TCSS 462/562: (Software Engineering for) Cloud Computing
Fall 2022

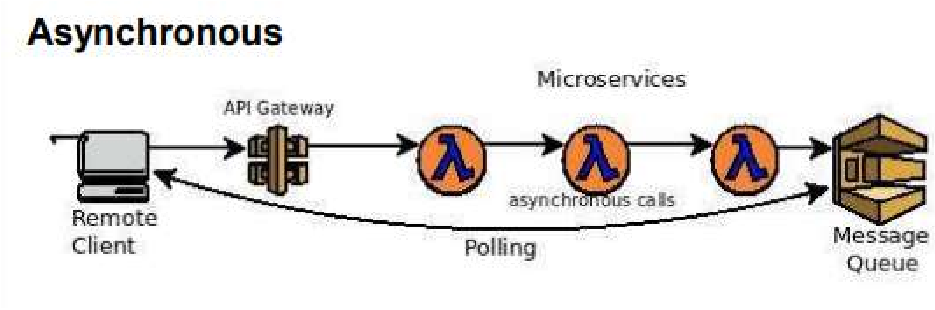
School of Engineering and Technology
University of Washington – Tacoma

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. For Tutorial 4, you may have chosen to implement the encode and decode functions using a single project.
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".
Also refer to Tutorial #4 regarding selection of Java version (8 or 11) for AWS Lambda.

2. CreateCSV Lambda Function

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

If using the Netbeans IDE, in the project, right click on the classname, and select "Refactor | Rename". Rename the "HelloPOJO" class to "CreateCSV".

IMPORTANT TIP USING NETBEANS: Java Import statements can be automatically added to your code by pressing CONTROL – SHIFT - i for imports… this saves from having to manually determine and add the import statements at the top of the class file.

If using the command line, rename the file and use a text editor to modify the class name inside CreateCSV.java to match "CreateCSV".

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

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

Start by adding the class variables:

```
    private String bucketname;
    private String filename;
    private int row;
    private int col;
```

Then, if using NetBeans, RIGHT-CLICK on the CreateCSV class name and select "Refactor", then select "Encapsulate Fields". Using the dialog box, select the 4 new variables. Netbeans will then automatically create getter and setter methods.

Or alternatively, as an example look at existing "getName" and "setName" methods that encapsulate "String name" in Request.java. Copy this pattern to create new getter and setter methods for the four new variables above. These fields will allow a user calling the service to specify required parameters to create a new CSV file. The file will be 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, load the Request class variables into local variables:

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

Next, generate a random matrix of values, storing each row in a separate String. Generate the CSV file text as follows adding this code to the handleRequest() method:

```
        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");
            }
```

You will need to add the Java import statements. In Netbeans, press Control-Shift-i. The next step is to write Java code to generate the S3 bucket file.

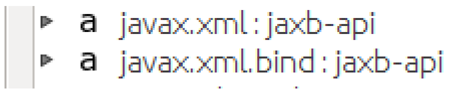
Working with the S3 Java API requires adding the Java support library.
In maven, the library can be added directly editing the pom.xml file.
The pom.xml file is under the java_template directory.

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 "**s3**" or "**aws-java-sdk-s3**".

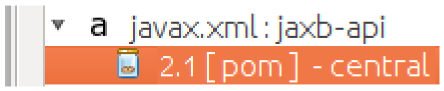
NOTE: When first using netbeans, the central repository index must be built. This could take several minutes and interfere with and/or impact the speed and/or correctness of the dependency searches.

Once found, select the version, such as ~ "1.11.907"…
This automatically includes all Java jar libraries required to work with Amazon S3.

Two more dependencies are required. Select the newest non-beta version. Beta versions have a "b" in the name. Search for "**jaxb-api**". Add "javax.xml: jaxb-api" and "javax.xml.bind".

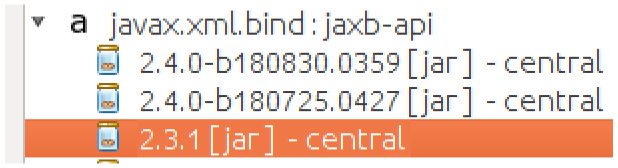


newest non-beta versions:



javax.xml : jaxb-api

2.1 [pom] - central



javax.xml.bind : jaxb-api

2.4.0-b180830.0359 [jar] - central

2.4.0-b180725.0427 [jar] - central

2.3.1 [jar] - central

Alternatively, if not using Netbeans, the dependencies 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-s3</artifactId>
            <version>1.11.907</version>
        </dependency>
        <dependency>
            <groupId>javax.xml</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.1</version>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.1</version>
        </dependency>
```

Note that the version of these dependencies are for Java 11. If deploying with Java 8 they may be somewhat different.

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. Add the following code to the handleRequest() method following the generation of the CSV text:

```
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 JSON response object returned by the CreateCSV service populates the "value" attribute. *(just like the Hello function !)* For CreateCSV set the value to be a string that describes the CSV file which is created. Modify the response.setValue() to:

```
response.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 the procedure to create an AWS Lambda function.

Be sure to specify the function handler:

```
lambda.CreateCSV::handleRequest
```

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.

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

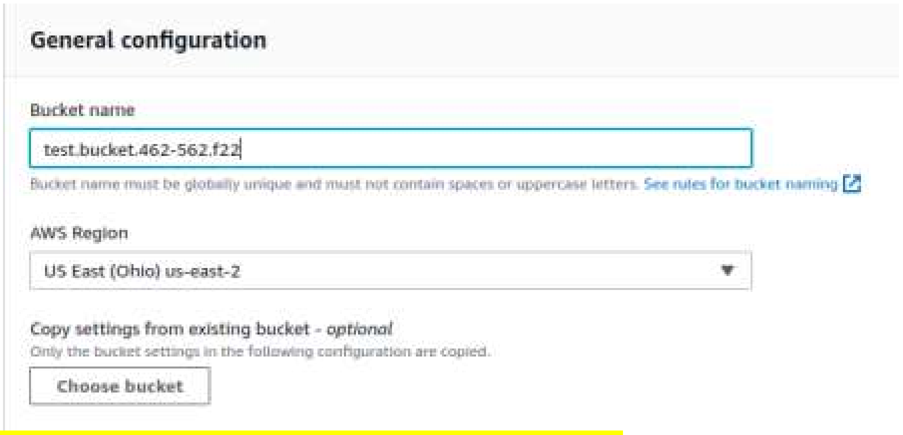
Begin by copying and adapting the callservice.sh script from tutorial 4. This is in SAAF/java_template/test/callservice.sh. Copy this script to the tutorial 5 project and redefine the input JSON object as follows:

```
json={"\"row\"":50,"\"col\"":10,"\"bucketname\"":\"test.bucket.562f21.aaa\"","\"fi
lename\"":\"test.csv\""}
```

Next, create an S3 bucket named "test.bucket.562f21.aaa" on the AWS management console replacing "aaa" with your initials.

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:


General configuration

Bucket name

test.bucket.462-562.f22

Bucket name must be globally unique and must not contain spaces or uppercase letters. See rules for bucket naming

AWS Region

US East (Ohio) us-east-2

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

S3 Bucket names must be globally unique within a naming partition. AWS currently has three partitions: aws (Standard Regions), aws-cn (China Regions), and aws-us-gov (AWS GovCloud (US)).
No other user anywhere can duplicate the same bucket name !!!

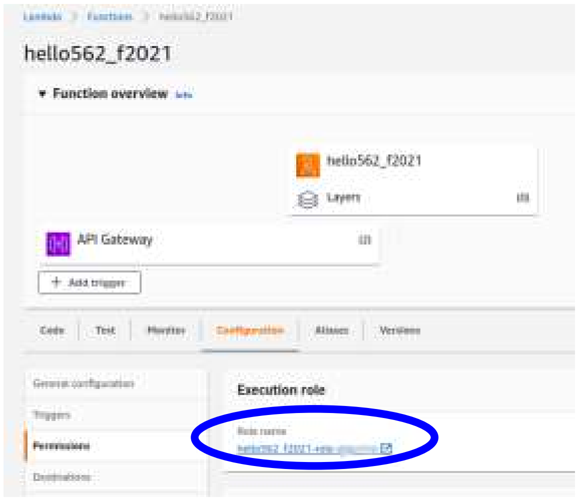
For the bucket name use: **test.bucket.462-562.f22.aaa**

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.

For remaining options accept the default values, and press the **Create bucket** button.

Next, it is necessary to configure permissions for your AWS Lambda functions to access your S3 bucket.
In the AWS Management Console, navigate to Lambda, and inspect your function’s execution role’s configured policies.

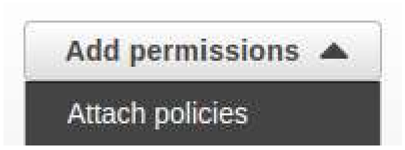
In Lambda, select the “Configuration” tab, followed by “Permissions” on the left.
Lambda provides a BLUE link to open your security role under Execution Role and Role Name:



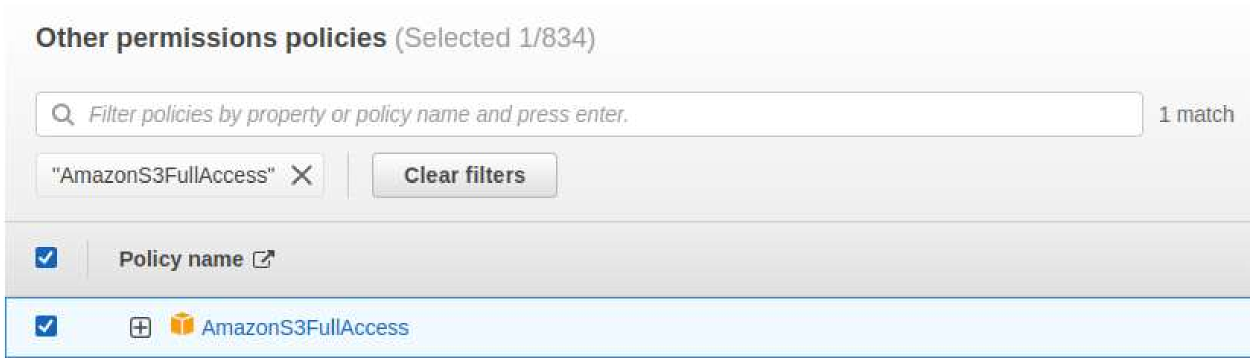
Note this is your dynamically generated security role name. Click on this blue link to navigate to your role under the IAM Management Console to edit the role to grant Lambda permission to access your S3 bucket.

Without the helpful link, this role can also be found by going to the upper right-hand corner, selecting your name, and in the drop-down choosing "My Security Credentials". Then on the left-hand side go to "Roles", and then search for the Lambda function's security role name.

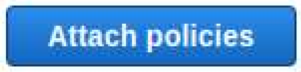
Once you're looking at the Role Summary, on the right, click the "**Add Permissions"** button and select "**Attach policies**":



Next, search for the name of the policy under **Other permissions policies:**
Type the name of the policy: "**AmazonS3FullAccess**" and press ENTER.
Next select the policy by clicking the checkbox on the LEFT:


Other permissions policies (Selected 1/834)
Filter policies by property or policy name and press enter.
1 match
"AmazonS3FullAccess"
Clear filters
Policy name
AmazonS3FullAccess

Then press the blue button on the right to attach the policy:



The policy should then be added to the Role.
This grants your Lambda function the permission to work with S3.
Every Lambda function that will work with S3 will need permission.
Fine grained security policies can be specified by creating an inline policy with the Role editor 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.inspectAllDeltas();** with another option. See tutorial #4 – page 21 for a list of inspect methods.

Now, test your CreateCSV Lambda function.
You will need to update the bucketname in callservice.sh to match your new bucket.

Optionally, 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.
Now try calling your function using the **`callservice.sh`** script:

```
$ ./callservice.sh
{"row":50,"col":10,"bucketname":"test.bucket.462-562.f22.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.462-562.f22.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:

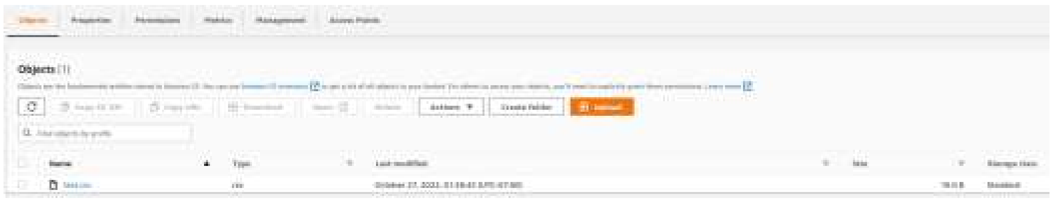
```
$ aws s3 ls test.bucket.462-562.f22.aaa
2018-10-25 21:44:42       1938 test.csv

$ aws s3 ls s3://test.bucket.462-562.f22.aaa
2018-10-25 21:44:42       1938 test.csv

$ aws s3 cp s3://test.bucket.462-562.f22.aaa/test.csv .
download: s3://test.bucket.562f21.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.462-562.f22.aaa/test.csv” is considered a URI or a Uniform Resource Identifier which is analogous (similar to) a URL.

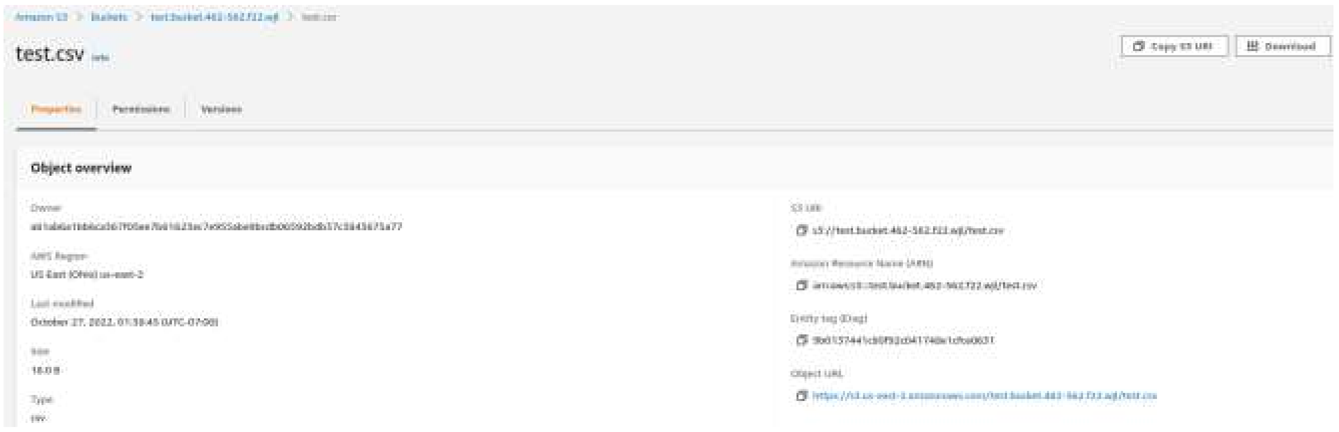
The “aws s3 ls” command doesn’t require “s3://”, while “aws s3 cp” does.

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 using the download button, but without allowing public access to your bucket (not recommended) the **Object URL web link** does not work.


Amazon S3 > Buckets > test.bucket.462-562.f22.wjl > test.csv
Copy S3 URI
Download
Properties
Permissions
Versions

test.csv

Object overview

Owner
[illegible]

AWS Region
US East (Ohio) us-east-2

Last modified
October 27, 2022, 01:38:45 (UTC-07:00)

Size
18.0 B

Type
csv

S3 URI
s3://test.bucket.462-562.f22.wjl/test.csv

Amazon Resource Name (ARN)
arn:aws:s3:::test.bucket.462-562.f22.wjl/test.csv

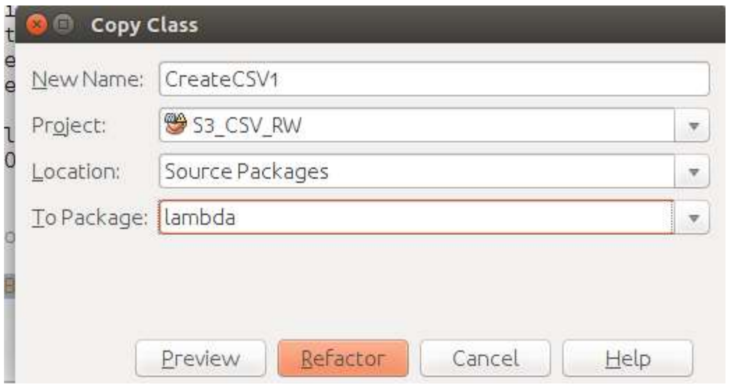
Entity tag (Etag)
[illegible]

Object URL
https://s3.us-east-2.amazonaws.com/test.bucket.462-562.f22.wjl/test.csv

5. Create ProcessCSV Lambda Function

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

A refactor popup appears:



New Name: CreateCSV1
Project: S3_CSV_RW
Location: Source Packages
To Package: lambda

Rename the class to “ProcessCSV”.

If not using Netbeans, copy the “CreateCSV” source file and create a new class called “ProcessCSV”. Adapt the Lambda function to read a CSV file called “filename” from the S3 bucket 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 text = "";
Scanner scanner = new Scanner(objectData);
while (scanner.hasNext()) {
      text += scanner.nextLine();
}
scanner.close();
```

For the s3Client.getObject() request, you'll need to provide the bucketname (srcBucket) and filename (srcKey) to the S3Client.getObject() API call. Properties from the Request object can be used to fetch the bucketname and filename. Define local variables for srcBucket and srcKey, and retrieve the values from the Request object.

### !!! ### !!! ### !!! ### !!! ### !!! ### !!! ###

IMPERATIVE WARNING

When copying CreateCSV to create the ProcessCSV Java class, it is IMPERATIVE that the PutObject() call to the Simple Storage Service (S3) be removed from ProcessCSV.

In the next step of the tutorial, an EventBridge rule is created to TRIGGER a call to ProcessCSV when a PutObject S3 Event occurs on the S3 Bucket.

****IF** the ProcessCSV Lambda function GENERATES S3 PutObject events, this will result in a CIRCULAR TRIGGER, and this will generate millions of AWS Lambda calls.**

This generates tremendous activity that exhausts cloud credits across the CloudTrail, CloudWatch, Lambda, and Simple Storage Service services.

****BE SURE TO DELETE** THE PutObject() CALL WHEN COPYING CREATE CSV TO MAKE THE NEW PROCESS CSV CLASS**

IMPERATIVE WARNING

### !!! ### !!! ### !!! ### !!! ### !!! ### !!! ###

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

Read each line of the CSV file and parse each individual comma-separated value. Use the code above to help start your solution.

Add the total of all values using a Java "long" primitive variable:

```
long total;
```

Track the total number of elements processed using another variable.

```
long elements;
```

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

```
double avg;
```

Calculate the average by dividing the total by the elements and storing in avg.

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

```
LambdaLogger logger = context.getLogger();
logger.log("ProcessCSV bucketname:" + bucketname + " filename:" + filename + "
avg-element:" + avg + " total:" + total);
```

Finally, adjust the "value" property of the response object for the ProcessCSV function:

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

Note:
AWS Lambda functions automatically receive permission to write out the CloudWatch logs.

This permission is included in the **AWSLambdaBasicExecutionRole** policy that is automatically created when you create the Lambda function.

Policies for the role assigned to your function can be viewed in the AWS Management Console from AWS Lambda by navigating to your function and then selecting the "Configuration" tab, followed by "Permissions" on the left. Lambda provides a BLUE link to open your security role under Execution Role and Role Name.

Click this link to inspect security policies assigned to your function.
Next EXPAND the "AWSLambdaBasicExecutionRole" to inspect the policy to be sure you have permission to write to the CloudWatch logs. Click on "CloudWatch Logs". Click on "{ } JSON".

A JSON description of the policy appears. An "**Edit policy**" button allows the JSON to be changed. You should have the Cloud Watch permissions as defined with the following JSON:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-2:465394327572:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                  "arn:aws:logs:us-east-2:465394327572:log-group:/aws/lambda/createCsv_562:*"
            ]
        }
    ]
}
```

Alternatively, if permissions are missing, you could attach 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 and is not recommended.

Now, compile the project.
This will create a JAR file with both Lambda functions.
Your Java JAR package contains two function handlers. This is the basis for implementing the switchboard pattern. A switchboard ***could be*** implemented by having an initial Handler process the JSON and redirect the function call inside the JAR based on the Request.java inputs.

CREATE A NEW Lambda function called "processCSV" using the jar file, and specify the new handler for ProcessCSV:

```
lambda.ProcessCSV::handleRequest
```

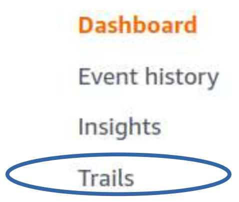
BE SURE to grant "processCSV" full S3 permissions as done for "createCSV".

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.
CloudTrail is considered a Management & Governance Tool:

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


Dashboard
Event history
Insights
Trails

And then click the button:

Create a trail as follows:

Trail name: **s3_1**

<u>Storage Location</u>

*** Create a new independent bucket for logging ***

Select "Create new S3 bucket" (*select radio button)*

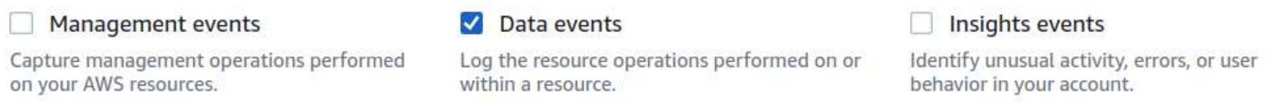
Trail log bucket and folder:
***Can just use** **the automatically generated unique name that is provided**, or provide your own name.*

Log file SSE-KMS encryption: *uncheck this – this should be disabled*

Then scroll down and click the **NEXT** button.

On the "**Choose log events**" screen, first uncheck **Management events**.

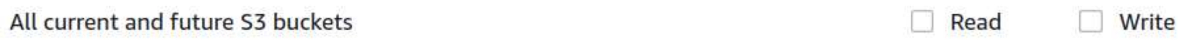
Then select "Data events".


☐ Management events
Capture management operations performed on your AWS resources.
✓ Data events
Log the resource operations performed on or within a resource.
☐ Insights events
Identify unusual activity, errors, or user behavior in your account.

In the Data events box, be sure the box says **Basic event selectors are enabled.**
If not click the button "**Switch to basic event selectors**".
Next set the "**Data event source"** to be "**S3**" .

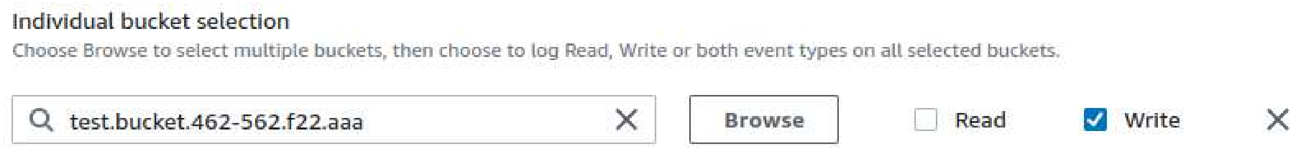
For the "Log selector template" select "Custom".

Uncheck: "All current and future S3 buckets" !


All current and future S3 buckets ☐ Read ☐ Write

You don't want to create an event trigger for ALL buckets !
Just the bucket we just created where CSV files are published.

Now fill in the "Individual bucket selection" to associate a CloudTrail with an S3 data event (e.g. *adding files to your bucket*).



Individual bucket selection

Choose Browse to select multiple buckets, then choose to log Read, Write or both event types on all selected buckets.

test.bucket.462-562.f22.aaa

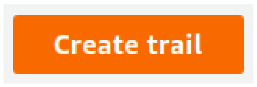
Read [✓ Write]

For the bucket name, **Browse** to find your bucket, or type in your bucket's unique name: test.bucket.462-562.f22.aaa

UNCHECK READ !!!
We only want to call a Lambda function when files are written to the bucket.

Then click the **NEXT** button.

Review the settings, and the create the trail:



Next, navigate to the "**CloudWatch**" service, also a Management tool:

On the left-hand side, near Events, select "Rules".

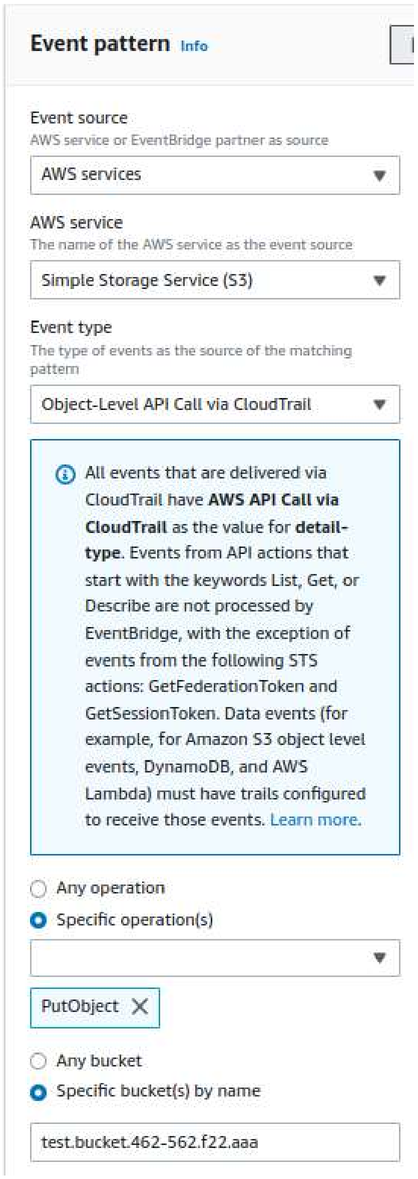
You should now be using the **Amazon EventBridge** GUI. A legacy CloudWatch Events GUI can also be used, but AWS is trying to retire the old GUI and approach.

Click the "Create Rule" button:

Then configure a rule as follows:

Name: Give your rule a name
Event bus: default
(X) Rule with an event pattern (*selected by default)*
Then select the **Next** button.

Scroll down and configure the **Event pattern**:


Event pattern Info

Event source
AWS service or EventBridge partner as source
AWS services

AWS service
The name of the AWS service as the event source
Simple Storage Service (S3)

Event type
The type of events as the source of the matching pattern
Object-Level API Call via CloudTrail

All events that are delivered via CloudTrail have **AWS API Call via CloudTrail** as the value for **detail-type**. Events from API actions that start with the keywords List, Get, or Describe are not processed by EventBridge, with the exception of events from the following STS actions: GetFederationToken and GetSessionToken. Data events (for example, for Amazon S3 object level events, DynamoDB, and AWS Lambda) must have trails configured to receive those events. Learn more.

Any operation [✓ Specific operation(s)]

PutObject

Any bucket [✓ Specific bucket(s) by name]

test.bucket.462-562.f22.aaa

For **Specific Operations** select **PutObject**.
The press the **Next** button.

For **Select target(s)** specific **AWS service** and select a target as **Lambda function**.
Provide the name of your Lambda function to call in response to the event.
Here is should be **ProcessCSV** (or the name you have used for the function).

Under **Additional settings**,
Under **Configure target input,** select **Constant (JSON text).**

In the **Specify the constant in JSON** provide JSON to pass to the function in the large text box:

Provide JSON:

```
{"bucketname":"test.bucket.462-562.f22.aaa","filename":"test.csv"}
```

Then select **Next.**
It is not necessary to add any tags, so select **Next** again.
Before creating the event rule, you'll have an opportunity to review the configuration.
If everything looks correct press "**Create rule**".

********************* 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 trigger to fire and call the Lambda function again endlessly. This 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.

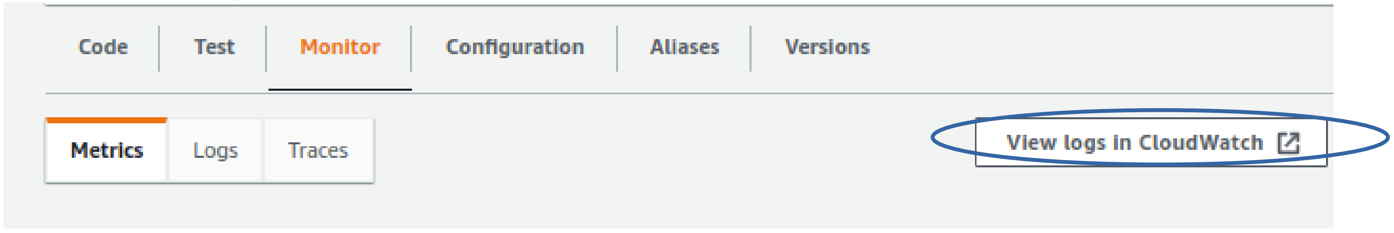
When createCSV finishes, the creation of a file in your S3 bucket should automatically trigger the processCSV Lambda function to process the file. The processCSV results will be written to the CloudWatch log.

7. Submitting the tutorial

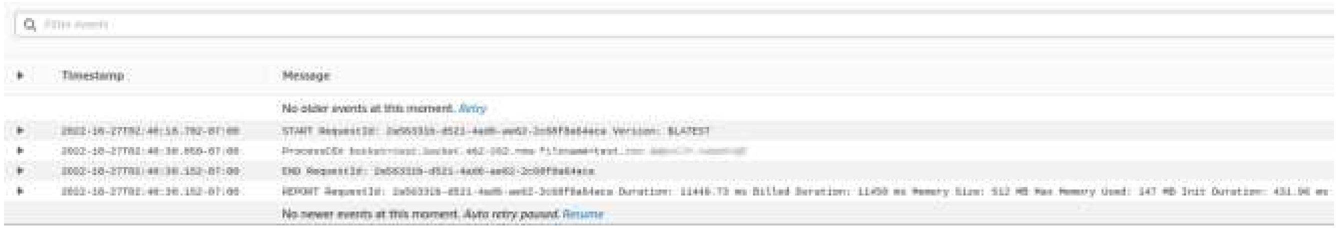
To submit the tutorial, submit the results of the ***ProcessCSV log file,*** when ***calling* CreateCSV** to generate a 50x10 CSV file (500 total elements, 50 rows by 10 columns). Each element should have a value selected randomly by the code provided above ranging from (1..1000). In the AWS Management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

Go to your "ProcessCSV" function.

Click the "**Monitor**" tab:
Select the "**View logs in CloudWatch**" button:



The log stream entry at the top of the list should contain the most recent output.
ProcessCSV should have been called when CreateCSV created the test.csv file in the bucket.
Click on the top log stream entry.
Now, CAPTURE THE SCREEN:



```
   Timestamp                        Message
                                    No older events at this moment. Retry
▸  2022-10-27T02:40:18.782-07:00    START RequestId: 2a563316-d521-4ad6-ae62-2c68f8a64eca Version: $LATEST
▸  2022-10-27T02:40:30.050-07:00    ProcessCSV bucket=[illegible] filename=[illegible]
▸  2022-10-27T02:40:30.152-07:00    END RequestId: 2a563316-d521-4ad6-ae62-2c68f8a64eca
▸  2022-10-27T02:40:30.152-07:00    REPORT RequestId: 2a563316-d521-4ad6-ae62-2c68f8a64eca Duration: 11446.73 ms Billed Duration: 11450 ms Memory Size: 512 MB Max Memory Used: 147 MB Init Duration: 431.96 ms
                                    No newer events at this moment. Auto retry paused. Resume
```

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 Tutorial Activity – Retrieve Bucket Name and Filename Dynamically

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.

When the CloudWatch rule invokes the Lambda function, it is possible to pass the event object instead of a hard coded object. The event object contains the name of the S3 filename and bucket that was created to fire the trigger.
First modify the CloudWatch Rule. In "CloudWatch", go to "Rules" on the left-hand side. Select your rule.
In the upper right-hand corner, select "Edit".

For the Target, under Additional settings, for "Configure target input" select "**Matched events"**.
Then press **Next**,
Skip configuring tags, and press **Next,**
Confirm the rule configuration, and press **Update rule**.

Selecting "Matched Events" will pass a JSON object describing the CloudWatch event that invoked your Lambda function to your Lambda Function.

Now, you'll need to modify your Java code to consume the bucket name and file name of the trigger.

Here is the structure of CloudWatch Event object sent to the triggered Lambda function. The bucket parameters are highlighted:

```
{
        "detail-type": "AWS API Call via CloudTrail",
        "resources": [],
        "id": "65a69778-bea5-6c65-cf52-c094e7a9ee34",
        "source": "aws.s3",
        "time": "2022-10-27T10:21:40Z",
        "detail": {
                "eventVersion": "1.08",
                "userIdentity": {
                        "type": "IAMUser",
                        "principalId": "AIIIIIIIIIIIIIIIIIIIIIIII",
                        "arn": "arn:aws:iam::123456789012:user/wlloyd",
                        "accountId": "123456789012",
                                "accessKeyId": "AKKKKKKKKKKKKKKKKKKKKKKKKK",
                                "userName": "wlloyd"
                        },
                        "eventTime": "2022-10-27T10:21:40Z",
                        "eventSource": "s3.amazonaws.com",
                        "eventName": "PutObject",
                        "awsRegion": "us-east-2",
                        "sourceIPAddress": "111.111.111.111",
                        "userAgent": "[aws-cli/1.19.53 Python/3.8.10 Linux/5.15.0-46-generic botocore/1.22.0]",
                        "requestParameters": {
                                "bucketName": "test.bucket.462-562.f22.aaa",
                                "Host": "s3.us-east-2.amazonaws.com",
                                "key": "test.csv"
                        },
                        "responseElements": null,
                        "additionalEventData": {
                                "SignatureVersion": "SigV4",
                                "CipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
                                "bytesTransferredIn": 21.0,
                                "SSEApplied": "Default_SSE_S3",
                                "AuthenticationMethod": "AuthHeader",
                                "x-amz-id-2":
"9Xnn1Cm35b+Wff8GyRCi37ZmZkM3dqePLF0EzLxx2Qqos4n2kxn1e9QVRMw2UFuDCpdbS5gHLfI=",
                                "bytesTransferredOut": 0.0
                        },
                        "requestID": "HXZX1QDVH51RNBEH",
                        "eventID": "73f7c855-36c3-4e1e-acb7-920e281e3835",
                        "readOnly": false,
                        "resources": [{
                                "type": "AWS::S3::Object",
                                "ARN": "arn:aws:s3:::test.bucket.462-562.f22.aaa/test.csv"
                        }, {
                                "accountId": "123456789012",
                                "type": "AWS::S3::Bucket",
                                "ARN": "arn:aws:s3:::test.bucket.462-562.f22.aaa"
                        }],
                        "eventType": "AwsApiCall",
                        "managementEvent": false,
                        "recipientAccountId": "123456789012",
                        "eventCategory": "Data",
                        "tlsDetails": {
                                "tlsVersion": "TLSv1.2",
                                "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
                                "clientProvidedHostHeader": "s3.us-east-2.amazonaws.com"
                        }
                },
                "region": "us-east-2",
                "version": "0",
                "account": "123456789012"
}
```

The properties of interest are:

detail.requestParameters.bucketName
detail.requestParameters.key

The following source code demonstrates acquiring the bucketName and filename and printing them to the log file. This code can be modified to suit your purposes:

```
public HashMap<String, Object> handleRequest(HashMap<String, Object> request, Context context) {

//Collect initial data.
Inspector inspector = new Inspector();
inspector.inspectAll();

//****************START FUNCTION IMPLEMENTATION*************************

Gson gson = new GsonBuilder().setPrettyPrinting().create();
LambdaLogger logger = context.getLogger();
logger.log ("bucketName=" + ((HashMap)((HashMap)request.get("detail")).get("requestParameters")).get("bucketName"));
logger.log ("filename=" + ((HashMap)((HashMap)request.get("detail")).get("requestParameters")).get("key"));
```

The event is provided as a nested HashMap object in Java. Each layer of JSON is a HashMap. Accessing the bucket parameters involves using the **detail** HashMap to access the **requestParameters** HashMap to read the **bucketName** and **key** fields.

Customize the sample code as needed to create a customized event handler that does not use a hard coded bucketname and filename.