[version_1.0.4]

Errors or corrections? Contact us at https://support.aws.amazon.com/#/contacts/aws-training

Lab Intro

Welcome to the labs section of Building Modern Web Applications on AWS.

If you have not watched my Lab Overview and Lab 1 intro. I suggest you stop here and go and watch them first so you know what your lab gameplay is going to be.

However just to quickly recap (and If you want to play along with the lab story). I'd like you to imagine this scenario:

The Lab Scenario

Context

I'm going to have you play the role of a **full-stack web developer**. Imagine that you've just picked up a nice little development gig at a local camera reseller where they sell lightly-used camera equipment.

The owner Mike, tells you that they have a great deal of foot traffic. However because they are only a small company with two sales people, they don't really have the resources to spend quality time with customers when it gets really busy.

He has decided to buy some in-store kiosks, and this is where you come in.

He wants you to write the application that is to be used on these kiosks. He doesn't care too much about the technology you use as long as it's on AWS and is leveraging a modern backend.

This is a high level overview of what Mike wants the kiosks (the app) to do.

Mike's Kiosk Idea

He wants customers to go to the kiosks that are strategically positioned throughout the store, and have them type in a key phrase. Like **4K** and/or **underwater camera**.

The app needs to provide a list of items (titles) that are related to the search.

He wants the search to be lighting fast, and also show the product id. This way if a user finds an item they like, they can simply tell one of the sales clerks and they can fetch it from the back.

In order to help customers decide if the searched item is "for them", he wants you to bring up one or more Amazon Reviews or average ratings for it.

He tells you that Sandra (his store manager) will provide you with a list of item ids and product ids that they have in stock. This was will be able to reference the respective Amazon reviews and ratings by product id via your code.

Mike doesn't want to spend time and money having you build and maintain a database just to keep a reviews database up to date. Instead, he asks you if there is any way to query the Amazon camera reviews in a text file (that Sandra can store and keep updated in Amazon S3) and just have the app query that.

You nod. Thinking about Amazon S3's cool **select functionality**, and you tell Mike, "Sure. That's absolutely do-able".

Your mental checklist / game plan

You look down at your notes, and then suggest the following game plan.

You offer to draft out a simple webpage that has a key phrase search feature which locally references a full list of products in stock.

Then you tell Mike, that you will then start creating an AWS backend API. That way, when a user clicks on the "get reviews" or "get ratings" buttons on a searched item, the app can make an API call that will query inside a file residing on S3. That file will simply be a stripped down version of the open source Amazon Reviews and Ratings dataset.

Mike seems happy with this, but asks for one more thing....if possible.

Mike's second ask:

He asks if you can allow his staff to **login** to any of the kiosks and **request a report** be **sent to their cellphones**. The report would need to have **all the items**, with their reviews and ratings, along with the **sentiment** and any **key phrases** (tags) to indicate what the focus or *intent* was behind each review.

He clarifies by giving you an example.

If his staff can quickly identify items in the report that have a generally negative review and then further identify items that have reviews that seem to focus on pricing.

By identifying these items, his staff can then think about reducing the price on just those products. As pricing may have been the *primary* reason why he has not been selling many of them.

You nod. You understand the benefit of that, and suggest that you can do the following:

Final (appended) game plan:

Although this makes things a little more complicated. You tell him, "sure that can be done".

You make the following assumptions, that he seems fine with.

You tell him you will use a Amazon Cognito hosted UI for the login system, as he told you he only has a small budget for this project. Besides, you don't want to build an authentication system from scratch if you can avoid it.

You tell him, that you can use a background process that leverages machine learning to evaluate the sentiment of reviews and discover key phrases using Amazon Comprehension Entity and Sentiment analysis. Then you tell him that you will use all of that to generate a full HTML report.

You warn him that there will be a slight delay from them requesting a report at the kiosk until it arrives on their cellphone. This is because the machine learning part may take some time, and sending out of the text with a link to the report is not instant either. Luckily, he seems fine with all of that.

You presume that his staff will want to still view reports when they are out of the office (at the coffee shop over the road) from their cellphones. However they still need to have some kind of security on this report. You suggest that you can have the report HTML page expire after a short while to prevent long term access to it. Again, he is good with that idea.

He tells you that he has a static IPv4 address at the store. So you plan on just allowing access to the app from that IP address. You will use an S3 pre-signed URL for access to a separate report HTML page, which will be designed to be viewed on a mobile phone.

Your overall game plan (bullet points)

- Create a web front-end leveraging a list of in stock items that they have produced for you as JSON, and use a client side search feature.
- Have the kiosk call an API to get a product's Amazon rating or review by id.
- Allow staff to login (using Cognito) at the kiosk, so they can further request a report.
- You will have a background process (AWS Step Functions) do all the various steps, and build out an HTML page that will get uploaded it to a private area on Amazon S3.
- Have your background process create a pre-signed URL for that report in S3 and send that URL (via Amazon SNS) to their cell phone that is gathered from Amazon Cognito.

> BTW: If you see this symbol 👷 It is me (Rick) talking to you and offering my *2 cents* here and there. If you ever see the 📓 icon it's usually me highlighting something more interesting or notable.

👷 *FYI. I have structured all of this into the following 6 labs / exercises*

1. LAB 1 - Get the front end up, by uploading a simple front-end where a user can search for an item.
2. LAB 2 - Get the plumbing in place, by creating three placeholder API Mocks. You will have the website hit up 3 distinct API endpoints that just pipe back the same dummy data for reviews and ratings and create report regardless of what product ids your front-end sends.
3. LAB 3 - Set up the authentication for staff using Cognito user pools. Wire that into the create report API endpoint and have it reject non authorized requests.
4. LAB 4 - Replace the mock AWS Lambda Functions with functions that use S3 *SelectObject* to return real data to the 2 GET endpoints: `get_reviews` and `get_ratings`.
5. Lab 5 - Have the `create_report` API kick off a step function background process. Have the step function call various Lambda functions that do various things. Such as sentiment analysis on a review. Look for key phrases to help with tagging in the report, and create the HTML report and pre-sign it. Finally sending that URL to the logged in user's cellphone.
6. Lab 6 - Add metrics to instrument how it is performing (AWS X-Ray). Find ways to improve one of your Lambda functions with context reuse, and cache the response.

👷 *Ok, best of luck with all these labs. If you get stuck, head over the forums.*

Exercise 1: Working with S3

The Story So Far

Ok, so its time to start your project.

You set about creating a basic webpage that allows a customer to search for camera equipment via a key-phrase.

👷 *The website is built by the way. You wont be building one. Just uploading it.*

You have been provided with a JS file from Sandra which contains all the products that they have in stock.

She has matched the product titles with those found on Amazon.com. To keep things simple she is using the same Amazon.com product ids for their own inventory. She figured having the same product ID as found on Amazon.com would help you reference the respective reviews later in your project. You are starting to like this person already ;).

After a few painful days of pixel-shifting your website front-end. You have something to upload.

Normally you would have created a static website HTTP website on S3 to avoid having the user type in the `/index.html` stuff. As this is a Kiosk application where the user is not typing in a URL, you figure it's easier to skip that step. Besides (as you will find out later) when you come to use Cognito for authentication. The return URL that it uses has to be HTTPS. Using the virtual style S3 secure URL vs an S3 hosted website seems like the path of least resistance. 👷 *Excuse the pun.*

You need to lock access down to just the kiosks in the store. you figure the easiest wa to do this is to write a *bucket policy* with an *IAM-like rule* to *condition-out* all other IPs from even viewing the site. As a bonus, the sales staff (provided they are in store and on the WIFI) can access it even if customers are using all the available kiosks.

If you can get the website installed and working in the kiosks by the end of today, you are confident that Mike will be happy that things are moving along nicely, and all will be good in the world.

- You are going to create a bucket in Oregon using `CMD_LINE` then configure it for website hosting.
- Using the `SDK` you will adjust the bucket policy (permissions) and lock it down to be accessed exclusively from the office IPv4 address (i.e your IPv4).

👷 *My lab 1's are always nice and easy, to lull you into a false sense of security ;)*

What you will learn in this lab:

1. How to create a new S3 bucket using the command line (terminal) on Cloud9. Or the `CMD_LINE`.
2. How to upload specific access permissions for the hosted website, and how to use an IP condition rule to only allow access to a specific IPv4 address.
3. How to upload a local website to an S3 bucket, using a script leveraging the AWS SDK (called from Cloud9). All while setting metadata on those objects.
4. [*Optional*] You will learn what the JS code in the website is doing if you ~~care~~ are interested. This will give you a heads up for how the APIs are being liaised with for the later labs.

Accessing the AWS Management Console

1. At the top of these instructions, click **Start Lab** to launch your lab.
2. A Start Lab panel opens displaying the lab status.
3. Wait until you see the message "**Lab status: ready**", then click the **X** to close the Start Lab panel.
4. At the top of these instructions, click **AWS**

   This will open the AWS Management Console in a new browser tab. The system will automatically log you in.

> **TIP:** If a new browser tab does not open, there will typically be a banner or icon at the top of your browser indicating that your browser is preventing the site from opening pop-up windows. Click on the banner or icon and choose "Allow pop ups."

Arrange the AWS Management Console tab so that it displays along side these instructions. Ideally, you will be able to see both browser tabs at the same time, to make it easier to follow the lab steps.

Setup

1. Ensure you are in **Cloud9**. Choose **Services** and search for **Cloud9**. You should see an existing IDE called Building_2.0. Click the button **Open IDE**. Once the IDE has loaded, enter the following command into the terminal: *(This command will ensure that you are in the correct path)*

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

2. You will need get the files that will be used for this exercise. Go to the Cloud9 **bash terminal** (at the bottom of the page) and run the following `wget` command:

   ```
   wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-Building_2.0/lab-1-s3.zip
   ```

3. You should also see that a root folder called **Building_2.0** with a `lab-1-s3.zip` file has been downloaded and added to your AWS Cloud9 filesystem (on the top left).
4. To unzip the file. Run the following command:

   ```
   unzip lab-1-s3.zip
   ```

5. To keep things clean. Run the following commands to remove the zip file. *You can also close the other AWS console tab*.

   ```
   rm lab-1-s3.zip
   ```

   ⚠ **If you are using Java** you will also need to run the following script:

   ```
   chmod +x ./resources/java_setup.sh && . ./resources/java_setup.sh
   ```

Lab Steps

Stage 1 - Creating An S3 Bucket Using The `CMD_LINE`

You need to a create an S3 bucket to host your website.

You know how to create buckets via the AWS Console, but you've always wanted to try and do this from the Cloud9 terminal's `CMD_LINE`. Now's your chance.

The command (method) for creating new S3 buckets is ~~surprise surprise~~ `create-bucket`. You will pass in the desired bucket name and the region where it is to be created.

You will need to come up with a unique bucket name.

🤷 *To keep things consistent. I recommend you stick to this following format for your bucket name.*

Your initials in lowercase, then today's date YYYY-MM-DD and then the word s3site.

For example: I would use this `rh-2020-02-25-s3site`.

Once you have decided on your bucket name. You will use the `CMD_LINE` to create a bucket in `us-west-2` or `Oregon`.

1. Run this command via the `CMD_LINE` replacing the `<FMI>` with your bucket name.

   ```
   aws s3api create-bucket --bucket <FMI> --region us-west-2 --create-bucket-configuration
   LocationConstraint=us-west-2
   ```

   Here is an example (yours will be different).

   ```
   aws s3api create-bucket --bucket rh-2020-02-25-s3site --region us-west-2 --create-bucket-configuration
   LocationConstraint=us-west-2
   ```

   This would give you something like this: (*You will need to make a note of the bucket name for later `<FMI>`s.*)

   ```
   {
           "Location": "https://rh-2020-02-25-s3site.s3.amazonaws.com/"
   }
   ```

> *Lose the `https://`bit and the `/` stuff before you use it in your code's `<FMI>` later. Basically the bucket name in this instance would be `rh-2020-02-25-s3site`. << Make a note of yours.*

First some basic lab information

As you go though these labs you will have the option to choose either `python`, `node` or `java`. Whenever you come to a section in a lab which starts with this symbol you will have to choose your ~~least hated~~ most loved language, and navigate into that folder via the Cloud9 `CMD_LINE` using the stated `<FMI>` *(fill me ins).*

Stage 2 - Setting A Bucket Policy On Your Bucket

Now you have a bucket created and live. You need to set the permissions on it. Currently everything is blocked to the outside world, as this is the default status of objects in S3 buckets.

Note that we are also locking down a `report.html` page for access via a pre-signed URL for later on in the course ...more about that later.

You need to allow access to kiosks and cellphones that are on the WIFI network. You do this by adding a **bucket policy** to your bucket where you can enforce a **condition**.

1. Head to the **AWS Cloud9** tab. Ensure `Building_2.0` is the highlighted folder by clicking on it. Then right click and choose **New File** and call it `website_security_policy.json`.

Lab Intro

Welcome to the labs section of Building Modern Web Applications on AWS.

If you have not watched my Lab Overview and Lab 1 intro. I suggest you stop here and go and watch them first so you know what your lab gameplay is going to be.

However just to quickly recap (and If you want to play along with the lab story). I'd like you to imagine this scenario:

The Lab Scenario

Context

I'm going to have you play the role of a **full-stack web developer**. Imagine that you've just picked up a nice little development gig at a local camera reseller where they sell lightly-used camera equipment.

The owner Mike, tells you that they have a great deal of foot traffic. However because they are only a small company with two sales people, they don't really have the resources to spend quality time with customers when it gets really busy.

He has decided to buy some in-store kiosks, and this is where you come in.

He wants you to write the application that is to be used on these kiosks. He doesn't care too much about the technology you use as long as it's on AWS and is leveraging a modern backend.

This is a high level overview of what Mike wants the kiosks (the app) to do.

```
        {
            "Sid": "DenyOneObjectIfRequestNotSigned",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<FMI_1>/report.html",
            "Condition": {
                "StringNotEquals": {
                    "s3:authtype": "REST-QUERY-STRING"
                }
            }
        }
    ]
}
```

As an example: once you have replaced the `<FMI>`s it will look a bit like this:

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::rh-2020-02-25-s3site/*",
                "arn:aws:s3:::rh-2020-02-25-s3site"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "23.23.23.23/32"
                    ]
                }
            }
        },
        {
            "Sid": "DenyOneObjectIfRequestNotSigned",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::rh-2020-02-25-s3site/report.html",
```

```
            "Condition": {
                "StringNotEquals": {
                    "s3:authtype": "REST-QUERY-STRING"
                }
            }
        }
    ]
}
```

Save that file.

2. You now need to tell your S3 bucket to apply this new bucket policy. You will need to use the `permissions` file in your respective language folder to do this.

   🔬 You will find a file or folder called `permissions` inside your respective code folder.

   Navigate into your respective code folder using the `CMD_LINE`

```
cd <FMI>
#e.g cd node_10.8.1
```

   ⚠️ **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to navigate to the parent folder for the program to run. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `permissions` for node and python, therefore the folder you should navigate to in the java folder is the `permissions` folder. Under that folder you should see a `pom.xml` file - that is how you know you are in the right spot :).

3. Open the `permissions` file in Cloud9 by double clicking it

   ⚠️ **If you are using Java** open the `App.java` file under `src/main/java/com/mycompany/app/`

4. Now edit the file by replacing the `<FMI>`s with the name of your bucket that used before. Example: `rh-2020-02-25-s3site`.

   Once you have edited the `permissions` file. **Save** it, and run it as per the table below:

   🤷 *You should already be in the right folder in the `CMD_LINE`. If not, navigate to the correct folder.*

   The following table shows the respective **run command** needed to run your recently edited `permissions` file:

Language	Filename You Need To Edit	Command to run via `CMD_LINE`
Node.js (v10)	permissions.js	`npm install aws-sdk && node permissions.js`
Python (v3)	permissions.py	`sudo pip3 install boto3 && sudo pip3 install --upgrade awscli && python3 permissions.py`
Java	App.java	`mvn clean install`, then run `mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

   Once you run the file in the `CMD_LINE` you should see (regardless of language) the word **done**.

Stage 3 - Upload The Website

Now we have all the permissions set for our website, we need to upload the website to the bucket.

🤷 *The good news is that you don't need to create a website. A basic one has been created for you and is sitting in your Cloud9 resources folder. You just need to upload it.*

You are going to use a script that leveraged the respective SDK to upload this website content to your bucket.

1. There is a file or folder called `upload_items`. This is the file or folder where you will edit the code.

   In this file replace the `<FMI>` entry with your bucket name that you created earlier *(not the website URL, the bucket name)*. **Save** the file. Then run it.

   The following table shows the respective **run command** needed to run your `permissions` file:

   🤷 *Be in the right folder when you run that file. e.g (cd resources/node_10.8.1)*

Language	Filename You Need To Edit	Command to run via `CMD_LINE`
Node.js (v10)	upload_items.js	`node upload_items.js`
Python (v3)	upload_items.py	`python3 upload_items.py`
Java	App.java	`mvn clean install`, then run `mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

   Once you run the file in the `CMD_LINE` you should see (regardless of language) the word **done**.

2. It's time to test the site and make sure it works via a virtual secure endpoint. You made a note of your bucket name.

   You can construct the URL in the following way: Where `<FMI>` is your bucket name.

```
https://<FMI>.s3-us-west-2.amazonaws.com/index.html
```

   For example.

```
https://rh-2020-02-25-s3site.s3-us-west-2.amazonaws.com/index.html
```

3. Visit that URL in you web browser to test it. Verify your website is now showing up.

   Try accessing that URL out side of your network. 🤷 *You could test by using your cellphone off your WIFI to get a different IPv4.*

You should only be able to access this website **IF** you are at that IPv4. Pretend that you are at the camera store and you are at a Kiosk.

🤷 *Keep in mind this will not work from testing within the Cloud9 environment. Only from the Source IP that was allowed.*

The JavaScript code in this website essentially allows for interaction with an API (that does not yet exist). When you create the API *in the next lab*. You will have to edit the `config.js` file and then re-upload it using a similar `upload_items` script.

That is why if you choose an item from the dropdown and press either the `get reviews`, `get rating` or `request report` buttons. You will get a message saying you have no API to call.

This is expected behavior right now.

Stage 4 [optional] - Look At The Website Code

I recommend looking at the JS code in the `main.js` file. You don't need to be a JavaScripter or follow everything that is going on in that file, but I think a glance will help you put everything into context as you progress through the other 5 labs.

📓 *You can see it checks for presence of an API which we don't have yet.*

Perfect!

You have a website up and working on all their kiosks.

Mike is happy that things seem to be moving forward quickly, and you look forward to tomorrow's task. (Lab2) where you will start wiring up the plumbing of the site, in terms of building it's back-end mock API.

Lab Complete

Congratulations! You have completed the lab.

1. Click **End Lab** at the top of this page and then click **Yes** to confirm that you want to end the lab.
2. A panel will appear, indicating that "DELETE has been initiated... You may close this message box now."
3. Click the **X** in the top right corner to close the panel.

For feedback, suggestions, or corrections, please contact us at:https://support.aws.amazon.com/#/contacts/aws-training