[version_1.0.4]

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

Exercise: Process

Story so Far

Last day 🎉

You feel like you are in the final stages of your project, and fingers crossed you'll be done around lunch time for a nice early weekend.

All you really need to do this morning is update your `create_report` Lambda function to send the IP and cellphone over to a background process. This will then somehow generate an HTML report and have it sent (protected) to the respective (logged in) manager's cell.

Your idea for the report is that it should figure out (using machine learning) how the reviewer feels about the product, and highlight any relevant tags identifying things like common brand names or locations.

You plan on using AWS STEP functions to orchestrate all these background processes.

You sketch out a rough gam plan as follow:

Here is your current game plan.

- Create a basic "placeholder" step function that validates any passed cellphone and IPv4 address. Return a pre-signed URL allowing access to (a currently blank) HTML report.
- Update your old `create_report` function and have it access that new placeholder step function ARN. Start it off by passing in the IPv4 and cellphone that it gets from Cognito.
- Create 3 functions that do all the sentiment analysis, tagging, and report creation stuff.
- Link them all together by updating your step function, so it is no longer a placeholder and actually does what you want it to do. Test that it creates and sends out a report to your cellphone.
- Get this all working and signed off by Mike so you can get out of here after lunch, and enjoy a nice long weekend.

You will learn in this lab:

- Learn how to work with AWS Step Functions.
- How to use AWS Lambda Functions in parallel.
- How to work with and update step definition files.
- How to use Amazon Simple Notification Service (SNS) to send text messages directly from AWS Step Functions.

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-5-process.zip
   ```

3. Unzip:

   ```
   unzip lab-5-process.zip
   ```

4. Let's cleanup:

   ```
   rm lab-5-process.zip
   ```

5. Run the `resources/setup.sh` script that will grab the website contents and upload them to the S3 bucket created by our CloudFormation template.

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

   This will prompt you to provide your email, cellphone and IPv4. You can get your Ipv4 from https://www.whatismyip.com/. Please use your real email and cellphone (no + country code needed).

   ⚠ **If you are using Java** you will also need to run the following scripts and commands:

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

6. Take a take a look at `resources/website/config.js` file. The last script should have modified that file to include the API URL and the Cognito hosted endpoint.

   It will look a bit like this:

```
var G_API_GW_URL_STR = "https://5n0m22le02.execute-api.us-east-1.amazonaws.com/test";
var G_COGNITO_HOSTED_URL_STR = "https://fancy35-domain.auth.us-east-1.amazoncognito.com/login?client_id=1v236pbj5dtupo14umu4q3vspi&response_type=token&scope=openid+profile&redirect_uri=https://foo-s3bucket-vvoru4b4wa5q.s3-website-us-west-2.amazonaws.com/callback.html";
```

Prior to the *DONE*, there are some URLs.

7. Copy the **callback URL** and paste it into the browser replacing `/callback.html` with `/index.html`

   👨‍🏫 *This saves you looking up the website URL in S3.*

   You should now be looking at the kiosk (website).

   If you try and **request a report** it will say something similar:

   ```
   You need to be logged in to create a report
   ```

   👨‍🏫 *This is expected behavior when one is not logged in ;).*

   Click **Go to the login area** and you will be taken to your Cognito login page.

   Use the credentials for your Cognito pool account. Which are as follows:

   ```
   ricky
   !FooBar55
   ```

   This will redirect you back to the kiosk website. Just so you are aware of how this works:

   Once Cognito authenticates. It redirects to `/callback.html` along with an id token (amongst other things). The website JS then extracts that ID and stores it in `HTML5 localStorage`. Which then redirects to `/index.html`

   Now If you try and **request a report** it will say:

   ```
   Report processing, check your phone shortly!
   ```

   👨‍🏫 *As we have not written the backend process for this function yet, you will not get a text message like it tells you.*

This is where the lab really starts.

Context

From the last Lab we have a web front end that it set up to call API Gateway, and show real data for `get_average_rating` and `get_reviews`.

However the `create_report` we had in Lab 4 is only returning cellphone information, an IPv4 address, and a simple message

We need to update this `create_report` code to call out to AWS Step Functions, passing in the previously returned username and cellphone number. Then have it set off an orchestration of new Lambda functions to run in the background.

The first function that forms this orchestration will `validate` the IP and cellphone, and the second one will `create a presigned url` for an S3 HTML page called `report.html`.

Although we will need a updated `create_report` to do this, we *first* need to create a basic step function system that we can reference as the **STEP_ARN** in said function.

And...before that, we need two ARNs of two prebuilt Lambda functions that are to be used within the "soon to be created" step function system itself.

👩‍🏫 *To save on time, I have created 3 functions for you (in node) and have uploaded them already. I have left a copy of the code used for them in the resources folder, in case you are interested.*

The first function is the updated version of `create_report`.

The key difference between this function and the one you had in Lab 4 is that it looks for an environmental variable (the **STEP_ARN**).

If it finds one, it uses it to call the `startExecution` method with an input payload. Which of course would be the cellphone and IPv4 address.

The other two (background) functions that have been created for you: `validate` and `create_presigned_url`. They do the following:

The `validate` one checks that the IPv4 and cellphone match the ones on record. 👩‍🏫 *The FMIs you see in there are placeholders for code in the setup, you don't have to fill those in. This is just for your reference.*

The `presigned_url` one creates a URL that allows access to `report.html`, which is to be blank for now but will be overwritten with a real report later. 👩‍🏫 *If you try and access report.html right now, you will get access denied, as you would expect.*

In order to work with step functions you need a definition file, and that requires the ARNs of those two background Lambda's.

You can get them from the command line, as follows:

Steps

Head to the Cloud 9 terminal and uses these commands to get both these ARNs.

```
aws lambda list-functions | grep function:validate | cut -f2- -d: | tr -d "," | xargs
```

This should give you something a but like this:

```
arn:aws:lambda:us-west-2:000000000000:function:validate
```

Then *again* for the other function, use:

```
aws lambda list-functions | grep function:create_pre_signed_url | cut -f2- -d: | tr -d "," | xargs
```

To give you something like this:

```
arn:aws:lambda:us-west-2:000000000000:function:create_pre_signed_url
```

💁 *These two node functions have been created for you already, and you are free to check out the code. There is a copy of these functions in your resources folder just for reference. Later on in this lab when you add more Lambdas functions to the step function process, we will give you the opportunity to work with code in your preferred language.*

1. Open `resources/definition.json` and replace the two `<FMI>s` with the correct two ARNs. Don't forget to **save** the file. `<FMI_1>` is for the `validate` ARN and the `< FMI_2>` is for the `create_pre_signed_url` ARN.

   As an **example**: Your ARNs will be different.

   ```
   {
       "Comment": "Basic step function process to Validate IP and cellphone",
       "StartAt": "Validate",
       "States": {
           "Validate": {
               "Type": "Task",
               "Resource": "arn:aws:lambda:us-west-
   2:000000000000:function:validate",
               "Next": "CreatePreSignedUrl"
           },
           "CreatePreSignedUrl": {
               "Type": "Task",
               "Resource": "arn:aws:lambda:us-west-
   2:000000000000:function:create_pre_signed_url",
               "End": true
           }
       }
   }
   ```

   This definition file is going to be used by a file called `stepfun`. You can see it calls 2 Lambda functions one after the other.

   Once you have saved your `definition.json` file.

   🔬 Open your respective `stepfun` file for your language and read through the code.

   ⚠ **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to open the parent folder to find the App.java file. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `stepfun` for node and python, therefore the folder you should navigate to in the java folder is the `stepfun` folder. From there you can drill down into the project structure to find the App.java file.

   You will see that it gets the correct **IAM role** and **creates a state machine** using that definition you just created. 💁 You do not need to alter anything in that file as there are no `<FMI>s`.

2. Run this script `stepfun`. You will need to navigate to the respective language folder. Example: `cd node_10.18.1`.

Language	CMD
Node	`npm install aws-sdk && node stepfun.js`
Python	`python3 stepfun.py`
java_8	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

Now that you have run your `stepfun` command. You should have a step function in place.

So head over to the console to test it:

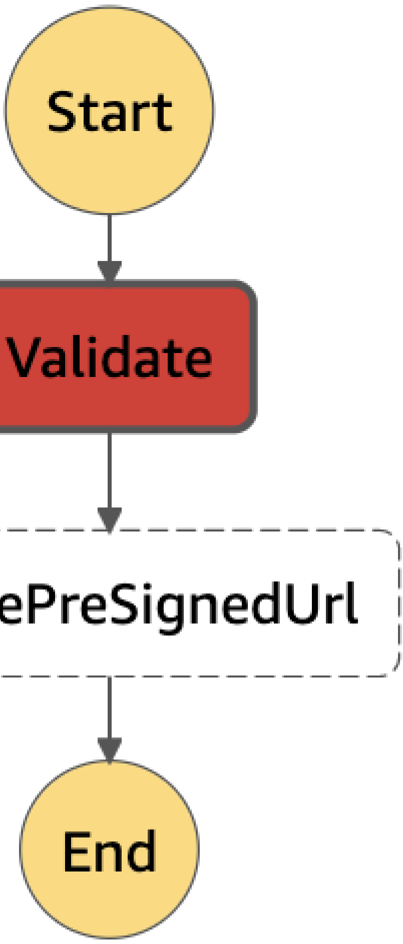
Stage 2 - Testing your step function

Let's see what our script built, and try and make it fail by not passing in the correct ipv4 or cellphone.

1. Choose **AWS Cloud9** and **Go To Your Dashboard**. Search for **Step Functions**. Choose the **Fancy-StateMachine**. Choose **Start execution** and leave the payload as the default: 💁 If the state machine is not showing use the menu on the left to choose State Machines.

```
{
    "Comment": "Insert your JSON here"
}
```

2. Choose **Start execution**. You should see the following:



3. This is to be expected, as you are not passing in the correct payload.

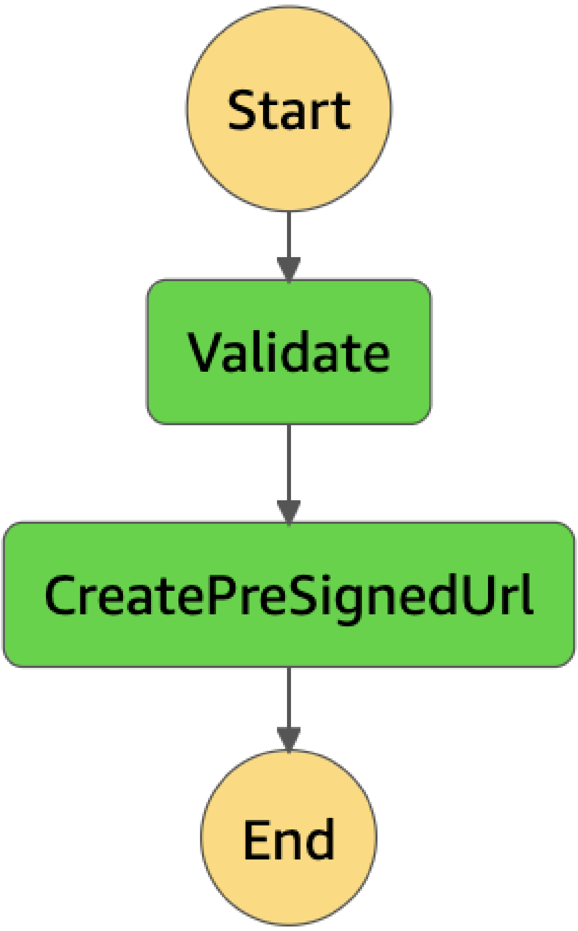
4. Now try again by choosing **New execution** at the top, and use the following payload. Swap out the two `<FMI>s` for a the valid cell phone and IP address you provided earlier during set up:

   This is the type of payload that we will ultimately be sending from `create_report`

   ```
   {
     "cellphone_str": "<FMI_1>", //use your cell in this format eg. "+1xxxxxxxxxx"
     "ipv4_str": "<FMI_2>" //use your IP in this format e.g. "x.x.x.x"
   }
   ```

   Remove the comments so it will be valid JSON.

   You should see the following after you click **Start execution**.



4. So it checked if your cell phone number is correct. If you try a fake number or an incorrect IP it will fail. **Perfect**. Now we can move onto adjusting our `create_report_code` code to send the payload to this step function.
5. Now we can grab the ARN. Choose **State machines** and **Fancy-StateMachine**. The ARN is listed at the top under **Details**:

   As an example: Yours will be different:

   ```
   arn:aws:states:us-west-2:000000000000:stateMachine:Fancy-StateMachine
   ```

   The code for `create_report` that was created and uploaded for you as part of the Lab set up is (as touched on earlier) looking for an environmental variable called `STEP_ARN` . If it doesn't find one it would just ignores the step function triggering bit, and returns the old message.
6. So in order for the code to know which step function to call, you need to pass in that ARN to the Lambda's environmental variables. Switch back to the **Cloud9** tab and run the following using your ARN where you see the `<FMI>` below:

```
aws lambda update-function-configuration --function-name create_report --
environment Variables="{STEP_ARN=<FMI>}"
```

Example: Yours will be different:

```
aws lambda update-function-configuration --function-name create_report --
environment Variables="{STEP_ARN=arn:aws:states:us-west-
2:000000000000:stateMachine:Fancy-StateMachine}"
```

Output will be similar to this:

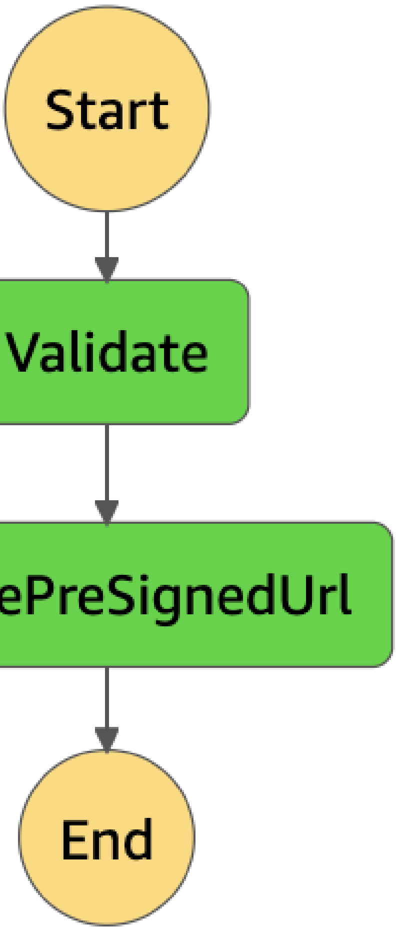
```
{
    "FunctionName": "create_report",
    "FunctionArn": "arn:aws:lambda:us-west-
2:000000000000:function:create_report",
    "Runtime": "nodejs10.x",
    "Role": "arn:aws:iam::000000000000:role/lab5-lambda-role",
    "Handler": "index.handler",
    "CodeSize": 863,
    "Description": "Lambda Function",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2020-06-11T16:08:44.308+0000",
    "CodeSha256": "dc1Qzd1Jy3S74OcJSPUXpviNRfXFbrMNRprFZ/BRzUQ=",
    "Version": "$LATEST",
    "Environment": {
        "Variables": {
            "STEP_ARN": "arn:aws:states:us-west-
2:000000000000:stateMachine:Fancy-StateMachine"
        }
    },
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "d37dbe5c-76e5-4feb-aed6-b05560cbd5e5",
    "State": "Active",
    "LastUpdateStatus": "Successful"
}
```

7. Switch back to your **website**.

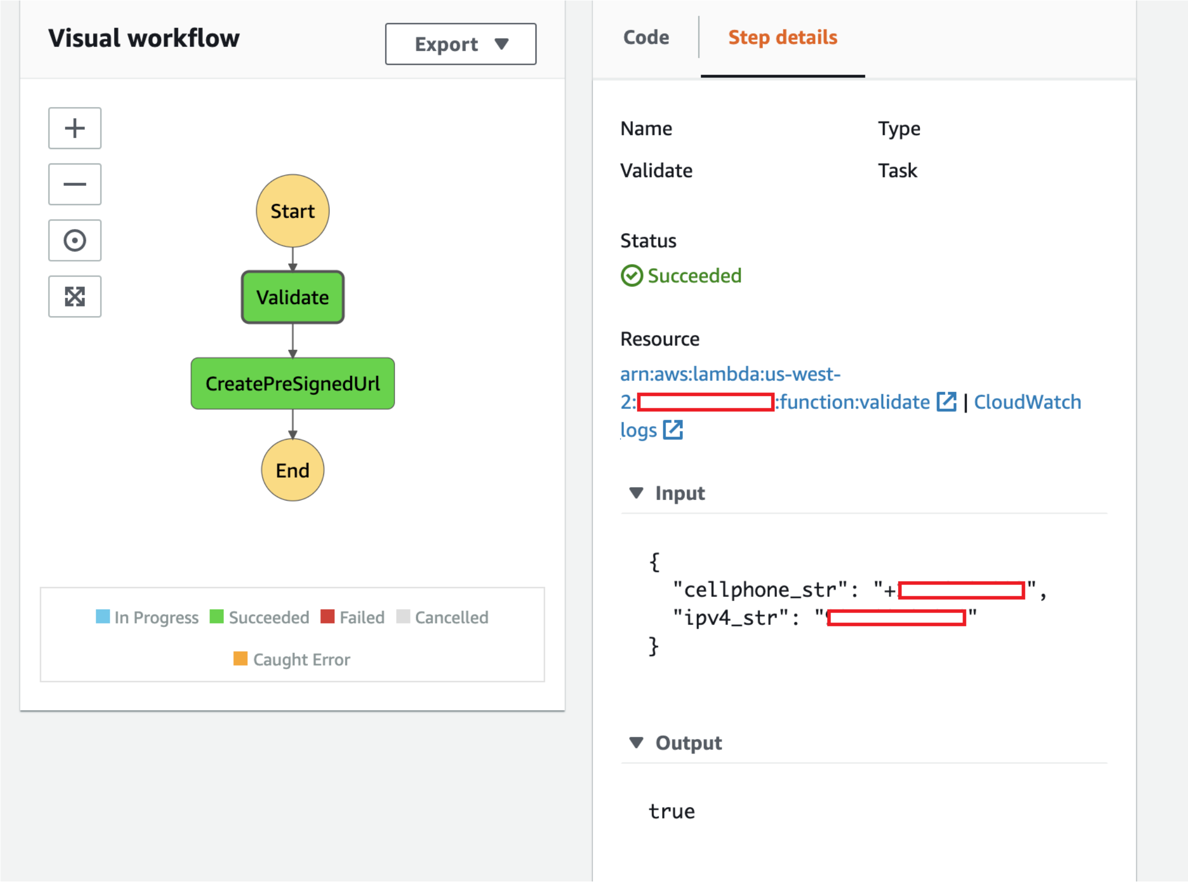
   🧑‍💻 Login creds if needed.

```
ricky and !FooBar55
```

And choose **REQUEST A REPORT**. Then switch back to the **Step Functions** tab and refresh the **Executions** to see that there should be an execution that was successful.



Most significantly. If we select the validate green button, look on the right, and open **Input** we will see that the input included the cell phone number (from your prior Cognito login) and your current IPv4 address. Like so.



Name Validate

Type Task

Status ✓ Succeeded

Resource arn:aws:lambda:us-west-2:[redacted]:function:validate | CloudWatch logs

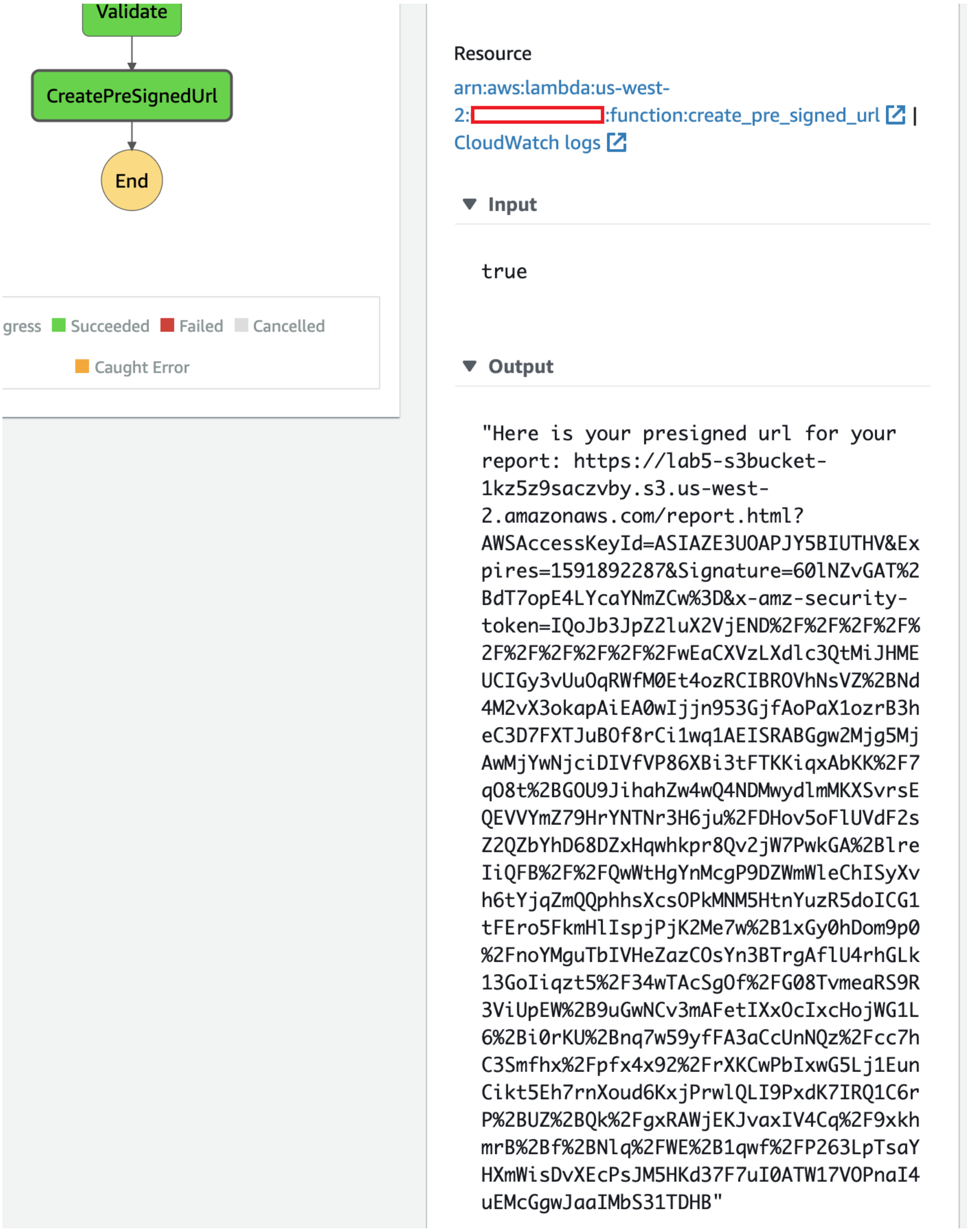
▼ Input

```
{
  "cellphone_str": "+[redacted]",
  "ipv4_str": "[redacted]"
}
```

▼ Output

```
true
```

Now choose the `CreatePreSignedURL` green button and look at the output:



Resource

arn:aws:lambda:us-west-2:[redacted]:function:create_pre_signed_url | CloudWatch logs

▼ Input

```
true
```

▼ Output

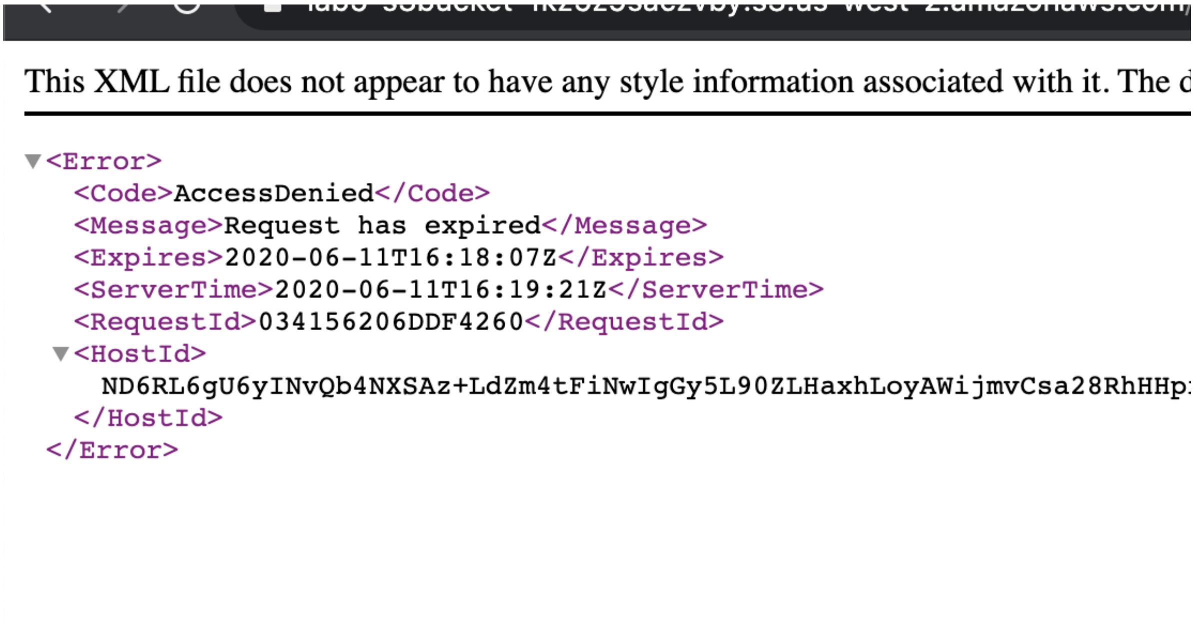
```
"Here is your presigned url for your
report: https://lab5-s3bucket-
1kz5z9saczvby.s3.us-west-
2.amazonaws.com/report.html?
AWSAccessKeyId=ASIAZE3UOAPJY5BIUTHV&Ex
pires=1591892287&Signature=60lNZvGAT%2
BdT7opE4LYcaYNmZCw%3D&x-amz-security-
token=IQoJb3JpZ2luX2VjEND%2F%2F%2F%2F%
2F%2F%2F%2F%2F%2FwEaCXVzLXdlc3QtMiJHME
UCIGy3vUuOqRWfM0Et4ozRCIBROVhNsVZ%2BNd
4M2vX3okapAiEA0wIjjn953GjfAoPaX1ozrB3h
eC3D7FXTJuBOf8rCi1wq1AEISRABGgw2Mjg5Mj
AwMjYwNjciDIVfVP86XBi3tFTKKiqxAbKK%2F7
qO8t%2BGOU9JihahZw4wQ4NDMwydlmMKXSvrsE
QEVVYmZ79HrYNTNr3H6ju%2FDHov5oFlUVdF2s
Z2QZbYhD68DZxHqwhkpr8Qv2jW7PwkGA%2Blre
IiQFB%2F%2FQwWtHgYnMcgP9DZWmWleChISyXv
h6tYjqZmQQphhsXcsOPkMNM5HtnYuzR5doICG1
tFEro5FkmHlIspjPjK2Me7w%2B1xGy0hDom9p0
%2FnoYMguTbIVHeZazCOsYn3BTrgAflU4rhGLk
13GoIiqzt5%2F34wTAcSgOf%2FG08TvmeaRS9R
3ViUpEW%2B9uGwNCv3mAFetIXxOcIxcHojWG1L
6%2Bi0rKU%2Bnq7w59yfFA3aCcUnNQz%2Fcc7h
C3Smfhx%2Fpfx4x92%2FrXKCwPbIxwG5Lj1Eun
Cikt5Eh7rnXoud6KxjPrwlQLI9PxdK7IRQ1C6r
P%2BUZ%2BQk%2FgxRAWjEKJvaxIV4Cq%2F9xkh
mrB%2Bf%2BNlq%2FWE%2B1qwf%2FP263LpTsaY
HXmWisDvXEcPsJM5HKd37F7uI0ATW17VOPnaI4
uEMcGgwJaaIMbS31TDHB"
```

Try that URL in your browser:

You should see a page showing the word "Report". It will have no content.

Meaning you have access to the report HTML page (temporarily at least).

Refresh the page and you will see it expires after 2 minutes, and then like normal you will no longer have access to that report page.



This XML file does not appear to have any style information associated with it. The d

```
▼<Error>
   <Code>AccessDenied</Code>
   <Message>Request has expired</Message>
   <Expires>2020-06-11T16:18:07Z</Expires>
   <ServerTime>2020-06-11T16:19:21Z</ServerTime>
   <RequestId>034156206DDF4260</RequestId>
 ▼<HostId>
     ND6RL6gU6yINvQb4NXSAz+LdZm4tFiNwIgGy5L90ZLHaxhLoyAWijmvCsa28RhHp:
   </HostId>
 </Error>
```

🧑‍🔬 *Check out* `public_policy.json` *in resources to see how we protect this resource, yet allow a pre-signed URL, if you are interested*

Perfect!

We have our website triggering a step function process.

Now to make it do useful stuff, and actually create a report..onto stage 3 of the lab.

Stage 3

Use the language of your choice to create three new Lambda functions, **two** of which can be run in parallel inside your step function process. The final **one** to run after both parallel functions have completed.

The first function you will upload will be called `sentiment`. It will tell you how people that reviewed the product felt about it, using Machine Learning!

🔬

1. Switch back to the **Cloud9** tab. We will start with the `sentiment` function.
2. Navigate to your respective code folder on the `CMD_LINE`.

⚠ If you are using Java, make sure you navigate to the folder that contains the `build.gradle` file.

3. Once you have read through your `sentiment_code` file, head over to the `CMD_LINE` and zip up the `sentiment_code` file into a file called `sentiment.zip`. Use the respective commands for this:

Language	CMD
python_3.6.8	`zip sentiment.zip sentiment_code.py`
node_10.8.1	`zip sentiment.zip sentiment_code.js`
java_8	`gradle build`

Expected output:

```
adding: sentiment_code.js (deflated 67%)
```

Expected output for Java:

```
BUILD SUCCESSFUL in 21s
4 actionable tasks: 4 executed
```

So now we have a zipped up asset with that code in it. **Awesome**!

We are now ready to tell Lambda to use this in a newly created function that we are going to call `sentiment`.

⚠ **If you are using Java** change directories up one level to the `java_8` folder to go through the next steps.

4. You will need to upload this newly created `sentiment.zip` to a S3 bucket using the respective `uploadToS3` script in your resources folder.

   👨‍🏫 *To save on unnecessary lab steps. I am going to have you shove this zip file in our existing website folder. Lazy I know, in prod you would obviously use a private bucket to store your lambda code.*

   So again you will need to choose your language specific `uploadToS3` script. As follows:

   ⚠ *If you get a permission warning on your cmd below, simply run a* `chmod + x` *on the command line for that file. Eg. for Python* `chmod +x ../resources/uploadToS3_python.sh`

Language	CMD
python_3.6.8	`../resources/uploadToS3_python.sh`
node_10.8.1	`../resources/uploadToS3_node.sh`
java_8.	`../resources/uploadToS3_java.sh`

   You should get an output similar to this:

   ```
   upload: ./sentiment.zip to s3://c11284a125438u294892t1w468500737961-s3bucket-iax7xho9pjdo/sentiment.zip
   ```

5. Ok **great**, you have your code all ready to be consumed by your "about to be created" Lambda function.

   You will use the SDK for this.

   👨‍🏫 *You don't need to alter anything in that code as there are no* `<FMI>s`*. You simply need to run the respective script.*

   You should still be in your correct language folder in the `CMD_LINE`.

   ⚠ **If you are using Java**, change directories into the `sentiment` folder that contains a `pom.xml` file.

   Run `sentiment` from that folder. This will create your sentiment lambda function from the code you just uploaded.

   Like this:

Language	CMD
python_3.6.8	`python3 sentiment.py`
node_10.8.1	`node sentiment.js`
java_8	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

Output should give you

```
Done
```

⚠ **If you are using Java**, the output will say BUILD SUCCESS along with other information. Please ignore any warnings from org.slf4j.impl.StaticLoggerBinder about logging, as long as it says BUILD SUCCESS in the output you can move onto the next step.

Now lets do the second function which will run in parallel to sentiment.

This function is going to be called `tag` and will asses a review and identify certain keywords, such as a brand or an organization. This will be useful in our report.

Ok, now onto the other Lambda function called "tag". First things first, have a read through `tag_code` if you have not already done so.

⚠ **If you are using Java**, change directories into the `tag_code` folder that contains a `build.gradle` file.

Then **zip up the code** using the correct command for your language:

Language	CMD
python_3.6.8	`zip tag.zip tag_code.py`
node_10.8.1	`zip tag.zip tag_code.js`
java_8	`gradle build`

Output will be similar to this:

```
adding: tag.js (deflated 67%)
```

Open your respective shell script file in your resources folder called `uploadToS3_xxxxx.sh` in Cloud9 (double click it).

⚠ Before you run this you will need to modify it.

Once you have the file open in Cloud9 and make sure to do the following:

A) *uncomment* out the line that uploads the `tag.zip`

B) *comment* out the `sentiment` one (as you don't need to re-upload that one)

As in this example (*If you are working with Node of course*):

```
Before:
aws s3 cp ~/environment/node_10.18.1/sentiment.zip s3://$bucket/sentiment.zip
#aws s3 cp ~/environment/node_10.18.1/tag.zip s3://$bucket/tag.zip
#aws s3 cp ~/environment/node_10.18.1/publish.zip s3://$bucket/publish.zip

After:
#aws s3 cp ~/environment/node_10.18.1/sentiment.zip
s3://$bucket/sentiment.zip
aws s3 cp ~/environment/node_10.18.1/tag.zip s3://$bucket/tag.zip
#aws s3 cp ~/environment/node_10.18.1/publish.zip s3://$bucket/publish.zip
```

Choose **File** and **Save**.

Now run it as follows (just like before). You should still be in your respective language folder in the `CMD_LINE`.

⚠ **If you are using Java** change directories up one level to the `java_8` folder to go through the next steps.

Language	CMD
python_3.6.8	`../resources/uploadToS3_python.sh`
node_10.8.1	`../resources/uploadToS3_node.sh`
node_10.8.1	`../resources/uploadToS3_java.sh`

This will give you output similar to this:

```
upload: ./tag.zip to s3://c11284a125438u294892t1w468500737961-s3bucket-
cy38ogpmfrvz/tag.zip
```

6. Now you have your code zipped up and in S3. It's time to create a Lambda function from it.

Create your second Lambda function using the respective `tag` script:

⚠ **If you are using Java**, change directories into the `tag` folder that contains a `pom.xml` file.

👨‍🏫 *Read the code in `tag` so you can see what it is doing.*

Language	CMD
python_3.6.8	`python3 tag.py`
node_10.8.1	`node tag.js`
java_8	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

Output should give you:

```
Done
```

⚠ **If you are using Java**, the output will say BUILD SUCCESS along with other information. Please ignore any warnings from org.slf4j.impl.StaticLoggerBinder about logging, as long as it says BUILD SUCCESS in the output you can move onto the next step.

Now for the final 3rd and final function, called `Publish`.

This function is designed to take the results of the 2 parallel functions `sentiment` and `tag` and merge the results into a HTML report page and publish (overwrite) `report.html`

⚠ **If you are using Java**, change directories into the `publish-code` folder that contains a `build.gradle` file.

First things first. After reading `publish-code`. Zip up the code using the correct command for your language:

Language	CMD
python_3.6.8	`zip publish.zip publish_code.py`
node_10.8.1	`zip publish.zip publish_code.js`
java_8	`gradle build`

Output will be similar to this:

```
adding: publish.js (deflated 67%)
```

Open your respective shell script file in your resources folder called `uploadToS3_xxxxx.sh` in Cloud 9 (double click it).

⚠ Before you run this you will need to modify it.

Once you have the file open in Cloud9 and make sure to do the following:

A) *uncomment* out the line that uploads the `publish.zip`.

B) *comment* out the `tag` one (as you don't need to re-upload that one).

As in this example (*If you are working with Node of course*):

```
Before:
#aws s3 cp ~/environment/node_10.18.1/sentiment.zip
s3://$bucket/sentiment.zip
aws s3 cp ~/environment/node_10.18.1/tag.zip s3://$bucket/tag.zip
#aws s3 cp ~/environment/node_10.18.1/publish.zip s3://$bucket/publish.zip

After:
#aws s3 cp ~/environment/node_10.18.1/sentiment.zip
s3://$bucket/sentiment.zip
#aws s3 cp ~/environment/node_10.18.1/tag.zip s3://$bucket/tag.zip
aws s3 cp ~/environment/node_10.18.1/publish.zip s3://$bucket/publish.zip
```

Choose **File** and **Save**.

Now run it as follows (just like before).

⚠ **If you are using Java** change directories up one level to the `java_8` folder to go through the next steps.

You should still be in your respective language folder in the `CMD_LINE`.

Language	CMD
python_3.6.8	`../resources/uploadToS3_python.sh`
node_10.8.1	`../resources/uploadToS3_node.sh`
java_8	`../resources/uploadToS3_java.sh`

This will give you output similar to this:

```
upload: ./publish.zip to s3://c11284a125438u294892t1w468500737961-s3bucket-cy38ogpmfrvz/publish.zip
```

7. Now you have your code zipped up and in S3. The final step is to create the Lambda function.

   Create your final Lambda function using the respective `publish` script:

   ⚠️ **If you are using Java**, change directories into the `tag` folder that contains a `pom.xml` file.

   🧑‍💻 *Read the code in `publish` so you can see what it is doing.*

Language	CMD
python_3.6.8	`python3 publish.py`
node_10.8.1	`node publish.js`
java_8	`mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App`

   Output should give you:

   ```
   Done
   ```

   ⚠️ **If you are using Java**, the output will say BUILD SUCCESS along with other information. Please ignore any warnings from org.slf4j.impl.StaticLoggerBinder about logging, as long as it says BUILD SUCCESS in the output you can move onto the next step.

Ok on to stage 4, where will will update our step function process to use these 3 new functions.

Stage 4

We are going to use `definition_2.json` instead of `definition.json` at this stage.

1. Open up the `resources/definition_2.json` file in **Cloud9**.

   As you will see now when it passes validation, instead of hitting `create_presigned_url`, it kicks off a parallel process that runs `sentiment` and `tag`.

   You can see in the *JSON* file that the results of each of these 2 functions are sent to the input of `publish`.

   Once the `publish` function completes. It will then run `create_presigned_url`, which outputs a pre-signed URL.

2. Replace the **5** `<FMI>`s in `definition_2.json` with the correct ARNs.

   🧑‍💻 Each ARN for each Lambda function can be found in your Lambda console.

Keep in mind that you can just tag on the name of the function after grabbing the first ARN value.

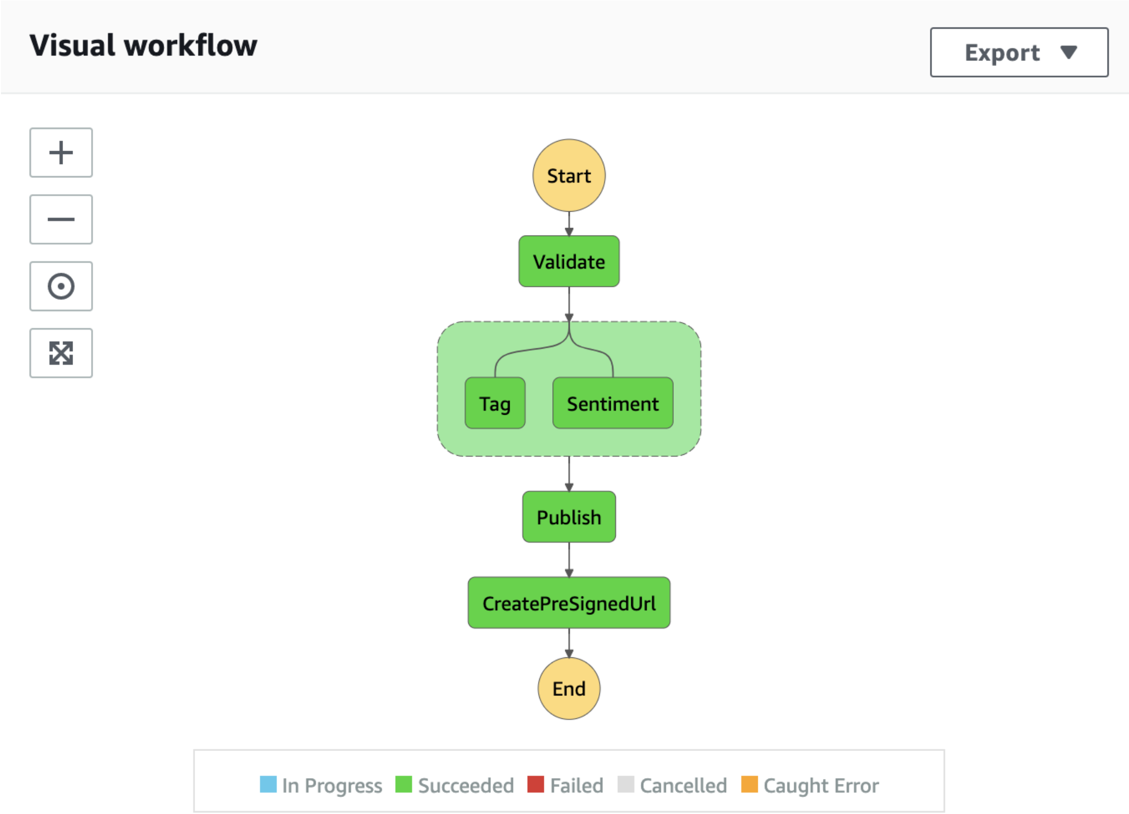
```
aws lambda list-functions | grep function:validate | tr -d "," | cut -f2- -d: | xargs
#Example
#validate arn:aws:lambda:us-west-2:000000000000:function:validate

aws lambda list-functions | grep function:sentiment | tr -d "," | cut -f2- -d: | xargs
#Example
#validate arn:aws:lambda:us-west-2:000000000000:function:sentiment

aws lambda list-functions | grep function:tag | tr -d "," | cut -f2- -d: | xargs
#Example
#validate arn:aws:lambda:us-west-2:000000000000:function:tag

aws lambda list-functions | grep function:publish | tr -d "," | cut -f2- -d: | xargs
#Example
#validate arn:aws:lambda:us-west-2:000000000000:function:publish

aws lambda list-functions | grep function:create_pre_signed_url | tr -d "," | cut -f2- -d: | xargs
#Example
#validate arn:aws:lambda:us-west-2:000000000000:function:arn:aws:lambda:us-west-2:179741345863:function:create_pre_signed_url
```

3. Save this `definition_2.json` file and **Go To Your Dashboard**. Choose **Services** and search for **Step Functions**.
4. Choose the **Fancy-StateMachine** and **Edit**. Paste in the `definition_2.json` and choose **Save**.
5. Head over to the website, and login again (in case your login expired).

   ```
   ricky and !FooBar55
   ```

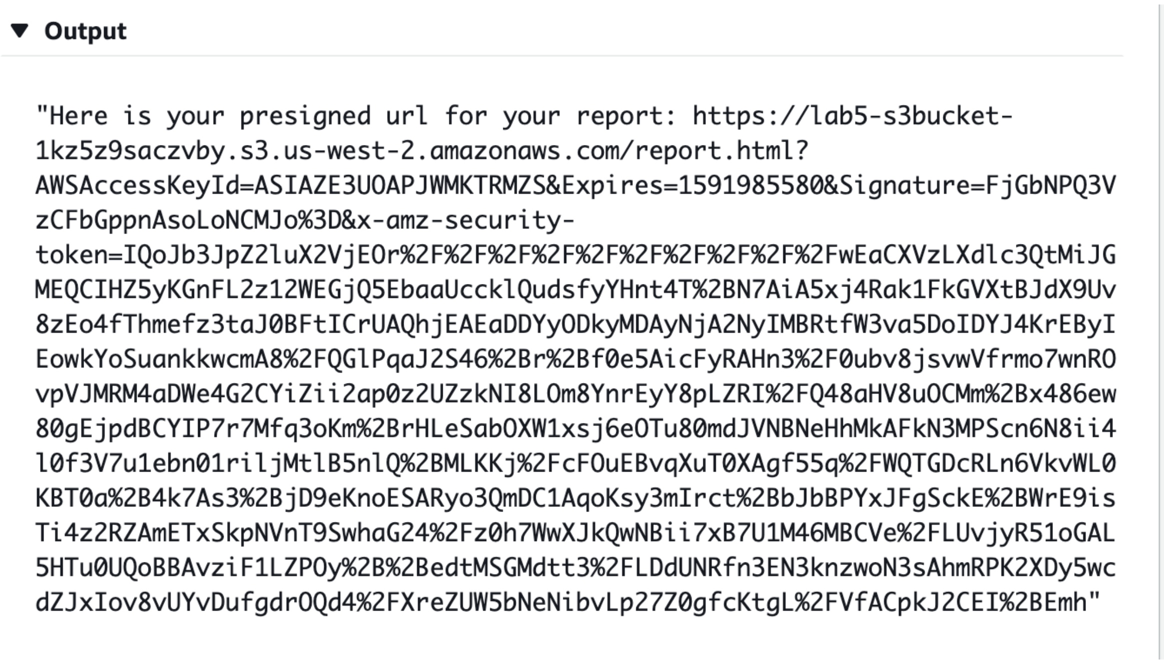
   Then click `REQUEST A REPORT`.
6. You can then go to the Step functions console and see its execution.

   You should see this:



💁‍♂️ Am I the only one that sees it all go green and thinks life is beautiful? 🤔

Click on CreatePresigned to get the URL under the output section.



▼ **Output**

```
"Here is your presigned url for your report: https://lab5-s3bucket-
1kz5z9saczvby.s3.us-west-2.amazonaws.com/report.html?
AWSAccessKeyId=ASIAZE3UOAPJWMKTRMZS&Expires=1591985580&Signature=FjGbNPQ3V
zCFbGppnAsoLoNCMJo%3D&x-amz-security-
token=IQoJb3JpZ2luX2VjEOr%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLXdlc3QtMiJG
MEQCIHZ5yKGnFL2z12WEGjQ5EbaaUccklQudsfyYHnt4T%2BN7AiA5xj4Rak1FkGVXtBJdX9Uv
8zEo4fThmefz3taJ0BFtICrUAQhjEAEaDDYyODkyMDAyNjA2NyIMBRtfW3va5DoIDYJ4KrEByI
EowkYoSuankkwcmA8%2FQGlPqaJ2S46%2Br%2Bf0e5AicFyRAHn3%2F0ubv8jsvwVfrmo7wnRO
vpVJMRM4aDWe4G2CYiZii2ap0z2UZzkNI8LOm8YnrEyY8pLZRI%2FQ48aHV8uOCMm%2Bx486ew
80gEjpdBCYIP7r7Mfq3oKm%2BrHLeSabOXW1xsj6eOTu80mdJVNBNeHhMkAFkN3MPScn6N8ii4
l0f3V7u1ebn01riljMtlB5nlQ%2BMLKKj%2FcFOuEBvqXuT0XAgf55q%2FWQTGDcRLn6VkvWL0
KBT0a%2B4k7As3%2BjD9eKnoESARyo3QmDC1AqoKsy3mIrct%2BbJbBPYxJFgSckE%2BWrE9is
Ti4z2RZAmETxSkpNVnT9SwhaG24%2Fz0h7WwXJkQwNBii7xB7U1M46MBCVe%2FLUvjyR51oGAL
5HTu0UQoBBAvziF1LZPOy%2B%2BedtMSGMdtt3%2FLDdUNRfn3EN3knzwoN3sAhmRPK2XDy5wc
dZJxIov8vUYvDufgdrOQd4%2FXreZUW5bNeNibvLp27Z0gfcKtgL%2FVfACpkJ2CEI%2BEmh"
```

Visit that URL in your browser you should see a full report:

Ok so almost done!

We just need to have this pre-signed URL sent to your cellphone.

Stage5 - getting the text message

Now we have a report, we just need to send the URL output from `create_presigned` to the respective (logged in) manager's mobile phone (yours).

We can leverage Step Function's ability to use Amazon SNS directly.

1. Open `definition_3.json` in Cloud 9 and you will see a final section that is now telling Step Function to use the SNS publish service to send a `message_str` from (the last output from pre-signed) to the `cellphone_str` variable of the initial input of the step function process.

   ⚠ You will also need FMIs 1 through 5, but you can copy that from `definition_2.json` that you created earlier.

2. Choose the **Fancy-StateMachine** and **Edit**. Paste in the `definition_3.json` and choose **Save**.
3. Head over to the website, and login again (in case your login expired). Then click `REQUEST A REPORT`.
4. You can then go to the step functions console and see its process being carried out.
5. Now check your cellphone, you should get a text with a pre-signed URL.
6. Click that URL and you should be able to see the final report on your phone. (it can take a few minutes to arrive)

Wow, you have completed the project! Mike will be happy when you show him the whole thing working. It's only midday on Friday, and you figure you've got the rest of the afternoon to make a start on your weekend!

As you pack up your laptop, you start thinking about what you are going to do at the weekend to celebrate a successful project. Skiing is looking like a good option right now, especially if you can get on the road soon.

So you head over to near Mike's desk, and usher him over to one of the kiosks to show him the final project.

You go though all the steps, and he is really excited about what you have done.

He is smiling ear to ear, and so are you.

Then his expression changes to a thoughtful one touching his chin, and he says "but how do I know this is fully optimized?"

annnnnd...there goes your Friday afternoon (along with any hope of skiiing).

💁‍♂️... yeah, there's a Lab 6 😈

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