function, by typing:

```
npm run predeploy
```

For Windows

You will need to **zip up all the files** in the **lab-3/lambda/custom-authorizer** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the custom-authorizer folder. Zip up the files inside of it**).

- In the AWS console's **Services** tab, click **Lambda** under **Compute**, and then click ***Create function***.
- Click the **Author from scratch** button.
- **Name** the function **custom-authorizer**.
- Under **Runtime**, select **Node.js 6.10**.
- Under **Role**, select **Choose an existing role** and your new role: *api-gateway-lambda-exec-role*.
- Click **Create function**.
- For **Code entry type,** select **Upload a .ZIP file.**
- Click **Upload**, and choose the zip file you just created: /lab-3/lambda/custom-authorizer/Lambda-Deployment.zip
- Click **Environment variables** to expand it, and create an environment variable with the key **AUTH0_DOMAIN** and set its value to the Auth0 domain from the last lesson.



- Under **Basic settings**:
  - To improve the performance of your function, increase the **Memory (MB)** slider to *1536 MB*. This will also allocate more CPU to your function.
  - Set the **Timeout** to 0 minutes, 30 seconds.
- Click the **Save** button at the top of the page.

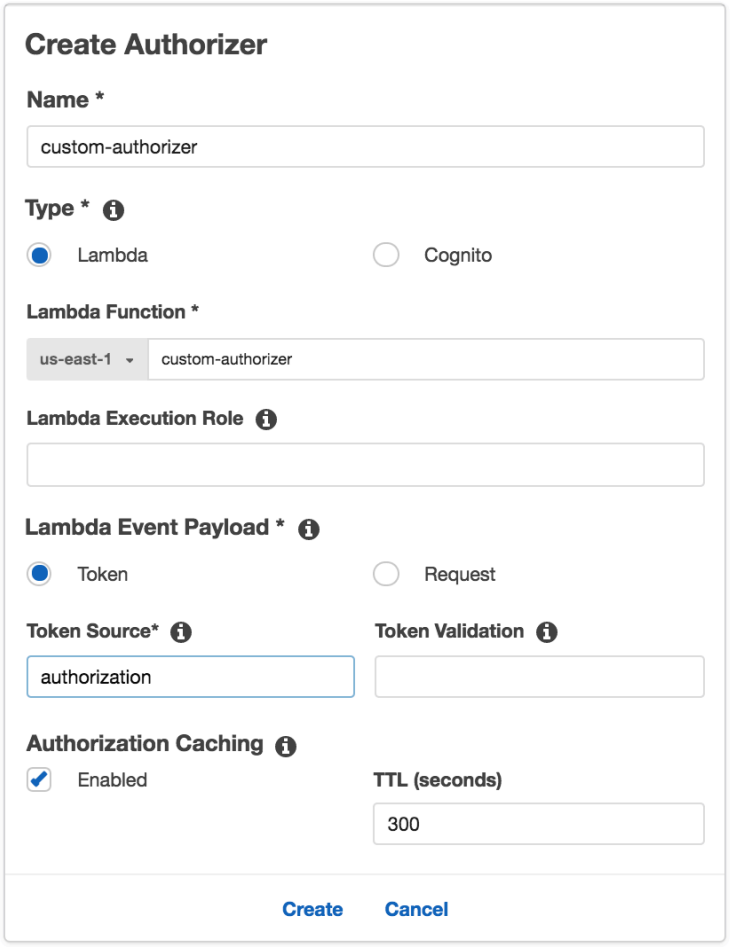
20. ASSIGN CUSTOM AUTHORIZER

Having deployed our custom authorizer function, we need to configure it so that it runs before our User Profile function.

- In API Gateway open the **24 Hour Video** API again.
- Click **Authorizers** from the left navigation menu.
- Click the **Create New Authorizer** button.
- Fill out the **New Custom Authorizer** form:
  - Type in **custom-authorizer** as the name of the Lambda function.
  - Set the region under **Lambda Function** to **us-east-1**
  - Set the **Lambda Event Payload** to **Token**.

- In the **Token Source** box, type "authorization"

- Click **Create** to create the custom authorizer.
- In the **Add permission** popup, click **Grant & Create** to confirm that you want to allow API Gateway to invoke the custom-authorizer function.



To make the custom authorizer invoke when the GET method is called, follow these steps:

- Click **Resources** under 24-hour-video in the left navigation menu.
- Click **GET** under **/user-profile**.
- Click **Method Request**.
- Click the pencil next to **Authorization**.
- From the dropdown select **custom authorizer** and click the tick/check mark icon to save.

  Note: if custom-authorizer isn't in the list, refresh the page
- Click on the **HTTP Request Headers** section to expand it.
- Click on the **Add Header** link, put "authorization" for the name, and click the tick/check mark icon to the right and save it.
- Click on the **URL Query String Parameters** section to expand it.
- Click on the **Add query string** link, put "accessToken" for the name, and click the tick/check mark icon to the right and save it.
- Deploy the API again.

- Select the **Actions** dropdown.
- Click **Deploy API**.
- Select **dev** as the **Deployment Stage**.
- Click **Deploy**.

21. TEST THE SYSTEM

Lesson 3 is complete! Now it's time to test.

- In your terminal or command-prompt, change to the following folder:

  lab-3/website

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

  npm install

- Now run:

  npm start

- Open the web-site in your browser:
  http://localhost:8100

To test whether everything has worked:

- If you're still logged in to your *24 hour video* website, click the **Sign Out** button in the upper right.
- Refresh the page
- Log in to the website by clicking on Sign In button.
- Click the profile button (it'll have your nickname and, possibly, your picture). After a short wait you will see a modal box with your user information.

When you're done with this lab, exit the "npm start" command in your terminal by pressing <Control>-c.
Isn't this fun!? There is actually more goodness to come ☺. See you in the next lesson.

Optional Exercises

Try to do the following exercises to confirm your understanding of concepts presented in this lesson.

1. Create a Lambda function (*user-profile-update*) for updating a user's personal profile. Assume that you can access the first name, last name, email address and the userId on the event object. Because we don't have a database yet, this function doesn't need to persist this information; however, you can log it to CloudWatch.
2. Create a POST method for the /user-profile resource in the API Gateway. This method should invoke the *user-profile-update* function and pass in the user's information. It should use the custom authorizer developed in this lesson.
3. Create a page in the *24 Hour Video* website to allow signed-in users to update their first name, last name, and email. This information should be submitted to the *user-profile-update* function via the API Gateway.

4. Modify the User Profile Lambda function to no longer validate the JSON Web Token. This validation isn't needed due to the custom authorizer. The function should still request user information from the Auth0 *tokeninfo* endpoint.

LESSON 4

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

1. Create a Lambda function to grant us credentials/policy to upload files to the 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.

NOTE: PLEASE CREATE ALL YOUR RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

22. CREATE A LAMBDA FUNCTION

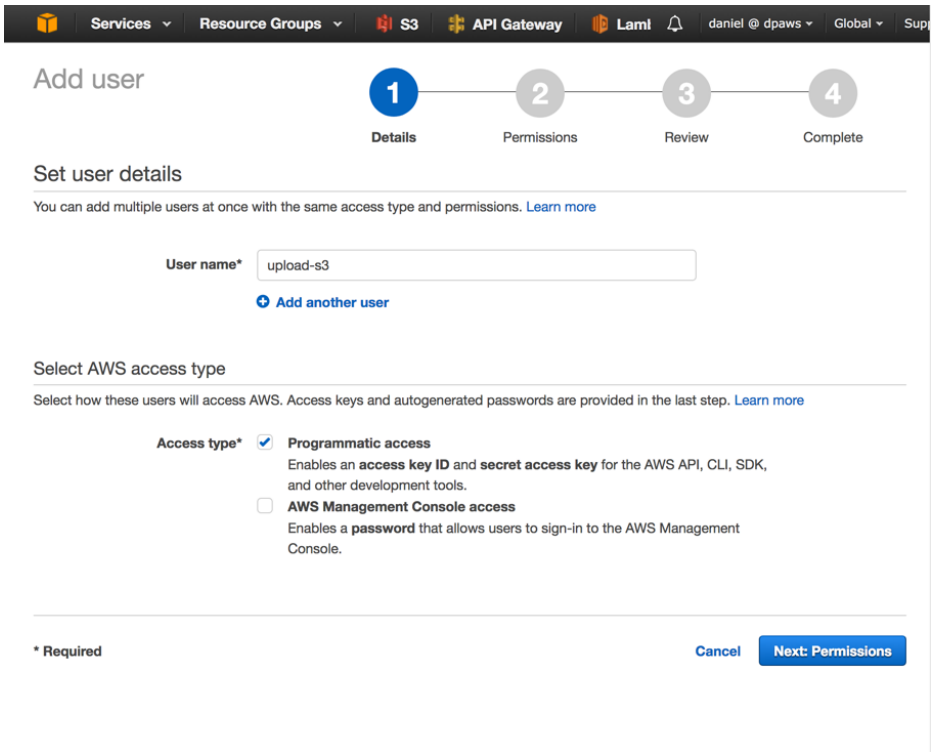
You will need to create a new Lambda function in the AWS console. This Lambda function will generate a policy document to allow your users upload videos to S3. To create the lambda function:

- Go to **Lambda** in the AWS Console.
- Create a new function as you did for your other Lambda functions.
- Name the function **get-upload-policy** and select **Node.js 6.10** as the Runtime.
- Assign the **lambda-s3-execution-role** policy to it (the same policy created in lesson 1).
- Set the **Timeout** to 30 seconds.
- Leave all other settings as their default values and **Save** the function.

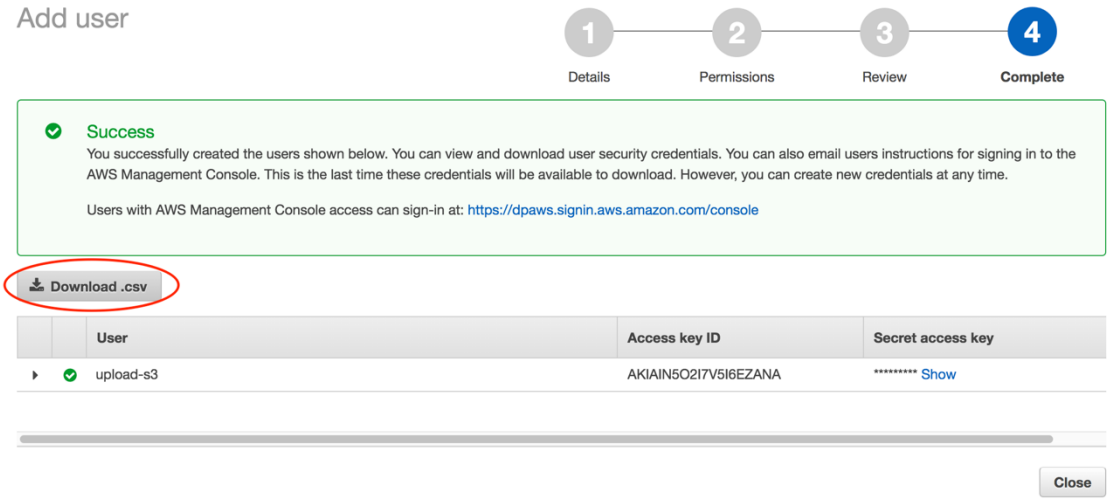
23. CREATE IAM USER

The policy and 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.

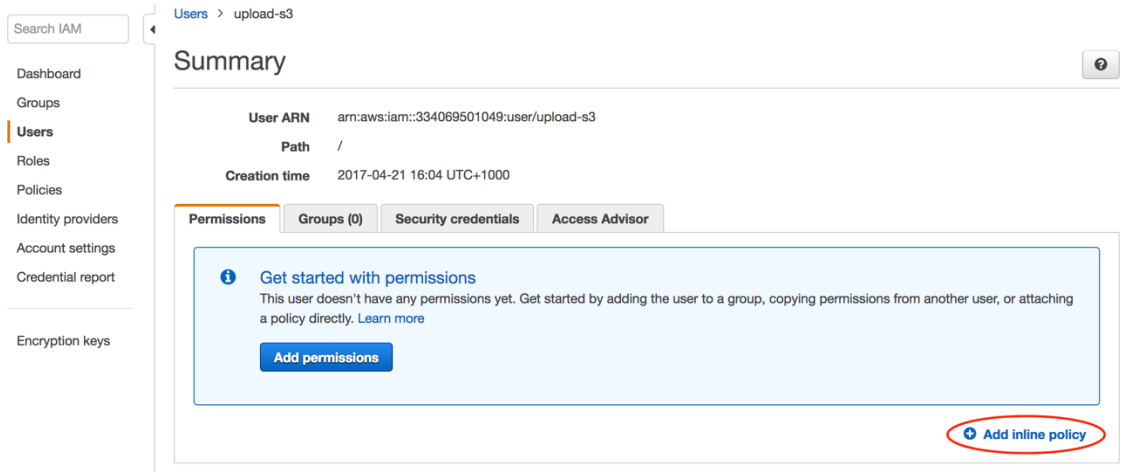
- Go to **IAM** in the AWS console.
- Click **Users** in the left navigation menu, then click the **Add user** button in the top left.
- Set the username to **upload-s3** and check the box labelled **Programmatic access**



- Click **Next: Permissions** and then skip adding permissions at this time by clicking **Next: Review.**
- Ignore the warning that *This user has no permissions* and click **Create user**.
- You will then be shown the following screen where you must download the user's Access key id and Secret access key as a CSV: click the **Download .csv** button and save the **credentials.csv** file to your computer.
- Click the **Close** button.



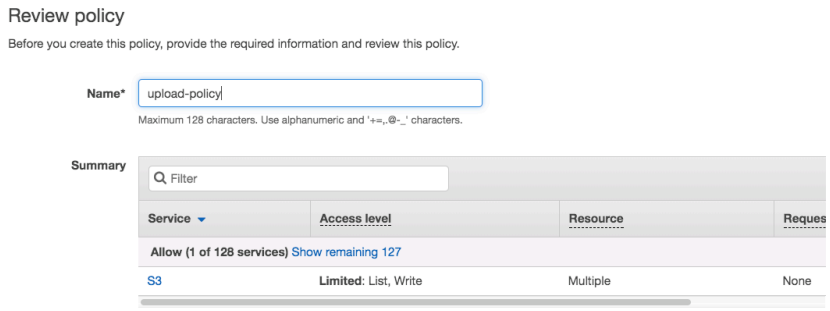
- Click the **upload-s3** user and click the **Permissions** tab.
- Click **Add inline policy**



- To create a new **Inline Policy**, select **Custom Policy**, and click **Select**.
- Click on the **JSON** tab
- 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/*"
            ]
        }
    ]
}
```

- Click on the **Review policy** button
- Set the name of the policy to **upload-policy**.
- Click Create policy



24. CONFIGURE FUNCTION

Set up and zip the Lambda function provided in Lab 4 on your computer. It's located in **lab-4/lambda/create-s3-upload-policy-document**.

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

  lab-4/lambda/create-s3-upload-policy-document

- Install npm packages by typing:

  npm install

- **Zip Lambda function**

  For OS X / Linux Users
  Now create create a ZIP file of the function, by typing:

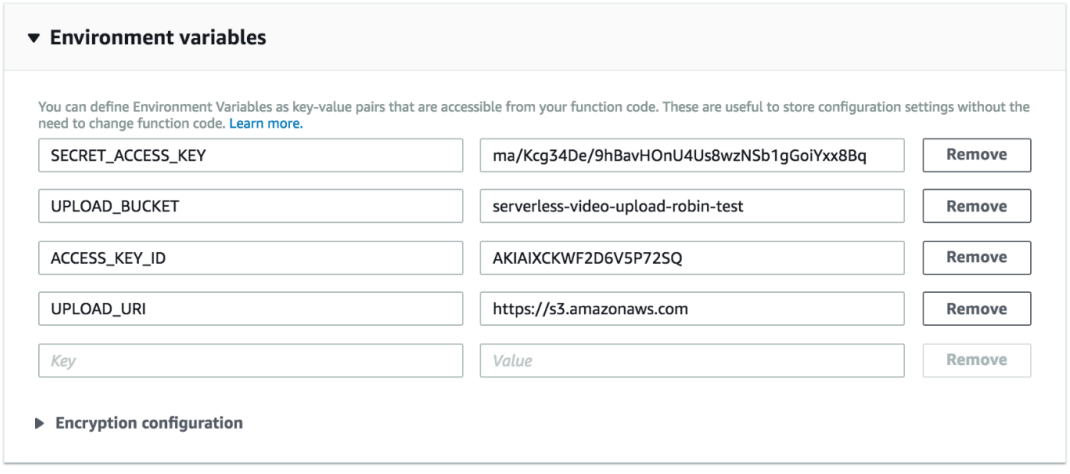
  npm run predeploy
  For Windows
  You will need to **zip up all the files** in the **lab-4/lambda/create-s3-upload-policy-document** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the create-s3-upload-policy-document folder. Zip up the files inside of it**).

25. 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.
- Select **Code entry type** and click **Upload a .ZIP** to upload the function, select the ZIP file created in the previous step and then click **Save.**
- Create two environment variables with the keys **ACCESS_KEY_ID** and **SECRET_ACCESS_KEY**. The values of these variables will be in the .csv file you downloaded in step 2 which you need to copy and paste into this screen.
- Create a third environment variable **UPLOAD_BUCKET** and set it to your video *upload* s3 bucket.

And finally create a fourth environment variable **UPLOAD_URI** and set it to https://s3.amazonaws.com

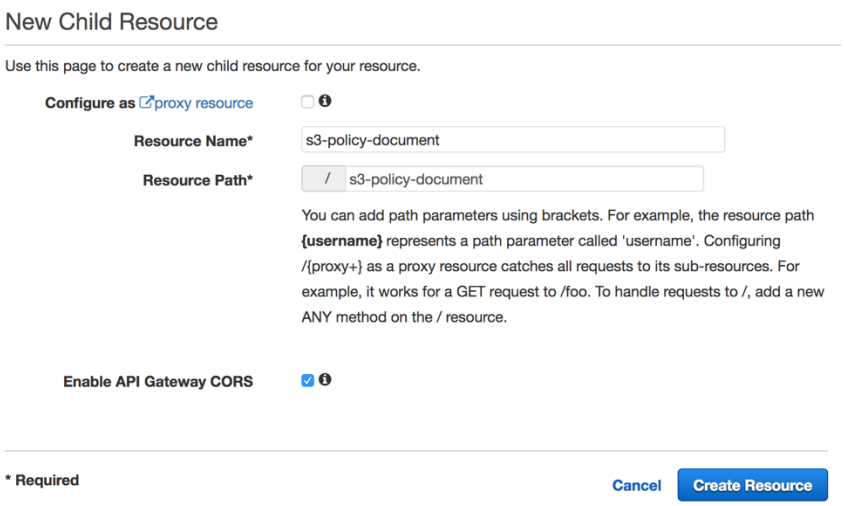


- Scroll to the top of the page and click the **Save** button.

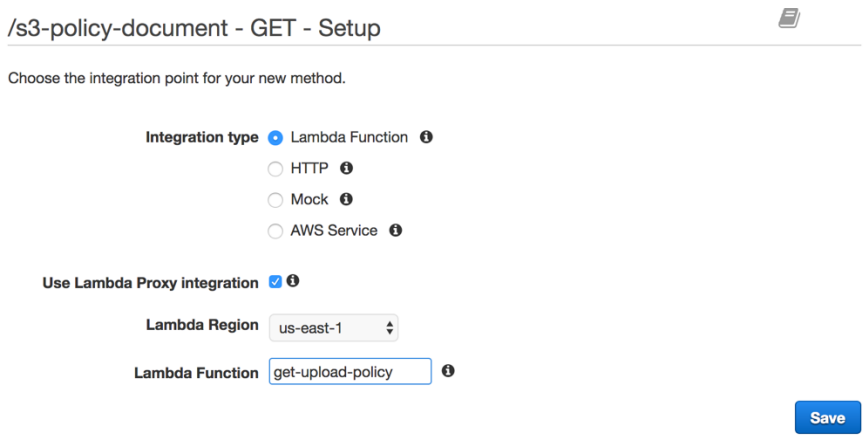
26. CREATE RESOURCE & METHOD IN THE API GATEWAY

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.

- Go to **API Gateway** in the AWS console.
- Select **24-hour-video**.
- Under **Resources** in the second column, ensure that the **/** resource is selected.
- Select **Actions** and then click **Create Resource**.
- Set the Resource Name to **s3-policy-document**.
- Ensure that the **Enable API Gateway CORS** box is checked.



- 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/check mark button to save.
- In the screen that appears:
    - Select **Lambda Function** radio
    - Check the checkbox with the label **Use Lambda Proxy Integration**
    - 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



To make the custom authorizer invoke on the GET method, follow these steps:

- Ensure you are still on the **Resources** tab in the left navigation menu.
- Click **GET** under **/s3-policy-document** in the second column.
- Click **Method Request** in the **/s3-policy-document - GET - Method Execution** section.
- Click the pencil next to **Authorization**.
- From the dropdown select **custom authorizer** and the little tick next to it.
- Click on the **HTTP Request Headers** section to expand it.
- Click on the **Add Header** link, put "authorization" for the name, and click the tick/check mark icon to the right to save it.
- Click on the **URL Query String Parameters** section to expand it.
- Click on the **Add query string** link, put "accessToken" for the name, and click the tick/check mark icon to the right and save it.

27. DEPLOY API GATEWAY

Finally, we need to deploy the API so that our changes go live.

- Click **Actions** at the top of the second column.
- Select **Deploy API**.
- In the popup select **dev** as the **Deployment stage**.
- Click **Deploy** to deploy the API.

28. ENABLE CORS FOR THE S3 BUCKET

To be able to upload directly to an S3 bucket we also need to enable CORS for the bucket.

- Go to **S3** in the AWS console.
- Click on the **upload** bucket (e.g. *severless-video-upload*).
- Click **Permissions** from the bucket menu.
- Click **CORS Configuration**.
- Paste in the following CORS configuration and click **Save**:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>POST</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

29. TESTING

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

- Copy the config.js file containing your account specific settings, from the last lesson:
  Copy lab-3/website/js/config.js to lab-4/website/js/config.js

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

  lab-4/website

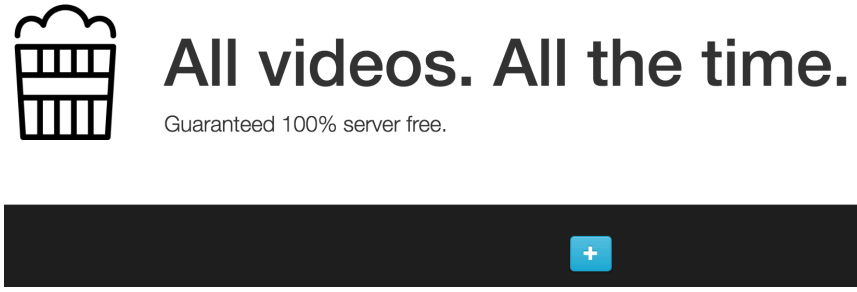
- Run the following command to make sure that required npm components are installed:

  npm install

- Run the following command to start the website:

  npm start

- Open the website (http://localhost:8100) and sign in. Click on the **plus** button at the bottom of the page to upload a movie file. You can use one of the example files from the first lesson. You will see a progress bar while the upload takes place.



Go 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? The files will be inside a folder with a randomly generated name, like a70e496a579b9fb21144fb108e6bf000747a98d3.
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.

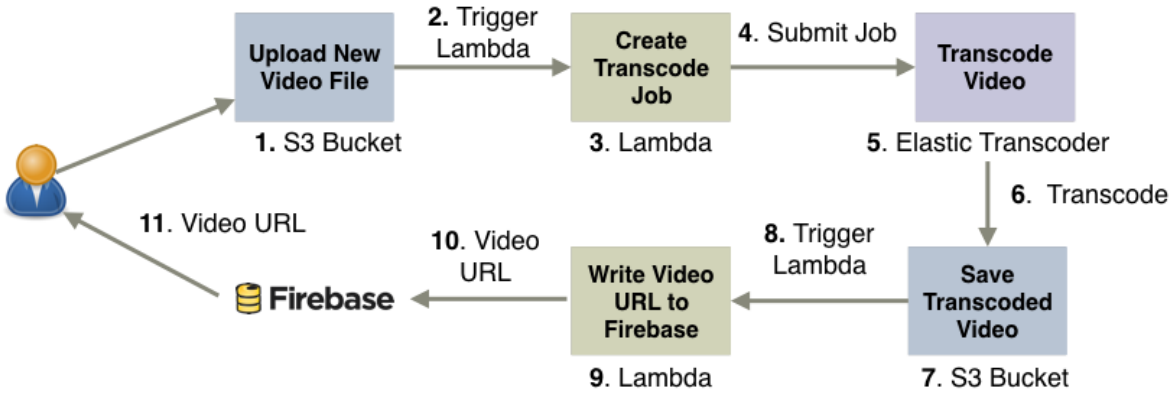
When you're done with this lab, exit the "npm start" command in your terminal by pressing <Control>-c.
We are nearly there! There's one more lesson left and you'll have your full YouTube clone ☺

LESSON 5

In this lesson, we'll connect our video transcoding pipeline in AWS with our website so that website users can view transcoded videos.
We'll do this using an online database service called Firebase. Firebase is a no-SQL database that has rich JavaScript support and can stream data updates directly to user's connected devices using web-sockets.
We'll do this by:

- Creating a Firebase database.
- Connecting our website to Firebase to fetch the URLs of the videos.
- Modifying our existing "transcode-video" lambda function to write to Firebase, so that connected browsers can show that a transcoding operation is taking place.
- Adding a new lambda function "push-transcoded-url-to-firebase", which writes the URLs of videos in S3 to Firebase.

- Configuring the transcoded s3 bucket to trigger this lambda function when a new transcoded video arrives, so that the locations of newly transcoded videos are published to Firebase and made available to users.

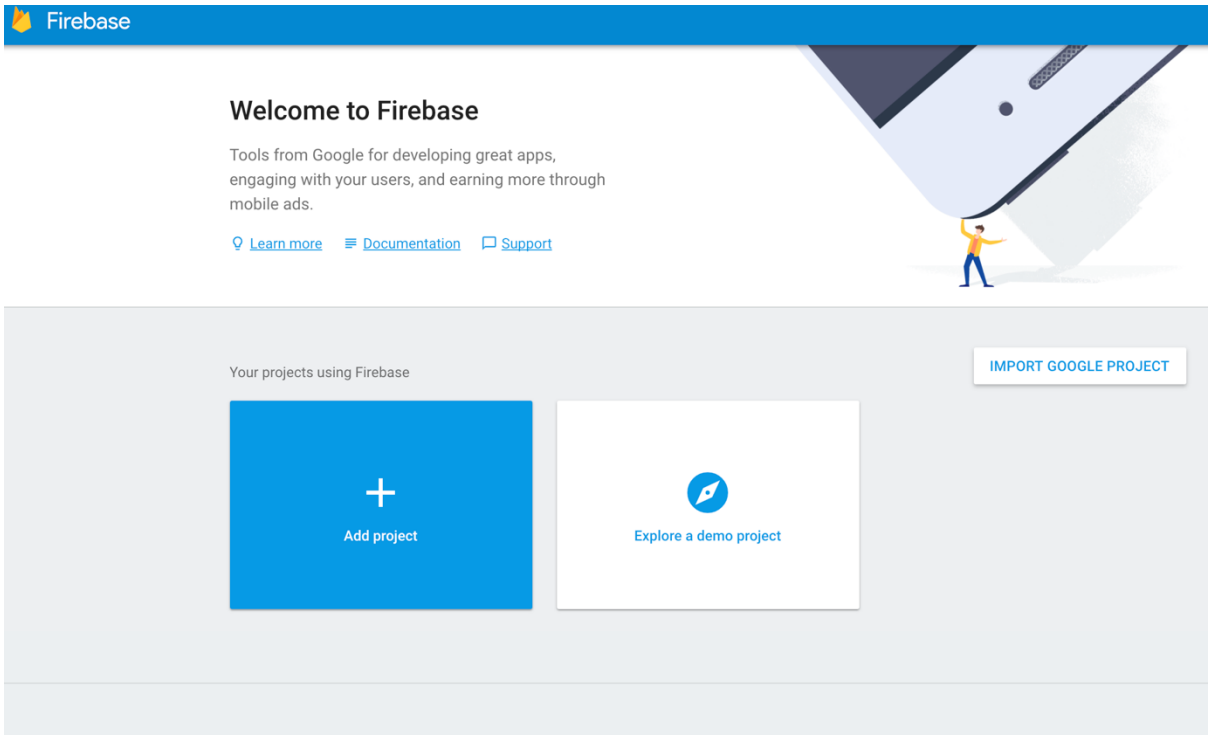


NOTE: PLEASE CREATE ALL YOUR RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

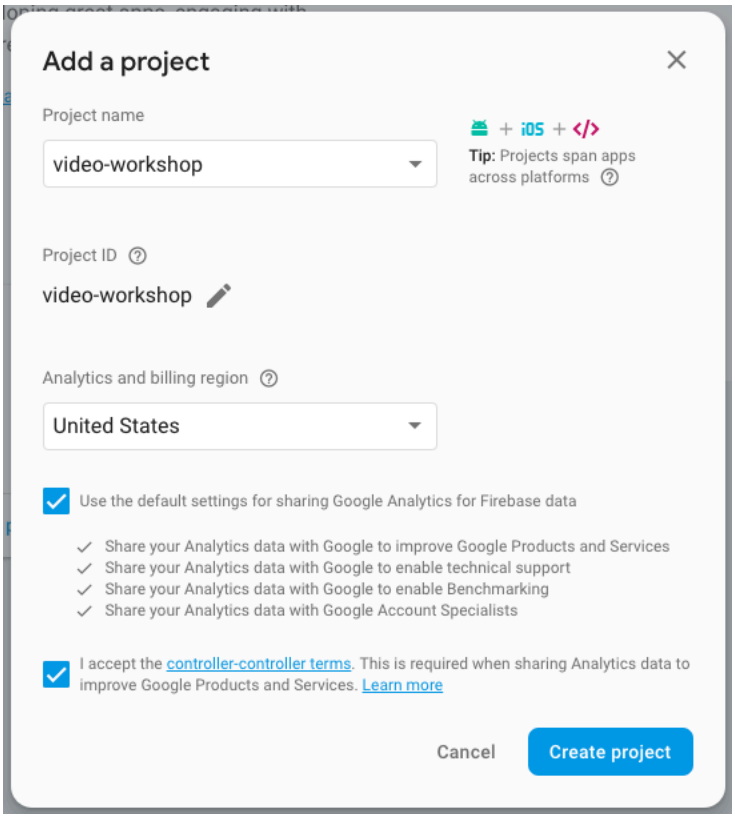
30. CREATE A FIREBASE DATABASE

First, you need to create a Firebase account.

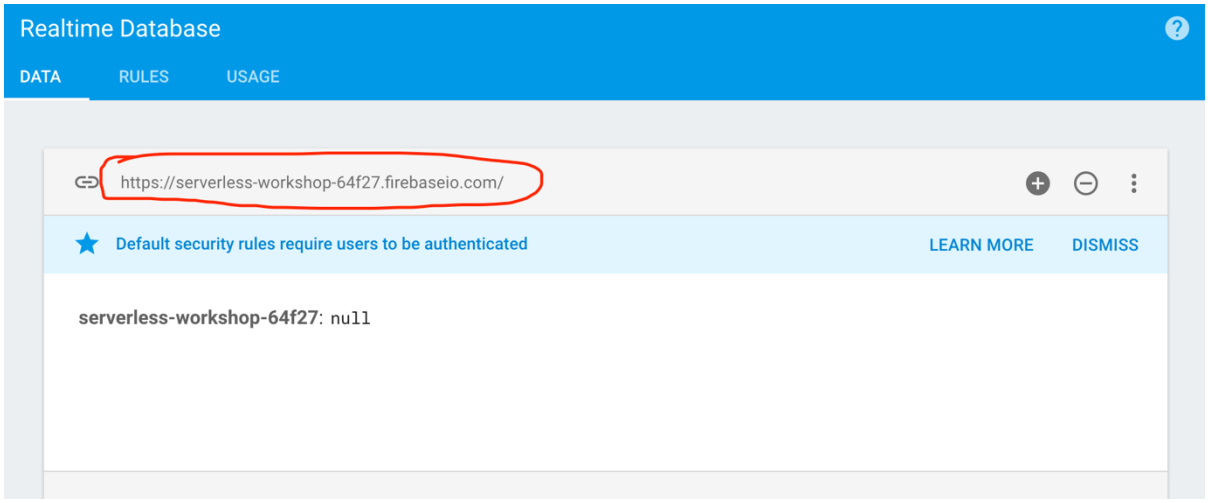
- Visit https://www.firebase.com/ to create a free account.
- Click the **Get Started** button.
- Register with your Google Account.
- You will be taken to the console. In the console click **Add Project**.



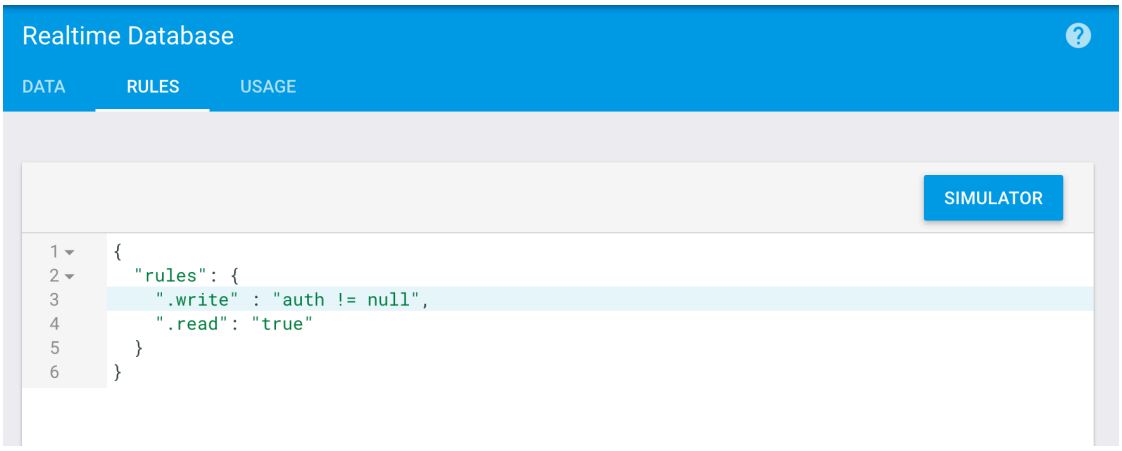
- Give your project a name like, “serverless-workshop”, and click **Create Project**.



- Your project will be created, which comes with a database. Let's check it out. Click **Database** in the left navigation menu. Then click the **Getting started** button in the **Realtime Database** tile
- You'll see you have an empty database. That's OK, we'll add some data later. For now, take note of the database URL. You will need it



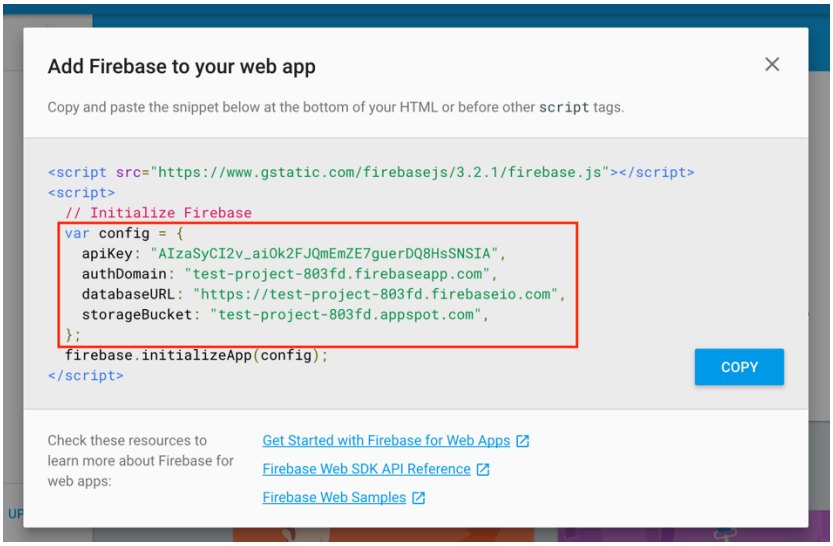
- Click on the **Rules** tab.
- Set ".read" to "true".
- Click the **PUBLISH** button to save.



31. MODIFY WEBSITE TO ACCESS FIREBASE

Now we're going to connect our web site to Firebase.

- Copy the config.js file containing your account specific settings, from the last lesson.
  Copy lab-4/website/js/config.js to lab-5/website/js/config.js
- Go the **Firebase** console and click on **Project Overview** in the upper left hand corner.
- Click on **Add Firebase to your web app** circle and copy the **config** object.



- Paste the config object to **lab-5/website/js/video-controller.js** where it says: **/* PASTE CONFIG HERE */** in connect your website to Firebase.
- In your terminal or command-prompt, go to the following folder:

```
lab-5/website
```

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

```
npm install
```

- Now run:

```
npm start
```

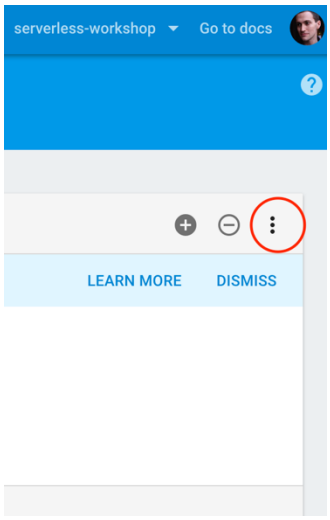
- Open the website in your browser:
  http://localhost:8100

  You should see a blue spinner in the center of the page. This will continue spinning until some videos are found in Firebase, which we'll add next…

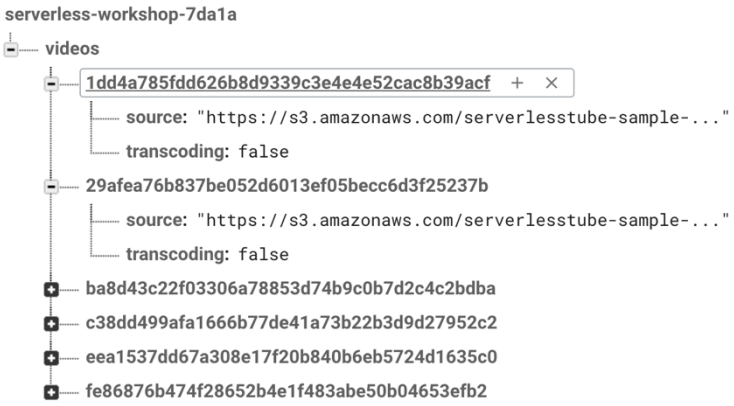
32. TEST WITH SOME SAMPLE DATA

To validate that our web site is connected to Firebase, we'll load some sample data into the Firebase database. This data points to some videos we've already transcoded for you.

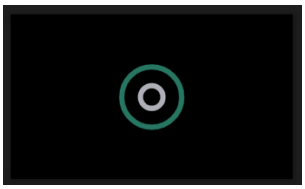
- To show off the push-based update features of firebase, make sure the 24-hour video site is open on your screen.
- In another browser window, open your Firebase project's database and go to the **Data** tab, just like you did in Step 1.
- Press the **Hamburger** button in the top right-hand corner of the database section.



- From the menu select **Import JSON**.
- Upload the json file from the following location:
  lab-5/data/firebase-sample-data.json
- Your Firebase data will now be populated:



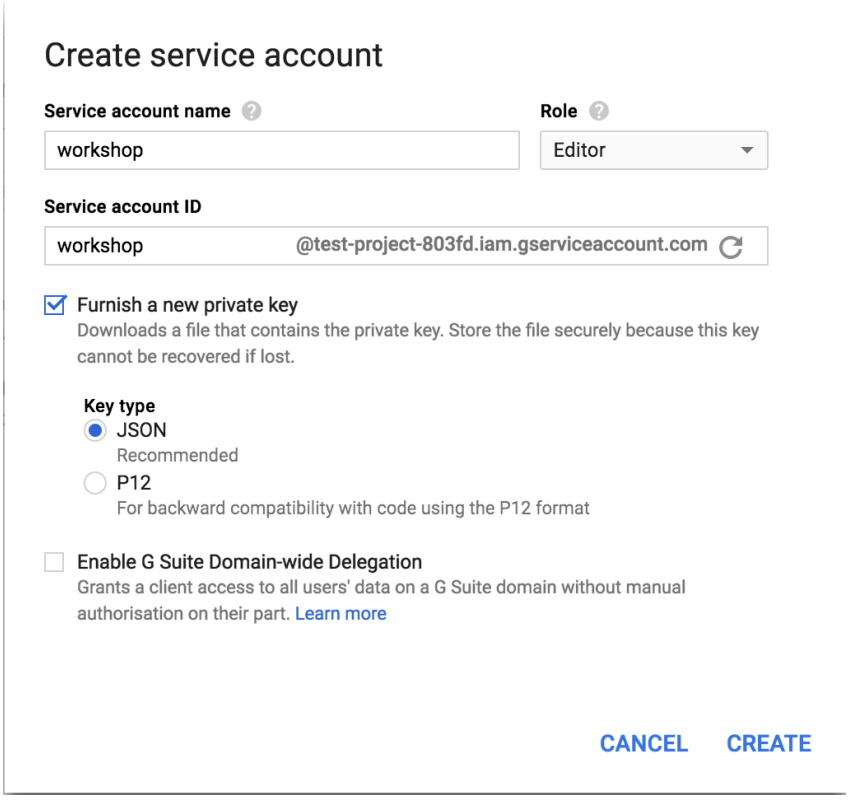
- Your web site will be automatically updated, as Firebase pushed the new data directly to your web browser via web sockets.
- Have a play, modifying the data in Firebase and watching the website update automatically:
  - **Delete** an entry by clicking the **x** icon to the right of the id, and watch it disappear from the website.
  - Change the **transcoding** flag on one of the entries from false to **true**.
    The transcoding flag is used to indicate that a file has been uploaded, but is currently being transcoded. This allows the UI to show an entry for it with an animation showing that something is in progress. Change the flag and watch the UI update to show the transcoding indicator:



33. MODIFY VIDEO TRANSCODE LAMBDA FUNCTION FOR FIREBASE

Before we proceed to modify Lambda functions we need to create service accounts in Firebase.

- In **Firebase** click on the **Gear** icon next to **Project Overview** in the left navigation menu, and click **Users and permissions**.
- Click the **Advanced permission settings** link in the lower right. This will open a different website in a new tab.
- Select **Service Accounts** from the left nav.
- Click the **Create Service Account** button.
- Set a service account name like “workshop”.
- From the **Role** dropdown, select **Project** and then **Editor**.
- Click “Furnish a new private key” and make sure that JSON is selected.
- Leave everything else as is and click **create**.



- Click **OK** on the popup that appears
- Copy the JSON file that was downloaded to lab-5/lambda/transcode-video-firebase-enabled/ Make a note of the filename

Now we're going to modify the existing video-transcode Lambda function, to have it push a new entry into Firebase with **transcoding:** set to **true**. With this in place, the user interface will show a placeholder of a video showing an animation while the video transcodes.
In Lesson 4 we configured file uploads, and this upload system created a unique key for each file that was uploaded (this key was used in the path when the file was stored in S3). We'll have our Lambda function use this key as the unique key for the video in Firebase.

- ZIP up your lambda function

  For OS X / Linux Users
  In the terminal / command-prompt, change to the directory of the function:

  ```
  cd lab-5/lambda/transcode-video-firebase-enabled
  ```

  Install npm packages by typing:

  ```
  npm install
  ```

  Now create create a ZIP file of the function, by typing:

  ```
  npm run predeploy
  ```

  For Windows
  You will need to zip up all the files in the **lab-5/lambda/video-transcoder-firebase-enabled** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the video-transcoder-firebase-enabled folder. Zip up the files inside of it**).

- In the AWS console, go to **Lambda**.
- Select the **transcode-video** function.
- Choose **Upload a .ZIP file** from **Code entry type** and click **Upload**.
- Select the ZIP file of the Lambda function you just created.
- Create two more environment variables:
  - **SERVICE_ACCOUNT**: The name of the **JSON Service Account** file you created earlier in this step.
  - **DATABASE_URL:** The URL from the **Database** tab in the Firebase.
- Click the **Save** button at the top of the page to upload the function and save your environment variables.

- Test that your function works by opening the 24-hour video web-site and uploading a video. Within a few seconds of the upload completing, you should see a new entry appear in the video list, showing the transcoding animation. This animation will remain forever, because we haven't yet connected anything to update firebase once the transcoding has completed.

34. CREATE NEW LAMBDA FUNCTION: PUSH-TRANSCODED-URL-TO-FIREBASE

Now we're going to complete the final piece of the system: We're going to add a new lambda function, that will trigger every time a newly transcoded video arrives in the second, transcoded S3 bucket. This lambda function will write the public URL of the video to Firebase (so that the browser can play the video). It will also set **transcoding: false**, indicating that transcoding has completed.

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

  ```
  lab-5/lambda/push-transcoded-url-to-firebase
  ```

- Install npm packages by typing:

```
npm install
```

- **Copy the Firebase service account JSON file**

  Copy the Firebase service account file you created in the previous step to */lab-5/lambda/push-transcoded-url-to-firebase*. The JSON file must be included with the Lambda function for it to work.

- **Now ZIP up your lambda function**

  For OS X / Linux Users
  Now create create a ZIP file of the function, by typing:

```
npm run predeploy
```

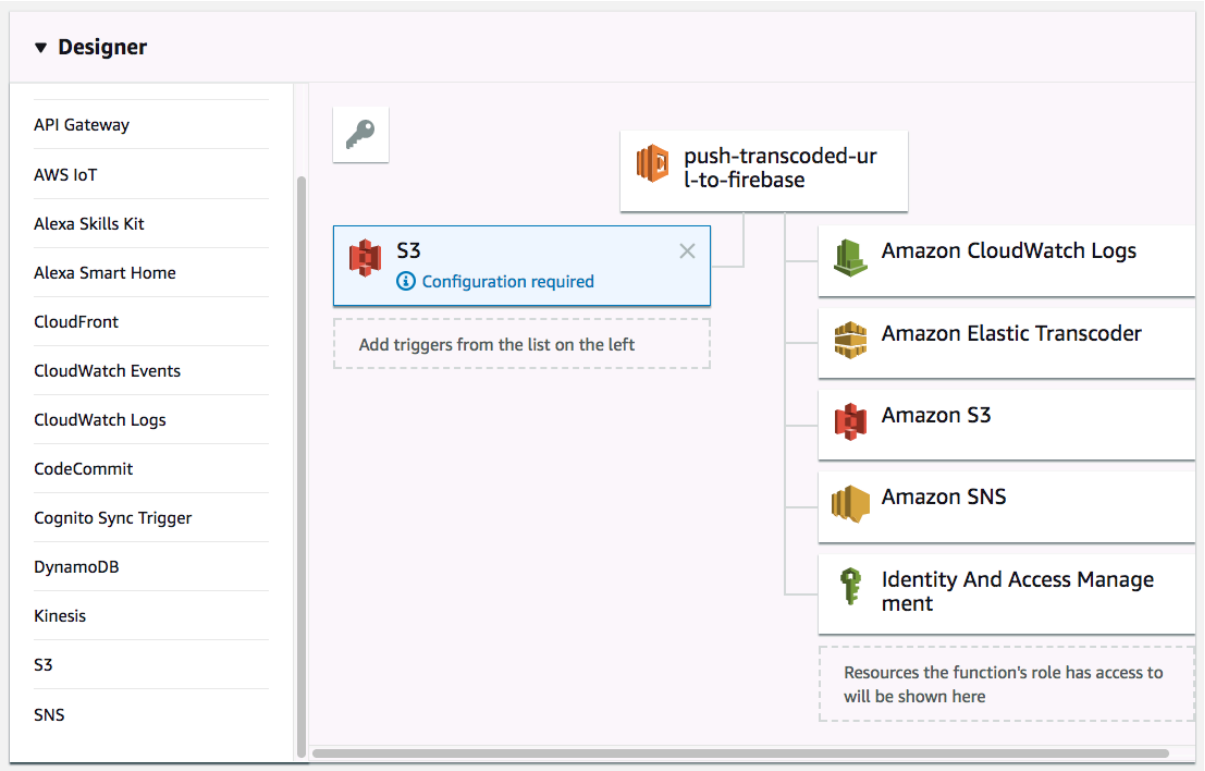
  For Windows
  You will need to zip up all the files in the **lab-5/lambda/`push-transcoded-url-to-firebase`** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the `push-transcoded-url-to-firebase` folder. Zip up the files inside of it**).

- In the AWS console, go to **Lambda** and create a function as before, with the following settings:
  - **Name**: push-transcoded-url-to-firebase
  - **Runtime**: Node.js 4.3
  - **Role**: lambda-s3-execution-role
  - **Function package**: The .zip file you just created.
  - **Timeout**: 30 seconds
  - Environment variables:
    - **SERVICE_ACCOUNT**: the name of the JSON Service Account file you created earlier
    - **DATABASE_URL**: the database URL from Firebase, just as you did for the last function
    - **S3_TRANSCODED_BUCKET_URL:** the URL of your second (*transcoded*) bucket, e.g. *https://s3.amazonaws.com/serverless-video-transcoded*.
    - **BUCKET_REGION**: *us-east-1*
- Don't forget to click the **Save** button at the top of the page.

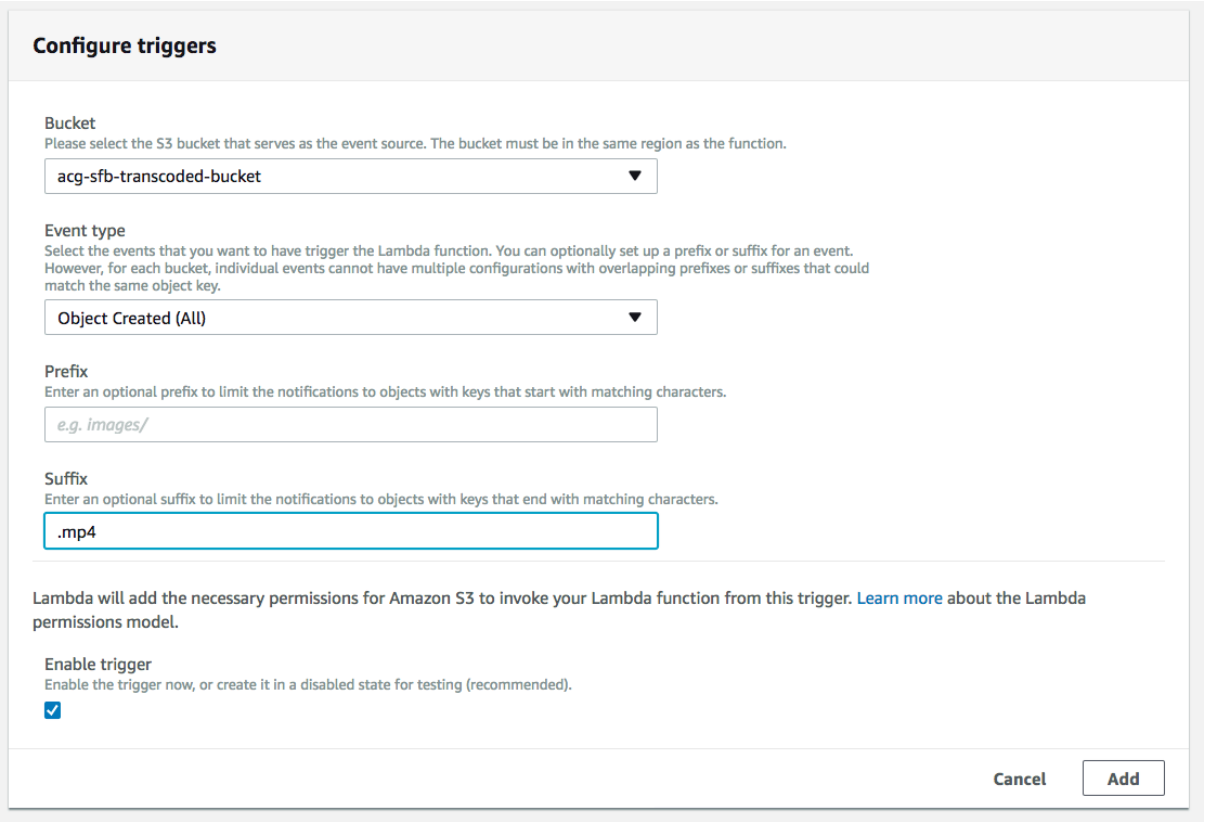
35. MODIFY TRANSCODED VIDEO BUCKET TO TRIGGER NEW LAMBDA FUNCTION

Now we need to configure S3 to invoke the new push-transcoded-url-to-firebase lambda function when a newly transcoded video arrives in the destination bucket:

- In the Lambda console for the push-transcoded-url-to-firebase function you just created, scroll to the **Designer** section
- Click on **S3** in the **Add triggers** list on the left



- Scroll down to the **Configure triggers** section
- Select the second bucket (e.g. *serverless-video-transcoded*).
- In the event type dropdown, select ***Object Created (All)***.
- Enter a suffix of **.mp4**
  We do this to ensure that the lambda function is only called when new videos arrived. The elastic transcoder may drop other assets in the bucket (such as thumbnails, or JSON files) which should not trigger the lambda function.



- Press **Add,** then click **save** up the top and AWS will link your S3 bucket and Lambda function.

36. TEST SYSTEM END-TO-END

The system is complete, it's time to test it end-to-end!

- Open the 24-hour video website in your browser.
- Upload a video file. The progress bar will show as the video uploads.
- Once upload completes, a tile will appear in the user interface representing the video. It will contain an animation, indicating that the video is being transcoded.
- Once transcoding is complete, the transcoding animation will be replaced by the video.
- Click on the video to watch it play. You'll notice that this is running in a web-friendly, lower quality 480p format.

Congratulations – you now have completely serverless website that authenticates users, allows them to upload video files, transcodes these videos to a web friendly format and then makes them available to all users of the web site.

Optional Exercises:

1. In the website, move the config object from video-controller.js to the config file.
2. Your website is still just running locally. Create a new S3 bucket with Static Web Hosting enabled for your **test** environment.
3. Update CORS and your firebase database to only allow requests from the origin for your **test** environment.