TCSS 562: Software Engineering for Cloud Computing
Fall 2019
http://faculty.washington.edu/wlloyd/courses/tcss562

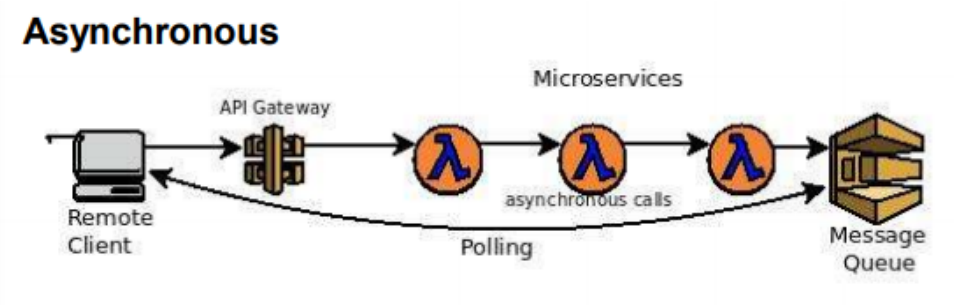
School of Engineering and Technology
University of Washington – Tacoma

Tutorial 5 – Introduction to Lambda II: Working with Files in S3 and CloudWatch Events

Disclaimer: Subject to updates as corrections are found
Version 0.11
Scoring: 20 pts maximum

The purpose of this tutorial is to introduce the use of the Amazon Simple Storage Service (S3) from AWS Lambda to support receiving, processing, and/or creating files. Additionally, this tutorial introduces combining multiple Lambdas into a single Java project. The tutorial also describes how to configure a CloudWatch event rule to trigger a Lambda function in response to an S3 Write Data event that is tracked by setting up a CloudTrail log "trail".

Goals of this tutorial include:
1. Create a new CreateCSV Lambda function to write a file to S3.
2. Create a new ProcessCSV Lambda function to read a file from S3.
3. Package these two Lambda functions into a single Java project to produce a single, composite jar file. The concept of a composite JAR provides the basis for setting up the "Switchboard" architecture by simply adding additional flow-control code.
4. Create a CloudWatch event rule to trigger the ProcessCSV Lambda function as a "target". The event rule is triggered when a file is uploaded to an S3 bucket by the CreateCSV Lambda function. This event is provided by setting up a CloudTrail log trail to track S3 Write Data events. CloudWatch event triggers provide a means to implement **asynchronous application flow control** as in:



For example, Lambda functions can be triggered in response to data being available in S3.

1. Create a new SAAF Lambda function template application

On your laptop, create a new directory for the project files and clone the git SAAF project to start:

```
git clone https://github.com/wlloyduw/SAAF.git
```

Refer to Tutorial #4 for information on the “SAAF”.

2. CreateCSV Lambda Function

In the SAAF project, rename the HelloPOJO.java class to CreateCSV.java.

```
cd {base directory where project was cloned}/SAAF/java_template/src/main/java/
mv HelloPOJO.java CreateCSV.java
```

If using the Netbeans IDE, right click on the classname, and select “Refactor | Rename”.

In the Request.java class in the same directory, add 4 input parameters for this Lambda function. Define getter and setters methods accordingly:

Property Name	Property Type
bucketname	String
filename	String
row	Int
col	int

These properties will allow a client to request the creation of a new CSV file.
The file is stored in the S3 Bucket described by “Bucketname”. The filename is described by “Filename”.
The CSV file will consist of comma-separated random numbers (range 1 to 1000). Row and Col specify the number of total rows and columns in the CSV file.

In the CreateCSV class handleRequest() method, consume the request variables into local variables:

```
        int row = request.getRow();
        int col = request.getCol();
        String bucketname = request.getBucketname();
        String filename = request.getFilename();
```

Then, generate the random matrix of values, storing each row in a separate String:

```
        int val = 0;
        StringWriter sw = new StringWriter();
        Random rand = new Random();

        for (int i=0;i<row;i++)
            for (int j=0;j<col;j++)
            {
                val = rand.nextInt(1000);
                sw.append(Integer.toString(val));
                if ((j+1)!=col)
```

```
                sw.append(",");
            else
                sw.append("\n");
        }
```

The next step is to write Java code to generate the S3 bucket file.

Working with the S3 Java API requires adding the Java support library.
In maven, the library can be added in the pom.xml file.
The pom.xml file is under the java_template directory.

The dependencies can *alternatively* be added through the Netbeans IDE, by RIGHT-clicking on dependencies, and select “Add Dependency”, and then in the Query box type “aws-java-sdk-s3”.

Once found, select the version, such as ~ “1.11.306”…
This automatically includes all Java jar libraries required to work with Amazon S3.

Alternatively, if not using Netbeans, the dependency can be added in the pom.xml file in between the tags “<dependencies> </dependencies>” directly as follows:

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-bom</artifactId>
            <version>1.11.327</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-s3</artifactId>
            <version>1.11.306</version>
        </dependency>
```

Once S3 dependencies have been included in the project, acquire the StringBuilder output as a Byte Array, and use this to create a new input stream. Then create metadata for describing the file to be written to S3 and create the new file on Amazon S3:

```
byte[] bytes = sw.toString().getBytes(StandardCharsets.UTF_8);
InputStream is = new ByteArrayInputStream(bytes);
ObjectMetadata meta = new ObjectMetadata();
meta.setContentLength(bytes.length);
meta.setContentType("text/plain");

// Create new file on S3
AmazonS3 s3Client = AmazonS3ClientBuilder.standard().build();
s3Client.putObject(bucketname, filename, is, meta);
```

The response object of the CreateCSV service will populate the “value” attribute. The value will be a string that describes the CSV file which is created. Modify the r.setValue() to:

```
r.setValue("Bucket:" + bucketname + " filename:" + filename + " size:" +
bytes.length);
```

Once these changes are completed, compile the project.

2. Deploy CreateCSV Lambda Function

Next, using the AWS Management Console, create a new CreateCSV Lambda service. Refer to Tutorial #4 to review this procedure.

For Tutorial #5, choose only ***one*** method for invoking your Lambda functions: either via curl and the API Gateway (REST endpoints), or using the “aws lambda” CLI. ***It is not necessary to configure both methods.***

If choosing curl for function invocation, be sure to configure API Gateway URLs (POST) for your Lambdas. Refer to tutorial #4.

Next, modify the callservice.sh to call your new CreateCSV service.

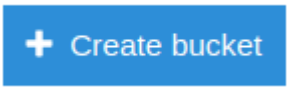
3. Prepare to call CreateCSV Lambda Function: Create S3 Bucket

Begin by copying and adapting the callservice.sh script from tutorial 4. This is in SAAF/java_template/test/callservice.sh. Copy this script to the tutorial 5 project and redefine the input JSON object as follows:

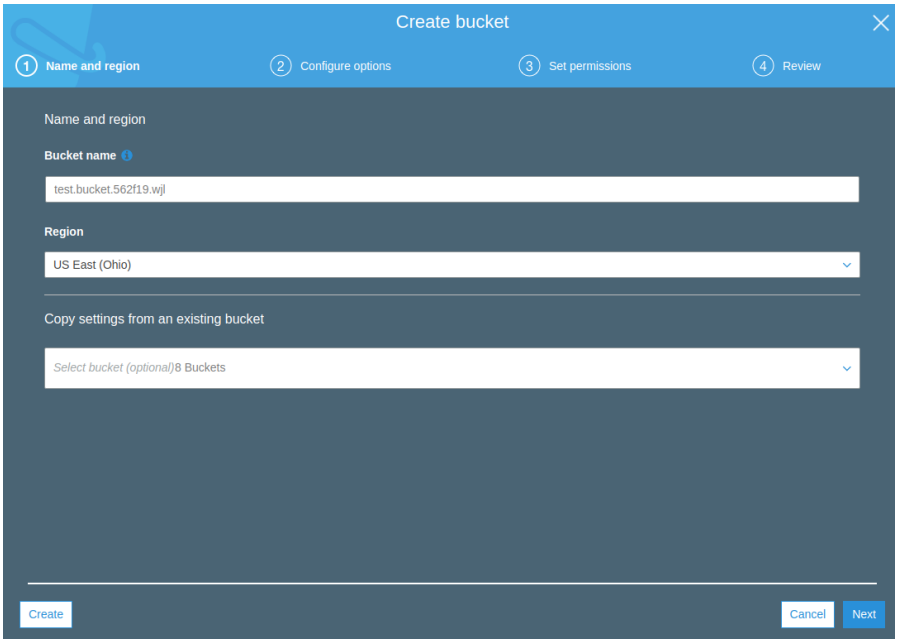
```
json={"\"row\"":50,"\"col\"":10,"\"bucketname\"":\"test.bucket.562f19.aaa\"","\"fi
lename\"":\"test.csv\""}
```

Next, create a bucket called “test.bucket.562f19” using the AWS management console.

Navigate to the “S3” Simple Storage Service from the dropdown list of services, and then inside S3 select the button:



Create the bucket as follows:

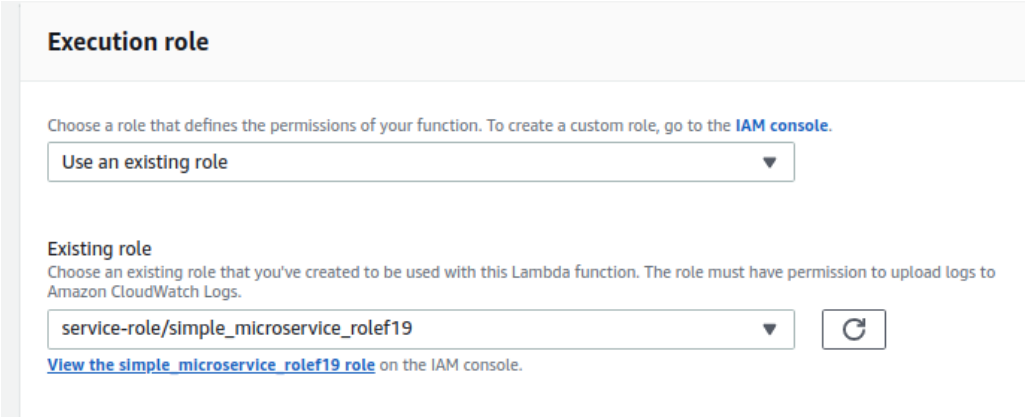


S3 Bucketnames must be unique.
For the bucket name use: **test.bucket.562f19.aaa**

Replace “aaa” with your initials.
If the name is still not unique, modify as needed until it is unique.
Revise the bucket name above in the JSON accordingly.
Press the [NEXT] button.

For all remaining steps in the Create Bucket wizard, accept the default values.

Next, it is necessary to configure permissions for Lambda to access your S3 bucket.
In the AWS Management Console, navigate to Lambda, and inspect your CreateCSV Lambda function. Scroll down to **Execution role**:



Below the existing role, Lambda provides a BLUE link to open your security role called: "**View the simple_microserivce_rolef19 role** on the IAM console". Click on this link to navigate to your role to grant Lambda permission to access your S3 bucket.

Next, in the upper right-hand corner, select your name, and drop-down the menu and choose "My Security Credentials".

(alternatively navigate to IAM in the AWS management console and on the left-hand side select "Roles", and then search for the name of your Lambda execution role (e.g. simple_microservice_rolef19).

Once found, click on the Role_name, and select the button:

Search for the policy: "**AmazonS3FullAccess**" and select the button:

The policy should then appear as added to the Role.
This grants any Lambda function with the Role permission to work with S3.
More fine grained security policies can be specified as needed.

4. Test your CreateCSV Lambda Function

To reduce/change verbosity (the number of metrics returned from SAAF) feel free to replace the call to **inspector.inspectAll();** with another option, or comment out **inspector.inspectAll(); and inspectAllDeltas();** for development purposes right now. See tutorial #4 – page 21 for a list of inspect methods.

Now, using the **`callservice.sh`** script, invoke your Lambda function.
Try creating different sizes of CSV files by increasing or decreasing the values for "row" and "col". Please note, for creating large CSV files, it may be necessary to increase timeout values in the API-Gateway and/or Lambda as creating large CSVs is slow.

```
$ ./callservice.sh
{"row":50,"col":10,"bucketname":"test.bucket.562.aaa","filename":"test.csv"}
Invoking Lambda CreateCSV function using API Gateway

real 0m10.662s
user 0m0.092s
sys 0m0.008s

CURL RESULT:
{"value":"Bucket: test.bucket.562.aaa filename:test.csv size:1938","uuid":
"eaa34121-d5fd-43b4-a19c-ebc381db5c56","error":"","vmuptime":1540528993,
"newcontainer":1}
```

Next, verify that the CSV file has been created in your S3 bucket.

First try this using the AWS CLI. Try out the following commands. Adjust bucketnames as needed:

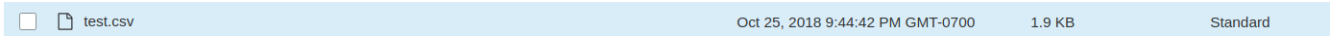
```
$ aws s3 ls test.bucket.562.aaa
2018-10-25 21:44:42       1938 test.csv

$ aws s3 ls s3://test.bucket.562.aaa
2018-10-25 21:44:42       1938 test.csv

$ aws s3 cp s3://test.bucket.562.aaa/test.csv .
download: s3://test.bucket.562.aaa/test.csv to ./test.csv

$ cat test.csv
38,869,146,8,578,793,8,713,581,259
49,994,324,882,412,287,402,428,401,922
971,584,184,972,717,611,14,660,978,867
...
```

Note that “s3://test.bucket.562.aaa/test.csv” is considered a URI or a Uniform Resource Identifier which is analogous (similar to) a URL.

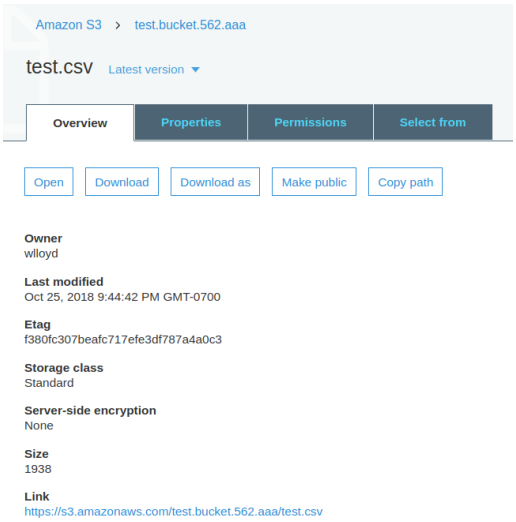
The “aws s3 ls” command doesn’t require “s3://”, while “aws s3 cp” does.

Next, using the S3 GUI, inspect your bucket and verify that the test.csv file exists.

Click your bucket name in the GUI.
Then click on the filename:



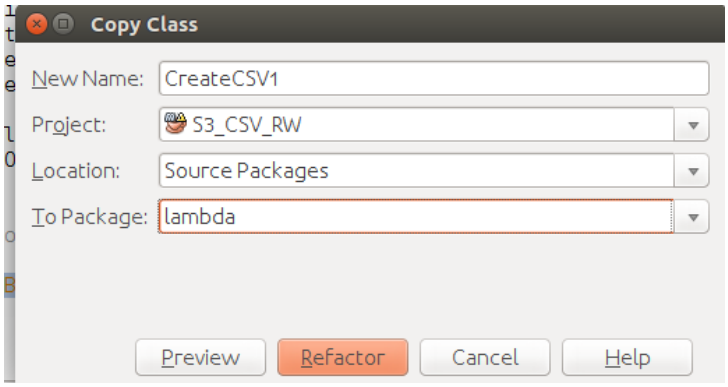
It is possible to download the file here, but without allowing public access to your bucket (not recommended) the **web link** at the bottom does not work:



5. Create ProcessCSV Lambda Function

Next, make a copy of the “CreateCSV” class called “ProcessCSV”.
If you’re using Netbeans, right click on “CreateCSV” and select “Refactor | Copy”.
Or alternatively press “ALT-C”.

A refactor popup appears:



Rename the class to “ProcessCSV”.

If not using Netbeans, copy “CreateCSV” and rename to create a new class called “ProcessCSV”. Adapt the Lambda function to read a CSV file called “filename” from the S3 called “bucketname”.

Adapt the example code provided from the URL and in the box below to read a file from S3 line-by-line. URL: https://blog.webnersolutions.com/use-aws-lambda-function

Here is the most relevant sample code for this activity:

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard().build();

//get object file using source bucket and srcKey name
S3Object s3Object = s3Client.getObject(new GetObjectRequest(srcBucket, srcKey));

//get content of the file
InputStream objectData = s3Object.getObjectContent();

//scanning data line by line
String textToUpload = "";
Scanner scanner = new Scanner(objectData);
while (scanner.hasNext()) {
      textToUpload += scanner.nextLine();
}
scanner.close();
```

The ProcessCSV Lambda function should add all numbers from the CSV file to calculate the average value and total value of all elements.

Read each line of the CSV file to parse each individual comma-separated value.
Add the total of all values using a Java “long” primitive variable:

```
long total;
```

Track the total number of elements processed.

At the end, use a Java “double” primitive, to calculate the average value for all elements in the entire CSV file:

```
double avg;
```

Add a logging statement to print the average value to the AWS Lambda log:

```
logger.log("ProcessCSV bucketname:" + bucketname + " filename:" + filename + "
avg-element:" + avg + “ total:” + total);
```

Finally, adjust the “value” property of the response object:

```
r.setValue("Bucket: " + bucketname + " filename:" + filename + " processed.“);
```

Note:
It is necessary to add permission for AWS Lambda functions to write out the CloudWatch logs.

If the Lambda function was created using the role **AWSLambdaBasicExecutionRole**, then the logging permissions are already available. If not, it *MAY* be necessary to add cloud watch log permissions to your role.

The role can be edited directly in the IAM GUI to add fine grained logging permissions as shown below:

```
{
"Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Or alternatively, you could add the **CloudWatchEventsFullAccess** Policy to your Role. Refer to how we previously added the **AmazonS3FullAccess** policy. Adding this policy adds more permissions than needed and is less secure.

Now, compile the project.
This will create a JAR file with both Lambda functions.
From this JAR file, a switchboard ***could be*** implemented by having the Handler method call different code based on the Request.java inputs within the same package.

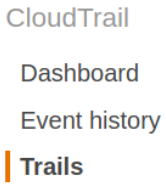
Deploy the new Lambda function using the jar file, and specify the new handler:

6. Automatically Trigger ProcessCSV when CreateCSV creates a file in S3

Next, we'll create a CloudWatch Event rule to fire the ProcessCSV Lambda function to run whenever a specific file is placed into the S3 bucket.

In the AWS Management Console, search for the "CloudTrail" service.
This is a Management & Governance Tool:

On the left-hand slide, select "Trails":



And then click the button:

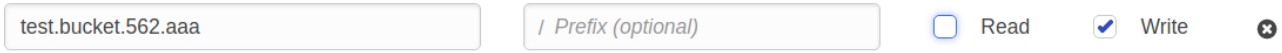
Create a trail as follows:

Create Trail
Trail name: **s3_1**
Apply trail to all regions: **no**

Management events
Read/Write events: **None** (*select the radio button*)

Data events
S3
Select:

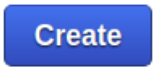
Then find your bucket name and configure Write events. Disable Read events:



Storage Location

** *Here, create a new independent bucket for logging* **
Create a new S3 bucket: **Yes** (*select radio button)*
S3 bucket: **tcss562.mylogs.aaa** (*give your bucket a name, where aaa are your initials)*

Then click the Create button:



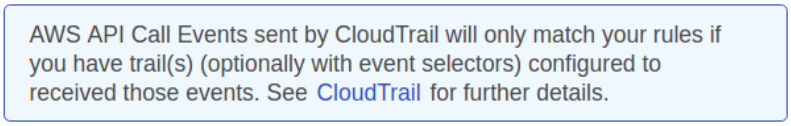
Next, navigate to CloudWatch, also a Management tool:

On the left-hand side, near Events, select “Rules”:
Click the “Create Rule” button:

Then configure a rule as follows:

(X) Event Pattern (*select this)*
Service Name: Simple Storage Service (S3)
Event Type: Object Level Operations

The message notifying about the requirement to configure CloudTrail should appear:



(X) Specific Operation (*select this)*
PutObject (*search to find “PutObject”)*
Specific bucket(s) by name:
test.bucket.562.aaa (*replace bucket name with your custom bucket name for Lambda)*
** *this is the Lambda bucket, not the logging bucket from immediately above* **

Next, on the right-hand side, click the “Add Target” button:

Select: Lambda Function

Function*: ProcessCSV
> Configure Input:
(X) Constant (JSON text) (*select this*)
Provide JSON:

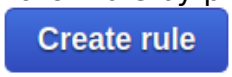
```
{"bucketname":"test.bucket.562.aaa","filename":"test.csv"}
```

Once everything is configured and ready, scroll down and in the lower right click:

For Step 2, provide a rule name:

Name*: processCSV_rule

Then complete creation of the CloudWatch rule by pressing:



*********** ***WARNING ABOUT CLOUDWATCH TRIGGERS*** **********

Previously a student created an invalid trigger where the Lambda that wrote a file to S3, was also the Lambda that was called by the trigger. This combination resulted in a circular Lambda call. Once invoked, the Lambda kept "putting an object" to S3 causing the call to be endless. This caused the Lambda to re-trigger and run again, and again, and again, and again, and was only discovered in the monthly bill after several weeks of run time. This resulted in a large number of Lambda calls, and charges that had to be reimbursed due to the bug. Be careful when defining triggers!

Now, run your callservice.sh script again to call createCSV.

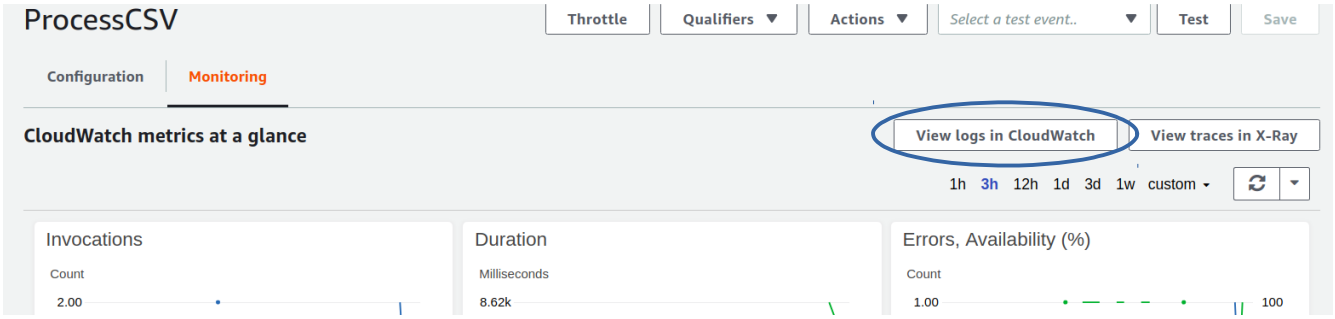
When createCSV finishes, the creation of a file in your S3 bucket automatically triggers the processCSV Lambda function to process the file !! The processCSV results will be written to the CloudWatch log.

7. Submitting the tutorial

To submit the tutorial, submit the results of the ***ProcessCSV log file,*** when ***calling* CreateCSV** to generate a 50x10 CSV file (500 total elements, 50 rows by 10 columns) with average elements selected randomly by the code provided above ranging from (1..1000). In the AWS Management Console, navigate to AWS Lambda.

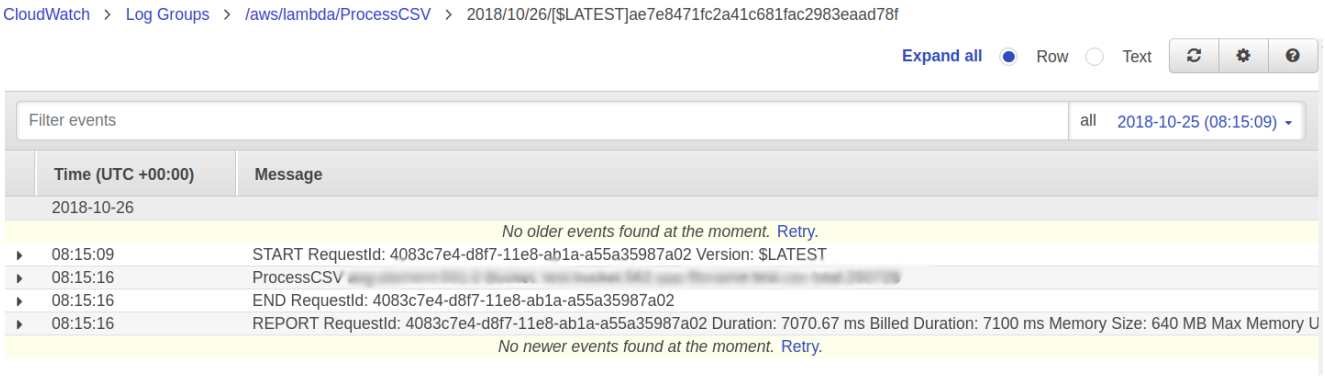
Go to your "ProcessCSV" function.

Click the "Monitoring" tab:



Then click the [View logs in CloudWatch] button.

Look through the log entries.
The top row should be your log entry where CreateCSV triggered the execution of ProcessCSV.
Click on this entry.
Now, CAPTURE THE SCREEN:



Using the CTRL-PrintScreen button to capture the entire screen,
or use CTRL-SHIFT-PrintScreen to draw a box around the relevant section of the screen to copy to the Clipboard an image of your log with the avg-element and total values for ProcessCSV.

In OpenOffice, Microsoft Word, or Google Docs, paste this image into a document.

Create a PDF file of the document, and submit this PDF file to Canvas.

Optional:

One shortcoming of the CloudWatch Event target here was that we used a hard coded value for the bucket name and the file name to pass to our ProcessCSV Lambda function. Ideally, we would like ProcessCSV to know the name of any new files added to S3 so our ProcessCSV will dynamically process the new file, and not a statically named one.

If interested in customizing your event handler, consult the instructor or research online to develop a solution for this problem.