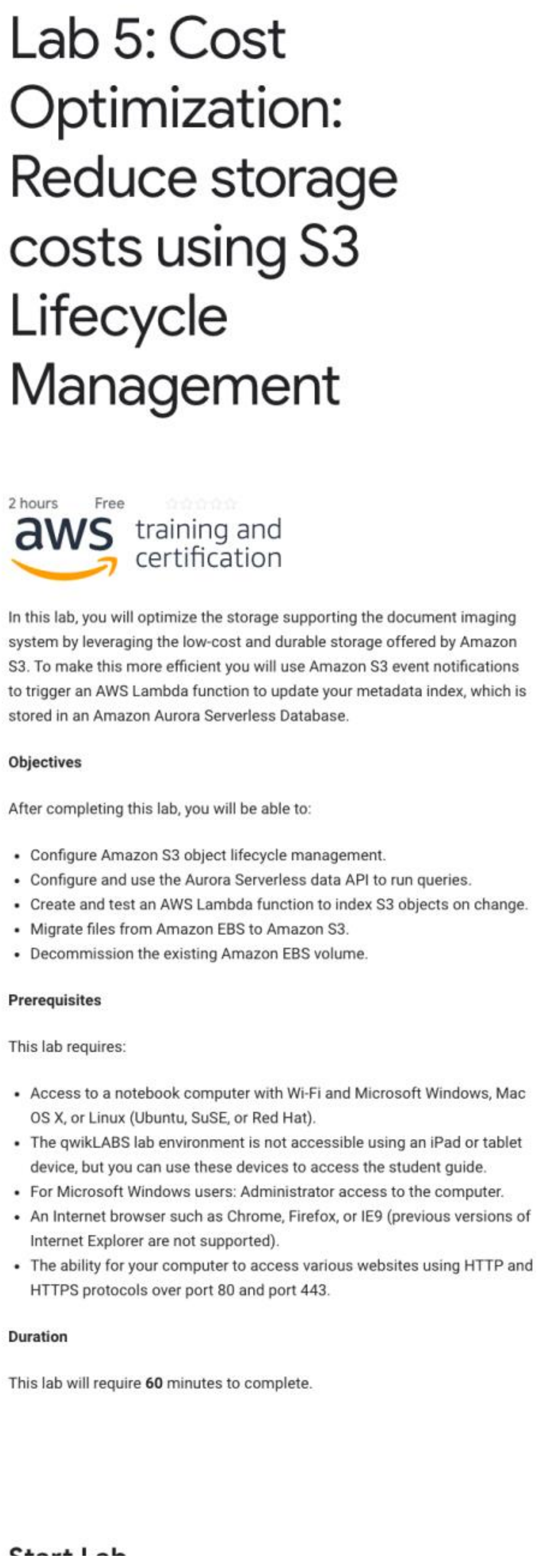
Lab 5: Reduce storage costs using S3 Lifecycle Management v1.0.5

Sunday, August 9, 2020 7:26 PM

Lab 5:

- Task 1 appears to be written for the old EC2 console. The EBS information in step 5 is now located under the storage tab after selecting the instance.
- Step 48, GUI Change, “Choose or create an execution role” is now “Change default execution role”



Lab 5: Cost Optimization: Reduce storage costs using S3 Lifecycle Management

2 hours Free

aws training and certification

In this lab, you will optimize the storage supporting the document imaging system by leveraging the low-cost and durable storage offered by Amazon S3. To make this more efficient you will use Amazon S3 event notifications to trigger an AWS Lambda function to update your metadata index, which is stored in an Amazon Aurora Serverless Database.

Objectives

After completing this lab, you will be able to:

- Configure Amazon S3 object lifecycle management.
- Configure and use the Aurora Serverless data API to run queries.
- Create and test an AWS Lambda function to index S3 objects on change.
- Migrate files from Amazon EBS to Amazon S3.
- Decommission the existing Amazon EBS volume.

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi and Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat).
- The qwikLABS lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- For Microsoft Windows users: Administrator access to the computer.
- An Internet browser such as Chrome, Firefox, or IE9 (previous versions of Internet Explorer are not supported).
- The ability for your computer to access various websites using HTTP and HTTPS protocols over port 80 and port 443.

Duration

This lab will require **60** minutes to complete.

Start Lab

Start Lab

1. At the top of your screen, launch your lab by choosing Start Lab

   This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

   ❶ If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. Open your lab by choosing Open Console

   This opens an AWS Management Console sign-in page.

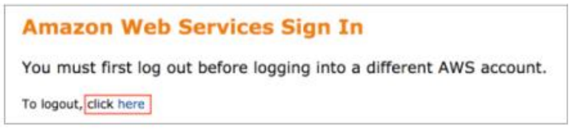
3. On the sign-in page, configure:

   - **IAM user name:** `awsstudent`
   - **Password:** Paste the value of **Password** from the left side of the lab page
   - Choose Sign In

   ⚠ **Do not change the Region unless instructed.**

Common Login Errors

Error: You must first log out



Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, click here

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose Open Console again

Introduction

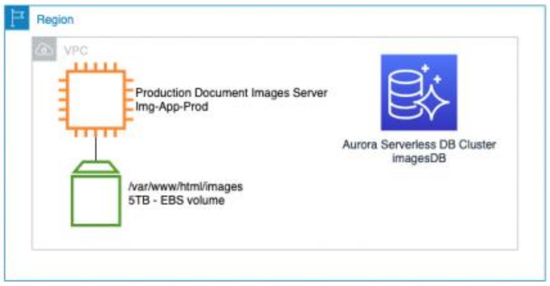
In evaluating your document imaging system, you noticed the Amazon Elastic Block Store (EBS) volumes provisioned are quite large and not fully utilized. The secondary volume attached to your EC2 instance, is exclusively storing the document images. If you can find a more appropriate location to store and access the image, you can decommissioned it for immediate cost savings.

Additionally, in order to migrate quickly to AWS, the application team is storing all images on the one EBS volume. The solution, no longer takes into consideration that the frequent access of recent document. While older images, rarely are accessed, but must be maintained to remain compliant with records retention requirements.

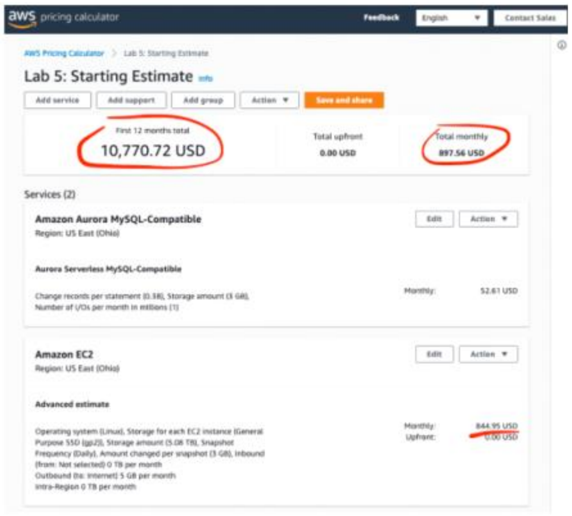
Finally, to meet your disaster recovery requirements, you should protect images from a total Availability Zone (AZ) failure. Currently, you are creating and storing a large number of EBS volume snapshots to achieve this. This adds additional cost, complexity, and makes it difficult to meet your Recovery Point and Time Objectives (RPO/RTO).

At the start of this lab, your environment consists of the following:

- An Amazon EC2 instance with your document imaging application
- An Amazon Aurora Serverless Cluster that contains your document index



Below is a screenshot of the AWS pricing calculator showing the cost of the environment at the start of the lab. The prices are as of 5/15/2020. You can select the link below to view an estimate online with current prices.



Lab 5: Starting Estimate

First 12 months total	Total upfront	Total monthly
10,770.72 USD	0.00 USD	897.56 USD

Services (2)

Amazon Aurora MySQL-Compatible
Region: US East (Ohio)

Aurora Serverless MySQL-Compatible

Change records per statement (0.38), Storage amount (3 GB), Number of I/Os per month in millions (1)

Monthly: 52.61 USD

Amazon EC2
Region: US East (Ohio)

Advanced estimate

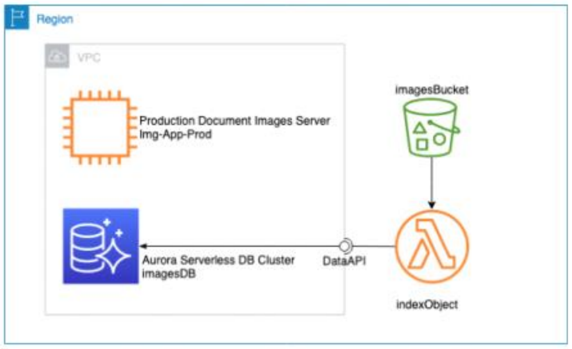
Operating system (Linux), Storage for each EC2 instance (General Purpose SSD (gp2)), Storage amount (5.08 TB), Snapshot Frequency (Daily), Amount changed per snapshot (3 GB), Inbound (from: Not selected) 0 TB per month
Outbound (to: Internet) 5 GB per month
Intra-Region 0 TB per month

Monthly: 844.95 USD
Upfront: 0.00 USD

https://calculator.aws/#/estimate?id=c88bafb2a4ed53a367b66ef1a28a4831c3a1248a

In this lab, you will optimize the storage supporting the document imaging system by leveraging the low-cost and durable storage offered by Amazon S3.

- You will configure an Amazon S3 Lifecycle Policy on an S3 bucket that handles the object storage class for your images, along with expiration settings that match your company's document retention policy.
- A Lambda function will be created to update the index in your existing database with the new location of the image files that are being added to S3. You will use an S3 event trigger to invoke your Lambda function so that your index always stays up to date. Based on the image's creation date in the database, the scripts will upload the files to the appropriate Amazon S3 storage class.
- To migrate the existing image files, you will use AWS Systems Manager Session Manager to access the instance and copy the files to S3 using a script provided for you.


Region
VPC
Production Document Images Server
Img-App-Prod
imagesBucket
Aurora Serverless DB Cluster
imagesDB
DataAPI
indexObject

The following has been pre-provisioned for you.

- An Amazon S3 Bucket for your images
- All of the scripts and Amazon
  Identity and Access Management (IAM) roles that you need to complete the lab
- A script that will automate migration of lab images from Elastic Block Store volumes to S3 buckets

Task 1: Explore the application environment

⚠You will be opening up multiple tabs and AWS consoles in this lab. For each task, compare the region you are in with the **region** value located in the left navigation pane of these instructions.

Before you can migrate your application's storage, you first need to understand the architecture.

This application consists of one application server that allows your company to upload documents and view them on the web. You can browse to the public DNS address in your web browser to view random PDF files. These files are stored locally on the server, on a 5TB EBS volume.

Task 1.1 Explore the application server

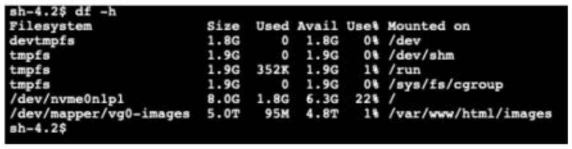
4. On the navigation bar, choose **Services ⌄**, and choose **EC2**.
5. In the left navigation pane, under **INSTANCES**, choose **Instances**.

   Scroll down in the details section until you find *Block devices*. You will notice the second Block device listed as */dev/sdf*. You can press */dev/sdf* and then select the EBS ID to review that volume if you would like to, but come back to the Instances screen when you are done.

   Next, You will review at the volume from the operating system and explore the folder structure.
6. Select the row of the **Img-App-Prod** instance.
7. Choose **Connect**
8. For **Connection method**, choose **Session Manager**.
9. Choose **Connect**

   The **df** command is used to display statistics and usage information about all mounted filesystems.
10. Enter the following command to view the file system structure and press ENTER.

```
df -h
```



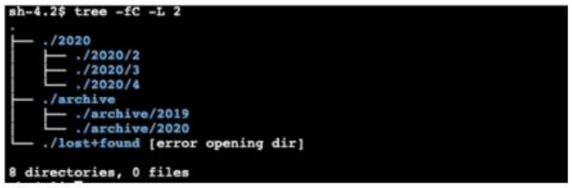
```
sh-4.2$ df -h
Filesystem                Size  Used Avail Use% Mounted on
devtmpfs                  1.8G     0  1.8G   0% /dev
tmpfs                     1.9G     0  1.9G   0% /dev/shm
tmpfs                     1.9G  352K  1.9G   1% /run
tmpfs                     1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/nvme0n1p1            8.0G  1.8G  6.3G  22% /
/dev/mapper/vg0-images    5.0T   95M  4.8T   1% /var/www/html/images
sh-4.2$
```

The images volume is the one on the list Mounted on ***/var/www/html/images***.

Inspect how the images are stored on the volume with the **tree** command, a linux utility that lists the contents of directories in a tree-like format.

11. Enter the following commands and press ENTER.

```
cd /var/www/html/images
tree -fC -L 2
```



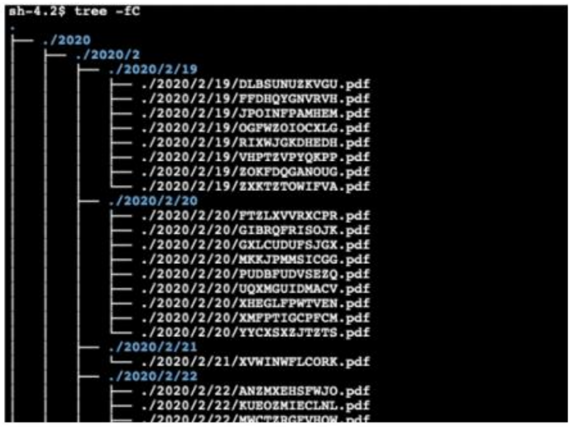
```
sh-4.2$ tree -fC -L 2
.
├── ./2020
│   ├── ./2020/2
│   ├── ./2020/3
│   └── ./2020/4
├── ./archive
│   ├── ./archive/2019
│   └── ./archive/2020
└── ./lost+found [error opening dir]

8 directories, 0 files
```

The imaging system was designed to allow the user to store images in multiple locations referred to as "bins". When this system was on-premises, they stored current files on fast disks and archived older files on less expensive storage. When the system was migrated to AWS, all bins were moved to the same EBS volume to save time. the directory structure shows this.

- The root of the images folder is the default bin for files less than 90 days old.
- The archive directory is the archive bin. Any image that is older than 90 days is moved here.
- Both bins store images the same way. There is a directory for the year, month, and day the image was created.

12. To clear the bash shell and display the entire directory, enter the following commands and press ENTER.

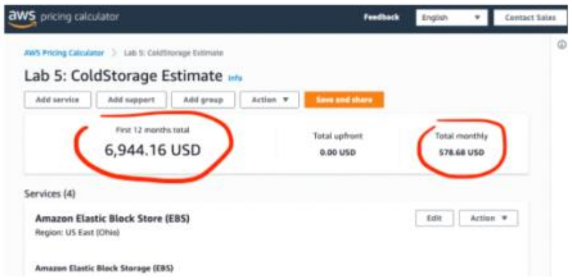
```
clear
tree -fC
```



```
sh-4.2$ tree -fC
.
├── ./2020
│   ├── ./2020/2
│   │   ├── ./2020/2/19
│   │   │   ├── ./2020/2/19/DLBSUNUZKVGU.pdf
│   │   │   ├── ./2020/2/19/FFDHQYGNVRVH.pdf
│   │   │   ├── ./2020/2/19/JPOINFPAMHEM.pdf
│   │   │   ├── ./2020/2/19/OGFWZOIOCXLG.pdf
│   │   │   ├── ./2020/2/19/RIXWJGKDHEDH.pdf
│   │   │   ├── ./2020/2/19/VHPTZVPYQKPP.pdf
│   │   │   ├── ./2020/2/19/ZOKFDQGANOUG.pdf
│   │   │   └── ./2020/2/19/ZXKTZTOWIFVA.pdf
│   │   ├── ./2020/2/20
│   │   │   ├── ./2020/2/20/FTZLXVVRXCPR.pdf
│   │   │   ├── ./2020/2/20/GIBRQFRISOJK.pdf
│   │   │   ├── ./2020/2/20/GXLCUDUFSJGX.pdf
│   │   │   ├── ./2020/2/20/MKKJPMMSICGG.pdf
│   │   │   ├── ./2020/2/20/PUDBFUDVSEZQ.pdf
│   │   │   ├── ./2020/2/20/UQXMGUIDMACV.pdf
│   │   │   ├── ./2020/2/20/XHEGLFPWTVEN.pdf
│   │   │   ├── ./2020/2/20/XMFPTIGCPFCM.pdf
│   │   │   └── ./2020/2/20/YYCXSXZJTZTS.pdf
│   │   ├── ./2020/2/21
│   │   │   └── ./2020/2/21/XVWINWFLCORK.pdf
│   │   ├── ./2020/2/22
│   │   │   ├── ./2020/2/22/ANZMXEHSFWJO.pdf
│   │   │   ├── ./2020/2/22/KUEOZMIECLNL.pdf
│   │   │   ├── ./2020/2/22/MWCTZRGFVHOW.pdf
```

You will return to the bash shell later. For now you can close out of this browser tab. It will time out before you return.

One thing you *could* do to reduce cost after migrating this system is utilize the bins as intended with different EBS volumes. Instead of using one large General Purpose SSD (gp2) volume, you could create a much smaller gp2 volume to hold recent images and create a Cold HDD (sc1) volume to hold older images in the archive bin. As you can notice in the following image, this configuration would result in significant savings and is the correct choice if your application must use block storage:



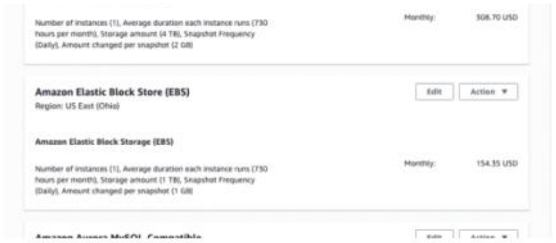
Lab 5: ColdStorage Estimate

First 12 months total 6,944.16 USD
Total upfront 0.00 USD
Total monthly 578.68 USD

Services (4)

Amazon Elastic Block Store (EBS)
Region: US East (Ohio)

Amazon Elastic Block Storage (EBS)


Number of instances (1), Average duration each instance runs (730 hours per month), Storage amount (4 TB), Snapshot Frequency (Daily), Amount changed per snapshot (2 GB) Monthly: 308.70 USD

Amazon Elastic Block Store (EBS)
Region: US East (Ohio)

Amazon Elastic Block Storage (EBS)

Number of instances (1), Average duration each instance runs (730 hours per month), Storage amount (1 TB), Snapshot Frequency (Daily), Amount changed per snapshot (1 GB) Monthly: 154.35 USD

Amazon Aurora MySQL-Compatible

https://calculator.aws/#/estimate?id=e40e0c6b80da649902adf2f1cd740d528d1bc86e

However, your document imaging application doesn't require block storage. It can use object storage instead, so you will migrate images from EBS volumes to an S3 bucket later in this lab.

Task 1.2: Explore the Database

Now that you know how the PDF images are stored on the file system, explore the database structure. When the team moved this application to AWS, they chose to use the Amazon Relational Database Service (RDS) to offload much of the underlying database server management to AWS. The on-premises database was MySQL compatible and had an unpredictable usage pattern. Amazon Aurora Serverless was selected because of its simple, scalable, highly available, and cost-effective design.

Throughout this lab, you will access the database using the query editor for Aurora Serverless. With this built-in query editor, you can run SQL queries in the RDS console. Any valid SQL statement can run on the Aurora Serverless DB cluster, including data manipulation and data definition statements.

The query editor requires an Aurora Serverless DB cluster with the Data API enabled. Let is enable the Data API for your cluster.

13. On the navigation bar, choose **Services ⌄**, and choose **RDS**.

14. In the left navigation pane, choose **Databases**.

15. Select the row of your database.

16. Choose **Modify**

17. Locate the **Connectivity** section and Check ☑ **Data API**.

18. Choose **Continue**

19. For **Scheduling of modifications** , choose **Apply Immediately**.

20. Choose **Modify cluster**

Now that the Data API is enabled, you can connect to your database to explore the structure. You will be using the Query Editor throughout this lab. To make navigation easier, you will open the Query Editor in a new browser tab.

21. In the left navigation pane, open the context (right-click) menu for **Query Editor** and choose to open the link in a new tab.

22. Enter the following

- For **Database instance or cluster**, choose your database cluster.
- For **Database username**, choose **Connect with a Secrets Manager ARN**.
- For **Secrets manager ARN** enter the ***secretsManagerArn*** value located in the left navigation pane of these instructions.
- For **Enter the name of the database or schema**, enter `imagesDB`
- Choose **Connect to database**

23. To clear the Editor window, choose **Clear**

24. Paste the following code in the Editor window.

```
SHOW TABLES;
DESCRIBE bin;
SELECT * FROM bin;
DESCRIBE imageIndex;
SELECT * FROM imageIndex ORDER BY RAND() LIMIT 5;
```

SHOW TABLES; DESCRIBE bin; SELECT * FROM bin; DESCRIBE imageIndex; SELECT * FROM imageIndex ORDER BY RAND() LIMIT 5;

25. Choose **Run**

The Query Editor lets you run multiple SQL commands at once, as you just did above. In the Output panel, it has opened a Result set tab for each query.

26. Choose the **Result set 1 (2)** tab.

The **SHOW TABLES;** query returns the two tables that are in your imagesDB.

Table	Description
bin	Definition of the storage locations
imageIndex	A record with information for each image in the system

27. Choose the **Result set 2 (2)** tab.

The **DESCRIBE bin;** query shows the structure of the bin table.

Field	Description
id	The id of the bin that is referenced in the imageIndex table
path	String that contains the path to the bin

28. Choose the **Result set 3 (2)** tab.

These results show the two current bins that are configured. Notice that the path starts with *http://*; this is the fully qualified domain name and path that is used to access the images from the web application.

29. Choose the **Result set 4 (4)** tab.

The **DESCRIBE imageIndex;** query shows the structure of the imageIndex table.

Field	Description
imageID	The file name, this must be unique
path	The file path relative to the location within the bin
entryDate	The date that the image was created
bin	The id of the bin that the image is stored in

30. Choose the **Result set 5 (5)** tab.

The final query shows 5 sample records from the imageIndex table.

You will use the Query Editor throughout the lab. Leave this tab open and return to the "RDS - AWS Console" browser tab to continue.

Task 2: Configure your S3 bucket for the new image repository

If your application requires block storage, you can save money by using disks that have the appropriate performance characteristics. However, your document imaging system does not require block storage, so you can use object storage to save even more.

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.

Images stored in your imaging application are frequently accessed for the first 60-90 days. After that, images are accessed much less frequently, but they still need to be immediately accessible. Your document retention policy states that you cannot keep images for more than 3 years.

If you do not know your data access patterns, you could choose to store them using the S3 Intelligent-Tiering storage class. However, in this case since you do know your data access patterns, there is a more cost-efficient option. To manage objects so they are stored efficiently throughout their lifecycle, configure an Amazon S3 Lifecycle. An S3 Lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects.

Task 2.1: Create an S3 Lifecycle rule for your S3 Bucket

In the "RDS - AWS Console" browser tab, not the one where your query editor is open.

31. On the navigation bar, choose **Services ⌄**, and choose **S3**.
32. Select the bucket name that matches the ***bucketName*** value located in the left navigation pane of these instructions.
33. Choose the **Management** tab.
34. Choose **Create lifecycle rule**
35. Configure the following settings:
    - For **Enter a rule name:**, enter `imageLifecycle`

    This bucket will only contain your images, nothing else. Everything in the bucket should be subject to this policy.

    - For **Choose a rule scope**, choose **This rule applies to all objects in the bucket**.
    - Check ☑ **I acknowledge that this rule will apply to all objects in the bucket**

    The **Transitions** screen is where you will configure a policy to move your images to a lower cost storage tier after 90 days, when the images are not accessed as often. You will use **S3 Standard-IA** for this. S3 Standard-IA was designed for older data that is accessed infrequently but still requires millisecond access.

36. In **Lifecycle rule actions** section,
    - Check ☑ **Transition current versions of objects between storage classes**.
    - Check ☑ **Expire current versions of objects**.
37. For **Transition current versions of objects between storage classes**
    - For **Storage class transitions**, enter `Standard-IA`
    - For **Days after creation**, enter `90`

    An Amazon S3 Lifecycle configuration can also delete expired objects on your behalf. The **Expirations** screen is where you will define the rule that removes objects from S3 after 3 years.

38. In **Expire current versions of objects** section, configure the following -
    - For **Number of days after object creation**, enter `1095`

- For **Number of days after object creation**, enter `1095`.

39. Choose **Create Rule**

Task 2.2: Create the new bin in your imaging application's database

Now that your S3 bucket has been configured to store and enforce the lifecycle rules, you can create your new bin in the database. Go to the "RDS - AWS Console" browser tab that is running the Query Editor. (if you need to connect again, please check Task 1.2 above.)

In the "RDS - AWS Console" browser tab

40. Choose **Clear**

REPLACE INTO bin (id, path) VALUES (3, 'https://[bucket-name].s3.amazonaws.com'); SELECT * FROM bin;

41. Paste the following code into the Editor window.

```
REPLACE INTO bin (id, path) VALUES (3, 'https://[bucket-
name].s3.amazonaws.com');
SELECT * FROM bin;
```

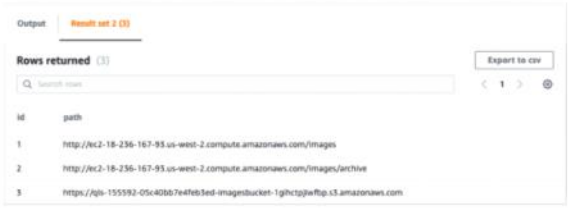
42. Replace the ***[bucket-name]*** placeholder with the ***bucketName*** value located in the left navigation pane of these instructions.

Example:

`REPLACE INTO bin (id, path) VALUES (3, 'https://qls-155592-05c40bb7e4feb3ed-imagesbucket-1gihctpjiwfbp.s3.amazonaws.com');`

43. Choose **Run**

44. Choose the **Result set 2 (3)** tab and confirm that you notice your new bin in results. Your results should look similar to the ones below.



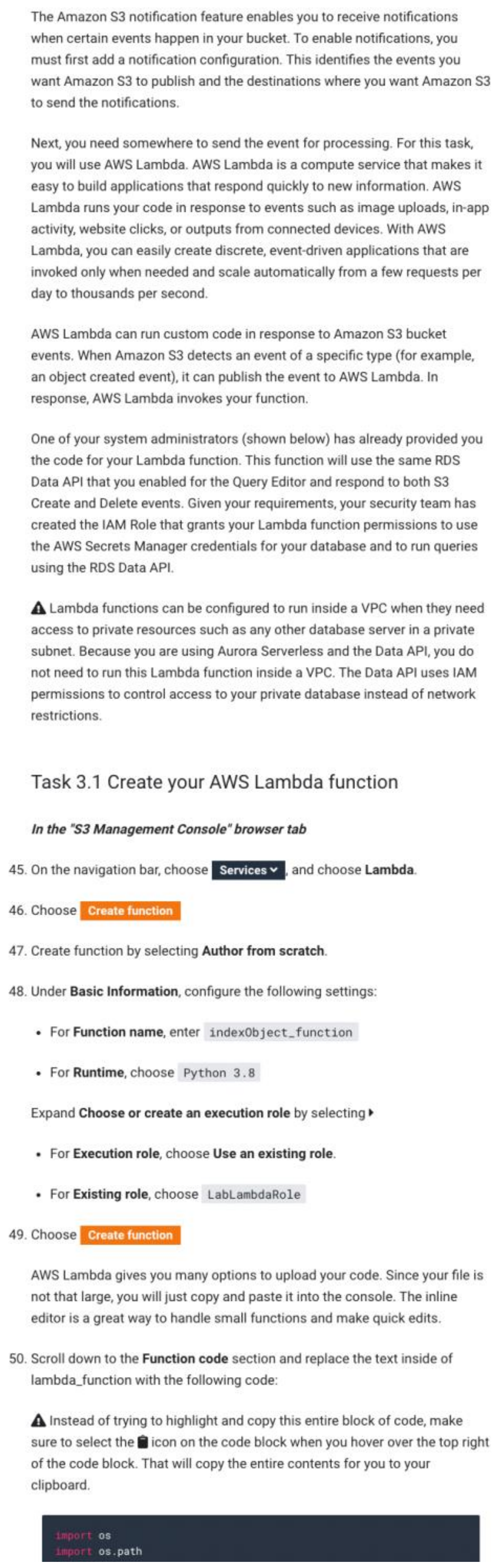
id	path
1	http://ec2-18-236-167-93.us-west-2.compute.amazonaws.com/images
2	http://ec2-18-236-167-93.us-west-2.compute.amazonaws.com/images/archive
3	https://qls-155592-05c40bb7e4feb3ed-imagesbucket-1gihctpjiwfbp.s3.amazonaws.com

⚠ Since, you are using a **REPLACE** instead of an **INSERT** query above, it would not cause errors if you run the command more than once. If your URL does not look correct, or your ran the command before replacing the placeholder, just fix the URL and run the command again to replace the incorrect bin.

In this task, you created an Amazon S3 Lifecycle rule to move your objects to the correct storage tier based on their expected access patterns. This rule also expires your objects to align with your company's document retention policy. In the second part of this task, you added this new S3 location as a potential place to store images in your document imaging application.

Task 3: Create an AWS Lambda function to index your Amazon S3 objects as they were uploaded

Now that your new image bin created is ready to accept images, you need to think about how to update your database when document images are added and removed. Migration scripts could update the location when they move images from EBS to S3, but that is a one time event. What would happen when your images update or expire? You will need a way for updates in S3 to reflect your database after migration is complete as well. This can be accomplished using Amazon S3 notifications.



The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. To enable notifications, you must first add a notification configuration. This identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications.

Next, you need somewhere to send the event for processing. For this task, you will use AWS Lambda. AWS Lambda is a compute service that makes it easy to build applications that respond quickly to new information. AWS Lambda runs your code in response to events such as image uploads, in-app activity, website clicks, or outputs from connected devices. With AWS Lambda, you can easily create discrete, event-driven applications that are invoked only when needed and scale automatically from a few requests per day to thousands per second.

AWS Lambda can run custom code in response to Amazon S3 bucket events. When Amazon S3 detects an event of a specific type (for example, an object created event), it can publish the event to AWS Lambda. In response, AWS Lambda invokes your function.

One of your system administrators (shown below) has already provided you the code for your Lambda function. This function will use the same RDS Data API that you enabled for the Query Editor and respond to both S3 Create and Delete events. Given your requirements, your security team has created the IAM Role that grants your Lambda function permissions to use the AWS Secrets Manager credentials for your database and to run queries using the RDS Data API.

⚠ Lambda functions can be configured to run inside a VPC when they need access to private resources such as any other database server in a private subnet. Because you are using Aurora Serverless and the Data API, you do not need to run this Lambda function inside a VPC. The Data API uses IAM permissions to control access to your private database instead of network restrictions.

Task 3.1 Create your AWS Lambda function

In the "S3 Management Console" browser tab

45. On the navigation bar, choose **Services ⌄**, and choose **Lambda**.

46. Choose **Create function**

47. Create function by selecting **Author from scratch**.

48. Under **Basic Information**, configure the following settings:

    - For **Function name**, enter `indexObject_function`
    - For **Runtime**, choose `Python 3.8`

    Expand **Choose or create an execution role** by selecting ▸

    - For **Execution role**, choose **Use an existing role**.
    - For **Existing role**, choose `LabLambdaRole`

49. Choose **Create function**

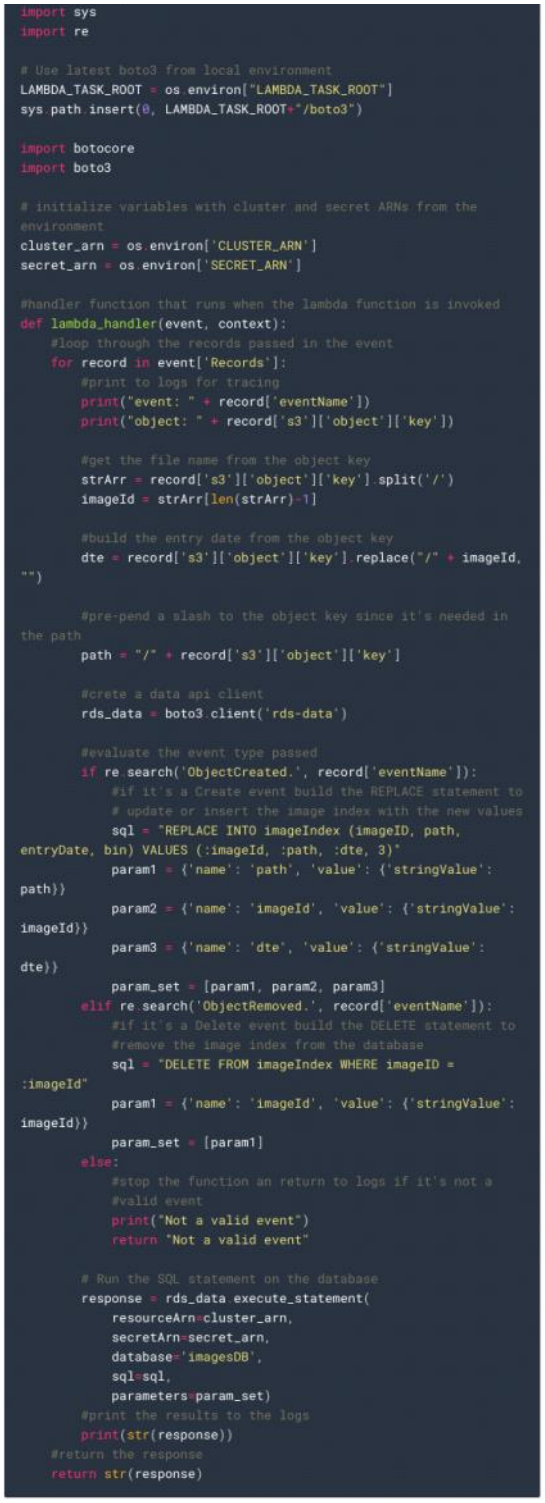
    AWS Lambda gives you many options to upload your code. Since your file is not that large, you will just copy and paste it into the console. The inline editor is a great way to handle small functions and make quick edits.

50. Scroll down to the **Function code** section and replace the text inside of lambda_function with the following code:

    ⚠ Instead of trying to highlight and copy this entire block of code, make sure to select the 📋 icon on the code block when you hover over the top right of the code block. That will copy the entire contents for you to your clipboard.

```
import os
import os.path
```

```
import os import os.path import sys import re # Use latest boto3 from local environment
LAMBDA_TASK_ROOT = os.environ["LAMBDA_TASK_ROOT"] sys.path.insert(0,
LAMBDA_TASK_ROOT+"/boto3") import botocore import boto3 # initialize variables with cluster and
secret ARNs from the environment cluster_arn = os.environ['CLUSTER_ARN'] secret_arn =
os.environ['SECRET_ARN'] #handler function that runs when the lambda function is invoked def
lambda_handler(event, context): #loop through the records passed in the event for record in
event['Records']: #print to logs for tracing print("event: " + record['eventName']) print("object: " +
record['s3']['object']['key']) #get the file name from the object key strArr = record['s3']['object']
['key'].split('/') imageId = strArr[len(strArr)-1] #build the entry date from the object key dte = record['s3']
['object']['key'].replace("/" + imageId, "") #pre-pend a slash to the object key since it's needed in the path
path = "/" + record['s3']['object']['key'] #crete a data api client rds_data = boto3.client('rds-data') #evaluate
the event type passed if re.search('ObjectCreated.', record['eventName']): #if it's a Create event build the
REPLACE statement to # update or insert the image index with the new values sql = "REPLACE INTO
imageIndex (imageID, path, entryDate, bin) VALUES (:imageId, :path, :dte, 3)" param1 = {'name': 'path',
'value': {'stringValue': path}} param2 = {'name': 'imageId', 'value': {'stringValue': imageId}} param3 =
{'name': 'dte', 'value': {'stringValue': dte}} param_set = [param1, param2, param3] elif
re.search('ObjectRemoved.', record['eventName']): #if it's a Delete event build the DELETE statement to
#remove the image index from the database sql = "DELETE FROM imageIndex WHERE imageID
= :imageId" param1 = {'name': 'imageId', 'value': {'stringValue': imageId}} param_set = [param1] else:
#stop the function an return to logs if it's not a #valid event print("Not a valid event") return "Not a valid
event" # Run the SQL statement on the database response =
rds_data.execute_statement( resourceArn=cluster_arn, secretArn=secret_arn, database='imagesDB',
sql=sql, parameters=param_set) #print the results to the logs print(str(response)) #return the response
return str(response)
```



```
import sys
import re

# Use latest boto3 from local environment
LAMBDA_TASK_ROOT = os.environ["LAMBDA_TASK_ROOT"]
sys.path.insert(0, LAMBDA_TASK_ROOT+"/boto3")

import botocore
import boto3

# initialize variables with cluster and secret ARNs from the
environment
cluster_arn = os.environ['CLUSTER_ARN']
secret_arn = os.environ['SECRET_ARN']

#handler function that runs when the lambda function is invoked
def lambda_handler(event, context):
    #loop through the records passed in the event
    for record in event['Records']:
        #print to logs for tracing
        print("event: " + record['eventName'])
        print("object: " + record['s3']['object']['key'])

        #get the file name from the object key
        strArr = record['s3']['object']['key'].split('/')
        imageId = strArr[len(strArr)-1]

        #build the entry date from the object key
        dte = record['s3']['object']['key'].replace("/" + imageId,
"")

        #pre-pend a slash to the object key since it's needed in
the path
        path = "/" + record['s3']['object']['key']

        #crete a data api client
        rds_data = boto3.client('rds-data')

        #evaluate the event type passed
        if re.search('ObjectCreated.', record['eventName']):
            #if it's a Create event build the REPLACE statement to
            # update or insert the image index with the new values
            sql = "REPLACE INTO imageIndex (imageID, path,
entryDate, bin) VALUES (:imageId, :path, :dte, 3)"
            param1 = {'name': 'path', 'value': {'stringValue':
path}}
            param2 = {'name': 'imageId', 'value': {'stringValue':
imageId}}
            param3 = {'name': 'dte', 'value': {'stringValue':
dte}}
            param_set = [param1, param2, param3]
        elif re.search('ObjectRemoved.', record['eventName']):
            #if it's a Delete event build the DELETE statement to
            #remove the image index from the database
            sql = "DELETE FROM imageIndex WHERE imageID =
:imageId"
            param1 = {'name': 'imageId', 'value': {'stringValue':
imageId}}
            param_set = [param1]
        else:
            #stop the function an return to logs if it's not a
            #valid event
            print("Not a valid event")
            return "Not a valid event"

        # Run the SQL statement on the database
        response = rds_data.execute_statement(
            resourceArn=cluster_arn,
            secretArn=secret_arn,
            database='imagesDB',
            sql=sql,
            parameters=param_set)
        #print the results to the logs
        print(str(response))
    #return the response
    return str(response)
```

51. Choose **Deploy** to publish changes.

    You need to configure two environment variables for your function to be able to connect to your database. The **CLUSTER_ARN** is the Amazon Resoure Name (ARN) for your database cluster, the **SECRET_ARN** is the ARN for your Secrets Manager secret.

52. Scroll down to the **Environment variables** section and choose **Edit**

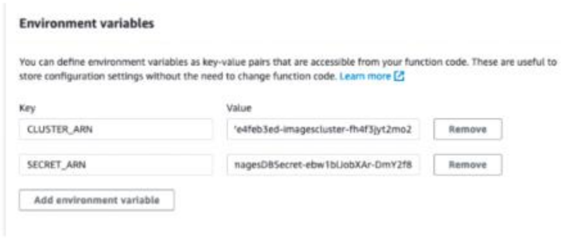
53. Choose **Add environment variable**

    - For **Key**, enter `CLUSTER_ARN`
    - For **Value**, enter the ***clusterArn*** value located in the left navigation pane of these instructions.

54. Choose Add environment variable

- For **Key**, enter `SECRET_ARN`
- For **Value**, enter the ***secretsManagerArn*** value located in the left navigation pane of these instructions.

⚠ The environment variable names are case sensitive and should look similar the ones below.



Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

Key	Value	
CLUSTER_ARN	'e4feb3ed-imagescluster-fh4f3jyt2mo2	Remove
SECRET_ARN	nagesDBSecret-ebw1bLJobXAr-DmY2f8	Remove

Add environment variable

55. Choose Save

{ "Records": [{ "eventName": "ObjectCreated:Put", "s3": { "object": { "key": "2020/2/13/test.pdf" } } }] }

Task 3.2: Create a test event for your function

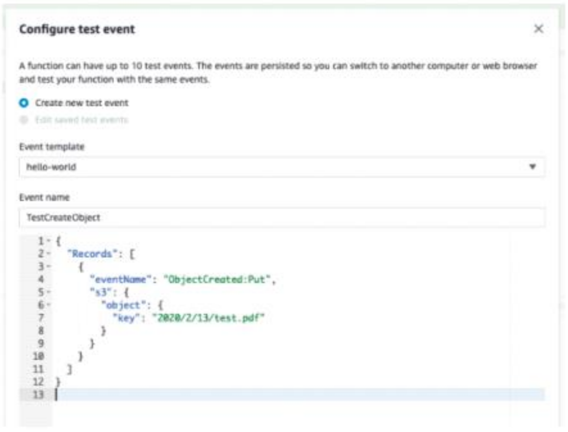
Now that your Lambda function is created, it is time to test. The function code is designed to accept an S3 *create* or *delete* event. You will create two separate tests that represent the information sent to Lambda from the S3 events.

56. Choose Test at the top of the page

57. For **Event name**, enter `TestCreateObject`

58. Replace the event with the following code:

```
{
  "Records": [
    {
      "eventName": "ObjectCreated:Put",
      "s3": {
        "object": {
          "key": "2020/2/13/test.pdf"
        }
      }
    }
  ]
}
```



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

(●) Create new test event
( ) Edit saved test events

Event template: hello-world

Event name: TestCreateObject

```
{
  "Records": [
    {
      "eventName": "ObjectCreated:Put",
      "s3": {
        "object": {
          "key": "2020/2/13/test.pdf"
        }
      }
    }
  ]
}
```

{ "Records": [{ "eventName": "ObjectRemoved:Delete", "s3": { "object": { "key": "2020/2/13/test.pdf" } } }] }

59. Choose Create

Now configure another test event to test object delete events.

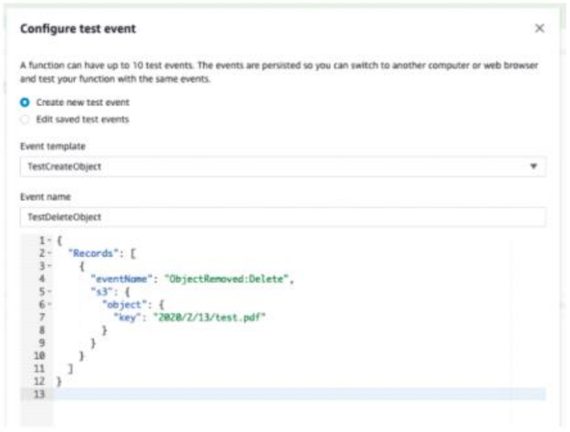
60. Choose **TestCreateObject ▾** at the top of the page to expand the Test Events selection and choose **Configure test events**.

61. Choose **Create new test event**.

62. For **Event name**, enter `TestDeleteObject`

63. Replace the event with the following code:

```
{
  "Records": [
    {
      "eventName": "ObjectRemoved:Delete",
      "s3": {
        "object": {
          "key": "2020/2/13/test.pdf"
        }
      }
    }
  ]
}
```



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

[✓ Create new test event]
Edit saved test events

Event template: TestCreateObject

Event name: TestDeleteObject

```
{
  "Records": [
    {
      "eventName": "ObjectRemoved:Delete",
      "s3": {
        "object": {
          "key": "2020/2/13/test.pdf"
        }
      }
    }
  ]
}
```

64. Choose **Create**

Task 3.3: Test your function

65. Choose **TestDeleteObject** ▾ at the top of the page to expand the Test Events selection and choose **TestCreateObject**.

66. Choose Test

The Execution result should show that it succeeded. If you expand ▸ **Details**, it should display the database response. You should notice that the HTTPStatusCode is 200 and that numberOfRecordsUpdated is 1.

select * from imageIndex WHERE imageID = 'test.pdf';

⚠ If your test failed, verify that you copied the code into your Lambda function correctly, verify your environment variables, and make sure you have the correct execution role selected on the Permissions tab.

Confirm your test in the Query Editor

67. Go to the "RDS - AWS Console" browser tab that you have the Query Editor open in.

68. Select Clear

69. Paste the following code in the editor window.

```
select * from imageIndex WHERE imageID = 'test.pdf';
```

70. Choose **Run**

In the **Result set 1 (1)** you should notice the record for test.pdf.

Test the function for delete events.

71. Go to the "indexObject_function - Lambda" browser tab.

72. Choose **TestCreateObject** ▾ at the top of the page to expand the Test Events selection and choose **TestDeleteObject**.

73. Choose Test

The Execution result should show that it succeeded. If you expand ▸ **Details**, it should display the database response. You should notice that the HTTPStatusCode is 200 and that numberOfRecordsUpdated is 1.

Confirm your test in the RDS Query Editor

74. Go to the "RDS - AWS Console" browser tab that you have the Query Editor open in.

75. With the same statement in the editor window, choose **Run**

In the **Result set 1 (0)** you should notice no records returned.

Task 3.4: Add an S3 event to trigger your function

Now that your Lambda function has been created and tested, you need to configure your S3 bucket to send an event to the function when an object is created or deleted.

In the "indexObject_function - Lambda" browser tab

76. On the navigation bar, choose **Services ⌄**, and choose **S3**.

77. Select the bucket name that matches the ***bucketName*** value located in the left navigation pane of these instructions.

78. Choose the **Properties** tab.

79. Scroll down and choose the **Event notifications** card.

80. Choose **Create event notification**

81. Configure the following settings:

    - In **General configuration** section, for **Name**, enter `indexObject`

    - In **Event types** section, select **All object create events** and **All object delete events**.

    - In **Destination** section, choose **Lambda Function**.

    - To **Specify Lambda function**, choose from your Lambda functions to pick **indexObject_function**.

    - Select **Save changes**

In this task, you created a lambda function that will maintain the external index of your objects in response to S3 create or delete events. You then configured your bucket to send an event to your Lambda function every time an object is created or deleted.

Task 4: Migrate your application storage to Amazon S3

Now that the resources supporting the new imaging application bin are in place, reconnect to your Session Manager console to copy your document images to S3.

In the "S3 Management Console browser" tab

82. On the navigation bar, choose **Services ⌄**, and choose **EC2**.

83. In the left navigation pane, under **INSTANCES**, choose **Instances**.

84. Select the row of the **Img-App-Prod** instance.

85. Choose **Connect**

86. For **Connection method**, choose **Session Manager**.

cat -n /lab/migrate_pdfs.py

87. Choose **Connect**

    You will be executing a python script that was created by your Application Administrators to migrate the files from your instance's EBS volume to S3. The script does the following:

    - Connects to the database
    - Gets the records not in the S3 bin
    - Loops through all records and uploads them to S3, placing anything in the archive bin in the Infrequently Accessed storage tier

    Inspect the script that was given to you so that you can review how it performs the tasks above.

python3 /lab/migrate_pdfs.py

88. Enter the following command and press ENTER.

```
cat -n /lab/migrate_pdfs.py
```

    Lines 45 and 58 call the method that actually uploads the files to S3. The *upload_file* method accepts a file name, a bucket name, and a target object name. It handles large files by splitting them into smaller chunks and uploading each chunk in parallel.

    Additionally, upload_file accepts an optional ExtraArgs parameter that can be used for various purposes. You are using this to set the access control list on the file, the appropriate Content-Type metadata for your browser to view the objects, and in the case of archive objects, setting the storage class to STANDARD_IA.

SELECT bin, count(imageID) AS total FROM imageIndex GROUP BY bin;

    You will run the script to upload your files to S3.

89. Enter the following command and press ENTER.

```
python3 /lab/migrate_pdfs.py
```

    You should notice all of the records scroll through the screen, 1250 in total. **Do not close the tab where your Session Manager console is open, you will need it again in Task 5.**

    Confirm your migration in the Query Editor

90. Go to the "RDS - AWS Console" browser tab that you have the Query Editor open in.

91. Press Clear

92. Paste the following code in the editor window.

```
SELECT bin, count(imageID) AS total FROM imageIndex GROUP BY bin;
```

93. Choose **Run**

    In the **Result set 1(1)**, you should only notice bin 3 if you migration is complete. If it is still copying in the AWS Systems Manager browser tab, you will notice the numbers update according to the progress each time you run the query. You can close this Query Editor tab, there will be no more database queries to run in this lab.

    In this task, you uploaded all of the document images to Amazon S3. Any objects that were in the archive bin were uploaded directly to the Standard_IA storage tier. All others were uploaded to the bucket default storage tier, which is S3_Standard. You then confirmed that your Lambda function updated the database with the correct location of the objects when they were uploaded.

Task 5: Decommission the Amazon Elastic Block Store volume

Now that you have uploaded all images from the EBS volume, you can remove it from the instance and delete it.

In the "AWS Systems Manager" browser tab.

94. Enter the following commands and press ENTER to unmount the file system and remove the volume.

```
sudo umount /dev/mapper/vg0-images
sudo lvremove /dev/mapper/vg0-images -y
```

Even though you unmounted the drive, it will be remounted (or at least the kernel will try to remount it) every time the operating system boots. To stop the system from trying to remount the drive at boot, you need to remove them from the *fstab* file.

95. To open your *fstab* file in vi enter the following command and pressing ENTER.

```
sudo vi /etc/fstab
```

96. Remove the third line from the file that reads **/dev/mapper/vg0-images /var/www/html/images ext4 defaults 0 2** by doing the following:

- Press the down arrow twice to move to line 3.
- Enter **dd** to remove the line.
- Enter `:wq` and press enter to save and exit vi.

97. Close the "AWS Systems Manager" browser tab and return to the "Instances | EC2 Manager" browser tab.

98. In the left navigation pane, under **ELASTIC BLOCK STORE**, choose **Volumes**.

99. Select the row of the volume that has a Size of **5120 GiB**.

100. Choose **Actions ˅**

101. Choose **Detach Volume**.

102. Choose **Yes, Detach**

When the volume is detached (you may need to select ↻ in the upper-right corner to refresh), you can delete it.

103. With the volume still selected, choose **Actions ˅**

104. Choose **Delete Volume**.

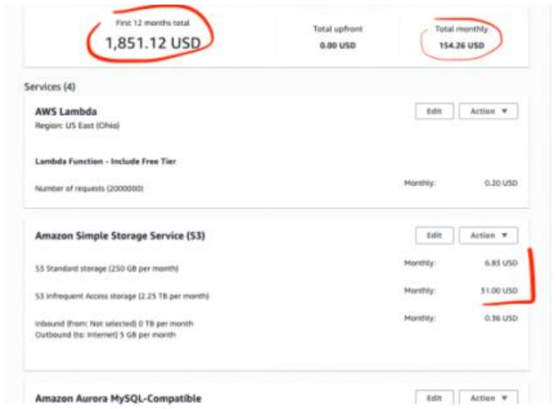
105. Choose **Yes, Delete**

In this task you removed the volume from the operating system of your application server, and deleted it from AWS.

Summary

Below is a screenshot of the AWS pricing calculator showing the cost of the environment after implementing the optimizations in this lab. The prices are as of 5/15/2020.

You can select the link below to view an estimate online with current prices. To match the scenario where an EBS volume is not fully utilized, this estimates 2.5 TB of storage used, 10% of that being S3 Standard.



First 12 months total
1,851.12 USD

Total upfront
0.00 USD

Total monthly
154.26 USD

Services (4)

AWS Lambda
Region: US East (Ohio)

Lambda Function - Include Free Tier

Number of requests (2000000)	Monthly:	0.20 USD

Amazon Simple Storage Service (S3)

S3 Standard storage (250 GB per month)	Monthly:	6.83 USD
S3 Infrequent Access storage (2.25 TB per month)	Monthly:	31.00 USD
Inbound (from: Not selected) 0 TB per month Outbound (to: Internet) 5 GB per month	Monthly:	0.36 USD

Amazon Aurora MySQL-Compatible

https://calculator.aws/#/estimate?id=80d49be982f5d79d57254e47089f0e855258885c

In this lab you learned how the correct storage service and type can drastically reduce your overall AWS budget. You learned how to use an external index with Amazon S3 to easily be able to locate the objects that you store there. Finally, you learned that utilizing serverless technologies, such as Amazon Aurora Serverless, allows you to only pay for capacity when you need it.

End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

106. Return to the AWS Management Console.

107. On the navigation bar, choose **awsstudent@<AccountNumber>**, and then choose **Sign Out**.

108. Choose **End Lab**

109. Choose **OK**

110. (Optional):

- Select the applicable number of stars ☆
- Type a comment
- Choose **Submit**

    - 1 star = Very dissatisfied
    - 2 stars = Dissatisfied
    - 3 stars = Neutral
    - 4 stars = Satisfied
    - 5 stars = Very satisfied

You may close the window if you don't want to provide feedback.

[redacted]