Exercise: Adding items to Amazon DynamoDB using the AWS Software Development Kit (AWS SDK)

Overview

In this exercise, you will learn how to *develop* with Amazon DynamoDB by using the AWS Software Development Kit (AWS SDK). Following the scenario provided, you will add items into an existing DynamoDB table and use the AWS SDK. This exercise gives you hands-on experience with Node.js, Amazon DynamoDB, and AWS Cloud9.

Objectives

After completing this exercise, you will be able to use the AWS SDKs to do the following:

- Upload items to the DynamoDB table.
- Query your DynamoDB table using code (i.e a full table scan).
- Create a role for an AWS Lambda function using AWS Identity and Access Management (IAM).
- Create an AWS Lambda function that talks to DynamoDB, using the Lambda console.
- Create an CORS enabled Amazon API Gateway that points to a Lambda function
- Upload an item to Amazon S3 via the AWS-SDK

Story continued

Now you have your database ready, it's time to seed it with some data.

You asked Mary for some card data and she has promised to email you a JSON document with all the card data and some dragon images.

However, she keeps delaying. So, you think it's a good idea to add a few items to help you create a basic proof of concept. The API will return all the data in the database to the website. You only have 2 images from her so far, so your database needs to have only 2 items.

Your next step is to add a couple of dragon items to the database table that you just created.

You think dragon info would look a bit like this (you are guessing). So you add this as a starting point while you wait for Mary to email you the real data.

Dragon table

Primary Key (dragon_name)	drgaon_type	description	attack	defense
sparky	green	breaths acid	10	7
tallie	red	breaths fire	7	10

You already have a basic website that you put together over the weekend, however you haven't built the back end API functionality for it yet. We can do that now.

We will be taking this step by step.

1. Upload your basic website to S3 and configure it for website hosting
2. Upload some dragon data using the AWS-SDK.
3. Build out the backend functionality.

Prepare the exercise

Before you can start this exercise, you need to import some files and install some modules in the AWS Cloud9 environment that you have created.

1. If you are not already in your AWS Cloud9 environment (from the last lab). Go to the AWS Management Console. Choose the **Services** menu and choose **Cloud9**, and choose **Open IDE** to open the AWS Cloud9 environment.
2. In the AWS Cloud9 system go to the AWS Cloud9 **bash terminal** and type

```
cd /home/ec2-user/environment
```

3. If you had any lab 1 stuff open collapse that folder now, and kill off any tabs you are not using from the last lab. *Things can get crowded very quickly in Cloud 9.*
4. Now you will need to seed your AWS Cloud9 filesystem, go to the Cloud9 **bash terminal** (at the bottom of the page) and run the following `wget` command:

```
wget \
https://s3.amazonaws.com/awsu-hosting/edx_dynamo/c9/dynamo-update/lab2.zip \
-P /home/ec2-user/environment
```

You should also see that a root folder called **dynamolab** with a `lab2.zip` file has been downloaded and added to your AWS Cloud9 filesystem (on the top left).

5. To unzip the `lab2.zip` file, run the following command:

```
unzip lab2.zip
```

6. To keep things clean, run the following commands to remove the zip files:

```
rm lab2.zip
cd lab2
echo "done"
```

7. Once you see "done". Select the black arrow next to the `lab2` folder (top left) to expand it. Notice inside this `lab2` folder there is a solution folder. **Try not to peek at the solution unless you really get stuck. Always TRY to code first.**
8. You will notice a `resources` folder inside `lab2` too (you can ignore this for now). We will use items in here later in the exercise.
9. Run this command to bring in the SDK into the `lab2` folder path.

   ```
   npm install aws-sdk
   ```

   You should see that some packages and modules have been installed. Ignore any warnings in the terminal. However, if you get an **error**, reach out in the forums before moving on.

As this course is self-paced, often people will start the lab then come back to it later. In the interim period, we may have made adjustments to the code.

Ensure that inside your lab2 folder that the name of the version markdown file is matching the version number at the top of this document.

If they are out of sync, you will run into problems. To get them synced, simply remove the old folder and run through the wget steps above one more time.

You are now ready to do the exercise tasks.

Step 1: Add Dragon data to your DynamoDB table using the SDK

We need to populate DynamoDB with some dummy dragon data so our website can eventually query it and display dragon data.

1. Open the SDK docs for `node.js`, and find the method for creating new items inside an existing DynamoDB table. Find out the correct method name and establish what parameters you need to pass in.

Language	AWS documentation deep link
NODE.JS (8.16.0)	https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#putItem-property

Time to write some code that adds a couple of items to your DynamoDB table.

1. In AWS Cloud9, open up the `upload_items` file inside the `lab2` folder by double clicking on it.
2. Have the SDK docs open (as above) to help you.
3. Replace the <FMI> sections of the code in that file. So that the code uploads dragon data to the table you created in `lab1`. Your table should have been called **dragons** and should be in **us-east-1**.
4. Here is the dragon info you wish to upload. As you created a flexible schema you can drop in the following data structure.

Primary Key (dragon_name)	dragon_type	description	attack	defense
sparky	green	breaths acid	10	7
tallie	red	breaths fire	7	10

5. Save the `upload_items.js` file.
6. In the AWS Cloud9 terminal, type this run command, *rather* then just pressing the run button.

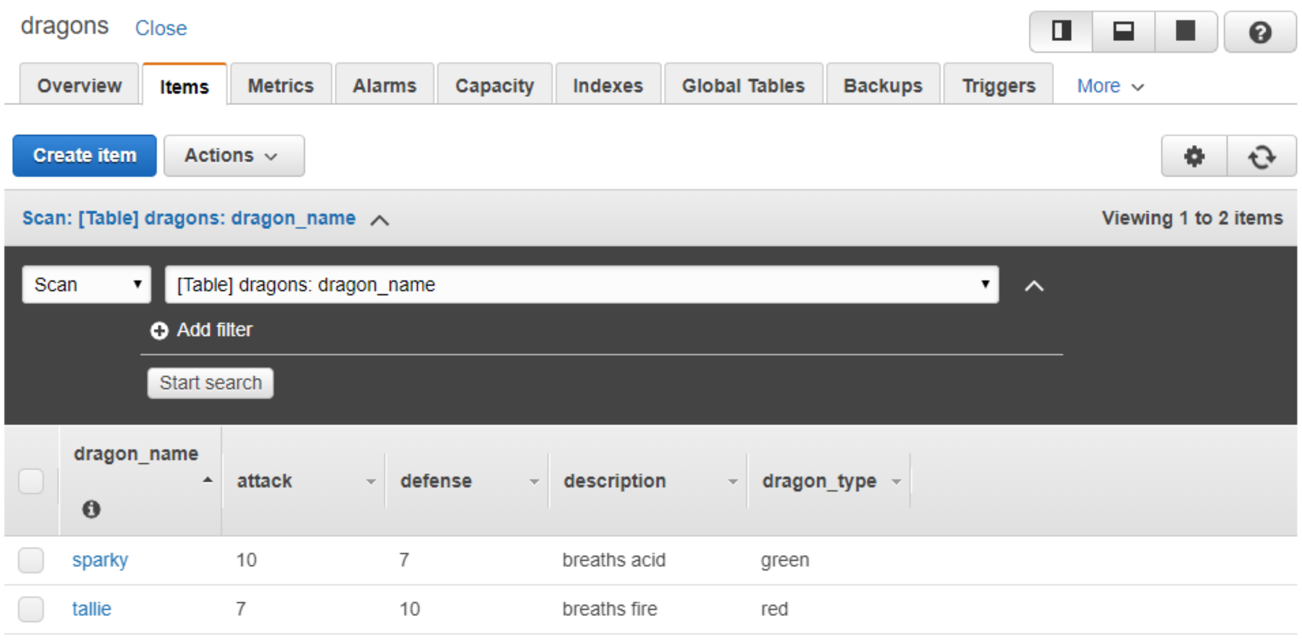
```
node upload_items.js
```

You should see something like this, showing two items have been added.

```
null { ConsumedCapacity: { TableName: 'dragons', CapacityUnits: 1 } }
null { ConsumedCapacity: { TableName: 'dragons', CapacityUnits: 1 } }
```

Confirm that your code worked.

1. Head back to your DynamoDB console *(probably still open in another browser tab from lab1)*. Refresh the table.
2. Click on your `dragons` table, and choose the **items** tab. Then press **scan**, you should see both dragon items (see image below).



dragon_name	attack	defense	description	dragon_type
sparky	10	7	breaths acid	green
tallie	7	10	breaths fire	red

Congrats you now have dummy Dragon data inside DynamoDB.

IF YOU GET STUCK, OR IT IS NOT WORKING. SIMPLY COPY THE CODE SITTING IN THE SOLUTION FOLDER.

Step 2: Wiring up the data to a web front end.

You spent time over the weekend putting together a basic HTML front end to display dragon card data. You now need to upload it to S3 and configure it to be hosted as a website.

Step 2A): Upload the website to S3

We have prepared a script for you that will upload your website (currently sitting in that resources folder we talked about earlier).

You need to write no code, and you will not be asked to ever modify the website, however for this upload script to work you will need to provide a unique bucket name in your account. As well as an IP address from where you are developing this exercise.

1. Choose **Services** and search for **s3**.
2. Choose **Create bucket**.
3. For **Bucket name** make sure you type in something unique but easy to remember. ***Example***: `er-101-2019-05-16-dragon-website`
4. Leave **Block all public access** checked.
5. Choose **Create bucket**.
6. Once the bucket is created choose it from the S3 buckets list.
7. Choose the **Permissions** tab and select **Bucket Policy**.
8. Copy and paste the following policy below:

1. Replace `"Resource": "arn:aws:s3:::yourwebsite/*",` with your bucket name. Example: `"Resource": "arn:aws:s3:::er-101/*",`
2. Also browse to https://www.whatismyip.com/ and replace `"0.0.0.0"` with your IPv4 address. This will only allow bucket access from your IP. So if you test this from another place, make sure the IP is updated.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::yourwebsite/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "0.0.0.0/32"
                    ]
                }
            }
        }
    ]
}
```

9. Choose **Save**.
10. Choose the **Properties** tab and choose **Static website hosting**.
11. Choose **Use this bucket to host a website**.
12. Type `index.html` in the Index document field.
13. Type `error.html` in the Error document field.
14. Choose **Save**.
15. Choose **Static website hosting** again and you will see the Endpoint similar to this: http://2019-05-16-dynamolab-er-102.s3-website-us-east-1.amazonaws.com. Click it to open it up in a new browser tab.

If you browse to the endpoint you will see **404 Not Found**. This is expected as you should have access to view the page but currently we don't have anything in our bucket.

Lab tip: Make a note of your bucket name somewhere, you will need it soon.

***NOTE:** Please use the chrome browser for viewing this dragon website, as we are not supporting other browsers in the forums.*

Step 2B): Upload items

1. Head back to your **Cloud9** tab. You will see a resources folder in the lab 2 folder.
2. Close any tabs and collapse any folders that you are not using in the AWS Cloud9 IDE.
3. Double click on `/lab2/resources/upload_website.js`.
4. You will only need to modify `line 42` where it asks for a bucket name. You do not need to modify any other parts of this file. However if you are interested you can see the code.
5. Replace the <FMI> (the fill me in) on line 42 with **your** bucket name. Example: `"2019-05-16-dynamolab-er-102"`
6. Choose **File** and **Save**.
7. Then run the following:

   ```
   pushd /home/ec2-user/environment/lab2/resources && node upload_website.js &&
   popd
   ```

The output should look like the following:

```
null { ETag: '"07df88017f8e994a0787eafe4a0db357"' }
.....<many more items>
```

We uploaded multiple files used in future labs to save you going through this step for every lab.

Now your website is all wired up to hit an API and return all the dragon data in the database as a proof of concept. The challenge is you don't have an API endpoint "to use" yet.

Go have a look at your website again.

So when you load the website it will say **"No API to call".** Which is to be expected, as you don't have one yet 😛

However you thought ahead, and have created a `config.js` file for your website. This can easily be updated with the API endpoint once you have one.

You will come back to this later.

Meanwhile you need to complete the following steps:

- Use Cloud 9 and create a script that queries your DynamoDB table. Test it to ensure that it returns all the dragon data it has (a full table scan).
- Create a role in IAM that can be used by a Lambda function that you are about to create that allows Lambda to talk to DynamoDB.
- Copy and paste the working code into the Lambda console, and using that role, create a working Lambda function and test it.
- Create a CORS enabled API gateway that points to your Lambda function, and test it.
- Upload (via the cli) a new config.js file containing your new API endpoint.
- Visit your website, and see all the dragon data (ready to show Mary as a proof of concept).

Step 3: Create a scan script

We will first write and test a script (using Cloud 9) that will query your dragon database.

You remember that to scan a DynamoDB table you would use a method like **scan**, but you double check the AWS SDK documentation to make sure.

From the table below, open the link to the method for scanning a DynamoDB table in the AWS SDK documentation. Confirm the method name and establish what parameters you need to pass in.

Language	AWS SDK Documentation deep link
Node.js (6.17.0)	https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#scan-property

In the AWS Cloud9 environment, do the following:

1. Close any Cloud 9 tabs you are not using such as `upload_website.js` and collapse the resources folder.
2. Open (double-click) the `scan_dragons` file inside the code folder you are working from.
3. Using the AWS SDK documentation to help, replace the <FMI> sections of the code in that file so that the code searches your **dragons** database and returns all the dragon data it has (a scan). Also modify the code to return the dragon information in a way the browser likes to receive data (as an array of data items).
4. Save the file.
5. *LAB TIP: Before you run this script, you should double check the solution code first. This way when you run the file it will just work. It is very easy to end up with some parts of a script working and some not, due to a typo or a small mistake, and end up with partial or inaccurate resources being created. It is better to check your code meticulously against the solution code before actually executing your code, rather than trying to undo partial changes after the script has run.*
6. In the AWS Cloud9 terminal, run command. (you should already be in the `lab2` path).

```
node scan_dragons.js test
```

Confirm that your code worked.

When you run the script, you should see following information.

```
Local test for all dragons
null [ { dragon_type: { S: 'green' },
    defense: { N: '7' },
    description: { S: 'breaths acid' },
    attack: { N: '10' },
    dragon_name: { S: 'sparky' } },
  { dragon_type: { S: 'red' },
    defense: { N: '10' },
    description: { S: 'breaths fire' },
    attack: { N: '7' },
    dragon_name: { S: 'tallie' } } ]
```

Next we need to create a Lambda function out of this. But first we will finish a few subtasks.

Step 3A): Create a new IAM role for use with Lambda

By creating a role for the Lambda function you are about to create, you are essentially allowing your function code to communicate freely with DynamoDB, write logs to CloudWatch Logs, report to AWS Xray and touch S3 which we will need in future labs.

1. From your Cloud9 dashboard choose **AWS Cloud9** in the upper left.
2. Then choose **Go to your dashboard**.
3. Go to **services** and choose **IAM**.
4. Choose **Roles** and choose **create role**.
5. Select **Lambda** and choose **Next: Permissions**.
6. In the **search** box type in **Dynamo** and select the **checkbox** next to `AmazonDynamoDBFullAccess`.
7. Don't press next yet.
8. Clear the search box and again in the **search** box type in **Lambda** and select the **checkbox** next to `AWSLambdaBasicExecutionRole`.
9. And again in the **search** box type in **AWSXrayWriteOnlyAccess** and select the **checkbox** next to `AWSXrayWriteOnlyAccess`.
10. And again in the **search** box type in **AmazonS3FullAccess** and select the **checkbox** next to `AmazonS3FullAccess`.
11. Choose **Next: Tags** and leave it as is. Select **Next:Review**.
12. Type the name `call-dynamodb-role` in the **Role name** box. Then choose **Create role**.

Lab note: We only asked you to add Xray and full Dynamo and S3 access to save you doing these steps again later in future labs.

Real world tip: In production environments you should apply the principal of least principle wherever possible.

Step 3B): Create a Lambda function

You already have your scan script ready and tested. So all you need to do now is create a Lambda function out of that code.

The code in the scan script was set up to work both in a Cloud9 testing environment and in a Lambda environment. You do not need to alter the code.

You just need to create a Lambda function, passing in that role you just created, then paste in your code "as is" from your existing **scan** script.

Just follow these steps:

1. Choose **services** and search for **lambda**.
2. Choose **lambda** from the drop-down list.
3. Choose **Create function**.
4. Type in `DragonSearch` for the **Function name**.
5. Use **Node.js 10.x** for the **Runtime**.
6. Under **Permissions** open **Choose or create an execution role**.
7. Under **Execution role** choose **Use an existing role**.
8. In the **Existing role** drop-down choose the role we created above `call-dynamodb-role`.
9. Finally choose **Create function**.
10. In the other browser tab where you have cloud9 open. Simply copy the code from `scan_dragons.js` and paste it into the Lambda editor replacing the contents of `index.js`.
11. Scroll down and under **Basic settings**. Change the timeout to `0` mins and `10` sec.
12. Choose **Save** at the top of the page.

Step 3C): Test our function

1. Next to the **Test** button select the drop-down arrow.
2. Choose **Configure test events**.
3. Leave the **Event template** as **Hello World**. In the **Event name** box type in `DragonScan` and paste in the following **blank object** code:

   ```
   {

   }
   ```

4. Choose **Create**.
5. Choose **Test**.

You should see the following in the **Execution result** go green, and when you expand the details sections you should see something like this:

```
[
  {
    "dragon_type": {
```

```
        "S": "green"
      },
      "defense": {
        "N": "7"
      },
      "description": {
        "S": "breaths acid"
      },
      "attack": {
        "N": "10"
      },
      "dragon_name": {
        "S": "sparky"
      }
    },
    {
      "dragon_type": {
        "S": "red"
      },
      "defense": {
        "N": "10"
      },
      "description": {
        "S": "breaths fire"
      },
      "attack": {
        "N": "7"
      },
      "dragon_name": {
        "S": "tallie"
      }
    }
  ]
```

Congrats! Your Lambda function can scan DynamoDB and return data. Now we need to put the API endpoint in front of it, so you can use it in your website.

Step 4: Create a simple API and connect it to your website

To create and test an API from the Amazon API Gateway console, do the following:

1. Choose **services** and search for **API Gateway**.
2. Choose **Get Started** and choose **OK** to remove the **Create Example API** pop-up.
    1. If you already have an API Gateway. Choose **Create API**.

3. Under **Choose an API type**, find **REST API** and Select **Build**
4. Under **Choose the protocol**, make sure **REST** is selected. Under **Create new API**, make sure **New API** is selected.
5. For settings, enter the following:

   API name: `DragonSearchAPI`

   Description: Add a brief description (optional)

   Endpoint Type: Regional
6. Choose **Create API**.
7. Choose **Actions** and then **Create Method**.
8. From the drop-down list under the **/** under **Resources**, choose **POST** and select the check mark icon.
9. For **Integration type**, choose **Lambda Function**.
10. Make sure the Lambda Region is set to ***us-east-1***.
11. Ensure that you **do not** select **Use Lambda Proxy integration**. That must remain **unchecked**.
12. Under Lambda Function start typing the word `Dragon`, so you can choose **DragonSearch** from the list.
13. Uncheck **Use Default Timeout**. Set **custom timeout** to 10 seconds `10000`.
14. Click **Save**.
15. Click **OK** to bypass the pop-up telling us about giving permissions for API gateway to talk to Lambda. This is fine, and just what we want it to do.
16. Click **TEST** and you should see a page where you can type the "request body", which you leave **blank**.
17. Scroll to the bottom to press the **Test** button.

Under **Response Body** you should see something like the following:

```
[
  {
    "dragon_type": {
      "S": "green"
    },
    "defense": {
      "N": "7"
    },
    "description": {
      "S": "breaths acid"
    },
    "attack": {
      "N": "10"
    },
    "dragon_name": {
      "S": "sparky"
    }
  },
  {
    "dragon_type": {
      "S": "red"
    },
    "defense": {
      "N": "10"
    },
    "description": {
      "S": "breaths fire"
    },
    "attack": {
      "N": "7"
    },
    "dragon_name": {
      "S": "tallie"
    }
  }
]
```

Important step

Before your website can talk to this back end API. We need to enable CORS before deploying it.

Here is why:

Your website is hosted in one domain (your S3 website URL) and your API is hosted in a different domain (the API Gateway endpoint URL). Your browser doesn't like this, and blocks the request.

You need to let your browser know that it is OK to call your API Gateway endpoint URL from your website. You do this by using what is known as a *preflight request*.

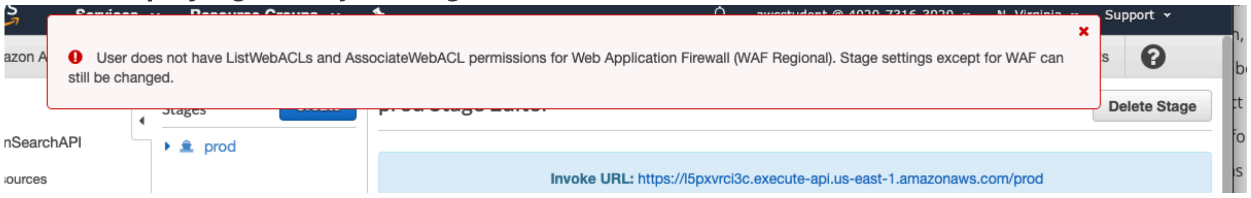
This entire process of allowing cross-domain access with preflight requests is called "Enabling CORS" (Cross-Origin Resource Sharing).

To enable CORS from the Amazon API Gateway console, do the following:

1. Click on the resource `/` above POST. *You should see a green POST box appear on the right.*
2. Choose **Actions** and select **Enable CORS**.
3. Check the top two boxes for **DEFAULT 4XX** and **DEFAULT 5XX**.
4. Leave the other settings as they are. Choose **Enable CORS and replace existing CORS headers**.
5. *IF* you see "replace existing values" choose **Yes, replace existing values**. *(Ignore any crosses and warnings)*.

Now your API is CORS enabled, you are ready to deploy your API.

1. Choose **Actions**. Under **API Actions**, select **Deploy API**.
2. On the **Deploy API** pop-up:
3. For **Deployment stage**, select **[New Stage]**
4. For **Stage name**, enter `prod` *(lowercase)*
5. For **Stage description**, enter `prod`
6. Leave **Deployment description** blank.
7. Choose **Deploy**. *(Ignore any warnings)*.


User does not have ListWebACLs and AssociateWebACL permissions for Web Application Firewall (WAF Regional). Stage settings except for WAF can still be changed.

Invoke URL: https://l5pxvrci3c.execute-api.us-east-1.amazonaws.com/prod

You should now be provided with a URL in your DragonSearch API dashboard that looks like this:

```
https://xxxxxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod
```

Copy this link to your clipboard.

You now need to edit your website configuration `config.js` file to point to your new API endpoint.

Step 5: Update a config file to point to new API (mock) endpoint

Head back to Cloud 9, and close the scan_dragons.js tab as you no longer need it open.

Currently, your website has the following in a `config.js` file in the `resources/website` folder. Double click it to edit it.

It will show this:

```
var API_ENDPOINT_STR = "<FMI>";
```

You need to edit this file to point to your new API endpoint:

1. Replace the with your API endpoint in quotes, like this: Remove any trailing slash after prod (if you have one), like so:

   ```
   var API_ENDPOINT_STR =  "https://xxxxxxxxxxxxx.execute-api.us-east-
   1.amazonaws.com/prod"
   ```

2. Save this `config.js` file and close that tab

3. Open `resources/upload_config.js` and then swap the for your bucket. Then issue this this command in the Cloud9 terminal.

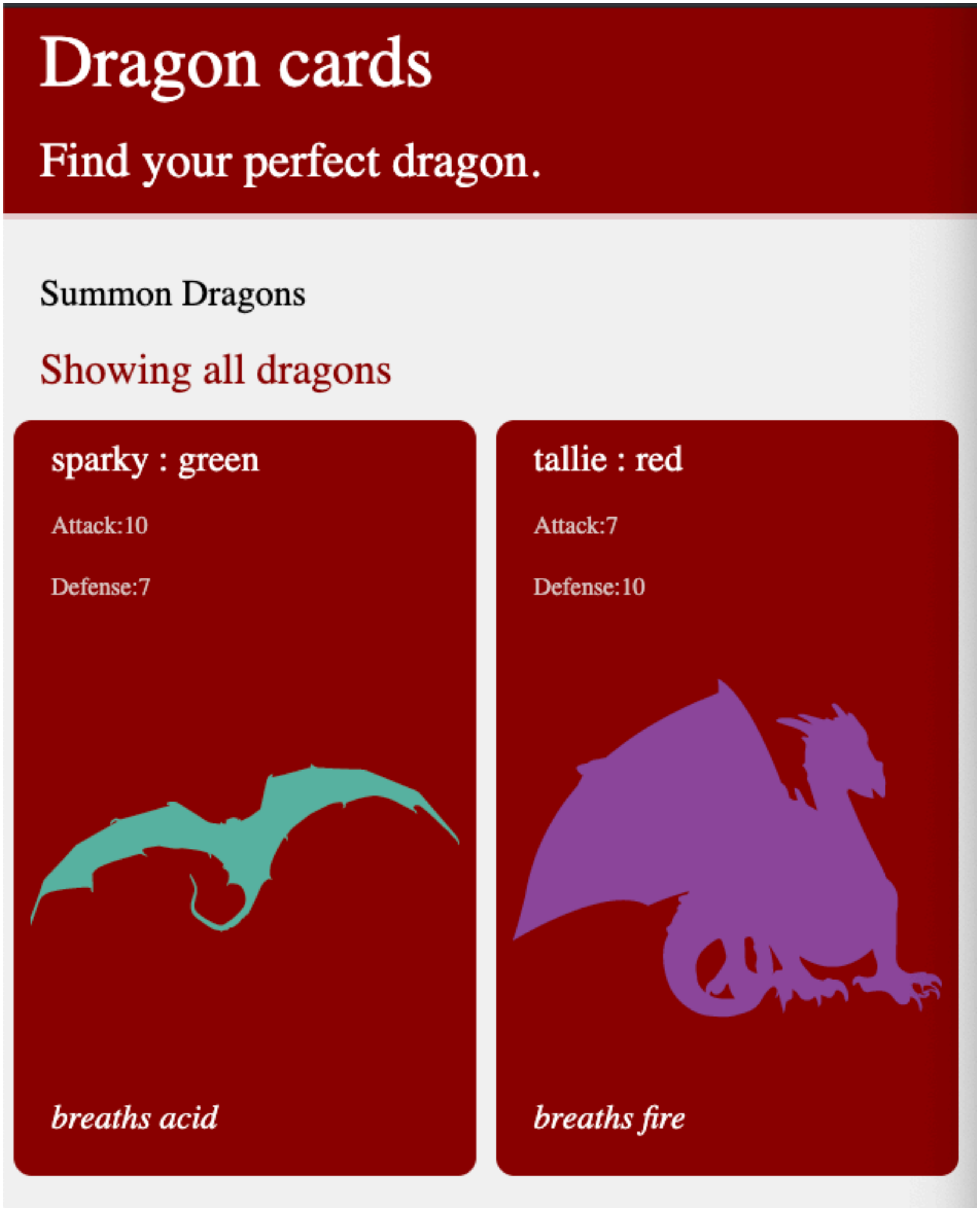
   ```
   pushd /home/ec2-user/environment/lab2/resources && node upload_config.js &&
   popd
   ```

You should see something a bit like this if it worked:

```
null { ETag: '"1b3c39cfa4c51b2d148635300d082c4d"' }
```

Now navigate back to your s3 website, and press refresh.

You should see both dragons.


Dragon cards

Find your perfect dragon.

Summon Dragons

Showing all dragons

sparky : green

Attack:10

Defense:7

breaths acid

tallie : red

Attack:7

Defense:10

breaths fire

Now all the website plumbing is out of the way. You have a simple proof of concept you can show Mary.

Hopefully she will provide you with real dragon data soon. So that you can adjust your function to query on specific dragons by name.

You are also crossing your fingers that she gives you a card template concept, and more dragon images, as she only gave you two images.

Congratulations! You have completed exercise 2.