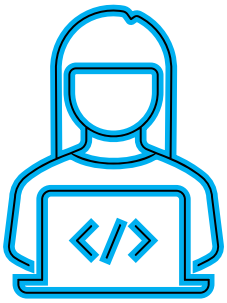
AWS JUMPSTART: AN INTRODUCTORY GUIDE



JANET YU

Table of Contents

Preface

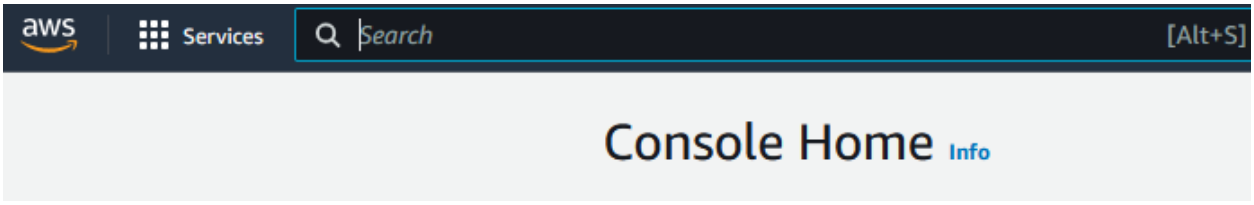
AWS (Amazon Web Services) is Amazon's offer of cloud computing solutions. It is now the world's largest cloud computing service provider. Its services include infrastructure (such as EC2, ECS, EKS), storage (such as S3, EBS, EFS), database (such as DocumentDB, DynamoDB, Aurora), analytics, machine learning, networking, security & identity. More and more businesses are moving their computing infrastructure, applications, and data onto AWS. Therefore, knowing AWS is becoming a necessary skill for not only IT professionals, but also regular business practitioners.

This book is written with entry-level materials to help the readers learn AWS by themselves. Detailed step-by-step instructions make the technology learning an enjoyable process. It is worth noting that AWS provides about a hundred different services and this book is only used as an introductory guide to the complex AWS ecosystem.

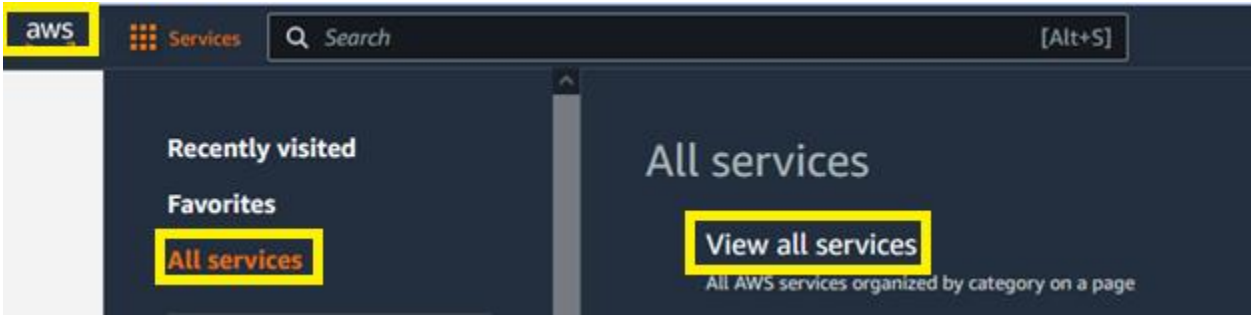
In Chapter 1, we will cover AWS infrastructure services. In Chapter 2, we will practice using AWS storage services and web services. In Chapter 3, we will learn AWS lambda functions. In Chapter 4, we will use AWS database services.

What you need to use this book

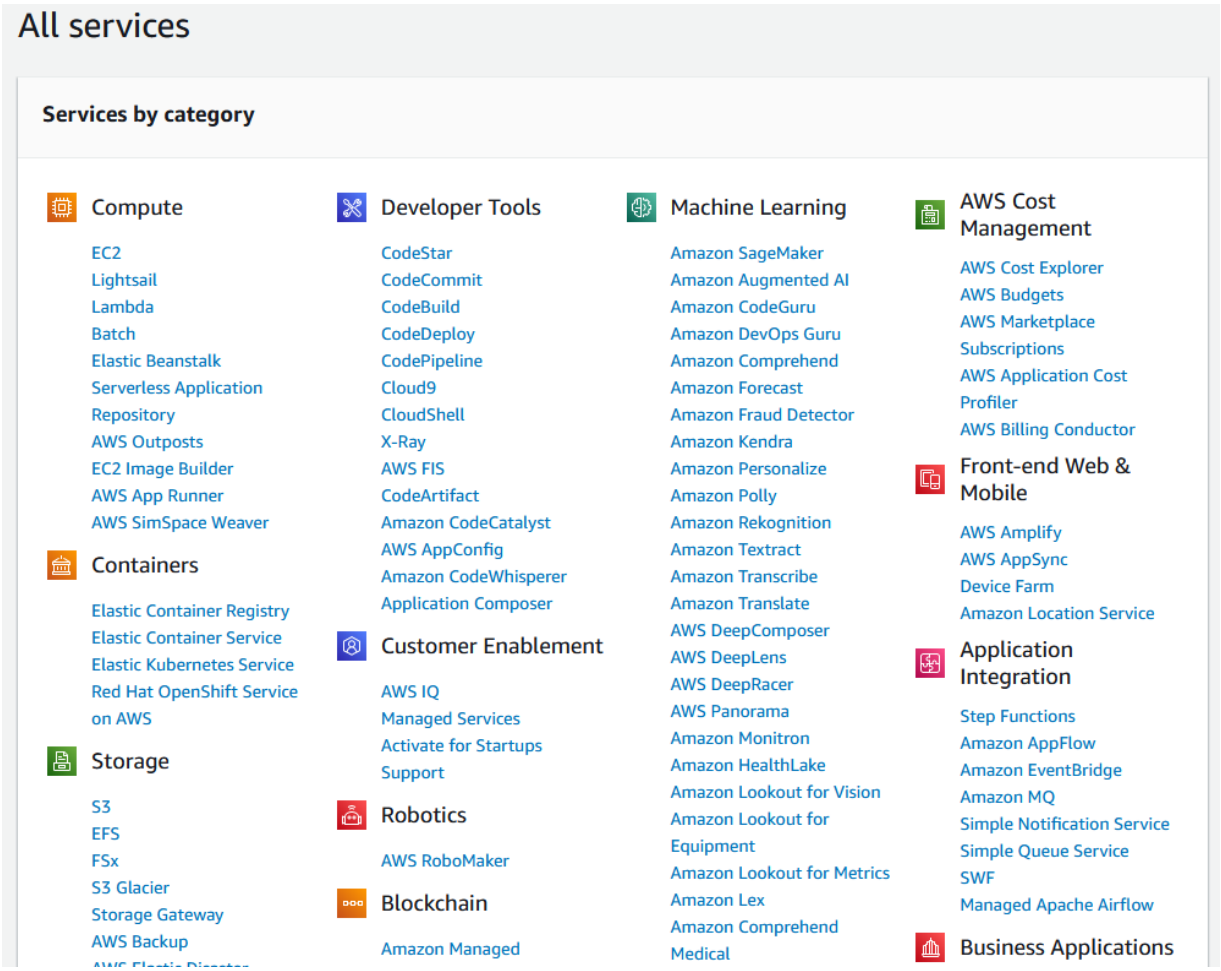
You need to create an AWS account at https://aws.amazon.com/resources/create-account/. After login, you should see the following AWS console home page.



Please click on aws, All services, then view all services.



You should be led to the following page. All our practices in this book will start here.



Downloading the example code

You can download the code files for this book from https://github.com/awsbookresource/files.

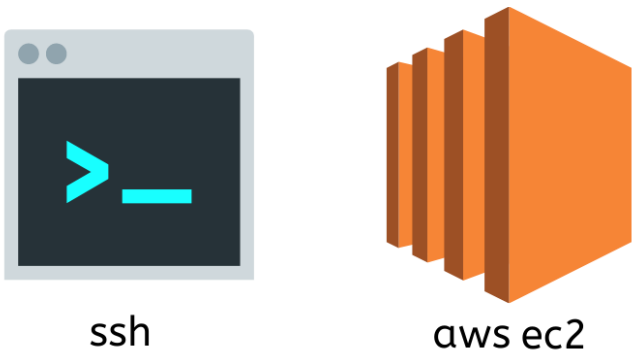
Chapter 1. Virtual Machines

AWS's Elastic Compute Cloud (EC2) is Amazon's solution to cloud-based virtual machines. It provides a wide selection of instance types with customizing CPU, memory, storage, network, and operating systems to meet different business needs. In this chapter, we will practice creating a Linux virtual machine and a Windows virtual machine.

1.1 Linux Virtual Machine

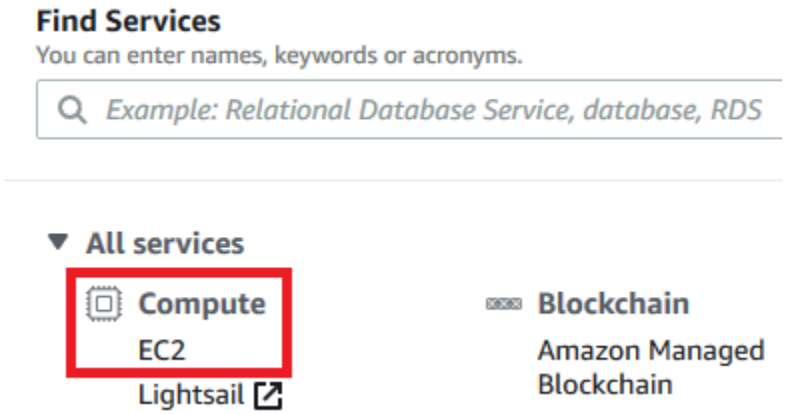
Objectives:

- Create an RSA key pair
- Create an AWS Linux virtual machine
- Access the AWS Linux virtual machine
- Terminate the AWS Linux virtual machine

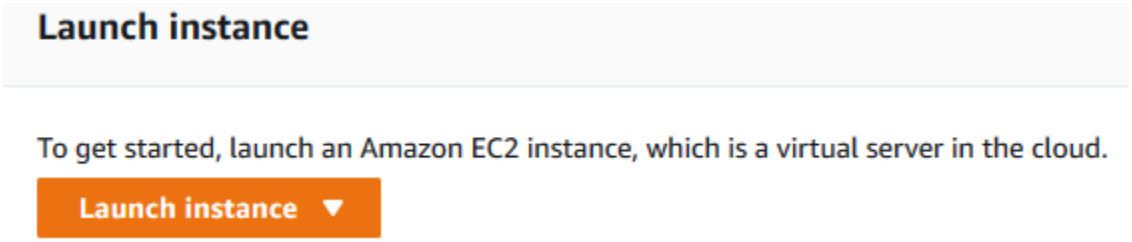


Part 1. Create a Linux virtual machine

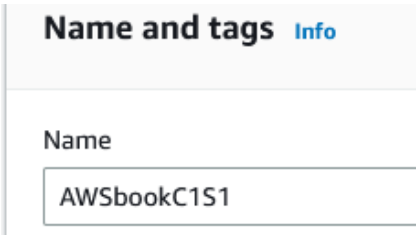
1. Log onto AWS.
2. Go to AWS Management Console.
3. Click “EC2” under Compute.



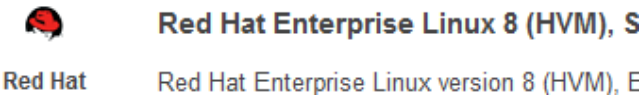
4. Click “Launch instance”.



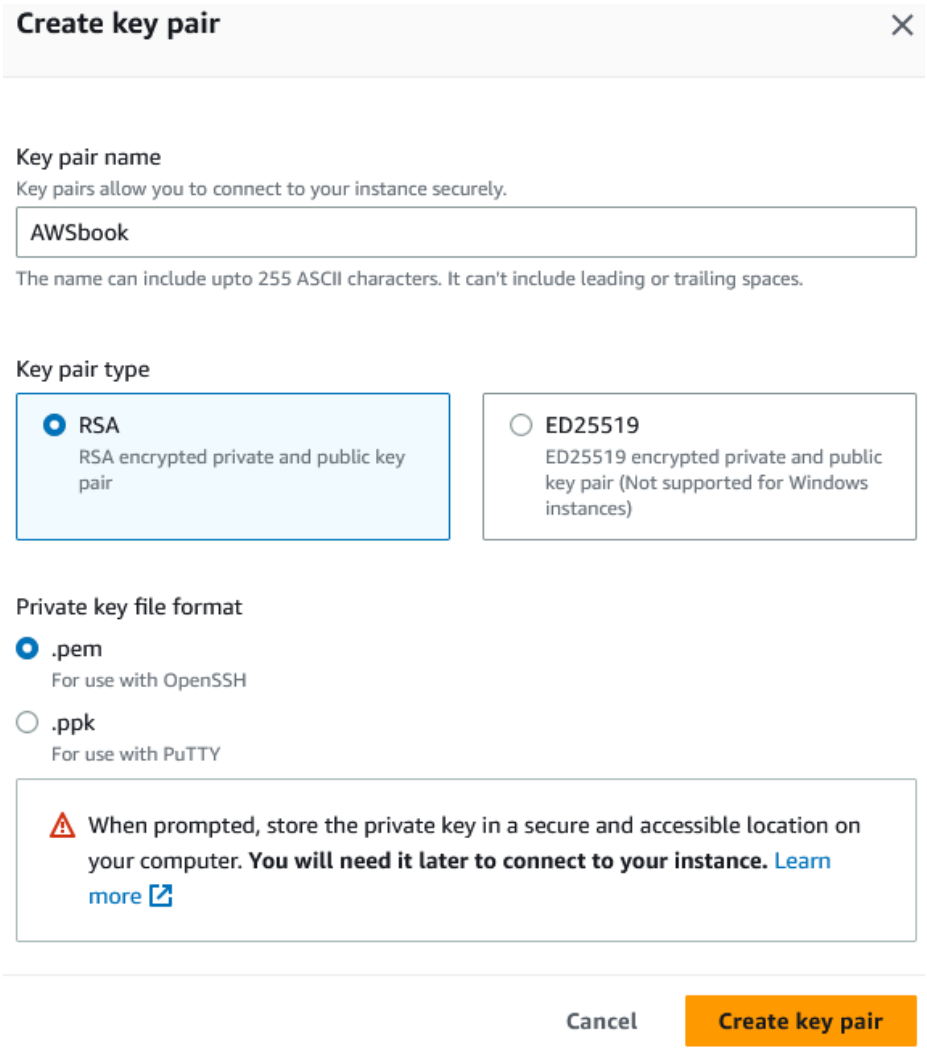
5. Assign a name.



6. Select "Red Hat".

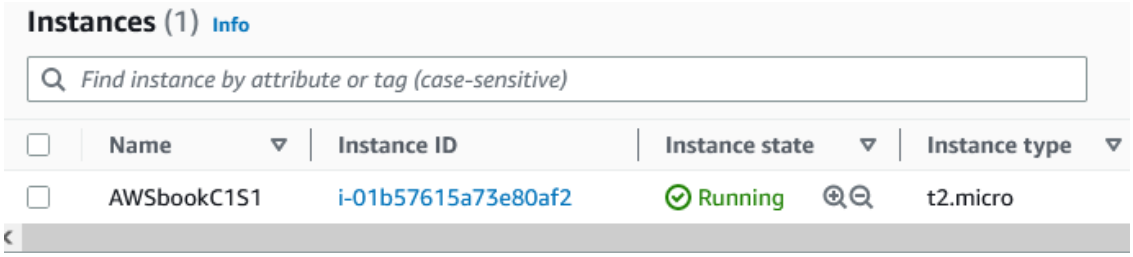


7. Create a new key pair, give it a name, select RSA type and **.pem** format. Click "**Create key pair**".



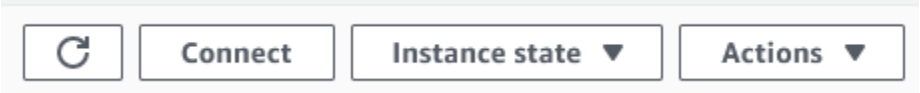
8. Your key will be downloaded. Save it at a safe place. You will need to use it many times later.
9. Click "Launch instances".
10. Your instances are now launching. Click "View all instances".

11. When launching is completed, you can see its state becomes "running" (green).

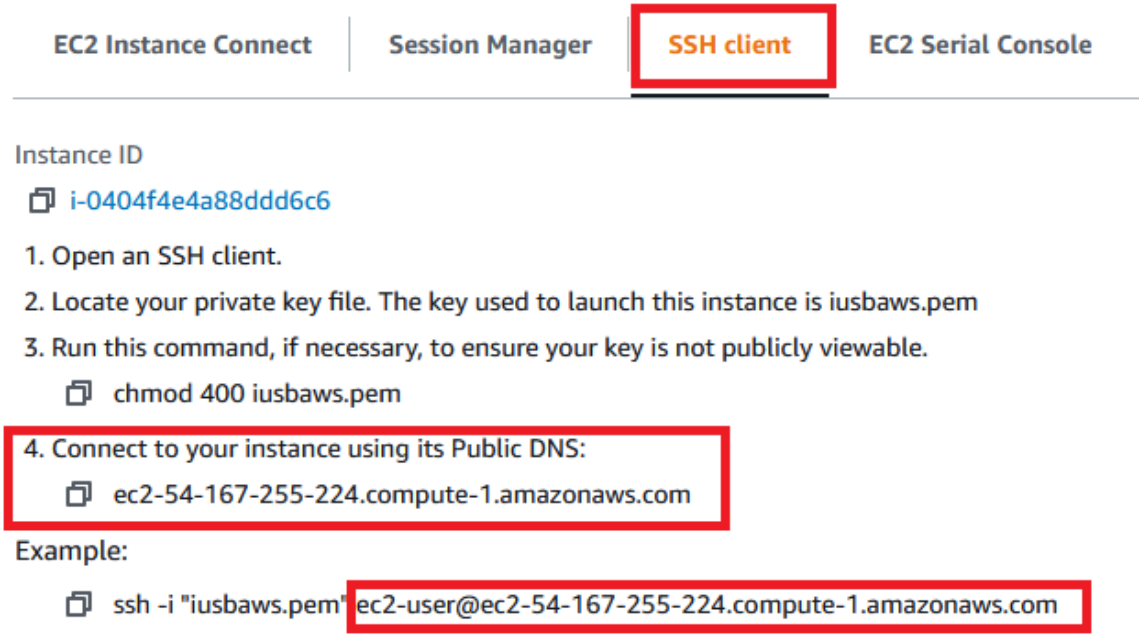


Part 2. Connect to the AWS EC2 with WinSCP

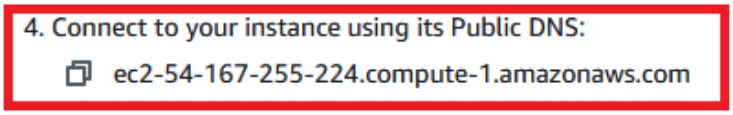
12. Click the Instance ID. Then click "Connect" to get your Connection Information.



13. Click **SSH client**.

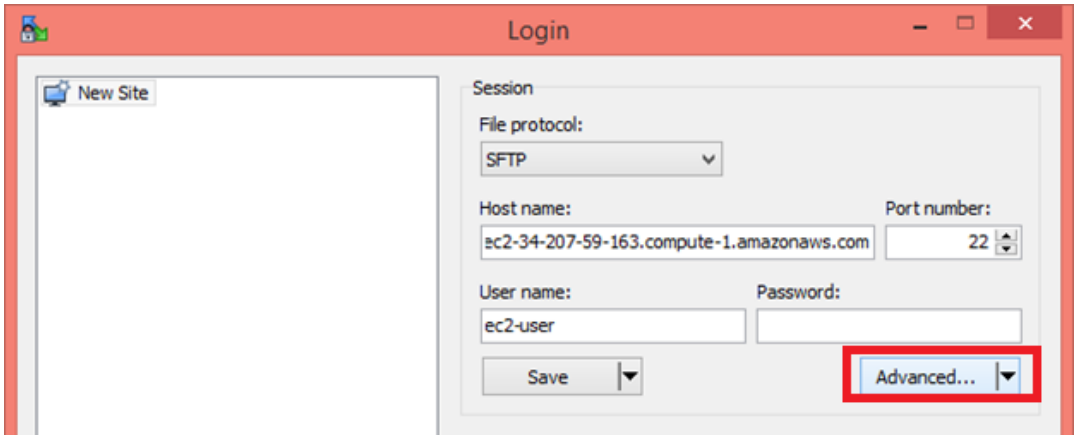


14. Start WinSCP. On Hostname, paste the one you see on Connection Page (Step 4).

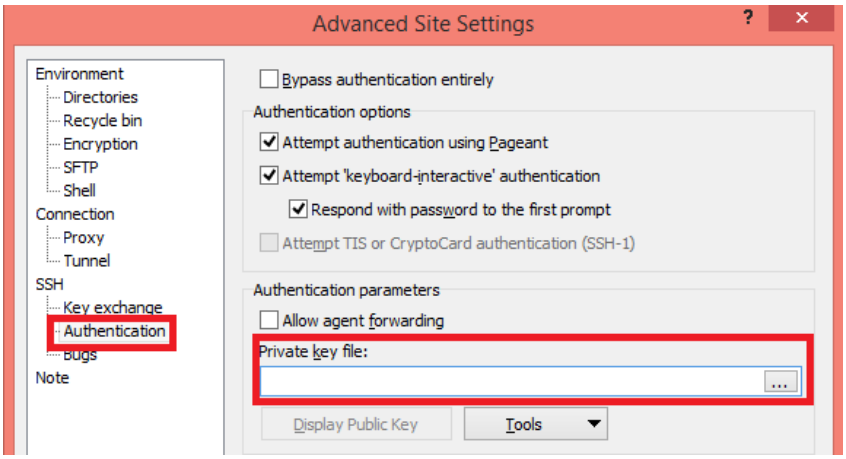


15. Username should be "ec2-user".

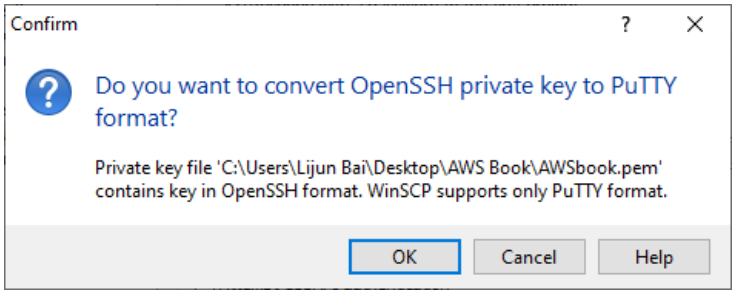
16. Click "Advanced".



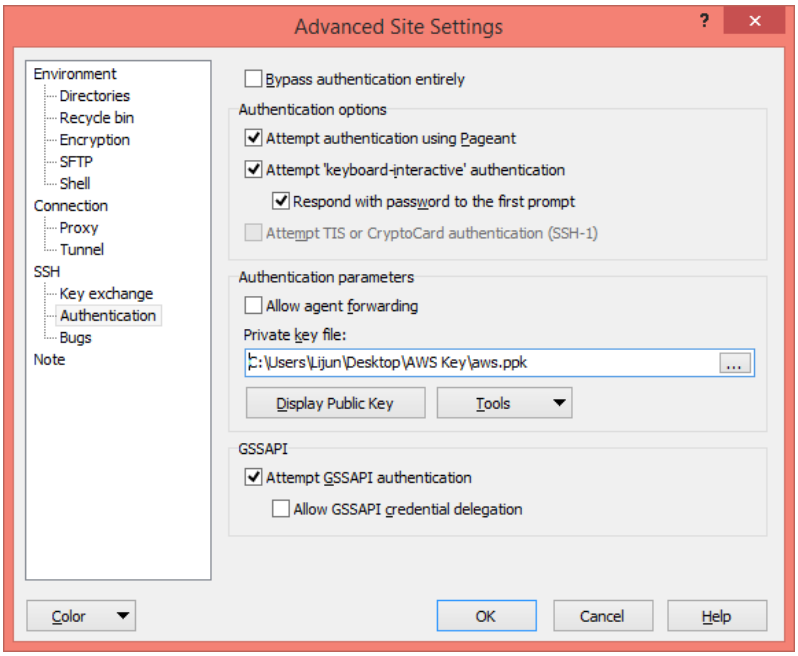
17. Click "Authentication". Browse to Private key file you downloaded earlier.



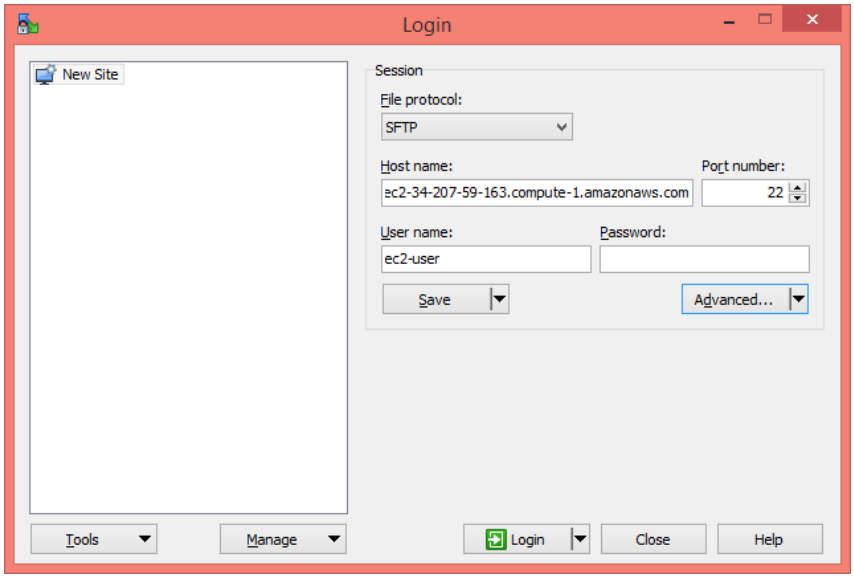
18. To show **.pem** file, select all file types. Then select the private key file (**.pem**) you saved at Step 8.
19. You will be asked to convert **.pem** to **.ppk** format. Click "OK".



20. Then click “Save” to save the .ppk file in the same folder as .pem file. Click “OK” .



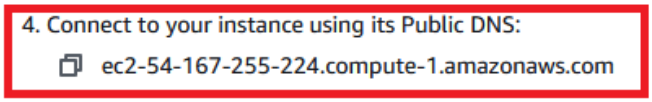
21. Click “Login”.



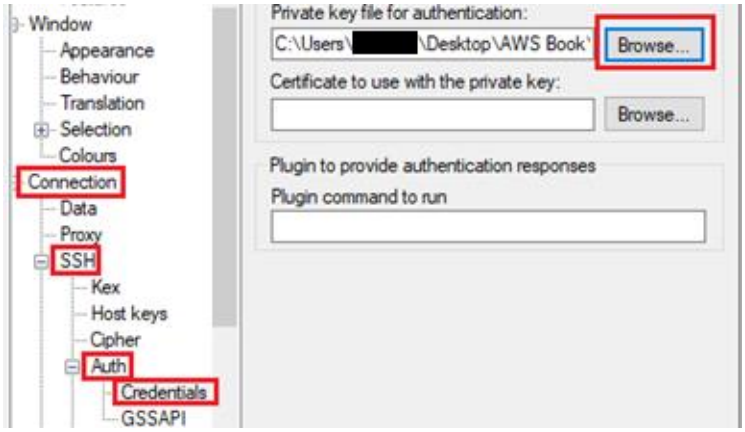
22. You will be logged into the virtual machine through WinSCP.

Part 3. Connect to the virtual machine with PuTTY

23. Start **PuTTY**.
24. In the Category pane, choose Session.
25. In the Hostname box, paste the one you see Connection Page (Step 4).

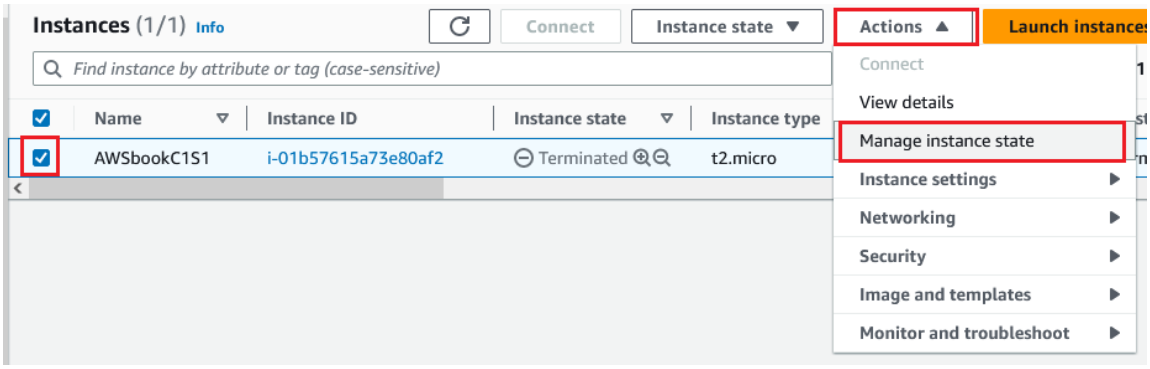


26. Under Connection type, select SSH. Ensure that the Port value is 22.
27. In the Category pane, expand Connection, expand SSH, and then choose Auth, Credentials. Choose Browse. Select the **.ppk** file that you generated for your key pair and choose Open.



28. Click Open, you will be asked for username (**ec2-user**), which can be found here.

29. You are now connected to the AWS virtual machine with PuTTY.
30. To terminate services: Go to console, click EC2, select your EC2 instances, click “Actions”, select “Manage Instance State”, and then “Terminate”.


Instances (1/1) Info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
Name
Instance ID
Instance state
Instance type
AWSbookC1S1
i-01b57615a73e80af2
Terminated
t2.micro
Connect
View details
Manage instance state
Instance settings
Networking
Security
Image and templates
Monitor and troubleshoot

1.2 Windows Virtual Machine

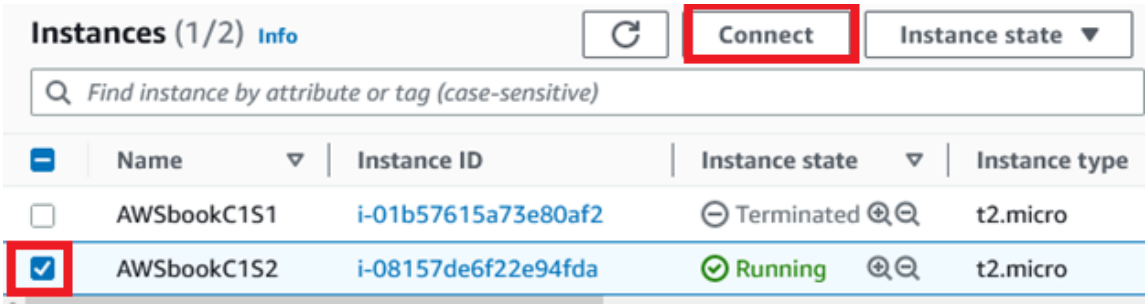
Objectives:

- Create an AWS Windows virtual machine
- Access the AWS Windows virtual machine

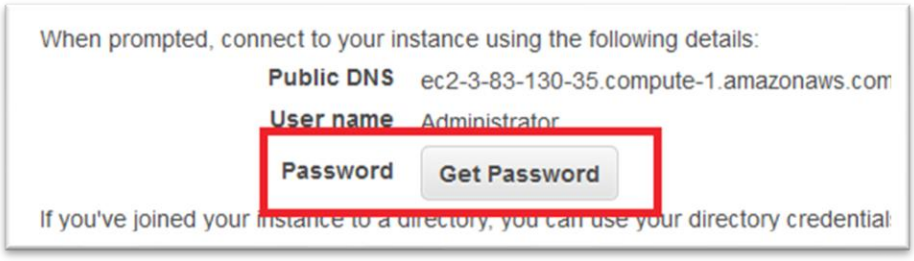
Part 1. Create and connect to a Windows virtual machine

1. Repeat Steps 1-5 in Section 1.1. Instead of choosing Red Hat, you should select a windows Free Tier machine.
2. Select your **existing** key pair, you do not need to create a new key pair.
3. Click "Launch instance".

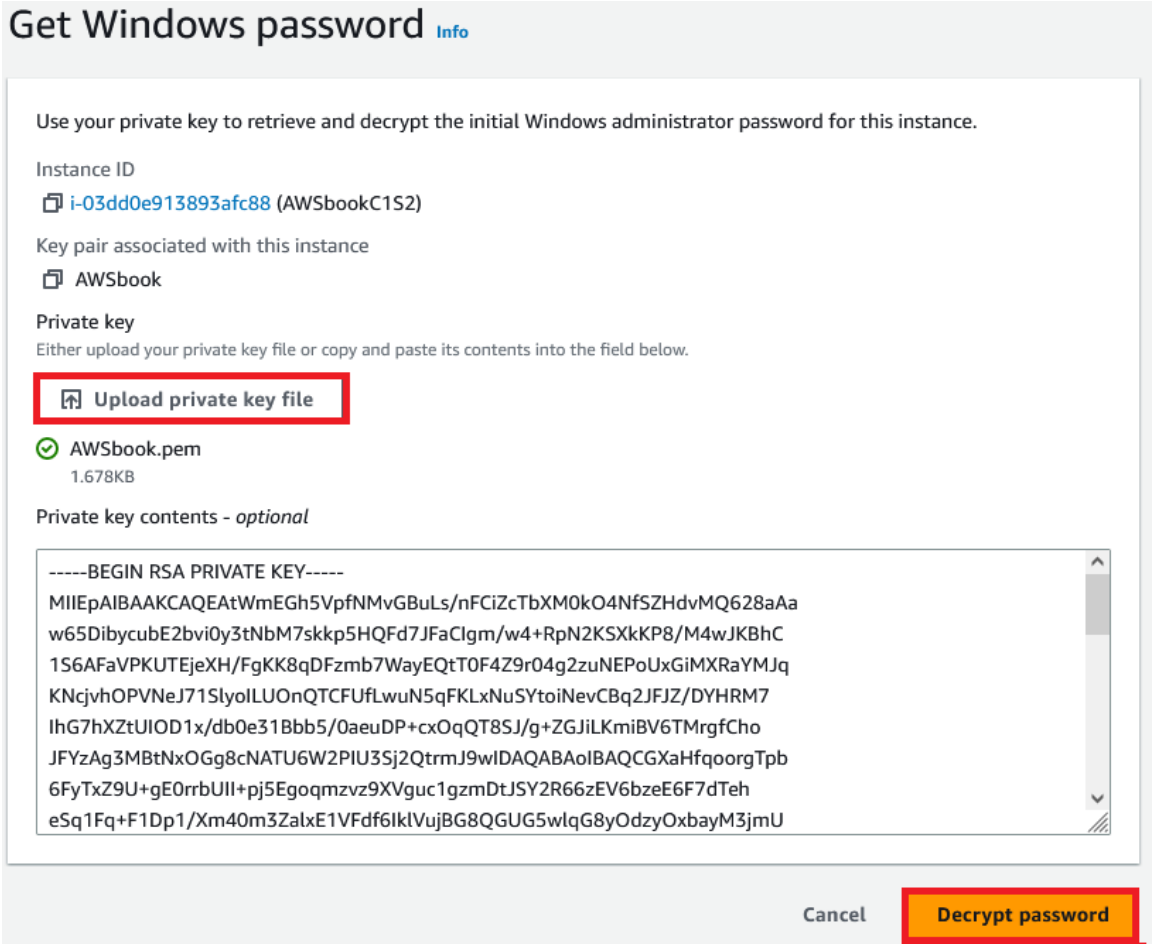
4. After launching is completed, go back to EC2→instances panel.
5. Select the instance and click "Connect".



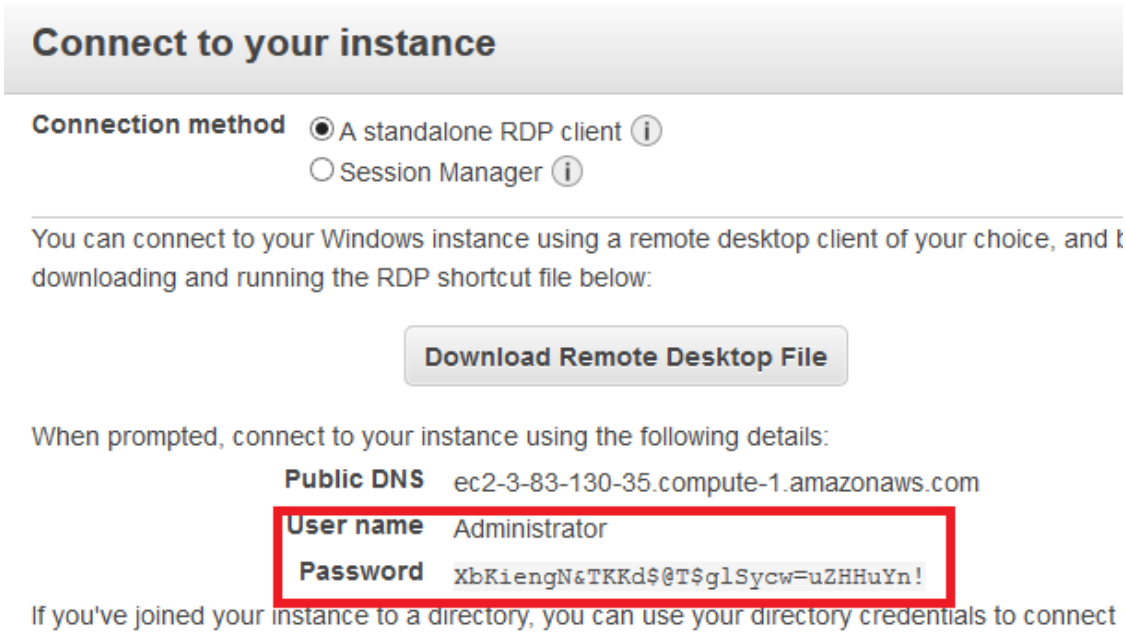
6. Click "**RDP client**" then click "Get Password".



7. Click “Upload private key file” and select the **.pem** file you downloaded in Section 1.1. Click “Decrypt Password”.

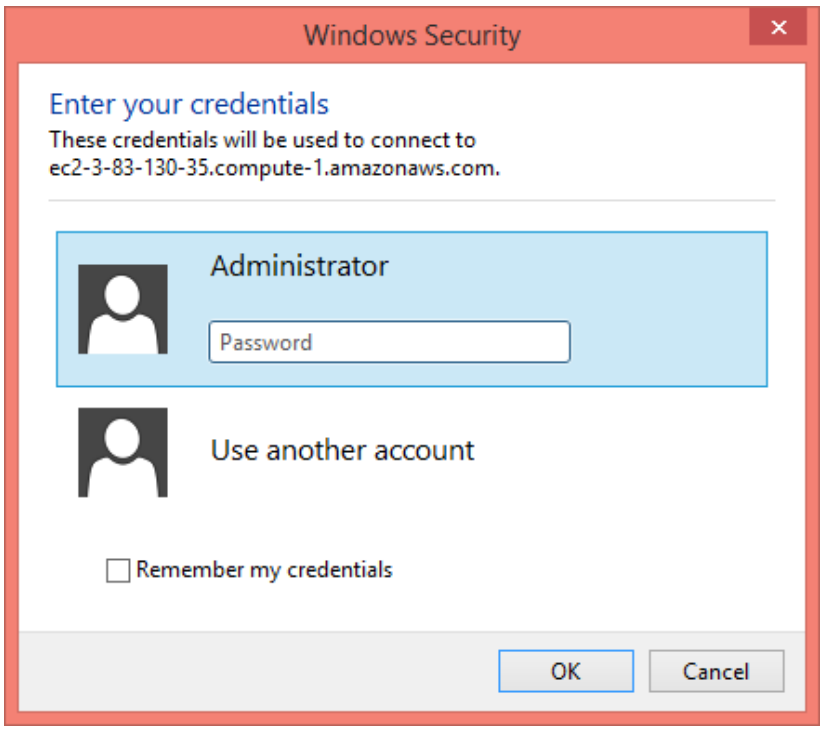


8. Copy the password. Close the window if needed.



9. On Connection page, click on "**RDP client**", then click on "Download Remote Desktop File".

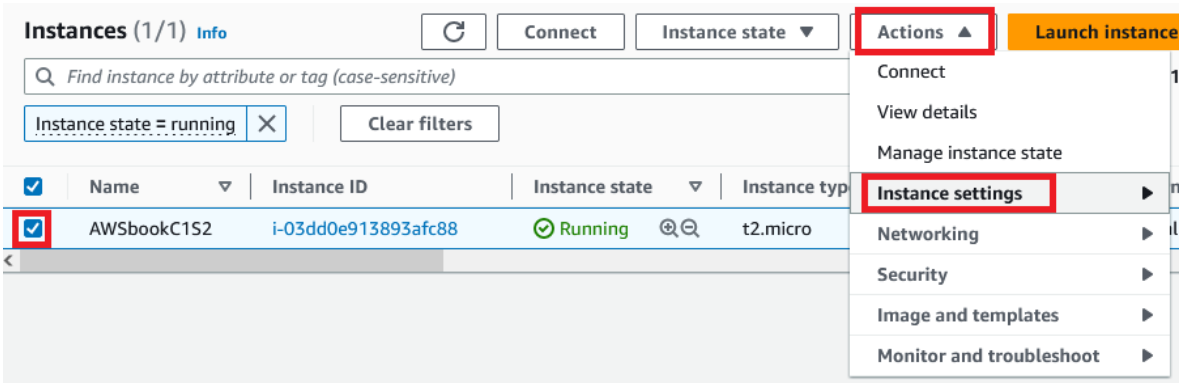
10. Open the file with Program "**Remote Desktop Connection**" on your computer.
11. Paste your password. Click OK.



12. You will be connected to the Windows virtual Machine.

Part 2. Terminate the EC2 instances

13. To terminate services: Go to console, click EC2, select your EC2 instances, click “Actions”, select “Instance Settings”, and then “Terminate”.



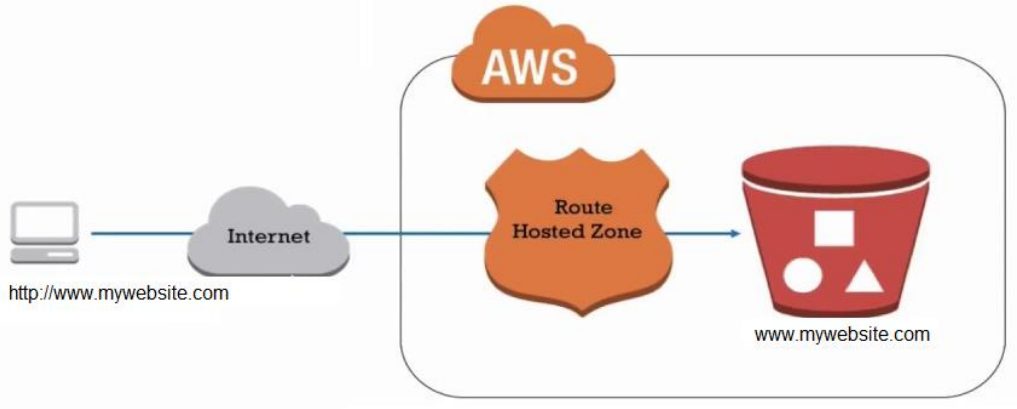
Chapter 2. Storage and Web Services

Amazon's cloud-based storage includes Simple Storage Service (S3), Elastic File System (EFS), Elastic Block Store (EBS), etc. In this chapter, we will practice using S3 service to host a static website.

2.1 S3 and Static Web Hosting

Objectives:

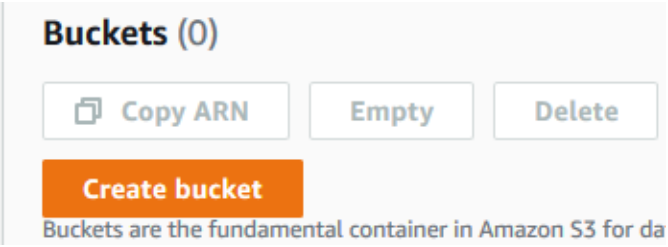
- Create and manage AWS Storage S3
- Publish static web pages with the S3 storage



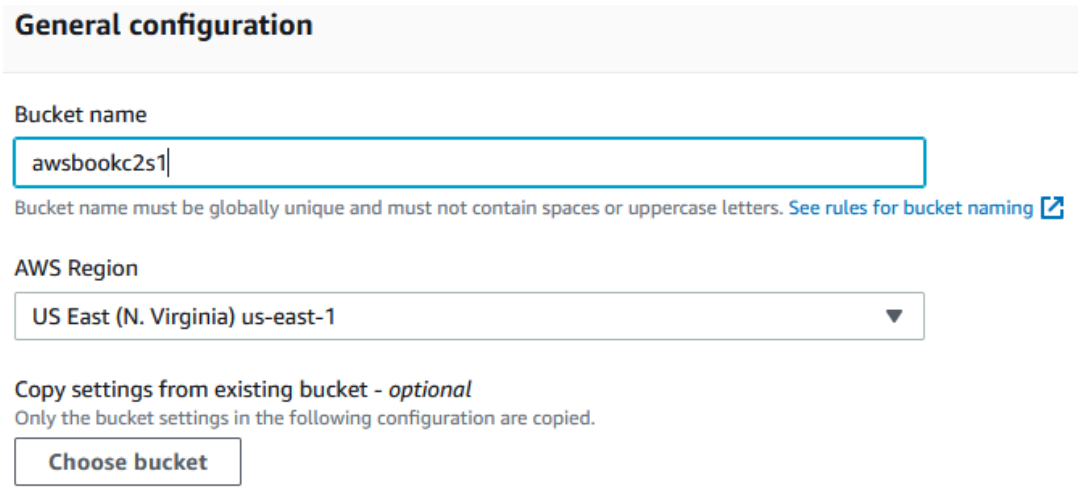
Part 1. Create and test an AWS storage S3 instance (container)

1. From AWS Console, click Services at the top.
2. From AWS Service, select Storage→S3.

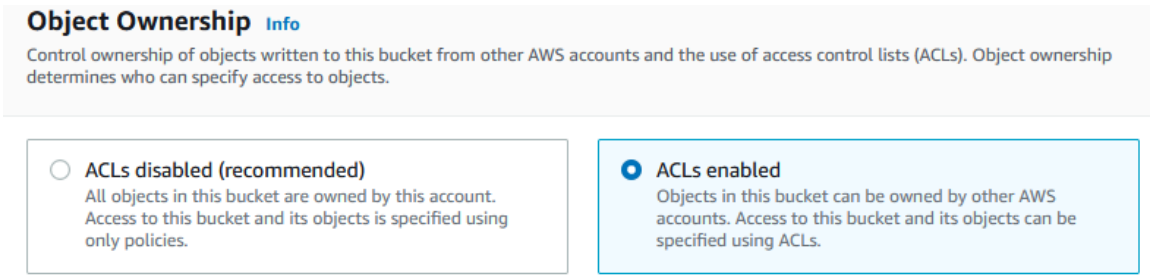
3. Click "Create bucket".



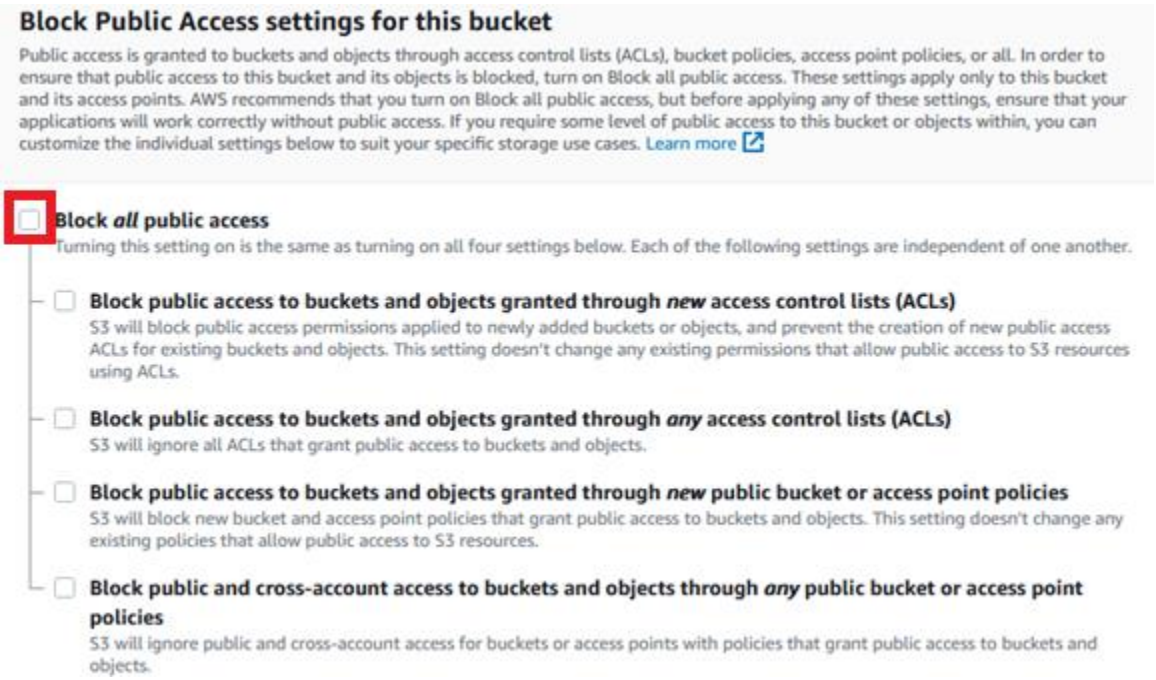
4. Assign a name to the bucket.



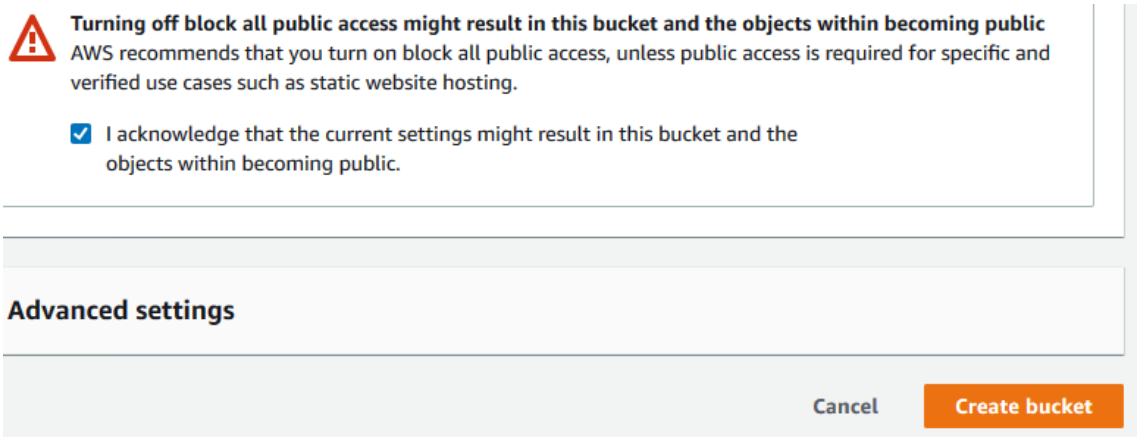
5. Choose “ACLs enabled”.



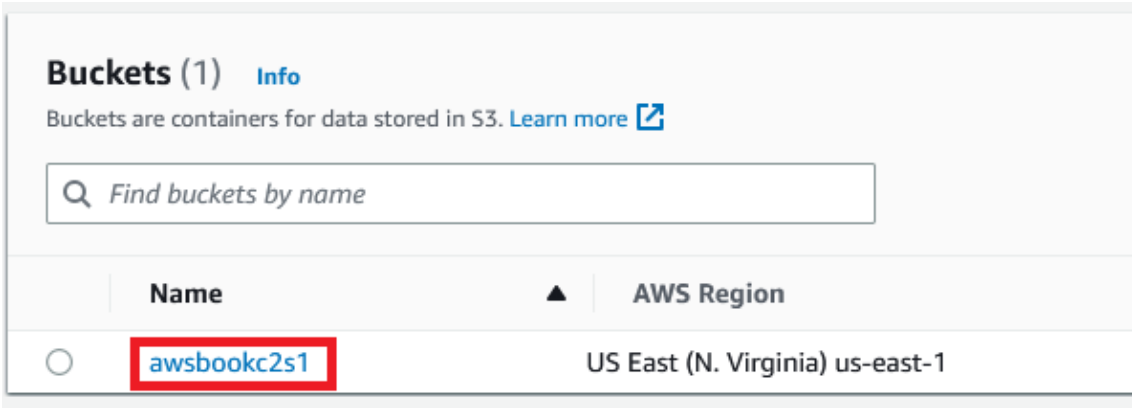
6. **Uncheck** “Block all public access”.



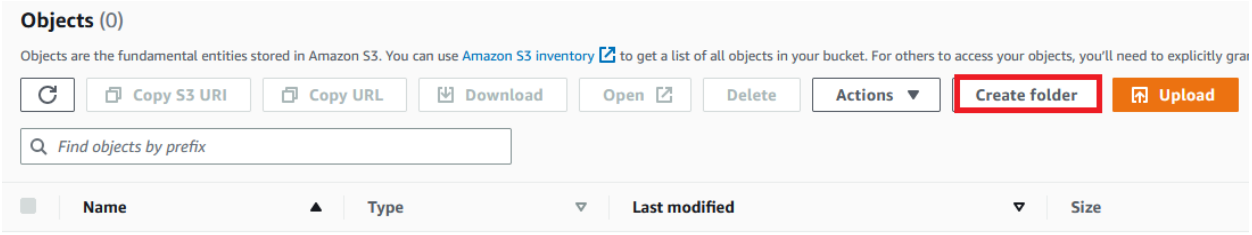
7. Acknowledge turning off the block of public access. Click "Create bucket".



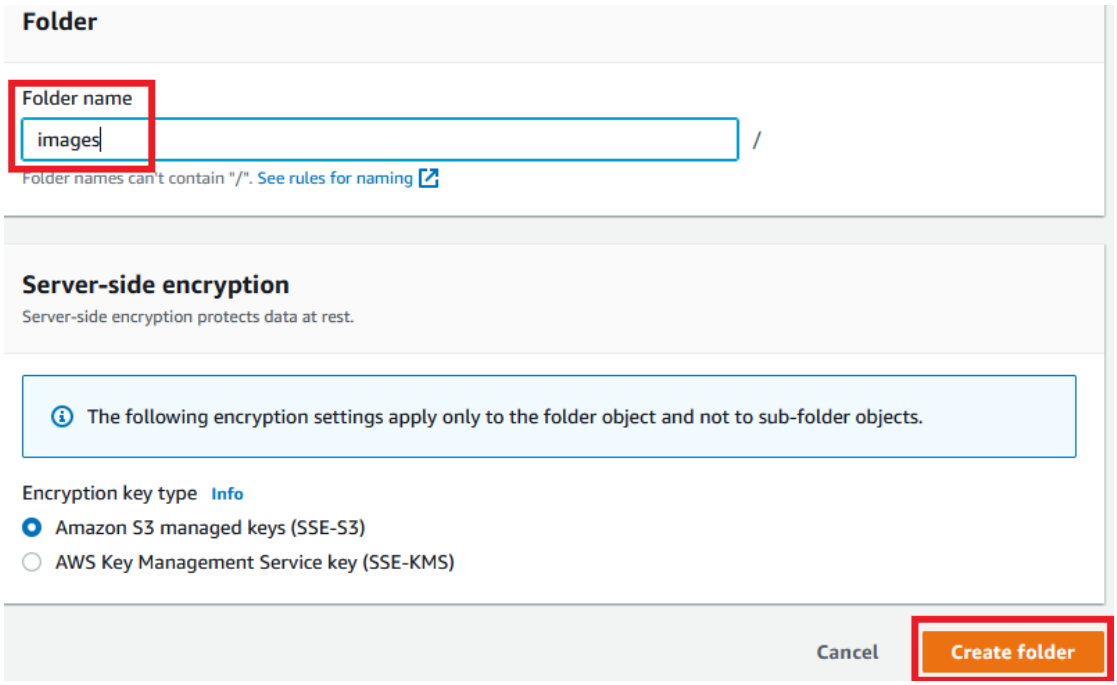
8. Click the bucket you just created.



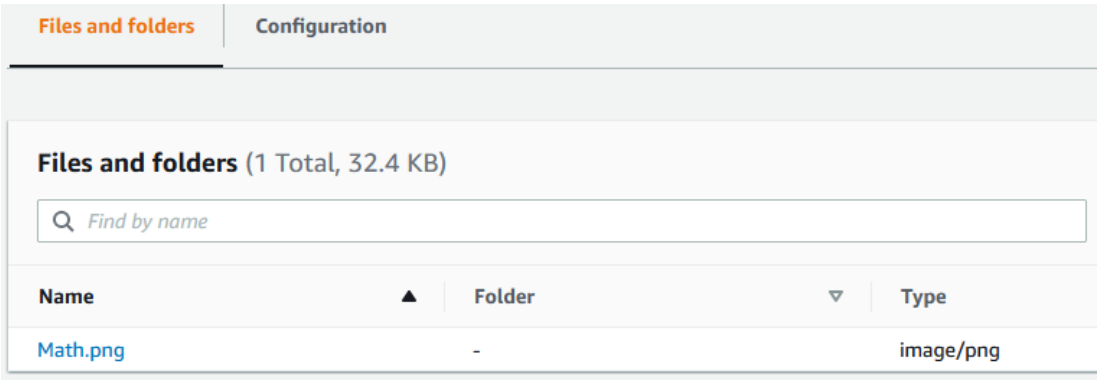
9. Click "Create folder".



10. Create a new folder.

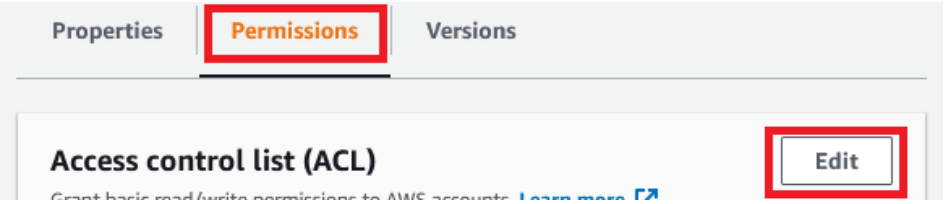


11. Switch to this folder and upload an image file.

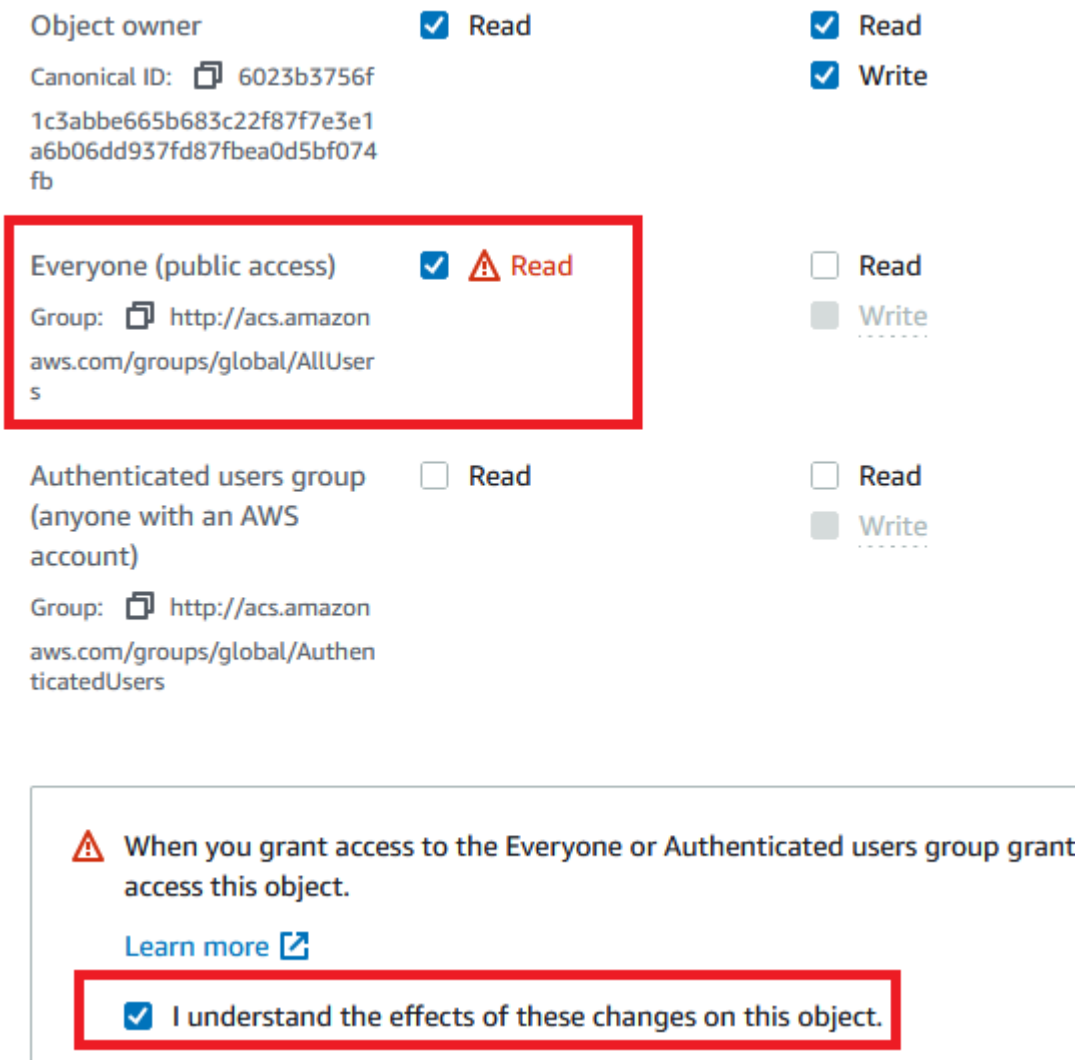


12. Click the file and copy "Object URL".

13. Paste the URL into a web browser. Your access will be denied.
14. Click "Permissions" for this file, then "Edit"



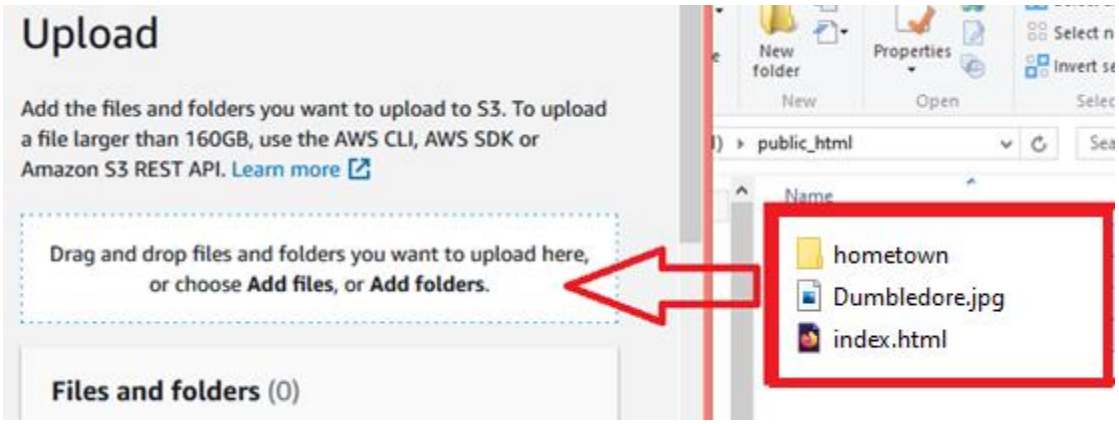
15. Move down, check “Read object” for everyone, check “I understand the effects …”. Click on “**Save changes**”



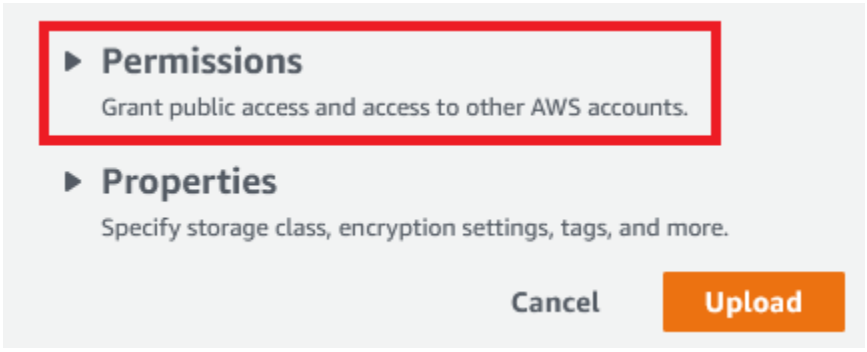
16. Copy and paste your URL into a web browser. You should be able to see the file now.

Part 2. Turn your AWS S3 storage into a static website

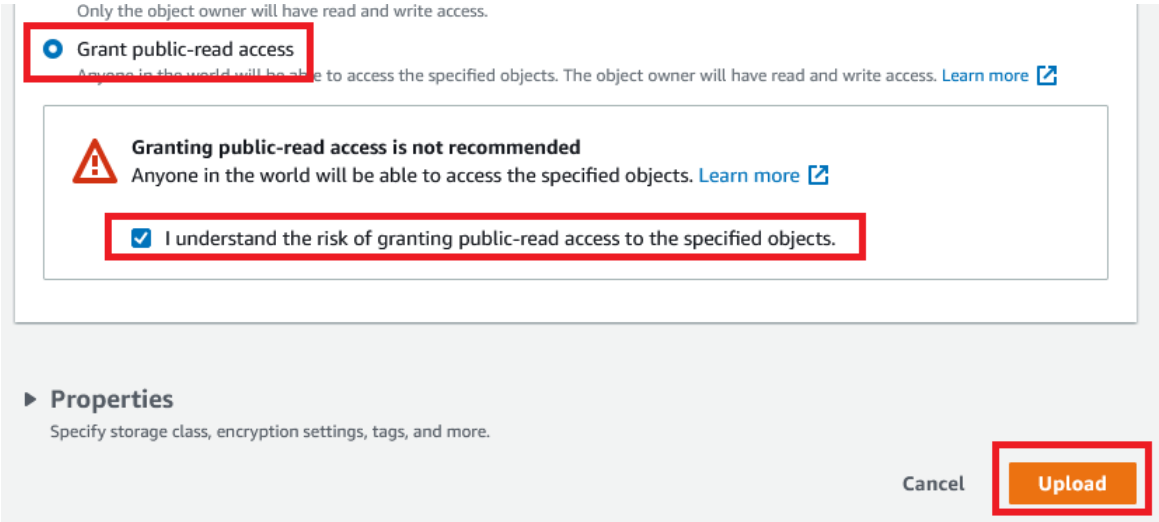
17. Download the zip file provided online (https://github.com/awsbookresource/files). Unzip it.
18. Go back to your Amazon S3 home. Click on the bucket you created.
19. Now you are inside the bucket (outside of the folder you created). Click “Upload”.
20. Upload all the unzipped files (without folder **public_html**) through dragging all files into this bucket.



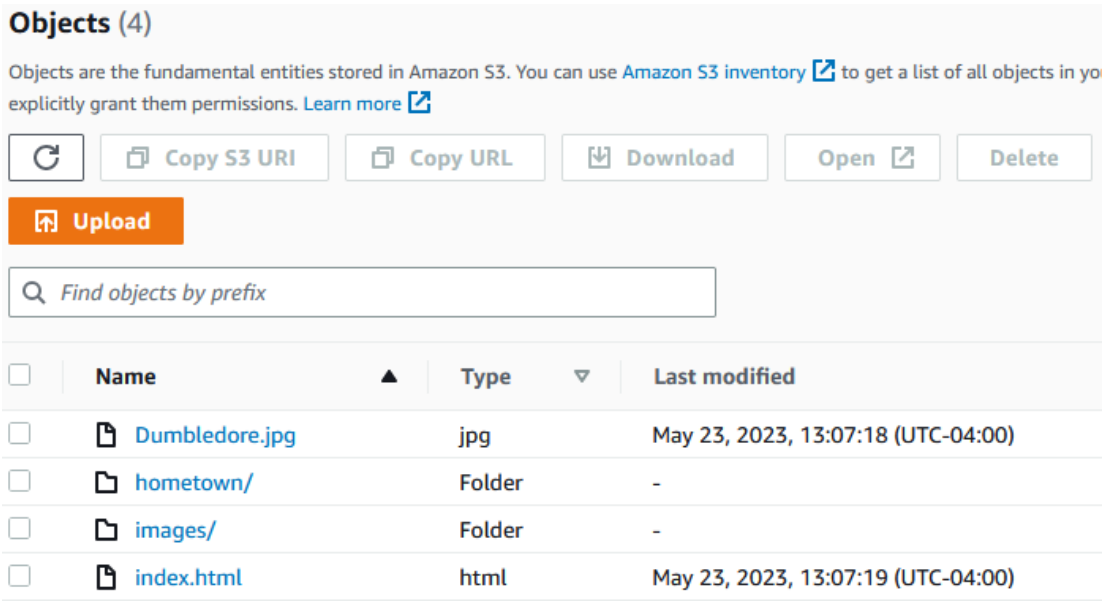
21. Click "Permissions".



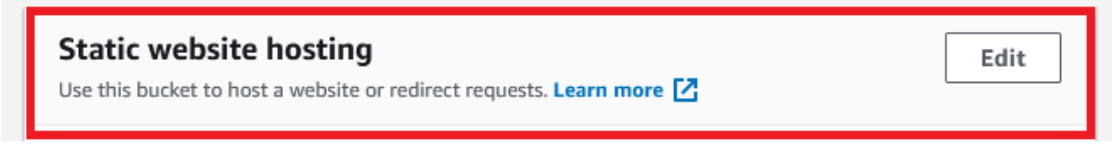
22. Select "Grant public-read access". Check "I understand the risk …". Click on "Upload".



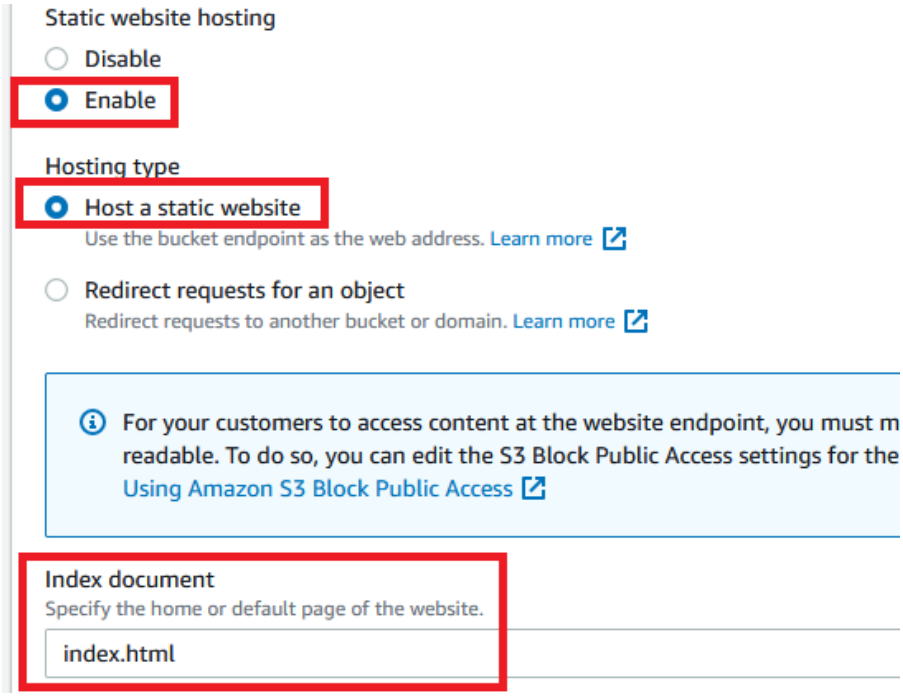
23. Click "Close" when upload is done. Your bucket might look like the following.



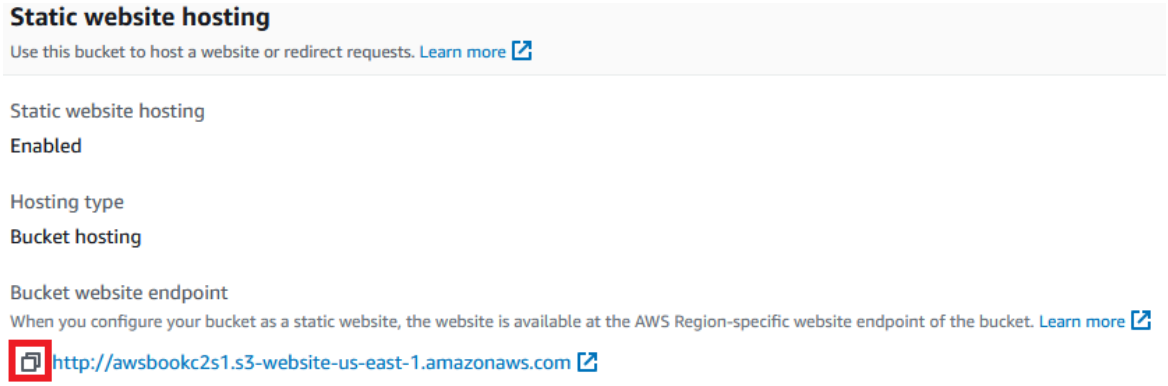
24. On you bucket page, click "Properties" tab, move all the way down. Click Edit "Static website hosting".



25. Select "Enable Static website hosting", "Host a static website". Enter index document "index.html". Click "**Save changes**".



26. Copy the website URL as shown below.

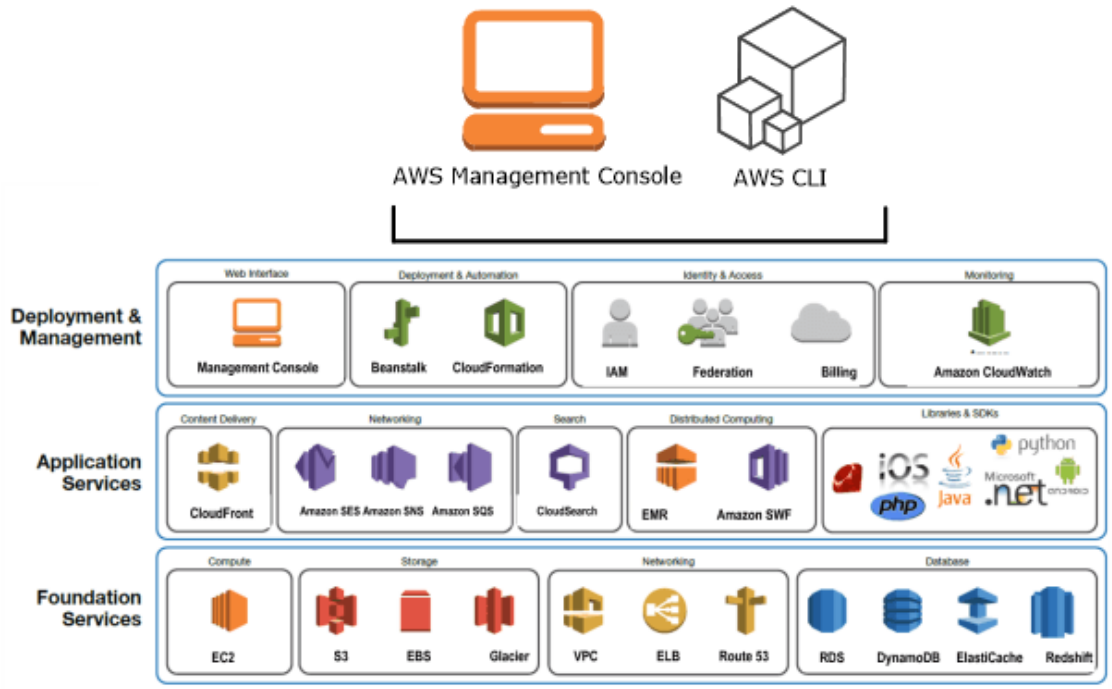


27. Paste your URL into a web browser. You can see the web pages.

2.2 Access S3 with CLI

Objectives:

- Download and install AWS CLI
- Access S3 with AWS CLI



Part 1. Install AWS CLI

1. Go to https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html.
2. Based on your operating systems, download AWS CLI version 2.
3. Install it.
4. If you are using Windows, you need to add the installation directory (C:\Program Files\Amazon\AWSCLIV2) to your PATH environment variable manually.
5. Type the following command to see if AWS CLI is installed correctly and you can access it.

```
C:\> aws --version
aws-cli/2.0.6 Python/3.7.4 Windows/10 botocore/2.0.0
```

Part 2. Configure AWS CLI

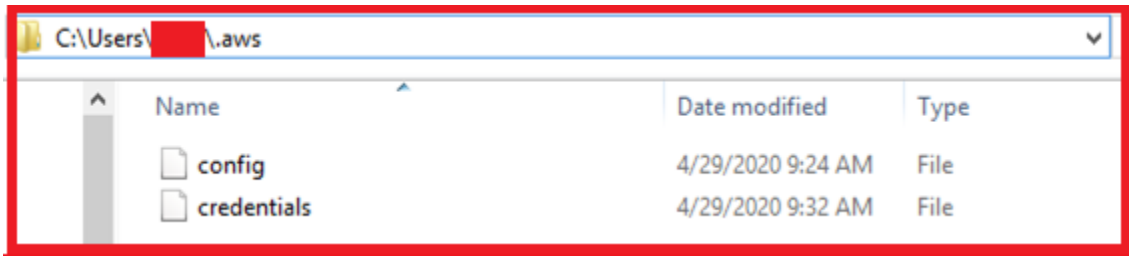
6. Execute the following command to configure AWS CLI.

aws configure

7. AWS Access KEY ID and AWS Secrete Access Key can be found following the instructions of Steps 10 and 11. Region can be found from AWS Console.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

8. Update configuration file, which can be found at C:\Users\yourname\.aws\config based on your AWS account location.
9. Open credentials, which can be found at C:\Users\yourname\.aws\credentials.



.

10. Log onto your AWS account. On this page, click "Account Details" or "AWS Details" button.
11. Copy the credentials and paste them into the file you just opened.

12. Save and close the file.

Part 3. Practice using AWS CLI

13. This website contains the reference to AWS CLI command:
    https://docs.aws.amazon.com/cli/latest/index.html
14. Since you have created S3 buckets, we can practice some commands, such as

```
aws s3 ls
aws s3 ls s3://bucket-name
```

15. Try other commands.
16. Note: the credentials you copied and pasted will expire after you are logged out of AWS. So, next time if you need to use AWS CLI, you need to copy new credentials from AWS and paste them into local file credential.

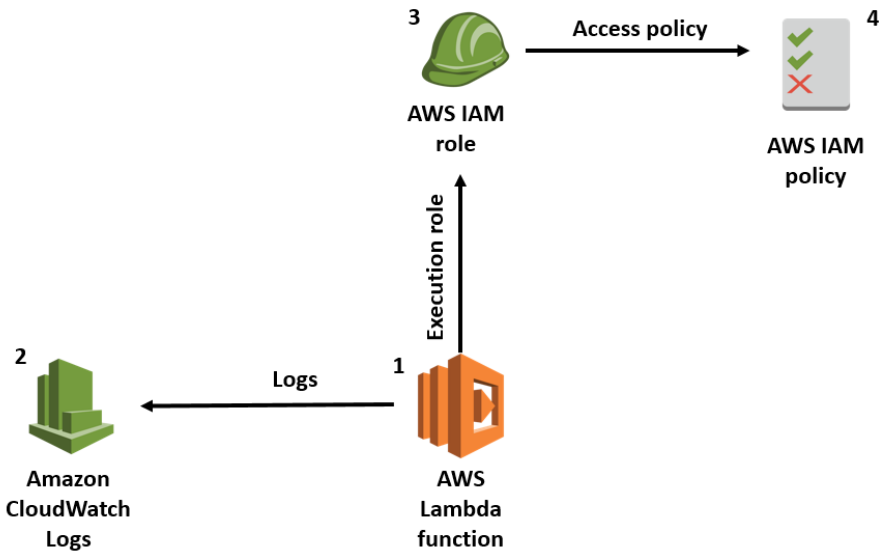
Chapter 3. Lambda Functions

Lambda function is AWS's solution to cloud-based function as a service (FAAS). It provides custom and flexible small scale reusable functionalities. In this chapter, we will practice creating and invoking Lambda functions, customizing its input source and output destinations.

3.1 Role, Policy, and Lambda Function

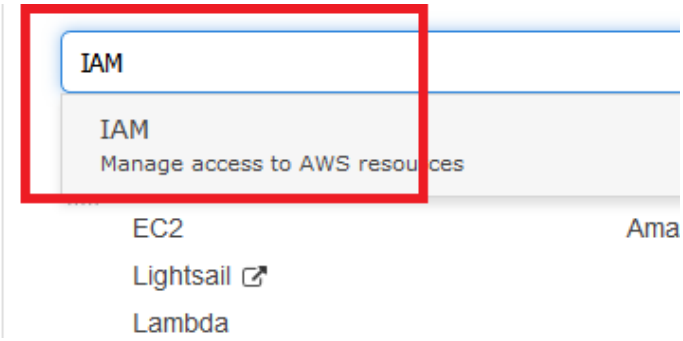
Objectives:

- Create an AWS Role
- Create an AWS Policy
- Create an AWS Lambda function



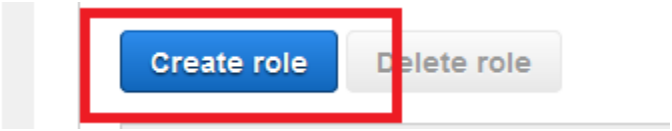
Part 1. Create an AWS role

1. Log onto Aws Console.
2. At your AWS Console home. Search "IAM", then click "IAM". You can also find it under "Security, Identity, & Compliance".

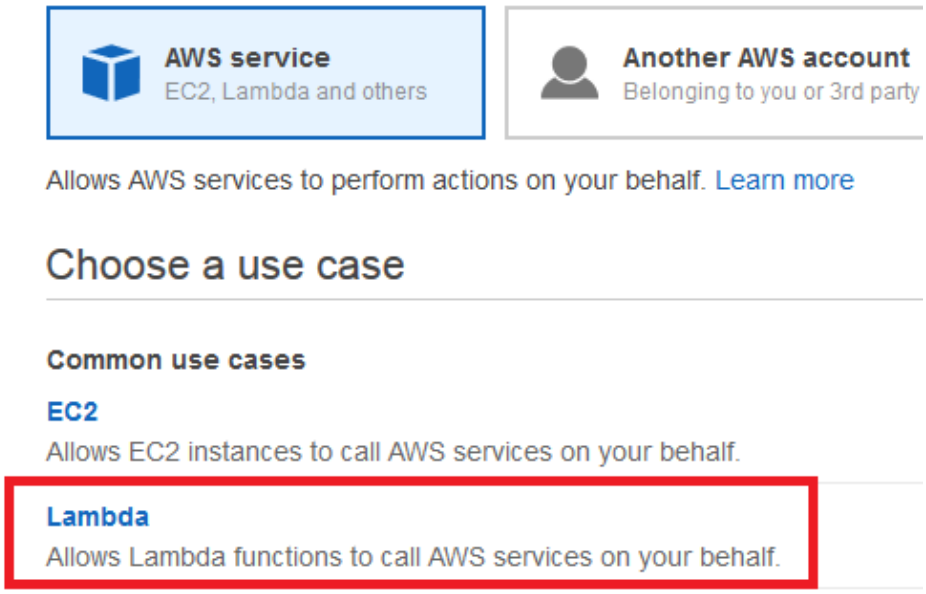


3. Click Roles.

4. Click “Create role”.

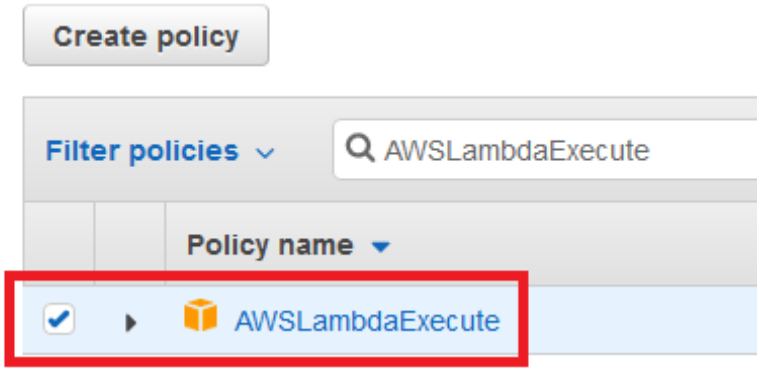


5. Under “AWS service”, click “Lambda”.

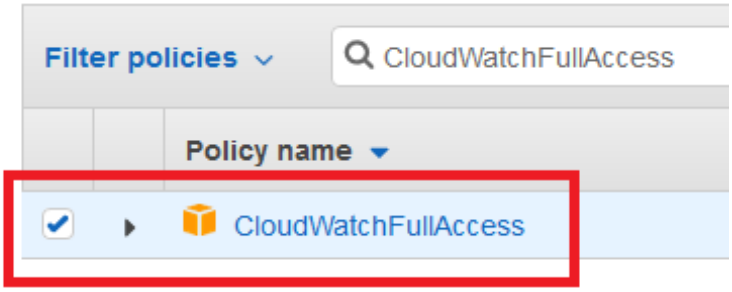


6. Click “Next”.

7. In the search box, search “AWSLambdaExecute” policy. Select the policy.

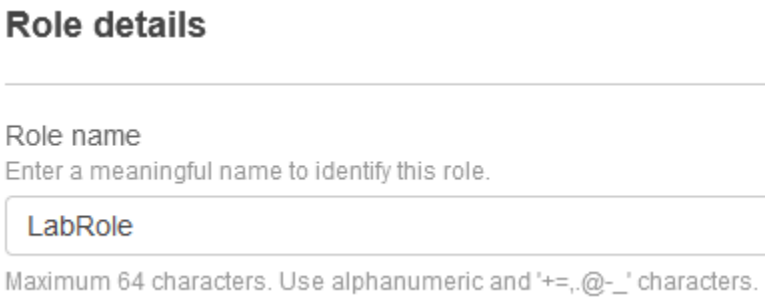


8. Search "**CloudWatchFullAccess**" and select it.

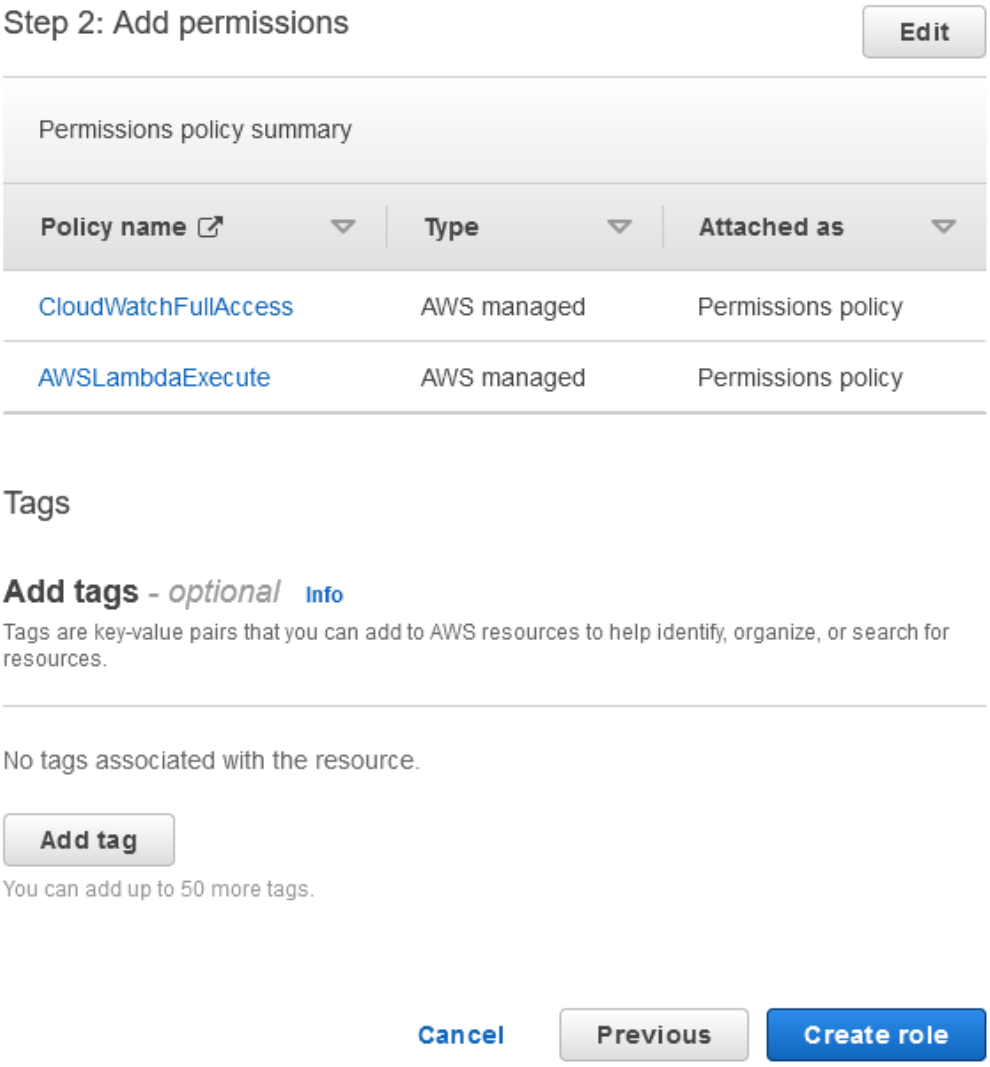


9. Click "Next".

10. Click "Next: Review".
11. Assign your **aws role a name**, **remember it**. We are going to use it many times.



12. Make sure this role is associated with two policies. Click "**Create role**".

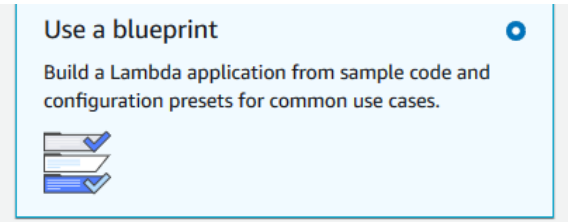


Part 2. Create a Lambda function

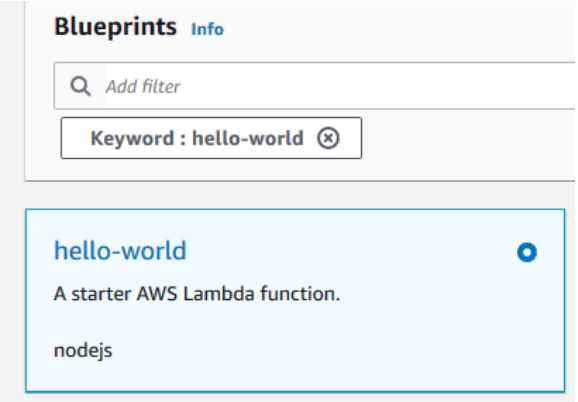
13. Go back to your AWS console page, click Lambda under Compute.

14. Click “Create function”.

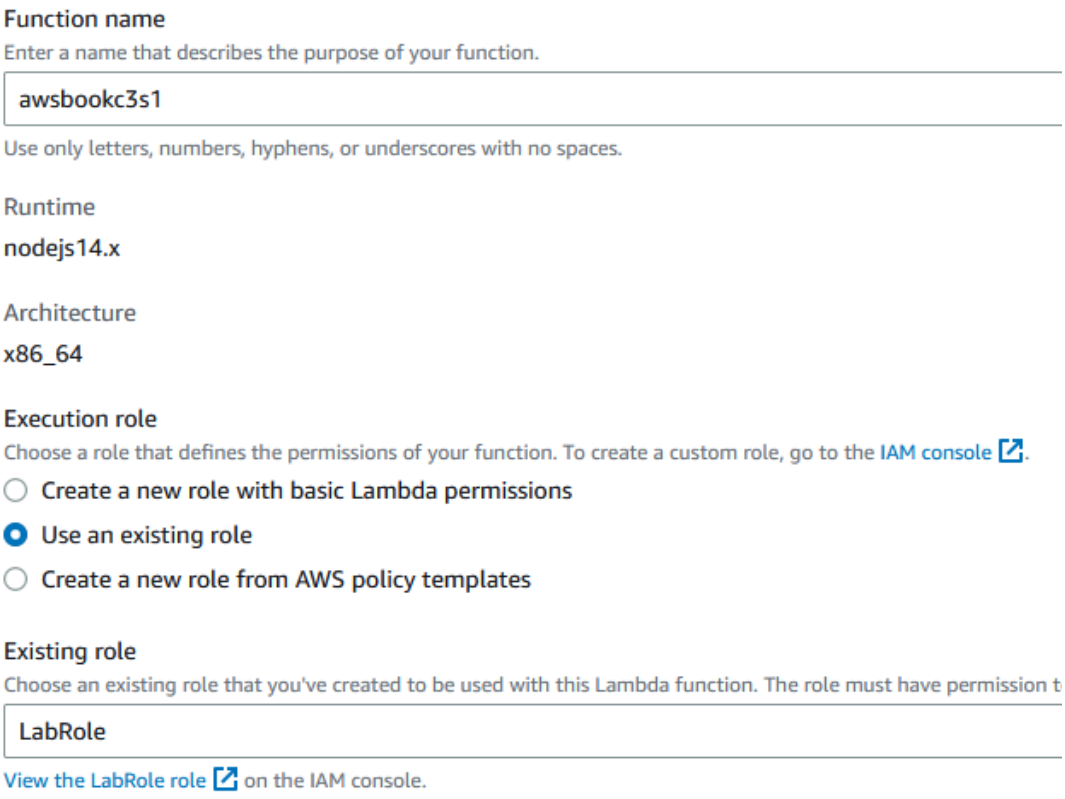
15. Select Blueprints.



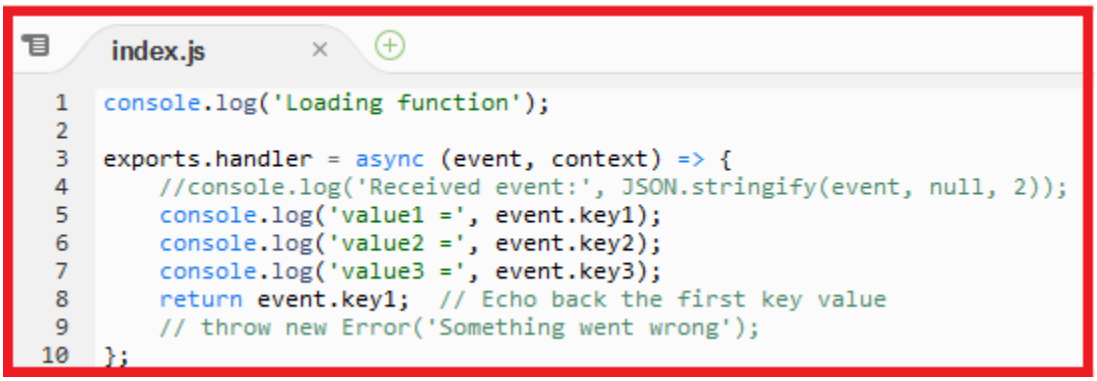
16. Search “hello-world” and select hello-world node.js. Click “Configure”.



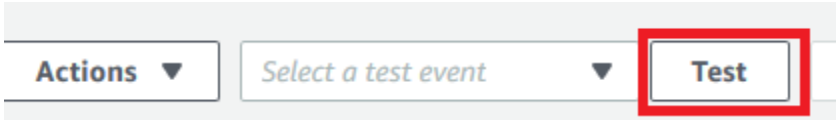
17. Assign a name. Choose “Use an **existing role**”. Pick the role you just created.



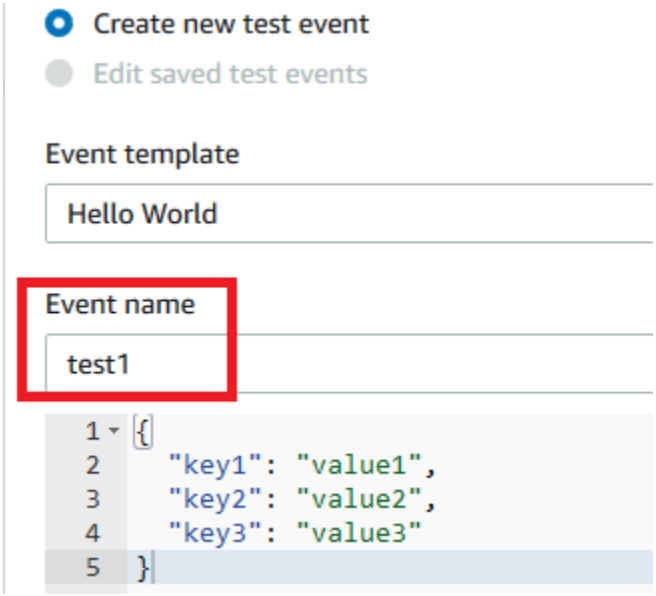
18. Click “Create function”.
19. Examine your code index.js. It takes three parameters from an HTTP call.



20. Click “Test”.

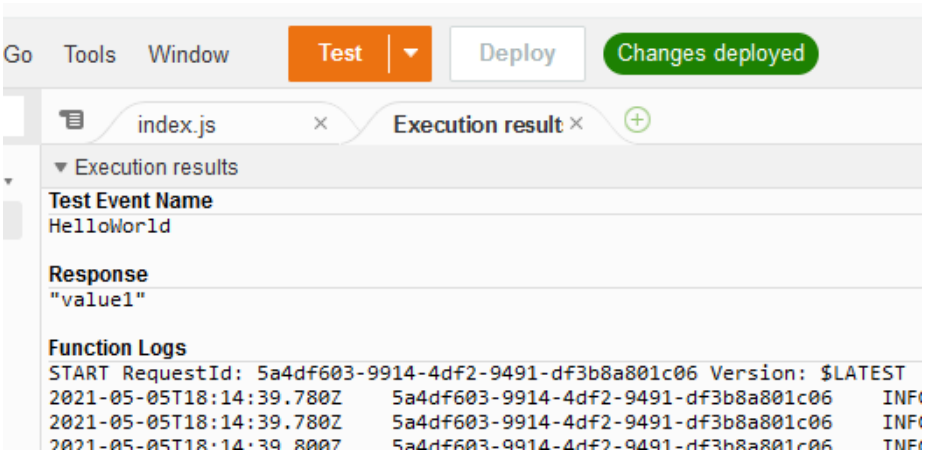


21. Assign a test Event name.

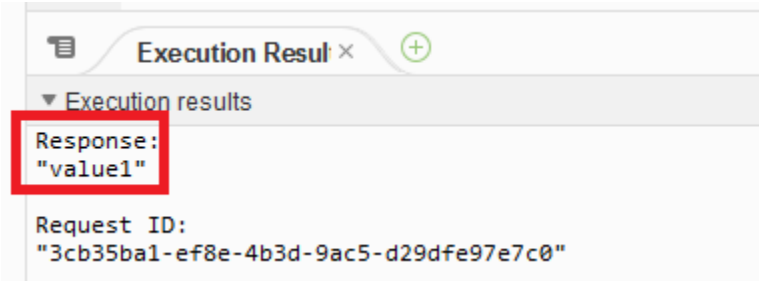


22. Click “Save”.

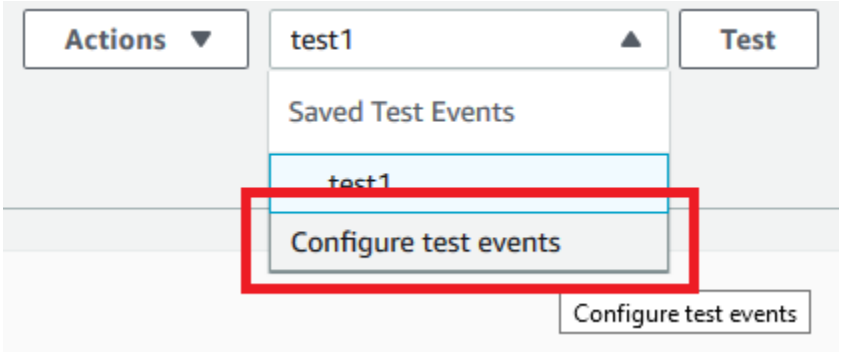
23. When your test event is selected, Click “Test”.



24. You can check the response under “Execution Result”.



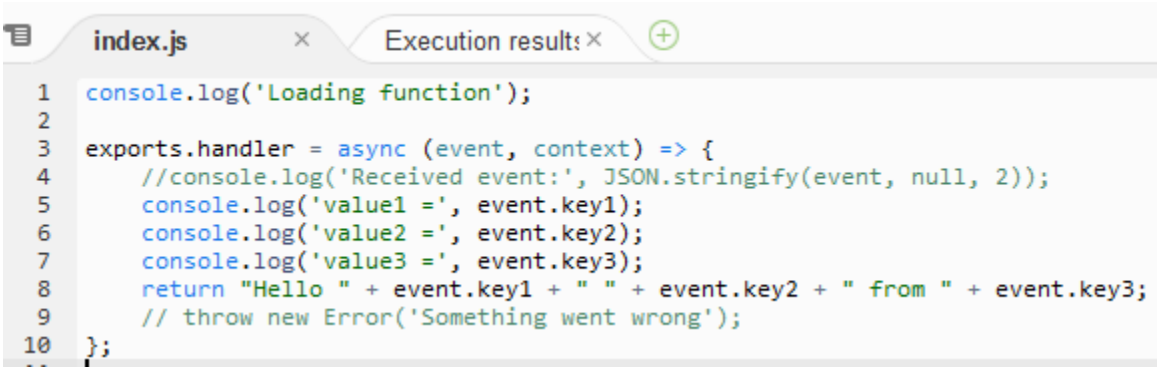
25. Now. Configure your test event.



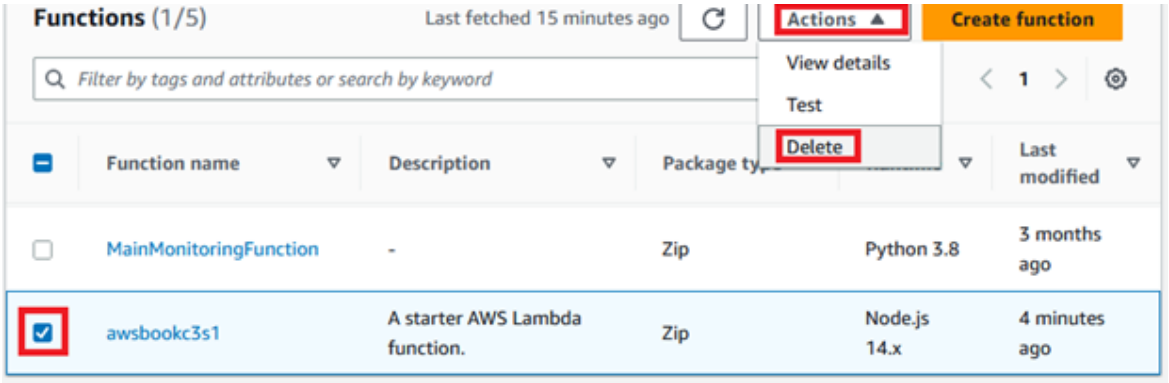
26. Modify the testing case so that Key1 contains first name of a person, key2 contains last name of a person, and key3 contains location of a person.

27. Click “Save”.
28. Modify the **index.js** code so that it will display the result below when it is invoked.

    Hello <Name> of <location>



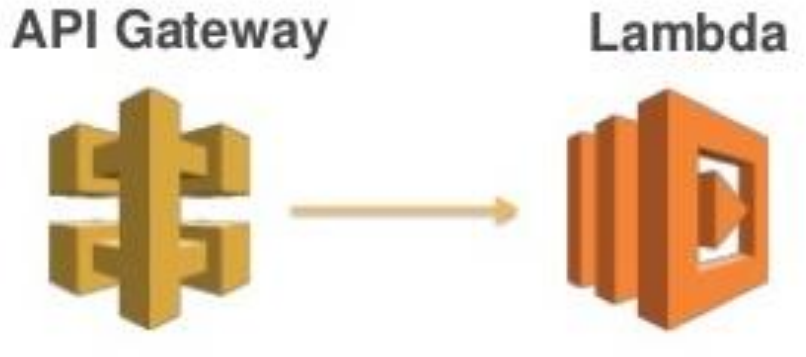
29. After changes made to index.js, the changes should be **deployed** (saved on the cloud).
30. Test your lambda function again and screenshot the output.
31. To delete your lambda function: From Function homepage, select the function, click “Actions”, then “Delete”.



3.2 API Gateway

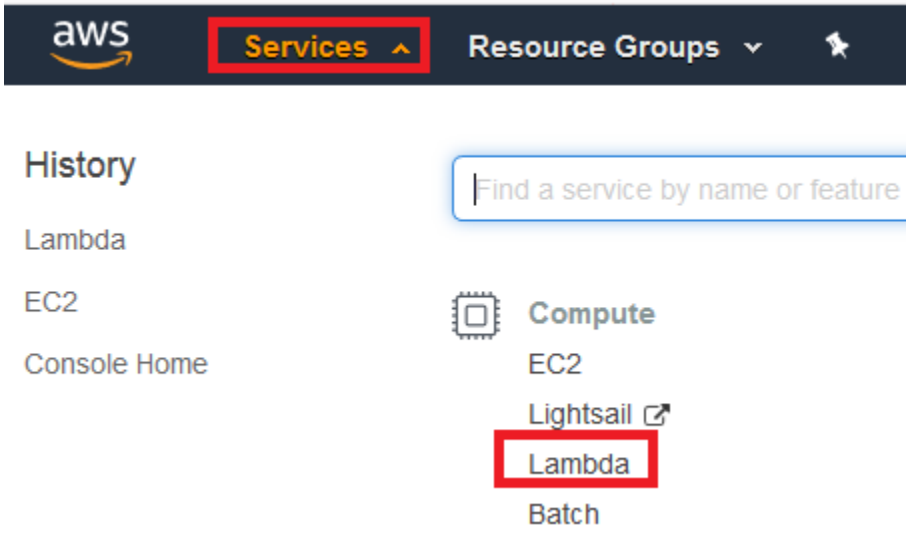
Objectives:

- Add API Gateway Triggers to Lambda function
- Test Lambda functions through API Gateway



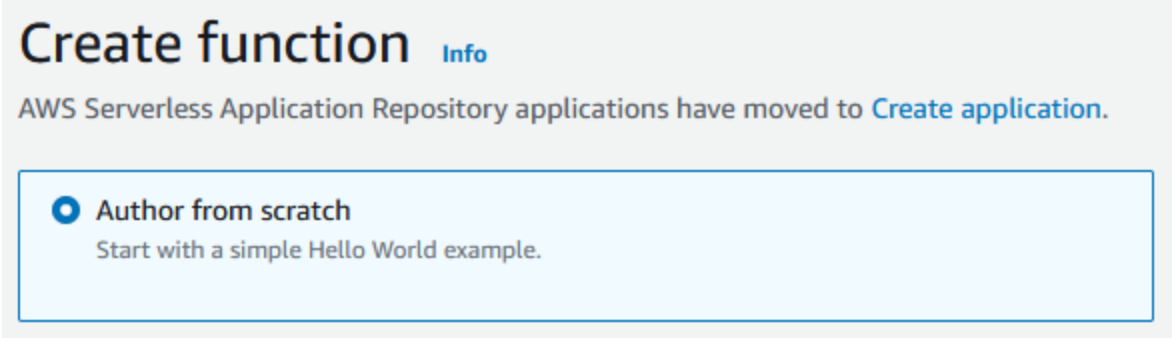
Part 1. Create a Lambda function and add a trigger

1. Go to your AWS Console page, click Services, then "Lambda" under Compute.

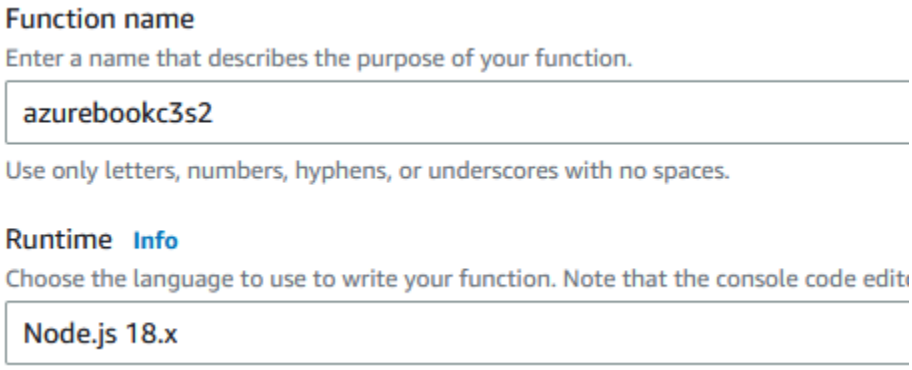


2. Click "Create function".

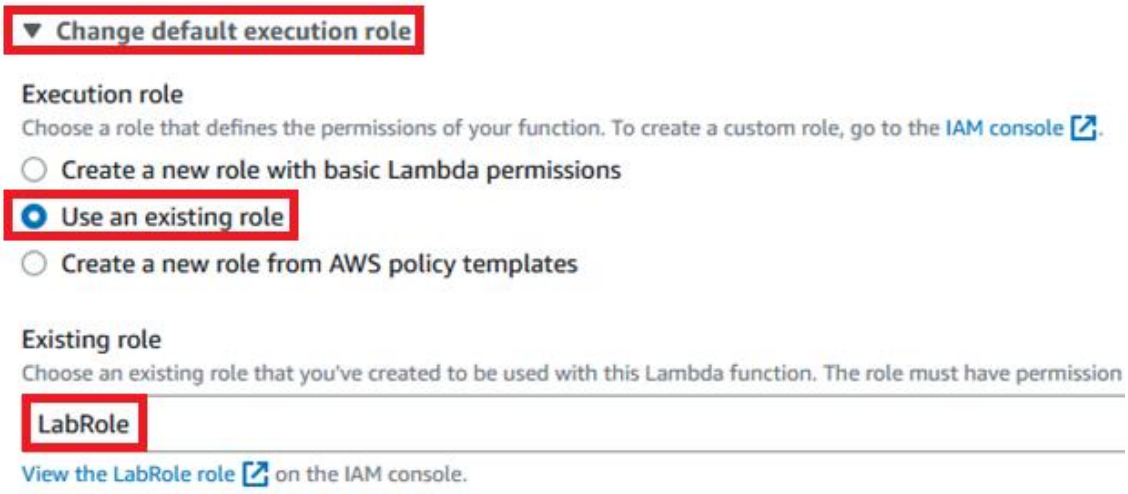
3. Choose "**Author from scratch**".



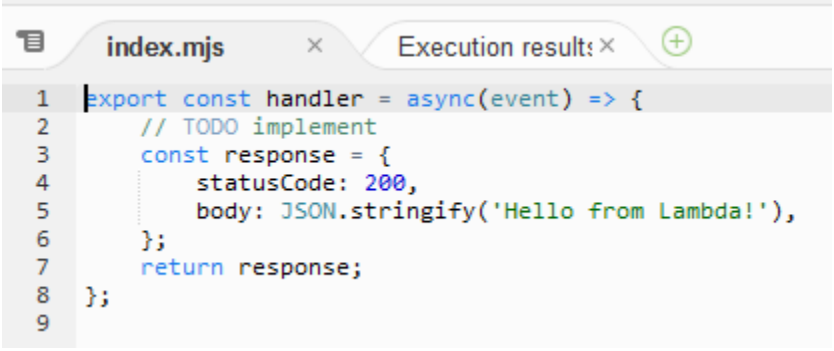
4. Assign a name.



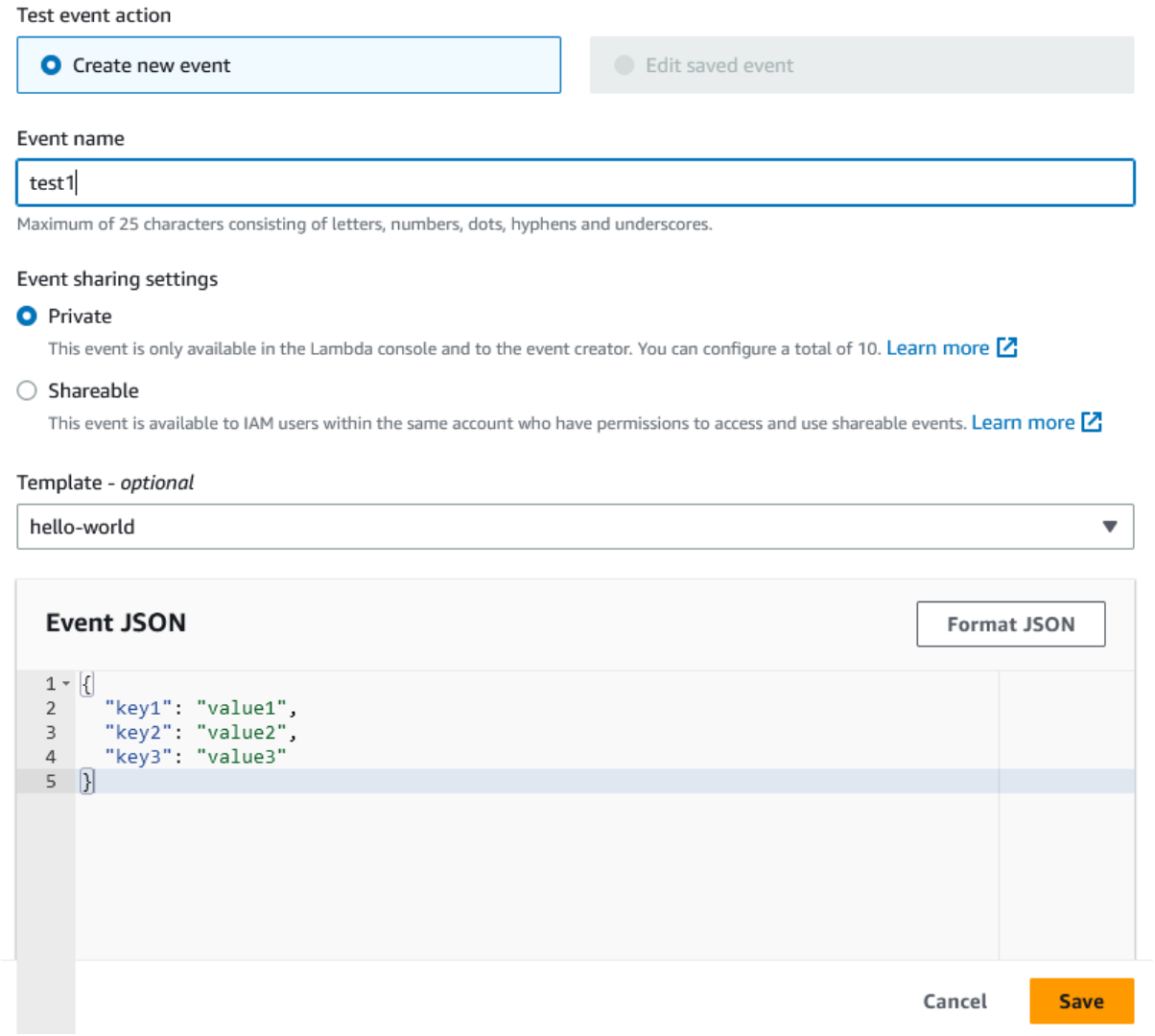
5. Expand “Change default execution role”, Choose “Use an existing role”. Pick the role you e created in Section 3.1.



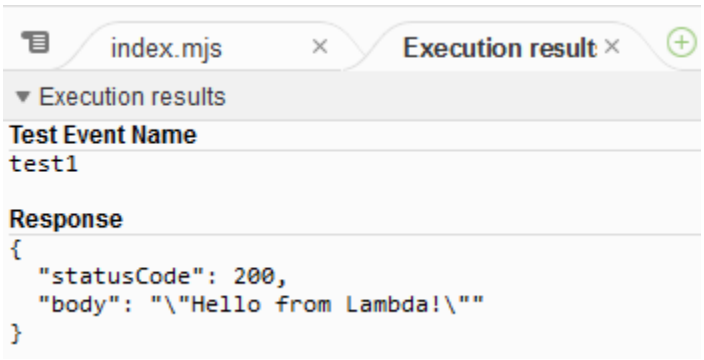
6. Click “Create function”.
7. Examine your code index.mjs.



8. Let's create a test event and save it.

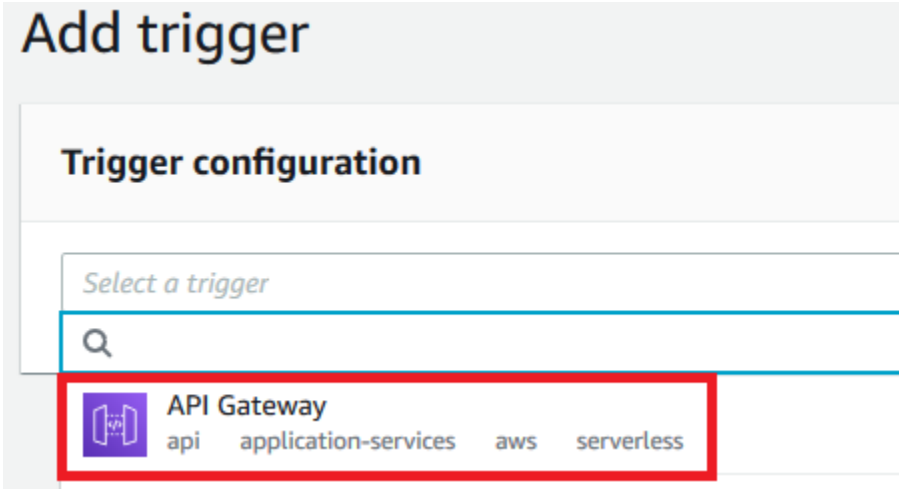


9. Let's execute the testing case: test1, and you should see the following execution result.

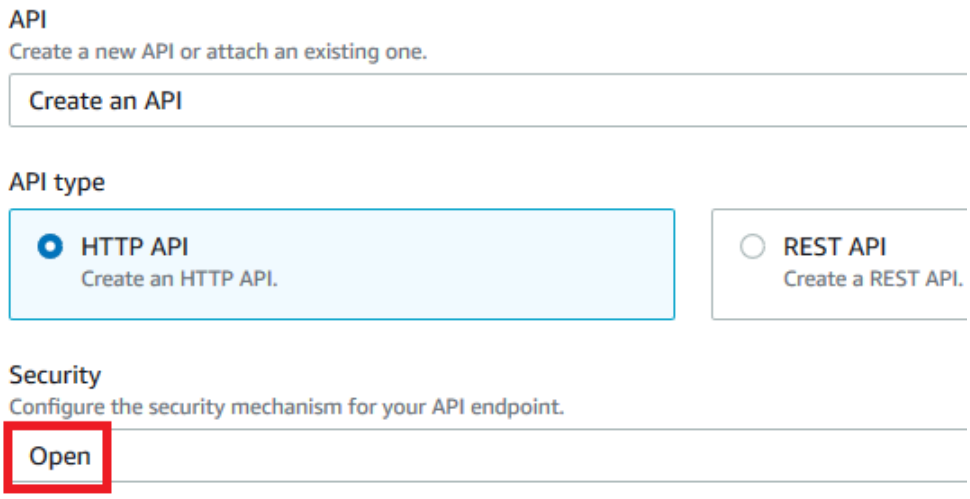


10. Click "+ Add trigger".

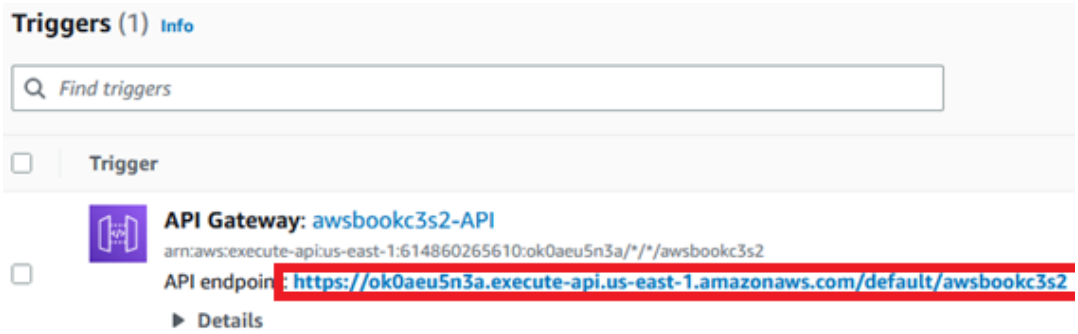
11. Select "API Gateway".



12. Select "Create a new API". API type should be "HTTP API", Security should be "Open". Click "Add".



13. The trigger is added.
14. Click on the API endpoint (URL) to test your lambda function.



15. You should see the following message.

```
"Hello from Lambda!"
```

16. Rewrite **index.mjs** as the following to process query strings.

```
export const handler = async(event) => {
    // TODO implement
    let result = {
        'First name': event.queryStringParameters.key1,
        'Last name': event.queryStringParameters.key2,
        'Location': event.queryStringParameters.key3
    };

    const response = {
        statusCode: 200,
        body: JSON.stringify(result),
    };
    return response;
};
```

17. Deploy the changes.
18. Let's add a query string to the end of the API endpoint URL (Step 14).

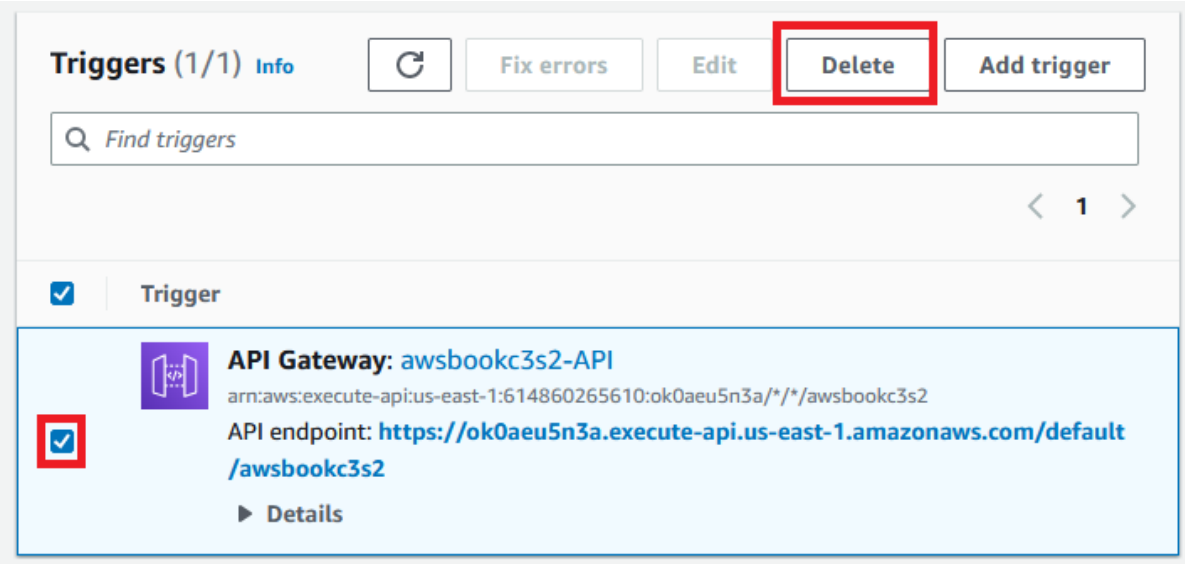
    ?key1=FirstName&key2=LastName&key3=South Bend

19. You will see the following response.

```
{"First name":"FirstName","Last name":"Lastname","Location":"South Bend"}
```

Part 2. Create another API Gateway trigger

20. Delete your HTTP API Gateway.



21. Repeat Steps in Part 1 to add a new API Gateway trigger. This time let's choose REST API (**Step 22**).

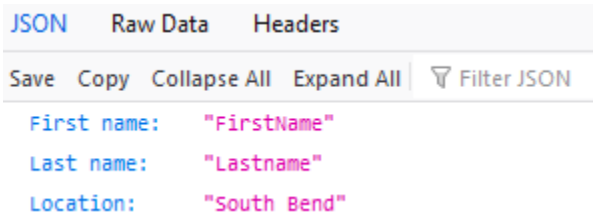
22. File index.mjs is not changed as shown below.

```
export const handler = async(event) => {
    // TODO implement
    let result = {
        'First name': event.queryStringParameters.key1,
        'Last name': event.queryStringParameters.key2,
        'Location': event.queryStringParameters.key3
    };

    const response = {
        statusCode: 200,
        body: JSON.stringify(result),
    };
    return response;
};
```

23. Click on the API endpoint (URL) to test your lambda function with the following same query string.

?key1=FirstName&key2=LastName&key3=South Bend

24. You will get the following response message.



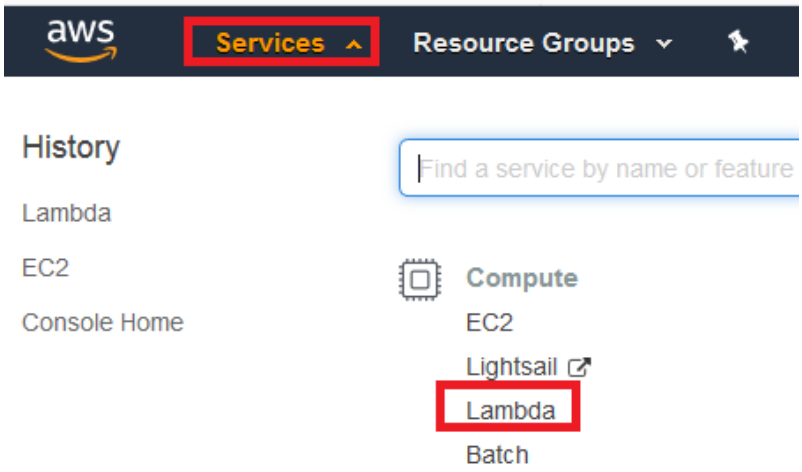
3.3 Trigger and Destination

Objectives:

- Create an AWS Lambda function
- Add triggers to Lambda function
- Add destinations to Lambda function
- Retrieve event information

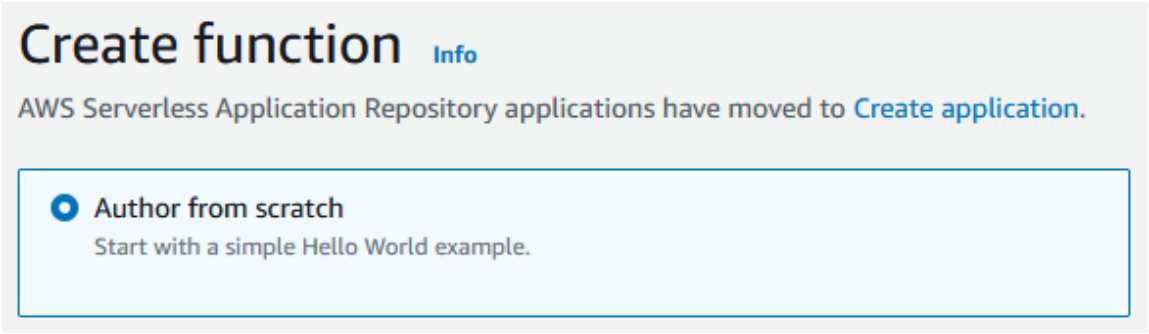
Part 1. Create a Lambda function

1. Log onto AWS Console home. Click Services, then "Lambda".

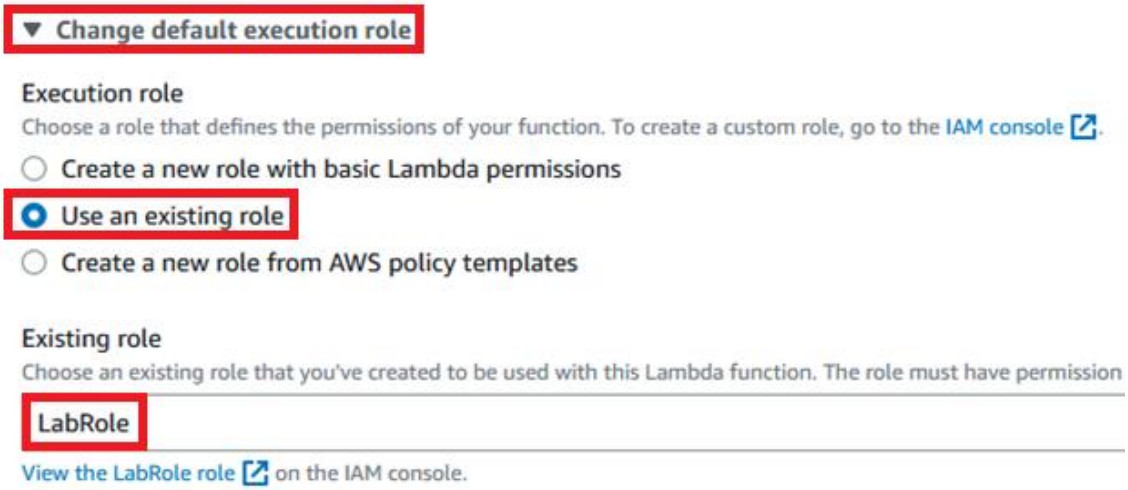


2. Click on "Create function".

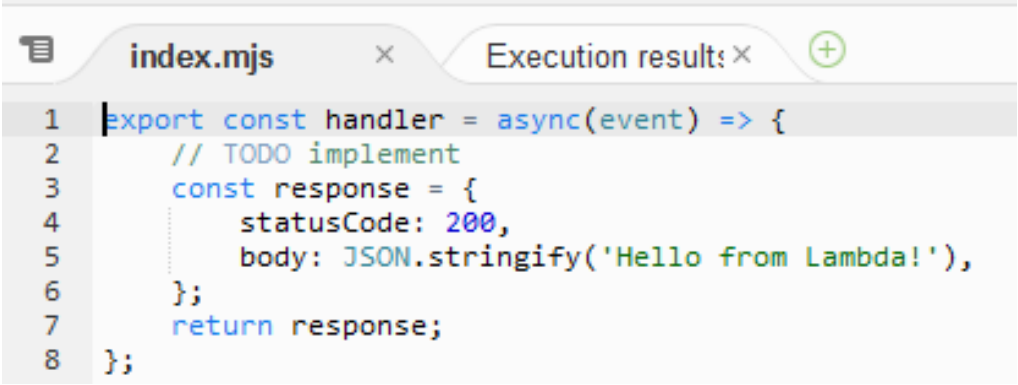
3. Choose "**Author from scratch**".



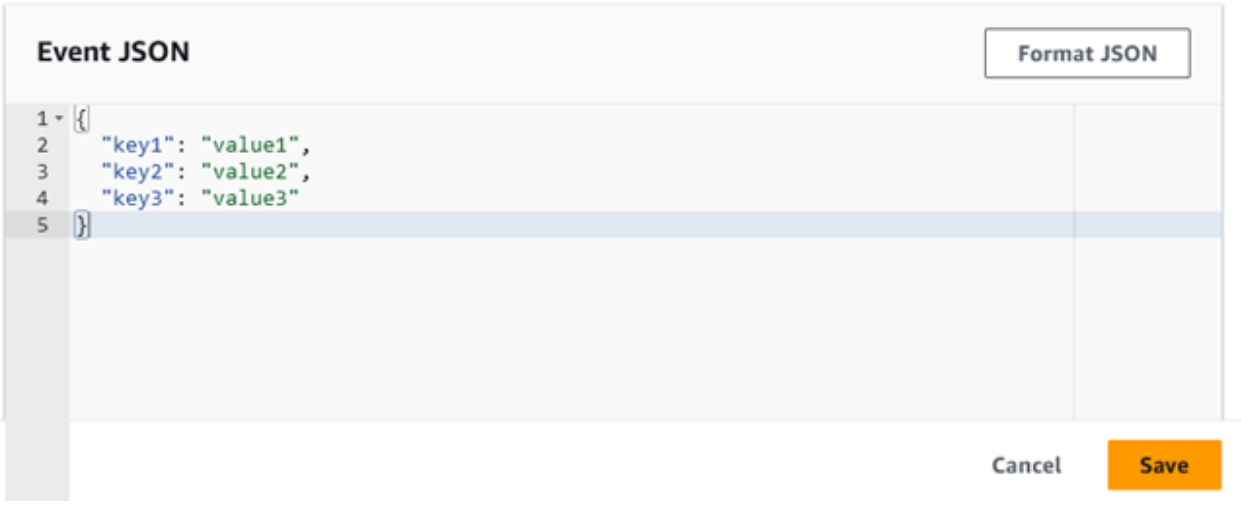
4. Assign a name.
5. Expand "Change default execution role", Choose "Use an existing role". Pick LabRole.



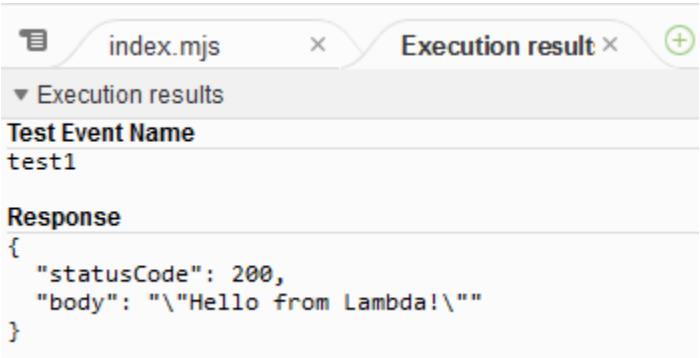
6. Click “Create function”.
7. Examine your code index.mjs.



8. Let’s create a test event and save it.

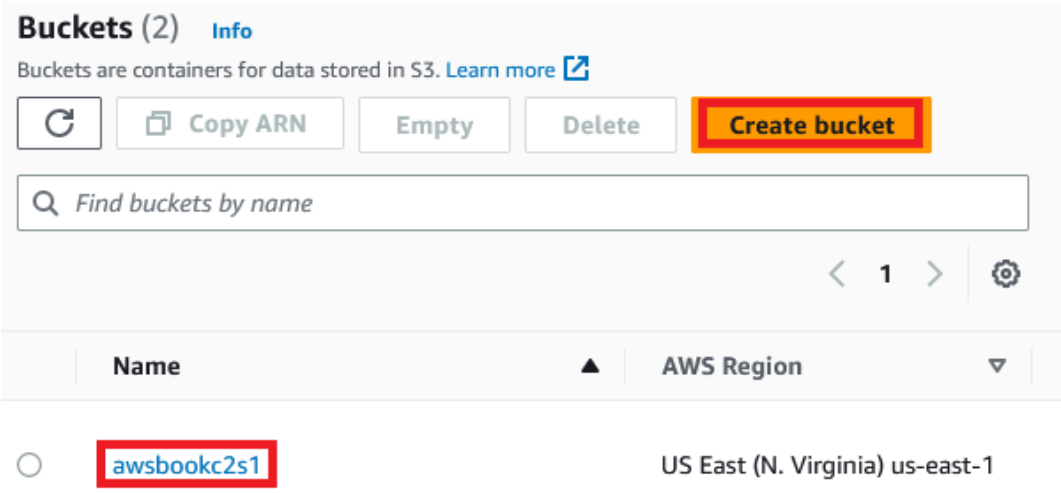


9. Let’s execute the testing case. You should see the following execution result.

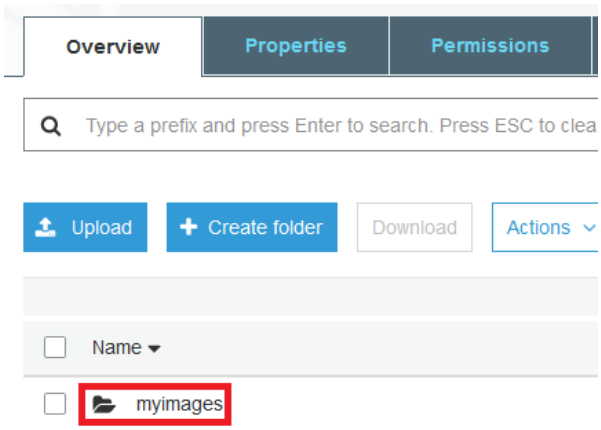


Part 2. Create a folder in your S3 bucket

10. Go to your Amazon S3 homepage. Click on the bucket we created before. Or you can create a new bucket.

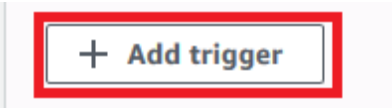


11. Create a new folder inside the bucket to hold images. I call it “myimages”.

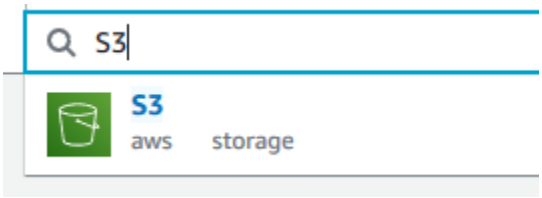


Part 3. Add a trigger to the Lambda function

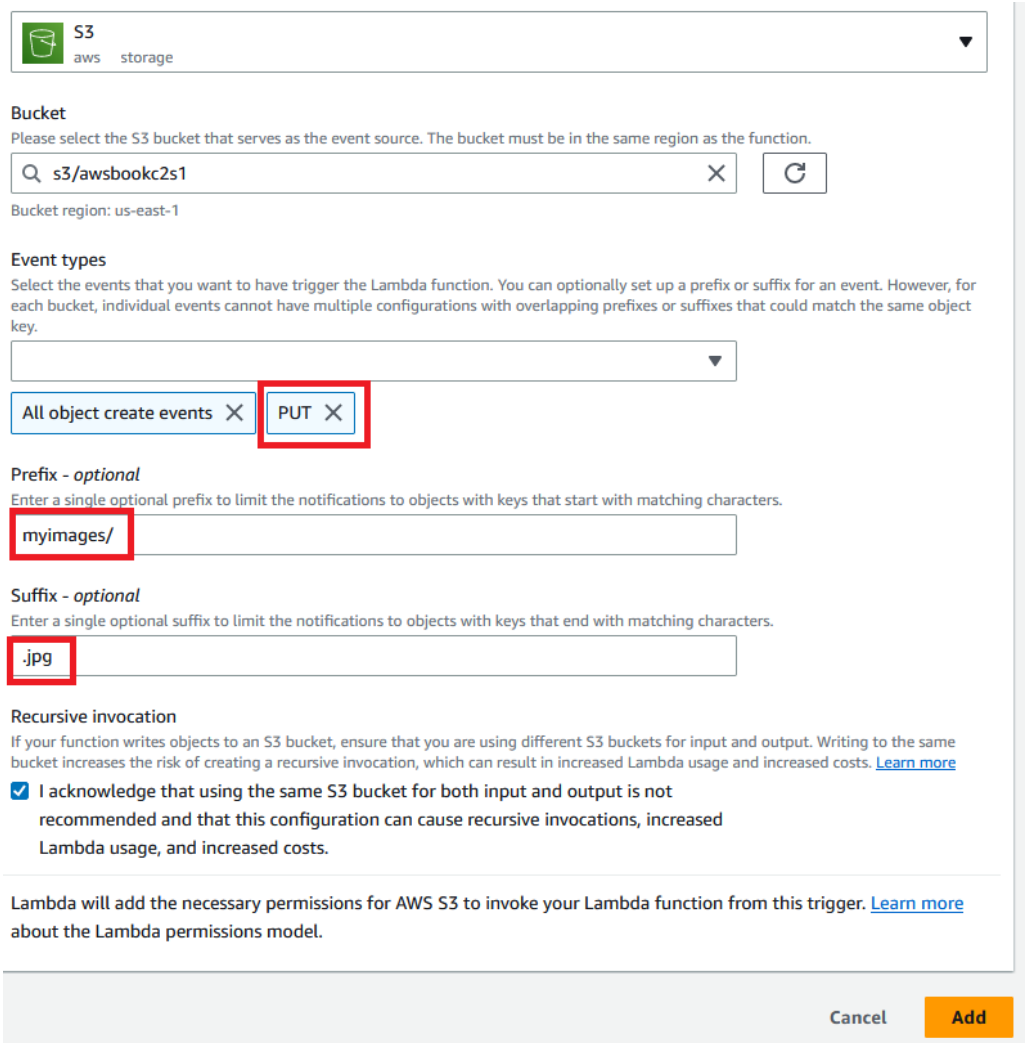
12. Go back to your Lambda function. Click “Add trigger”.



13. Select S3.



14. Select the bucket. Specify Event type as “PUT”. Specify folder name (prefix) and file suffix.



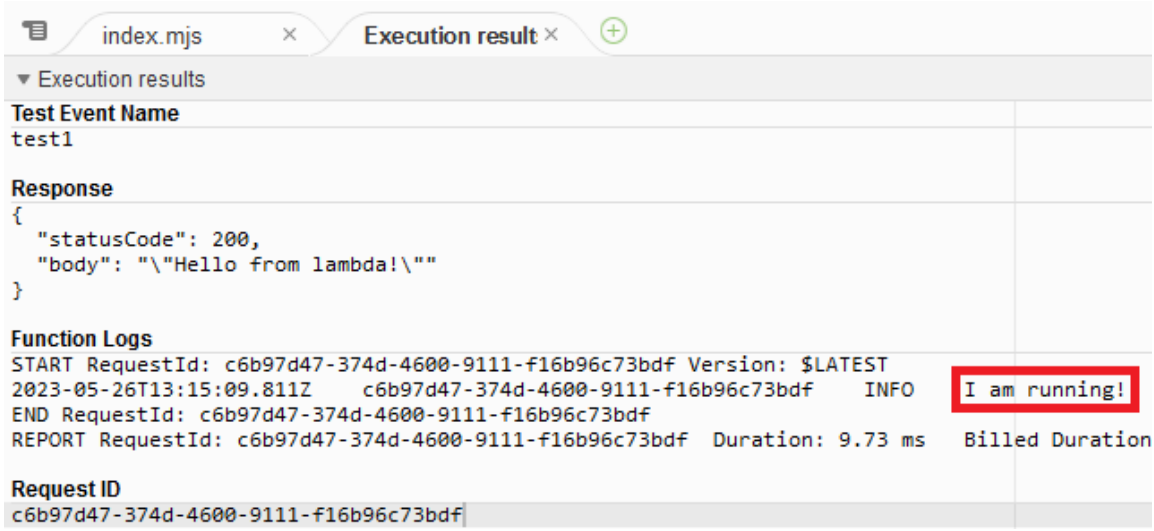
15. Check "I acknowledge that …". Click "Add".

Part 4. Test the trigger

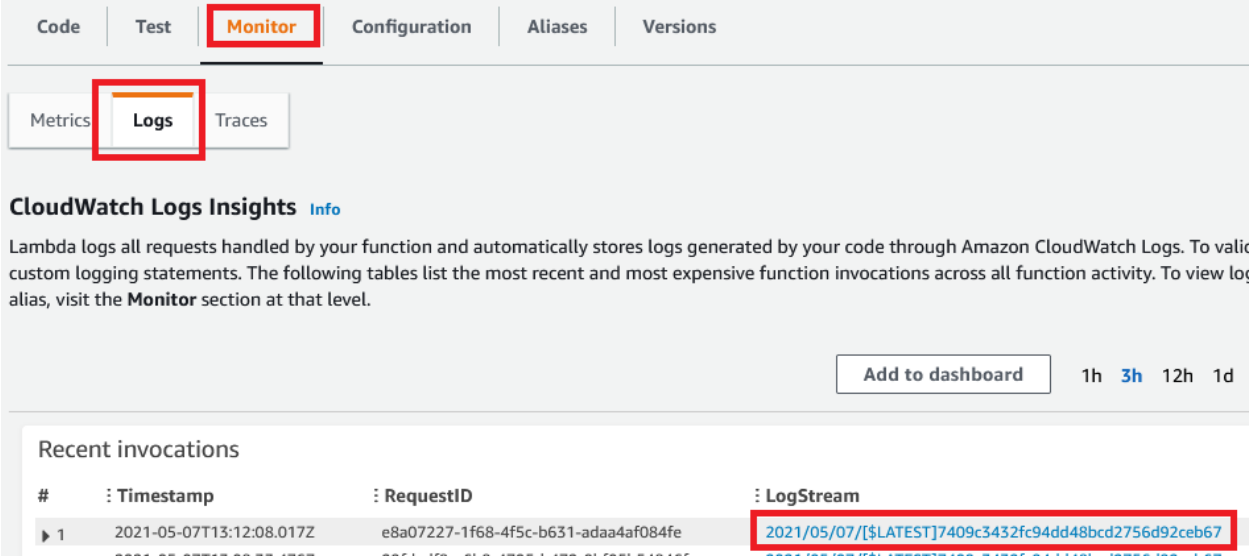
16. Go back to your AWS S3 page, upload an image (.jpg) into the folder you just created.
17. Check your Lambda function's response.
18. See nothing? That is OK.
19. Let's add a line code (#line 7) to index.mjs. We can check the log to see if this function is triggered.
20. Deploy the changes.

```
1  exports.handler = async (event) => {
2      // TODO implement
3      const response = {
4          statusCode: 200,
5          body: JSON.stringify('Hello from Lambda!'),
6      };
7      console.log("I am running!");
8      return response;
9  };
```

21. First, let's run the testing case locally. Click test and you should see the following response.



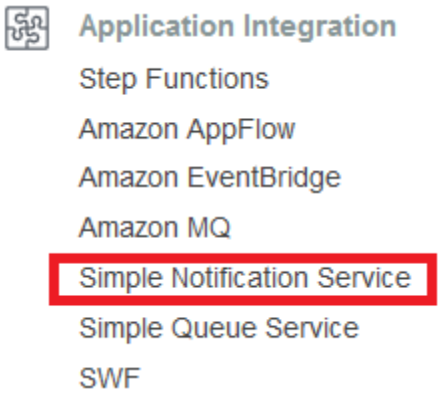
22. Now, let's test it through S3 trigger.
23. Upload another .jpg file onto your S3 bucket.
24. On your function page, click on "Monitor", then "Logs". Then, click on the top (latest) log entry.



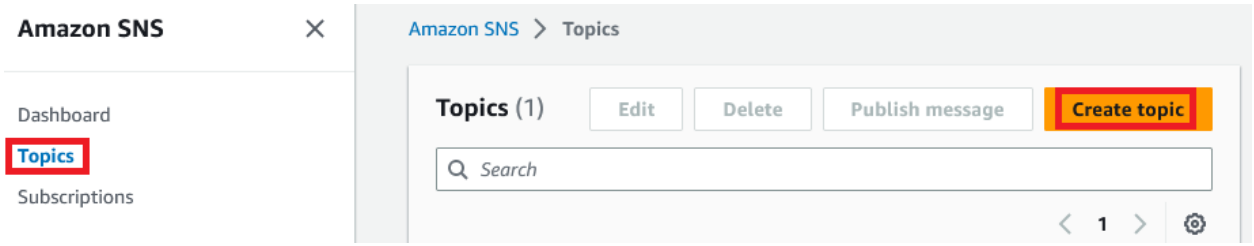
25. You can see your message is logged. (**It might take several minutes to see the result**).

Part 5. Add a destination to Lambda function

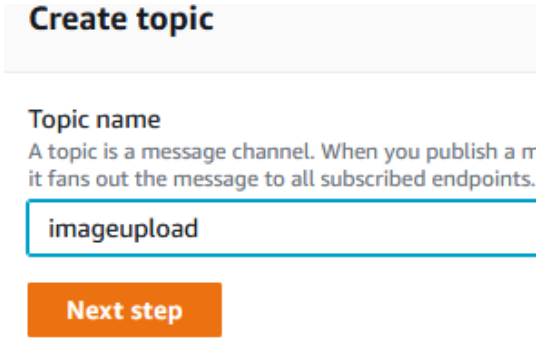
26. Let's first create an SNS topic.
27. On AWS Services page, select "Simple Notification Service" under "Application Integration".



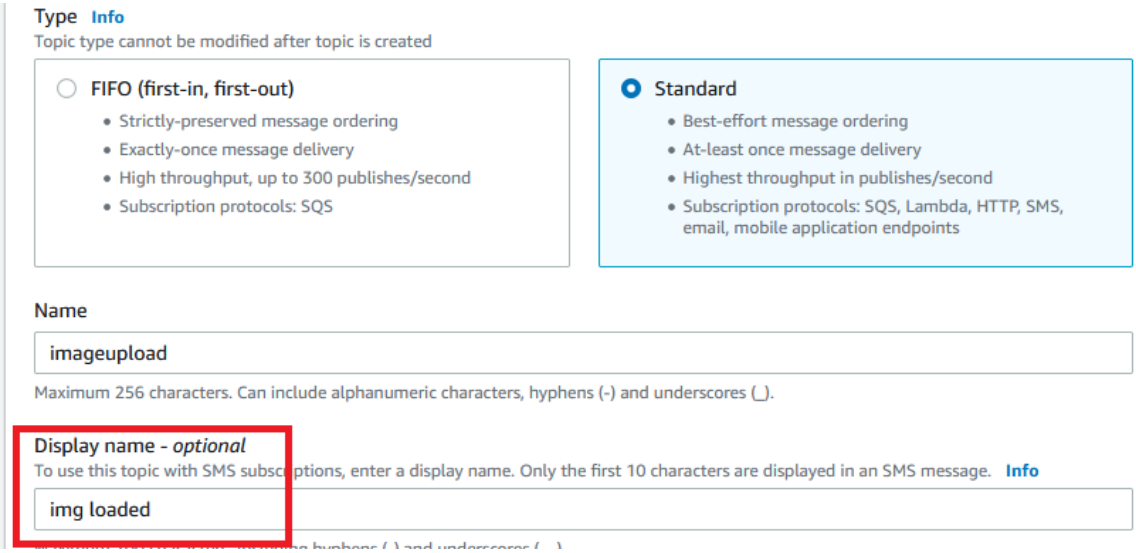
28. Click “Topics”, then “Create topic”.



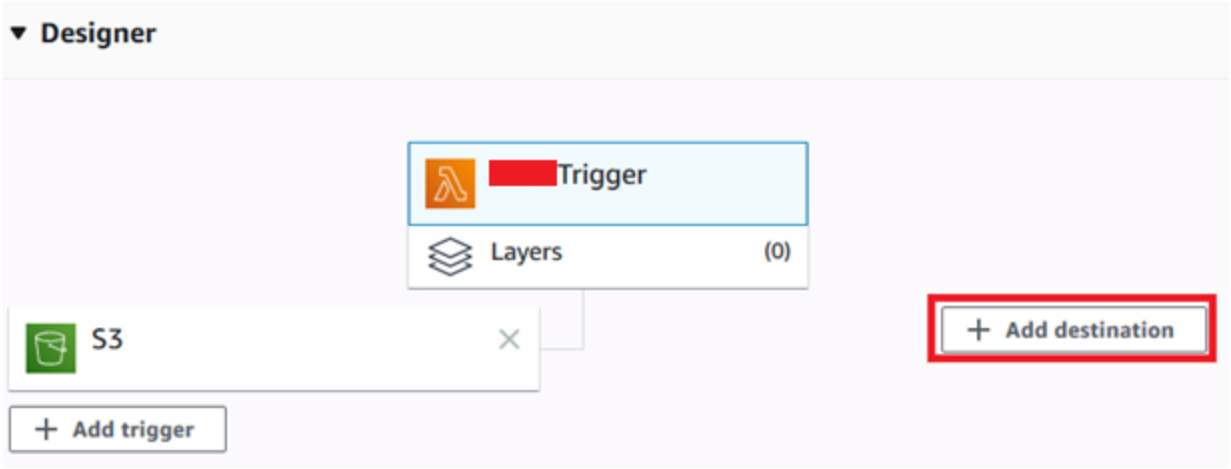
29. Assign a topic name.



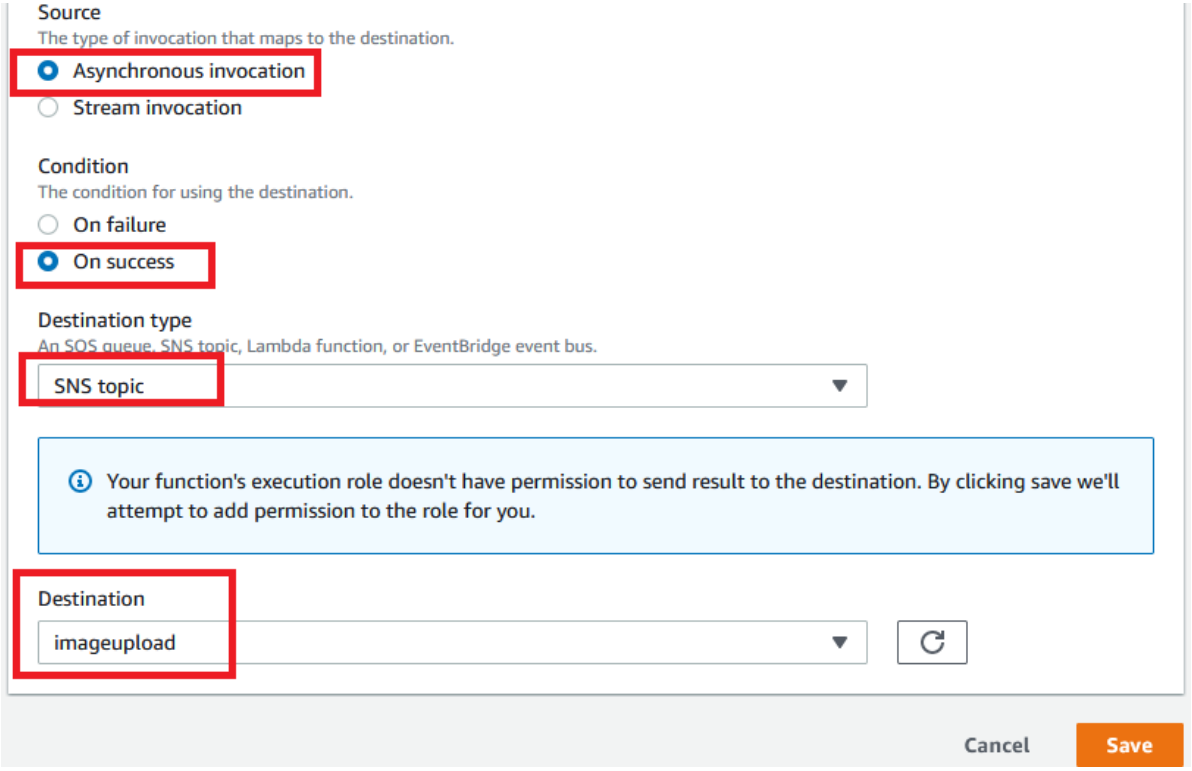
30. Assign a “Display name”. Click on “**Create topic**”.



31. Go back to your lambda function page, click “Add destination”.

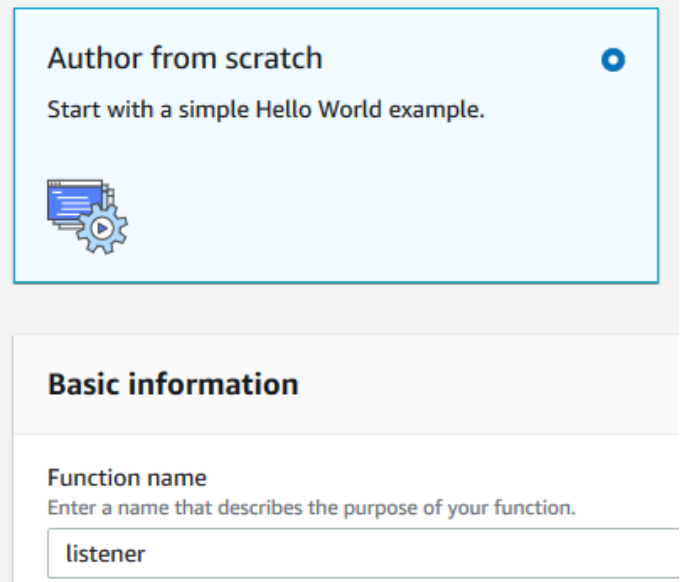


32. Specify Source, Condition, Destination type, and the SNS topic you just created. Click on "Save".



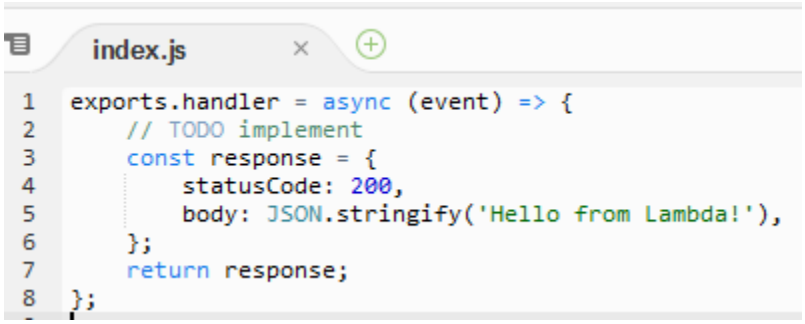
Part 6. Add a Lambda function listening to SNS

33. On your Lambda homepage, click on "Create function" to create a **new** lambda function.
34. Let's choose "Author from scratch". Assign a name to your function. Pick "Node.js".

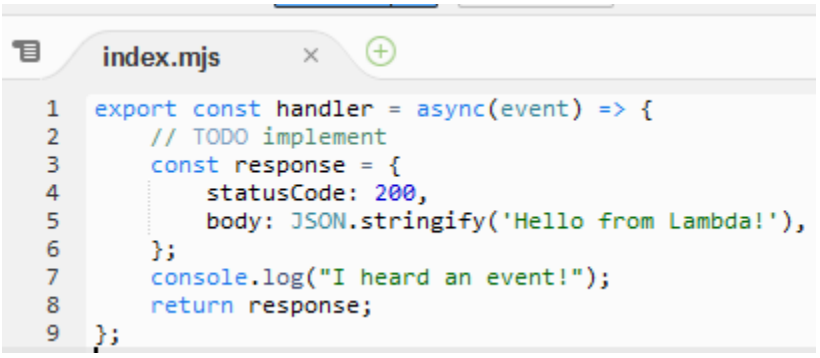


35. Click on “Change default execution role”.
36. Pick “Use an existing role”.
37. Choose the role “LabRole”.
38. Click “Create function”.

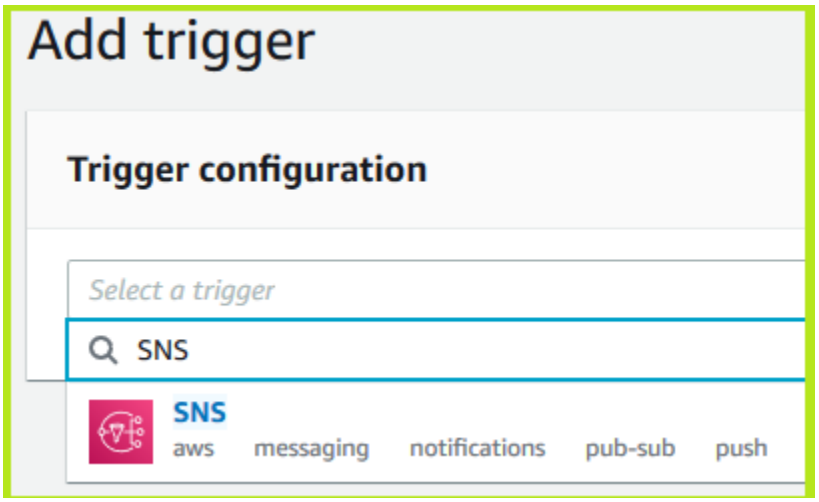
39. This is the code we have now.



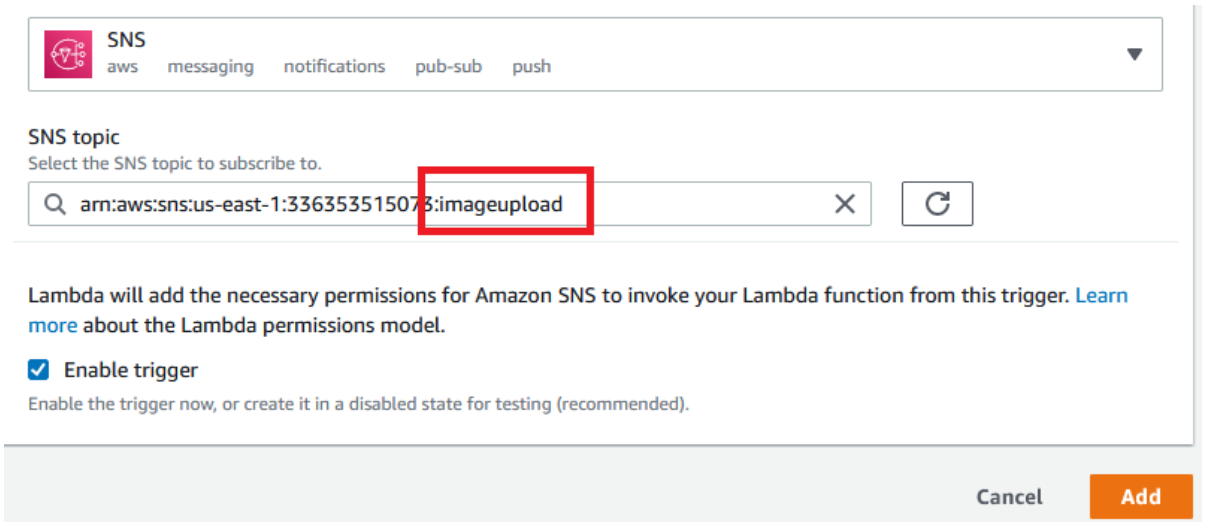
40. Modify the code to add a new line (#7). Deploy the changes.



41. Add a trigger to this function. Let's select SNS.

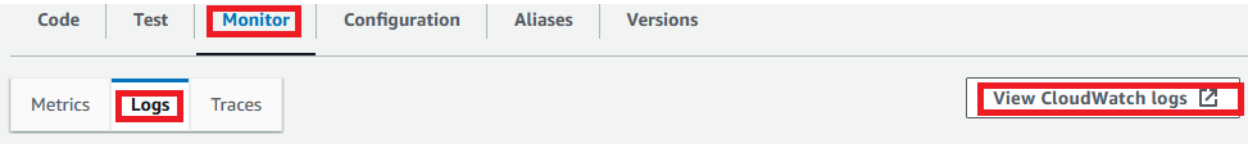


42. Pick the SNS topic we just created. Click on "Add".

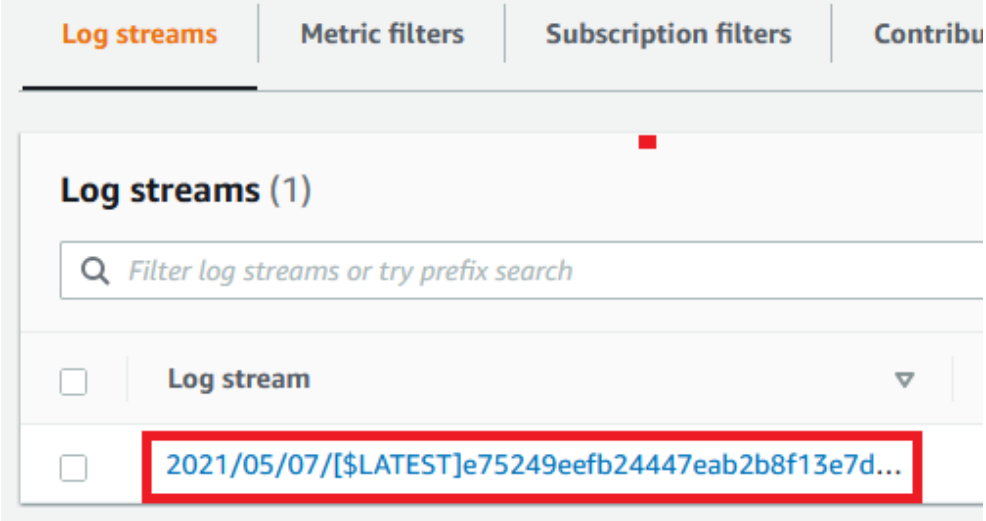


Part 7. Test the integration

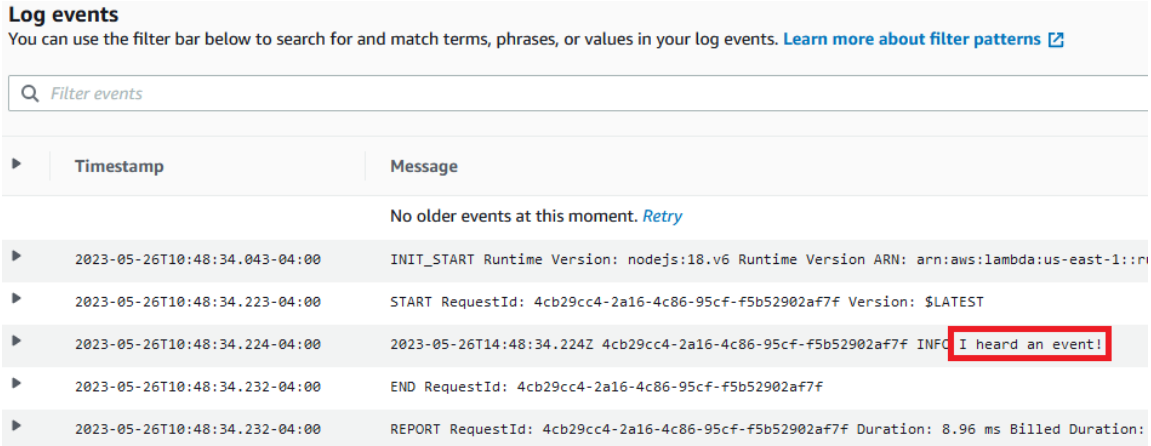
43. Upload a new image to your S3 bucket.
44. On your **"Listener"** lambda function page, click "Monitor", "Logs, then "View logs in CloudWatch".



45. Click on the top log entry.



46. You can see the log of your “Listener” function.



Part 8. Retrieve “event” parameter information

47. On your first lambda function, modify the response body and add a new line of code (#12) to display parameter event information.

```
1  exports.handler = async (event) => {
2      // TODO implement
3      const response = {
4          statusCode: 200,
5          body: {
6              id: 1001,
7              name: 'Jane Smith',
8              phone: '876-09-5432'
9          }
10     };
11     console.log("I am running!");
12     console.log(JSON.stringify(event));
13     return response;
```

48. Deploy the changes.
49. Test your function through uploading a new image.
50. Check logs of this function. You can see this information about parameter “event”.

51. Now, we know how to retrieve the uploaded file information. Add a new line to index.js (Line #13).

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: {
            id: 1001,
            name: 'Jane Smith',
            phone: '876-09-5432'
        }
    };
    console.log("I am running!");
    console.log(JSON.stringify(event));
    console.log(event.Records[0].s3.object.key);
    return response;
};
```

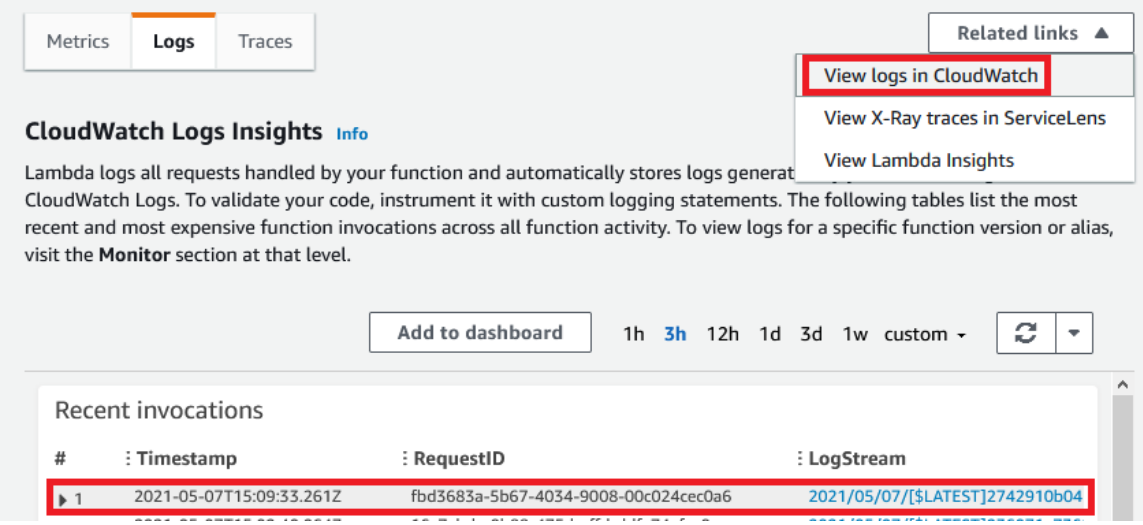
52. Deploy the changes. Test your function again through uploading a new image.
53. Check logs. You can see this information about “s3 object” in parameter “event”.

Part 9. Retrieve message information from SNS

54. On your **second** lambda function (in my case: listener), add a new line of code (#8) to display parameter event information, and modify Line#5.

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Listener!'),
    };
    console.log("I headed an event!");
    console.log(JSON.stringify(event));
    return response;
};
```

55. Deploy the changes. Test your function through uploading a new image.
56. From your second lambda function, check logs from “Recent invocations” or “CloudWatch”.



57. You can see this information about parameter "event".

```
2020-05-04T23:30:28.721Z        2a224607-cdc1-4f08-817e-d28f55c2c658    INFO
{
    "Records": [
        {
            "EventSource": "aws:sns",
            "EventVersion": "1.0",
            "EventSubscriptionArn": "arn:aws:sns:us-east-1:616875596481:bucketEvent:ec867d
            "Sns": {
                "Type": "Notification",
                "MessageId": "e3b7633f-85a1-5303-990b-34152275c8a6",
                "TopicArn": "arn:aws:sns:us-east-1:616875596481:bucketEvent",
                "Subject": null,
                "Message": "{\"version\":\"1.0\",\"timestamp\":\"2020-05-04T23:30:28.263Z\"
                "Timestamp": "2020-05-04T23:30:28.327Z",
                "SignatureVersion": "1",
                "Signature": "OW1ZqSJftmKfNfgJfPzDOC2zdNHMhO6t0q+zEOzHMmNxiOGQ4tJq4PnGGZuJ
                "SigningCertUrl": "https://sns.us-east-1.amazonaws.com/SimpleNotificationS
                "UnsubscribeUrl": "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe
                "MessageAttributes": {}
            }
        }
    ]
}
```

58. "Message is a long string. It is hard to read. Add a new line (Line #9) code to examine **Sns** Object.

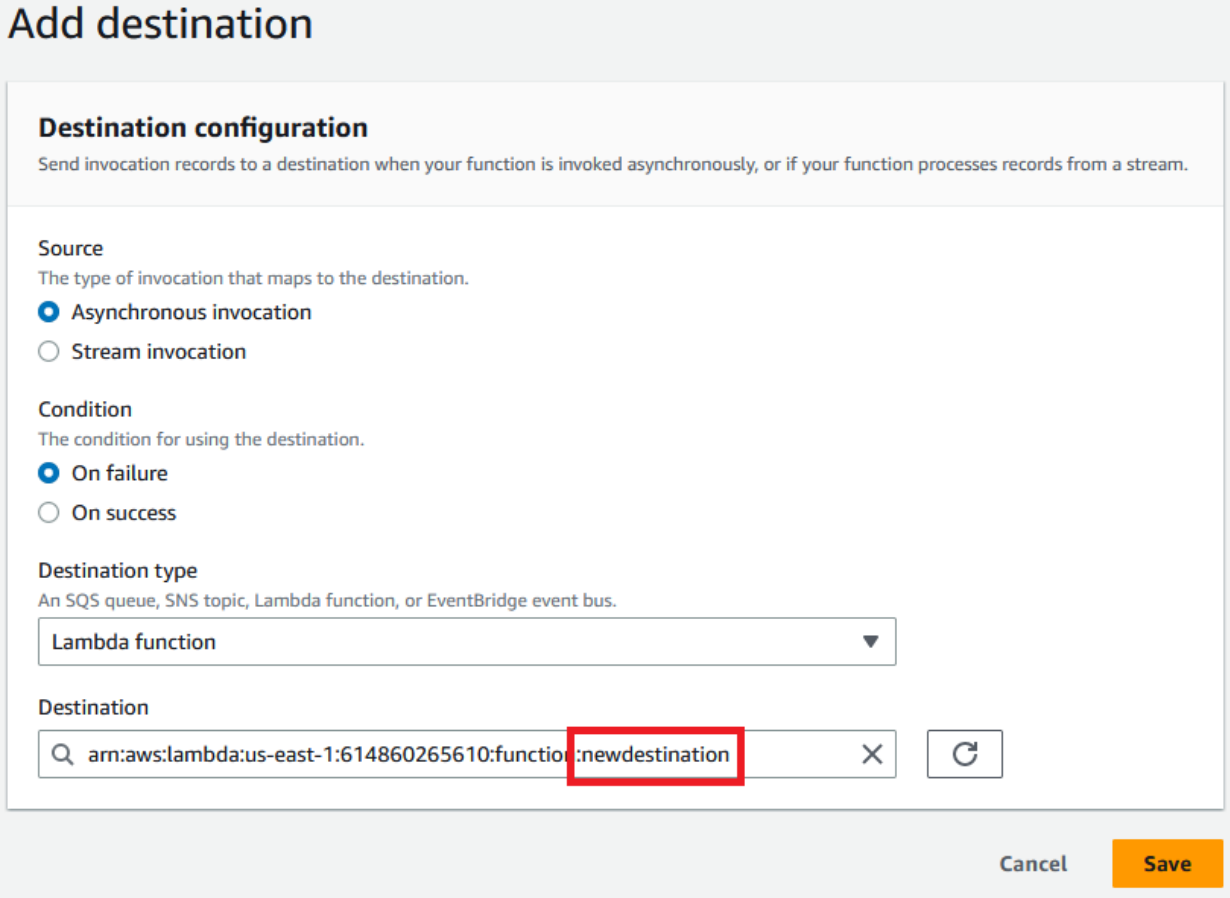
```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Listener!'),
    };
    console.log("I headed an event!");
    console.log(JSON.stringify(event));
    console.log(event.Records[0].Sns);
    return response;
};
```

59. Deploy the changes. Test your function through uploading a new image.
60. Check logs. You can see this information about Sns. All the information you might need is here.

61. SNS basically passes a message (a string). Information inside can be parsed and retrieved.

Part 10. Replace destination of our first Lambda function

62. Let's follow the previous mentioned steps to create a new Lambda function (from scratch). I will name it "newdestination". Do not forget to choose an "existing role".
63. Go back to your first Lambda function, add a new destination. Pick the Lambda function you just created.



64. Now, your first lambda function has another lambda function as its destination (the previous SNS is replaced by this new lambda function).
65. Your newest lambda function looks like the following.

```
1  exports.handler = async (event) => {
2      // TODO implement
3      const response = {
4          statusCode: 200,
5          body: JSON.stringify('Hello from Lambda!'),
6      };
7      return response;
8  };
```

66. Modify its code (Line #8) to retrieve “event” parameter information.

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from real handler!'),
    };

    console.log(JSON.stringify(event));
    return response;
};
```

67. Deploy the changes. Test your function through uploading a new image into S3.
68. Check logs of the **newest lambda function**. You can see this information about the event.
69. Delete all lambda functions.

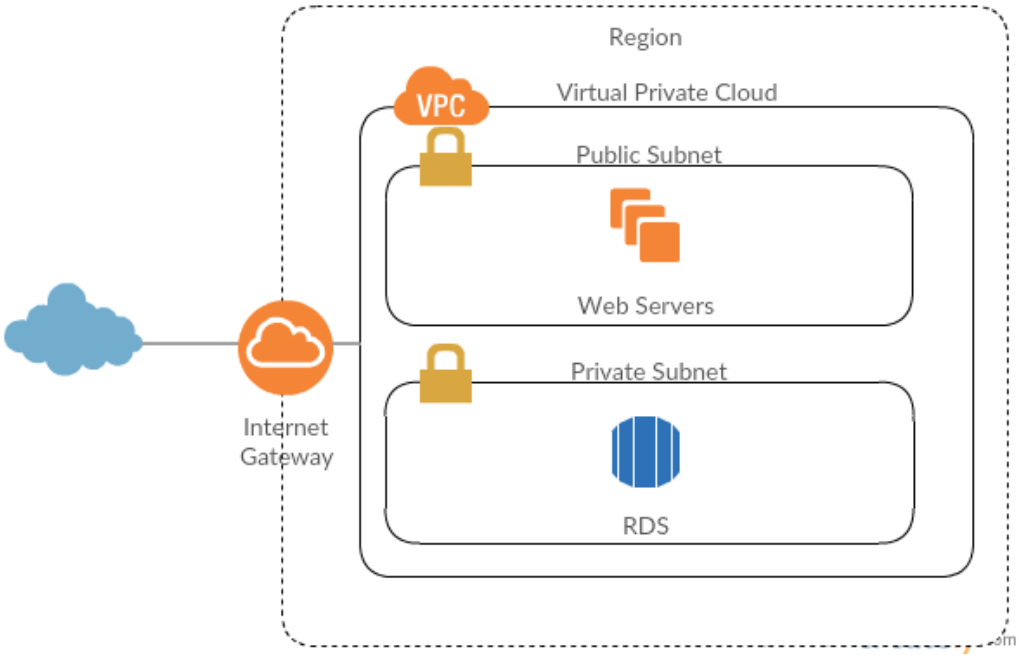
Chapter 4. Databases

AWS provides both relational database service and NoSQL database service. In this chapter, we will practice using Oracle relational database, as well as NoSQL database, such as DocumentDB and DynamoDB.

4.1 Relational Database

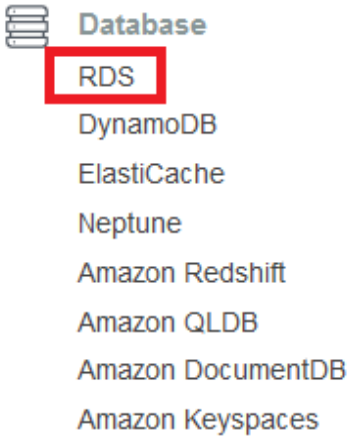
Objectives:

- Create an Oracle database
- Access the Oracle database with sqldeveloper

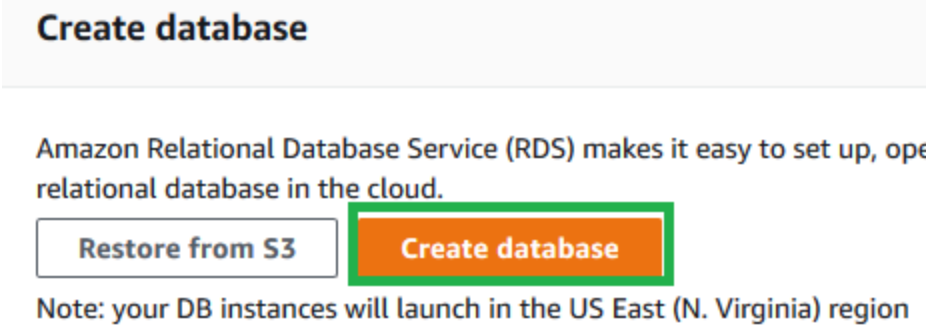


Part 1. Create an Oracle database

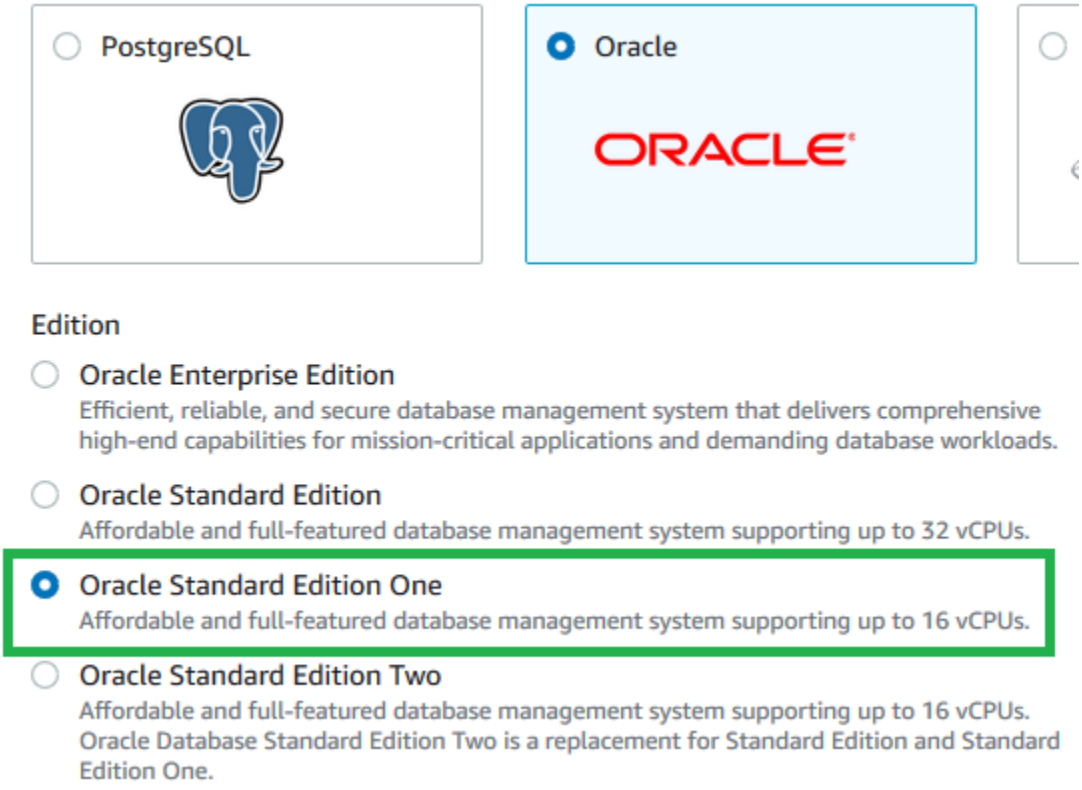
1. Logon to AWS Console, from Services page, select RDS under Database.



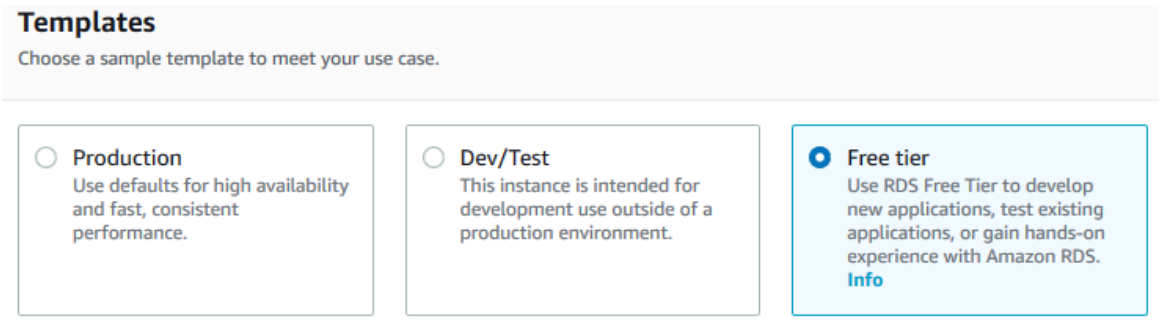
2. Select “Create Database”.



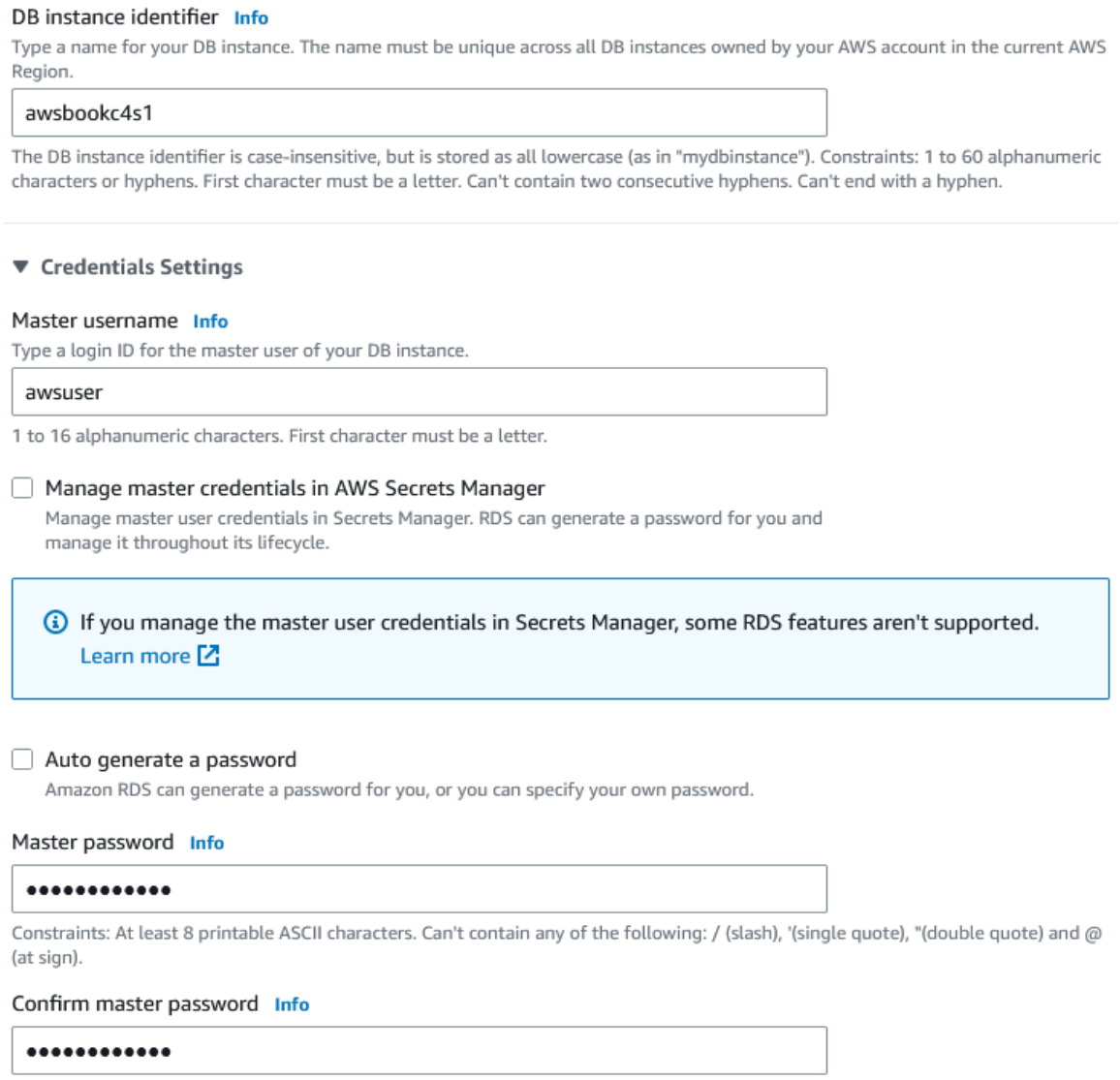
3. Select Oracle, Standard Edition One.



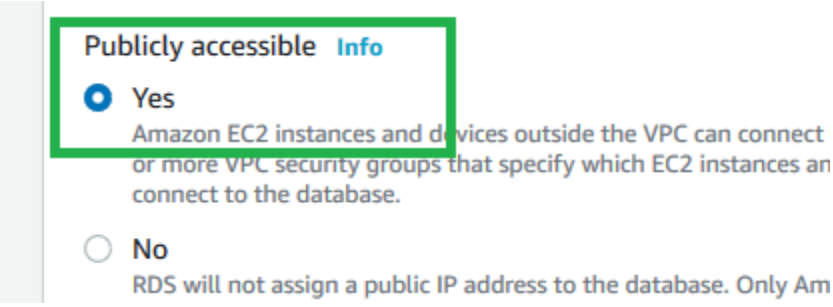
4. Select “Free tier” for Templates.



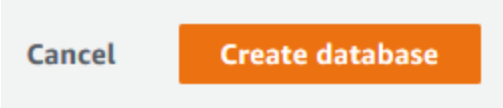
5. Enter DB name, user ID and password.



6. Allow public access.

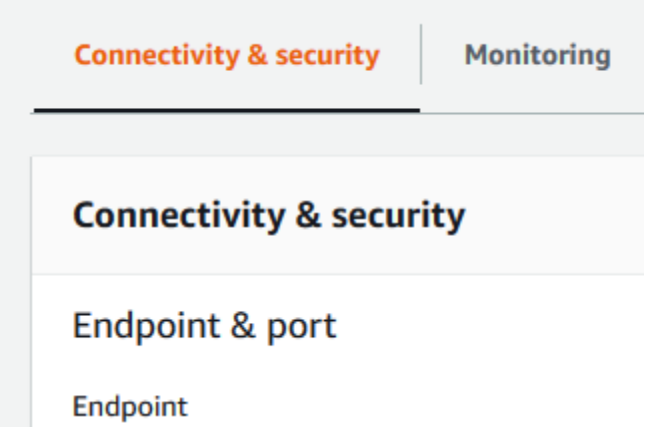


7. Click “Create database”.

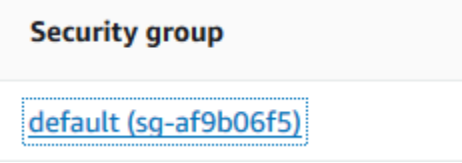


Part 2. Update inbound and outbound traffic rules

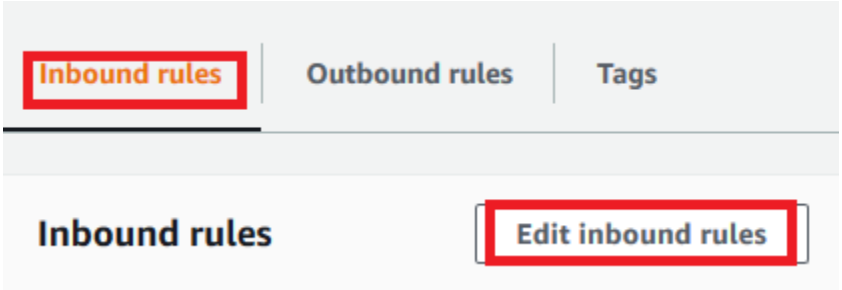
8. After creation is completed, click on your oracle database instance and select **Connectivity & security.**



9. Move down and click on your **security group.**

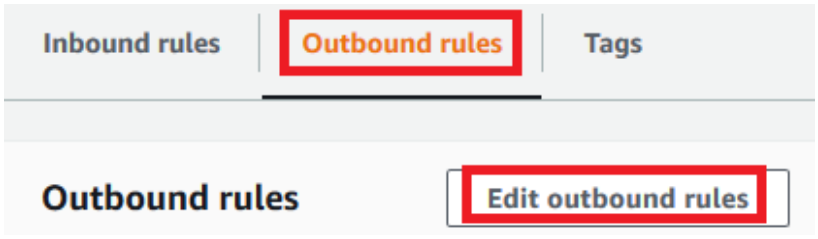


10. Click “Inbound rules”, “Edit inbound rules”.



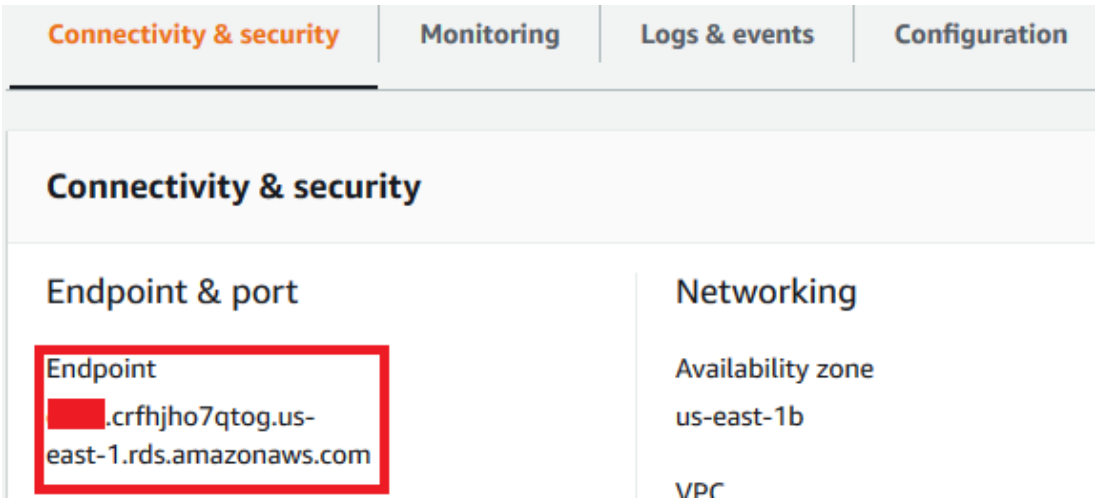
11. Click “Add rule”, select “All traffic”, and choose **'MY IP'** from the drop down. This automatically stores your IP address and allows your computer to connect to the Database. Click “Save rules”.

12. Click “Outbound rules”, “Edit outbound rules”.

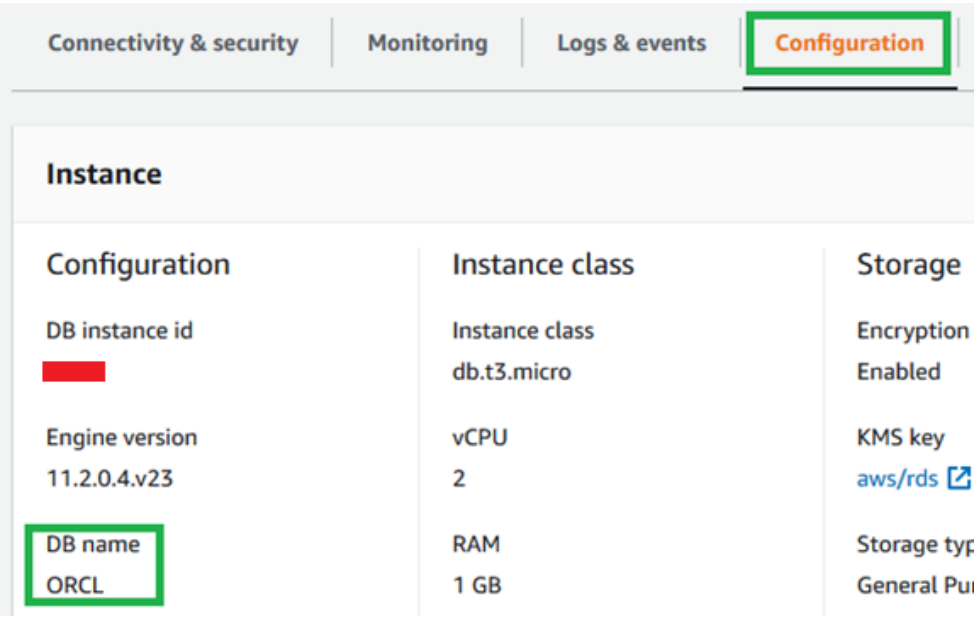


13. Click “Add rule”, select “All traffic”, and choose **'MY IP'** from the drop down. This automatically stores your IP address and allows your computer to connect to the Database. Click “Save rules”.

14. Go back to your database instance page. Copy Endpoint.

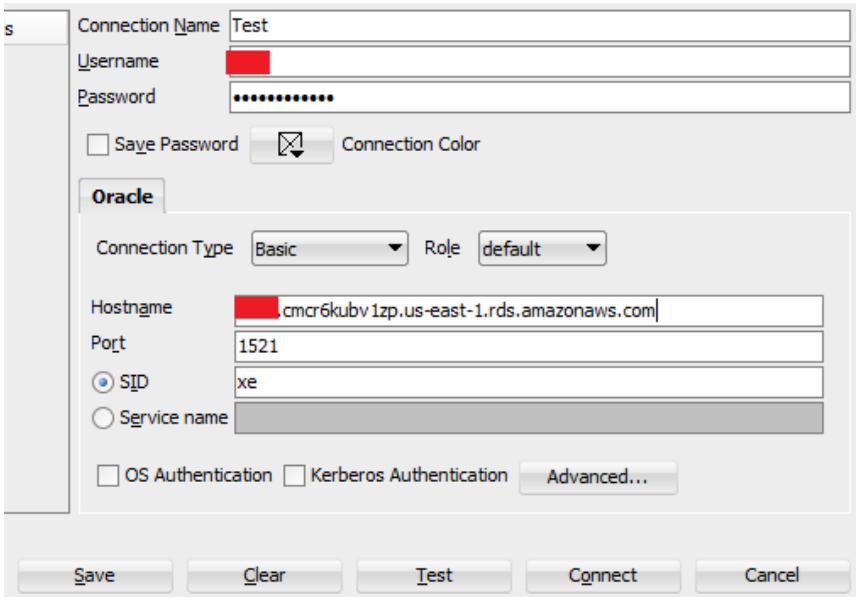


15. Click on “Configuration” and find out “DB name”.

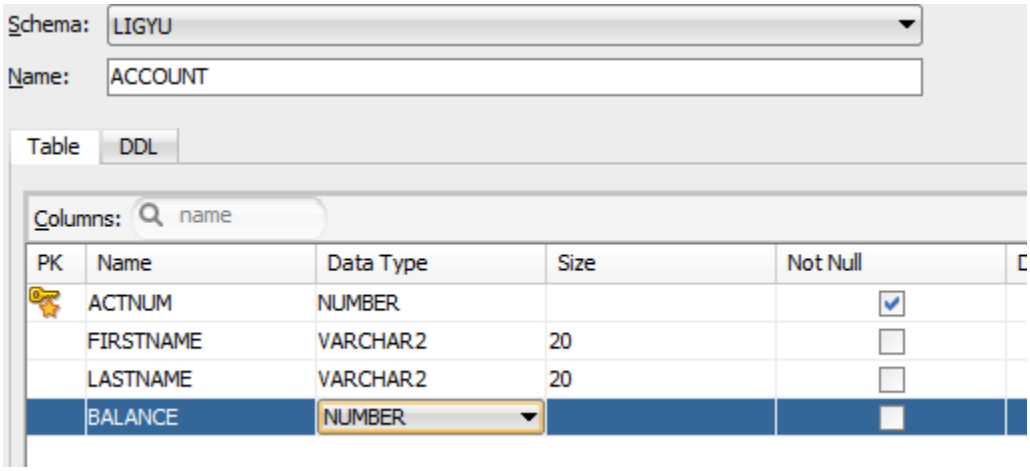


Part 3. Access database through SQL Developer

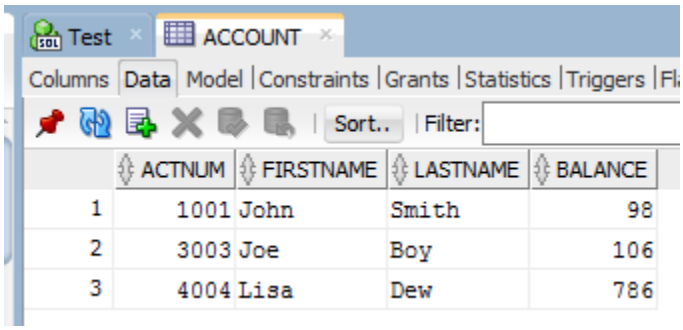
16. Download and Install Oracle SQL Developer
    https://www.oracle.com/tools/downloads/sqldev-v192-downloads.html
17. Start Oracle SQL Developer.
18. Specify a connection name.
19. Hostname is the "Endpoint" you copied earlier at Step 14.
20. SID is either XE or ORCL, which is "DB name" specified in database configuration (Step 15).



21. Click “Connect” and you should be able to connect to the database.
22. Create a Table called Account with the following attributes (columns).



23. Enter at least 3 records into Table Account. After the data is entered, do not forget to commit the changes.



24. Run a query to show all the accounts that have a balance of less than 200.

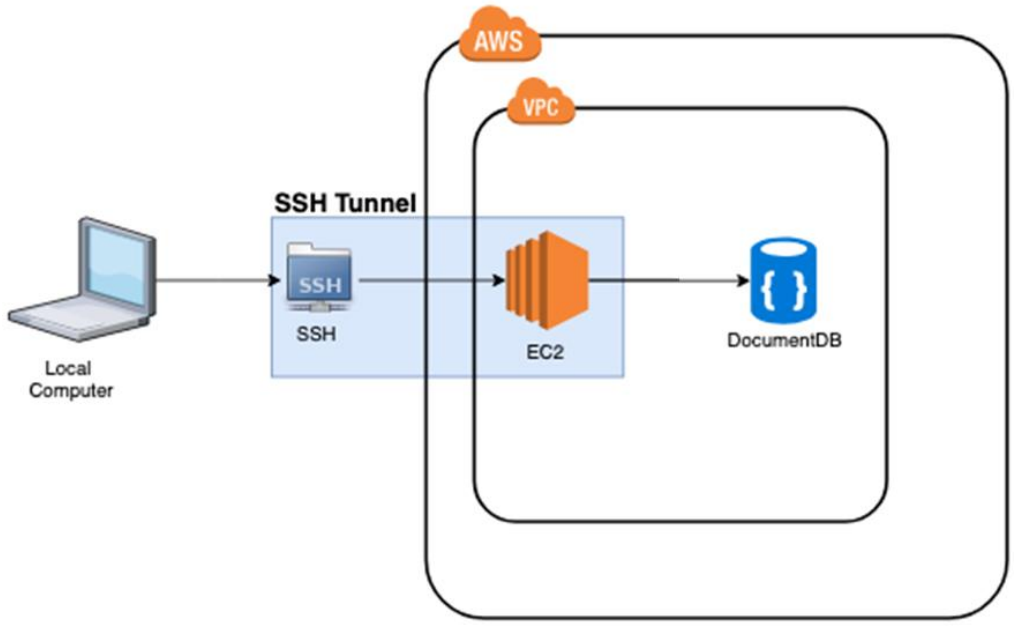
    *SELECT * FROM ACCOUNT WHERE balance < 200;*

25. Delete the database instance.

4.2 DocumentDB

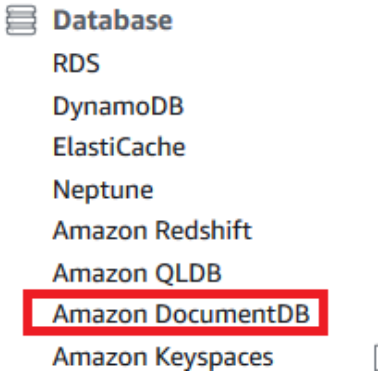
Objectives:

- Create an instance of AWS DocumentDB
- Connect to AWS DocumentDB with SSH tunnel
- Create collections and upload data using Studio 3T
- Query DocumentDB and export the result.

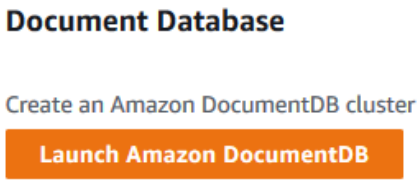


Part 1. Create an instance of AWS DocumentDB

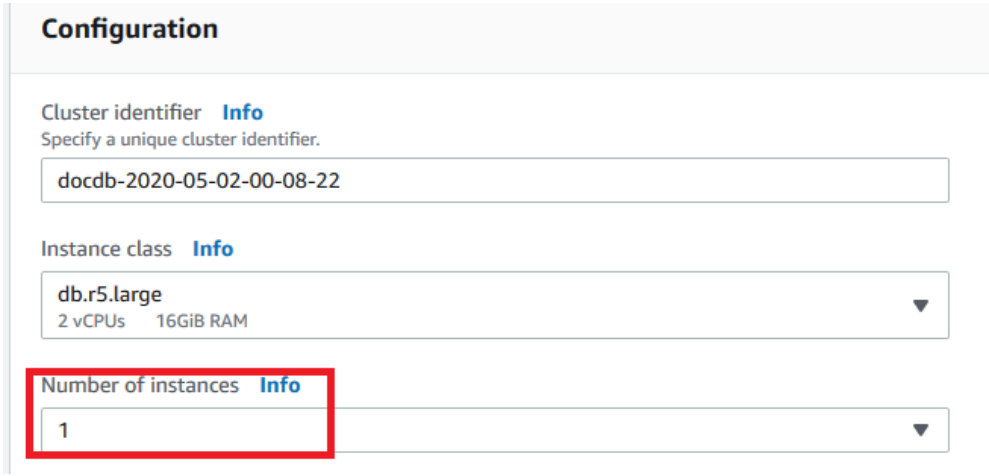
1. Log onto AWS console.
2. From Services, select Amazon DocumentDB under Database.



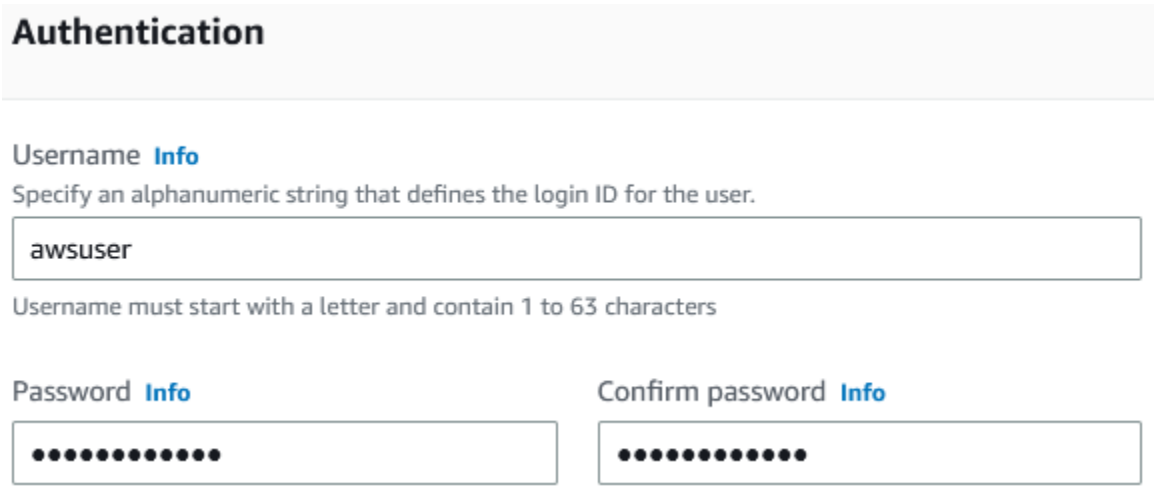
3. Click “Launch Amazon DocumentDB”.



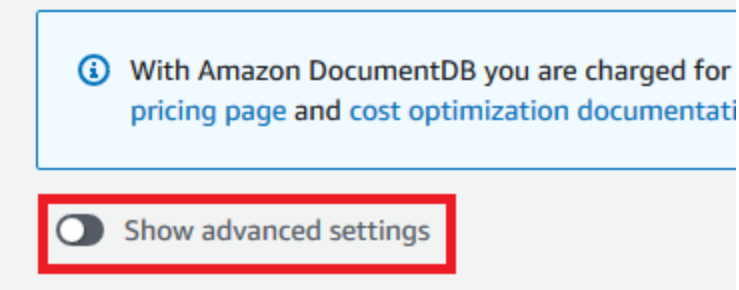
4. I will just pick 1 instance (to reduce the unnecessary cost).



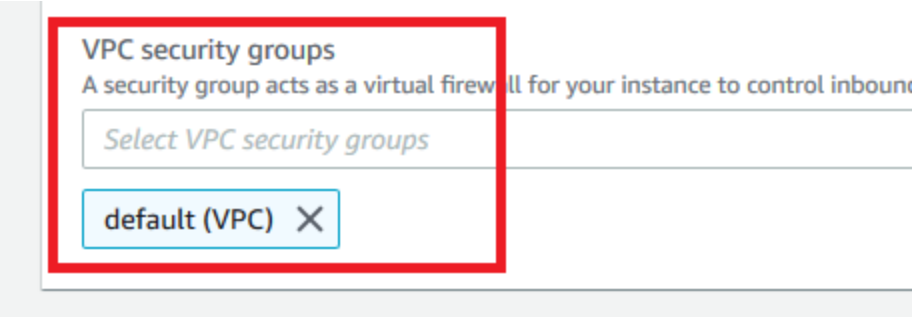
5. Specify username and password.



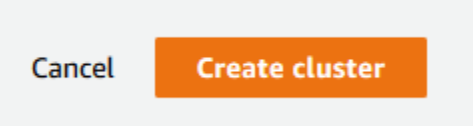
6. Enable “Show advanced settings”.



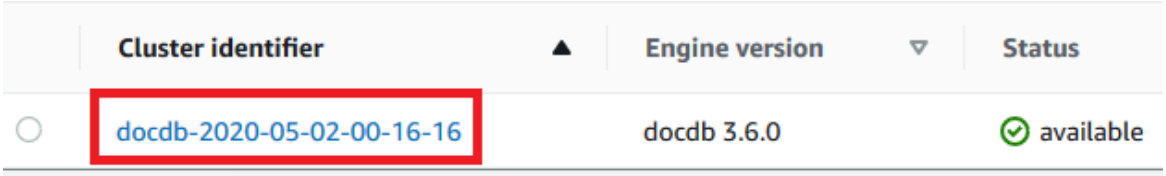
7. You can see that "default VPC" is selected. You do not need to change anything here.



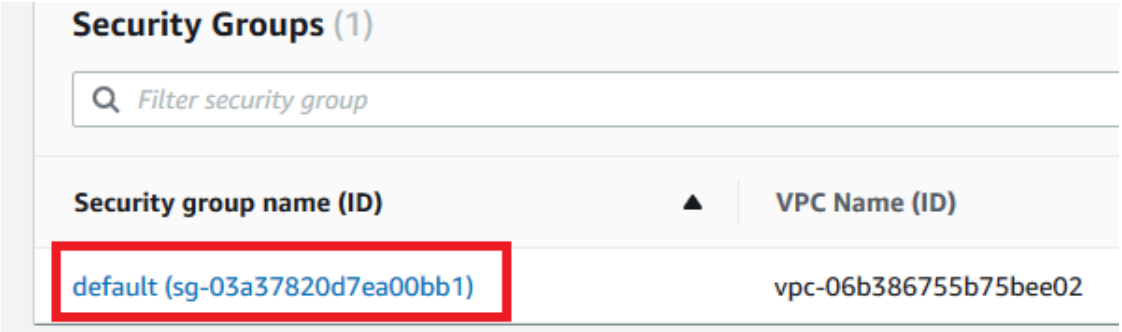
8. Click "Create cluster".



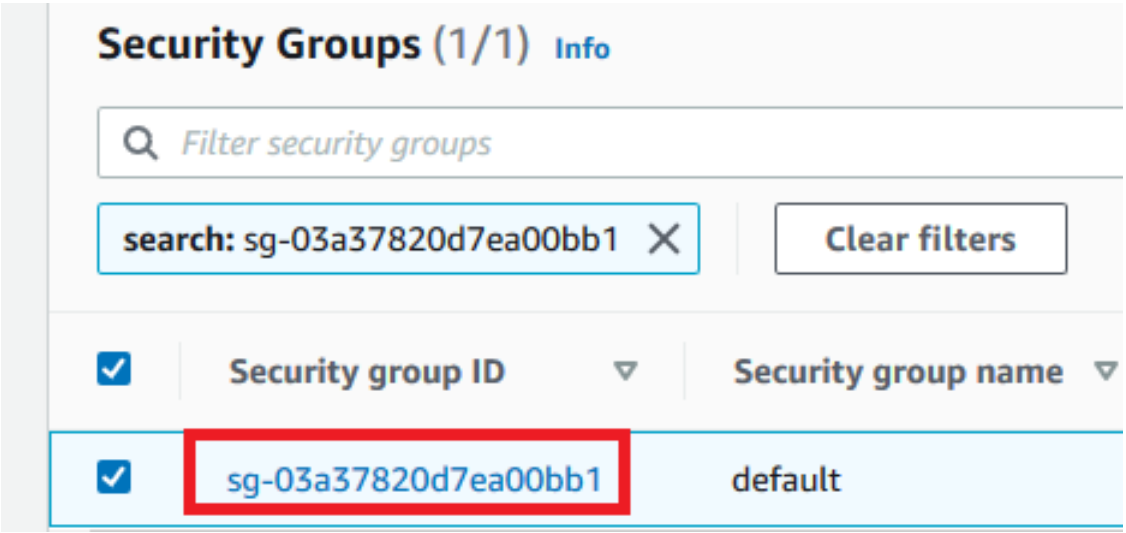
9. Your AWS DocumentDB cluster will be created.
10. Click your instance when it is available.



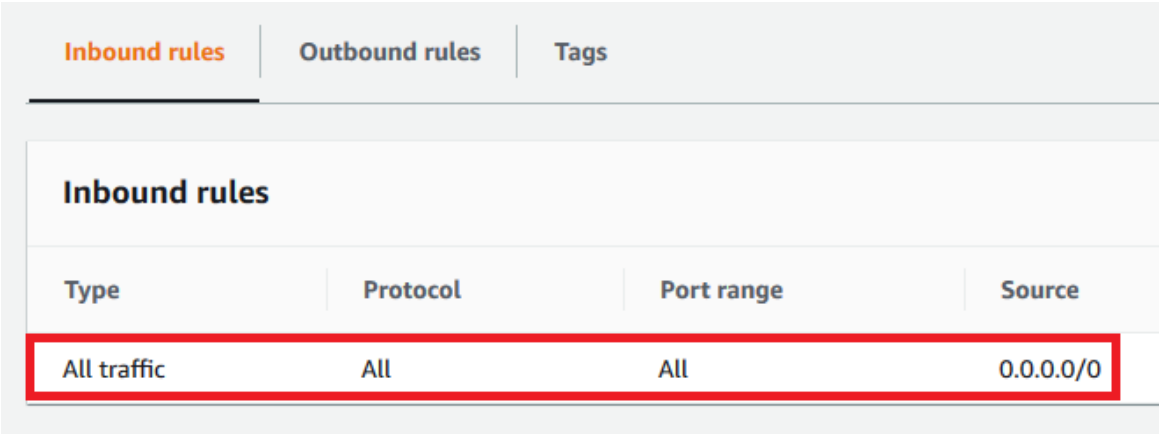
11. Move all the way down and click on "Security group".



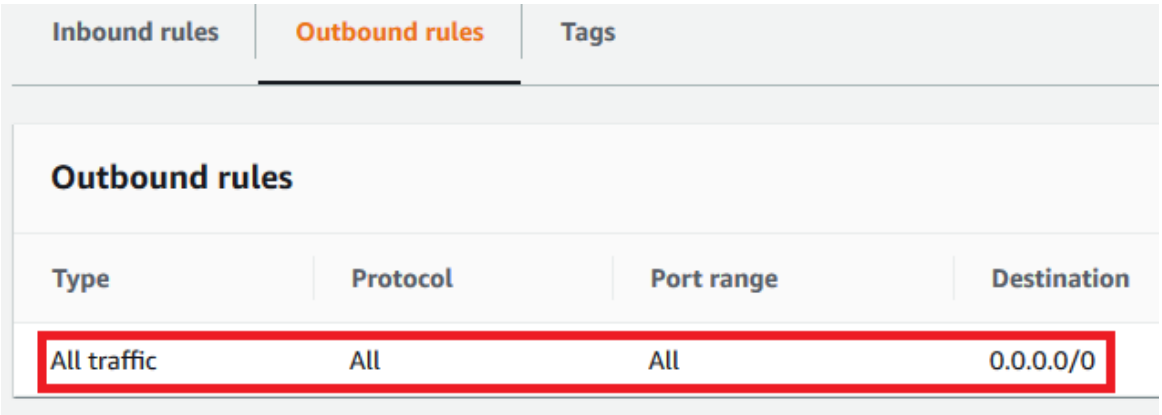
12. Click on the Security Group ID again.



13. Make sure you have the following Inbound Rule. If not, change it.



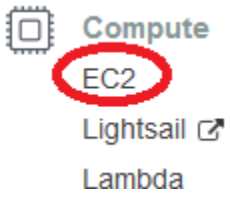
14. Make sure you have the following Outbound Rule. If not, change it.



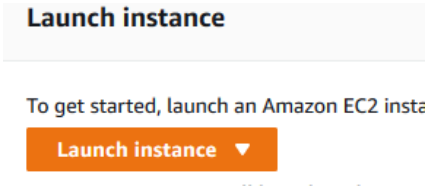
15. Save all the change.

Part 2. Create an AWS EC2 instance in the same VPC of the DocumentDB

16. Go back to your Services page, click EC2 under Compute.

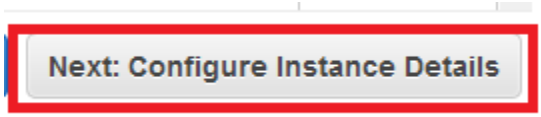


17. Click "Launch instance".

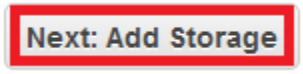


18. Select Amazon Linux 2 AMI.

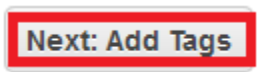
19. Click "Next: Configure Instance Details".



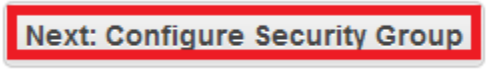
20. Click "Next: Add Storage".



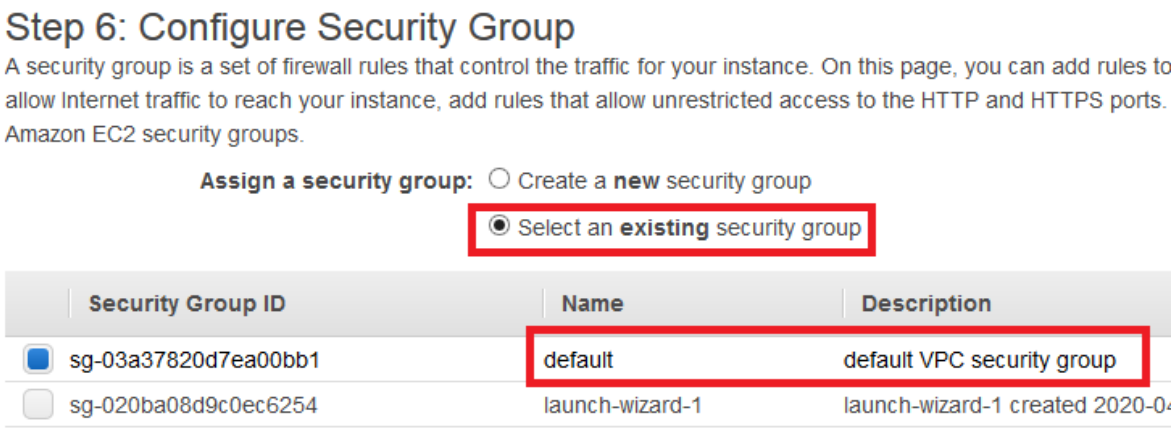
21. Click "Next: Add Tags".



22. Click "Next: Add Security Group".

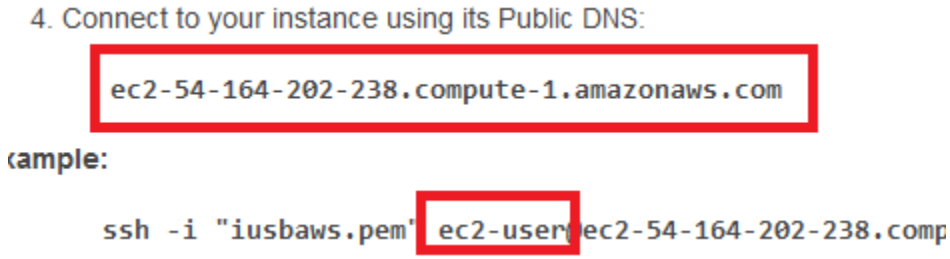


23. Select “Select an existing security group”, select “default VPC security group” (Your EC2 instance and DocumentDB instance must be in the same VPC group).



24. Click “Review and Launch”.

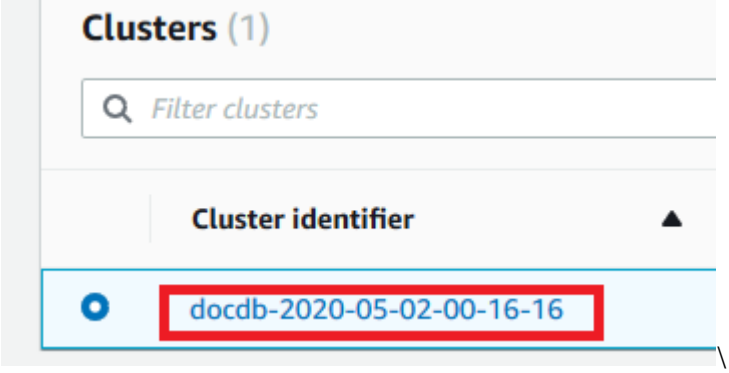
25. Ignore the warnings and click “launch”.
26. Select the key pair you generated before. Click “launch Instances”.
27. After your EC2 instance launching is completed. Get the connection information.



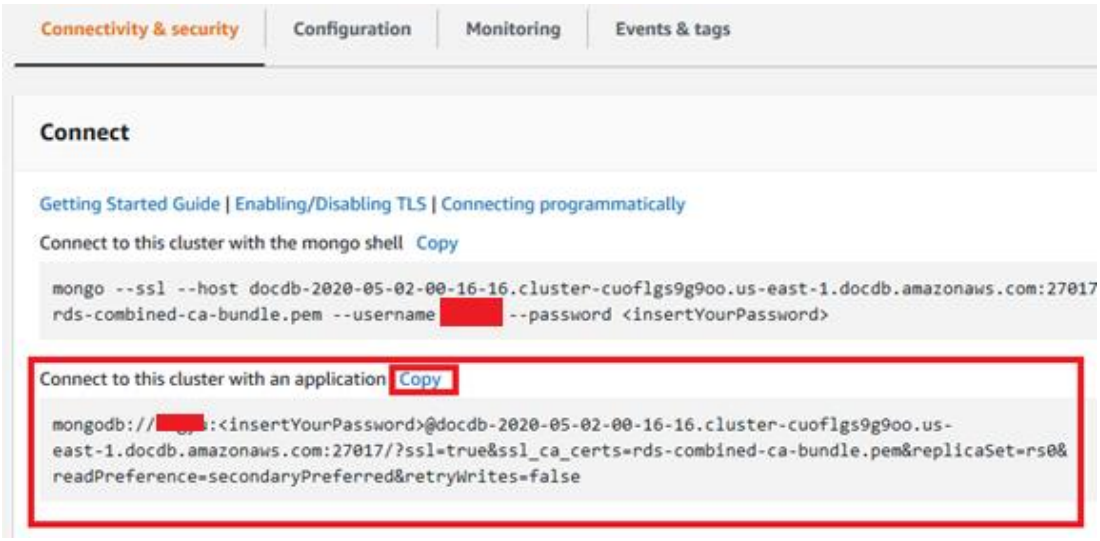
28. Use Putty to connect to this EC2 instance. Make sure it works. We have done this in Section 1.1. Refer to that section for instructions if needed.

Part 3. Download and install Studio 3T and connect to AWS DocumentDB

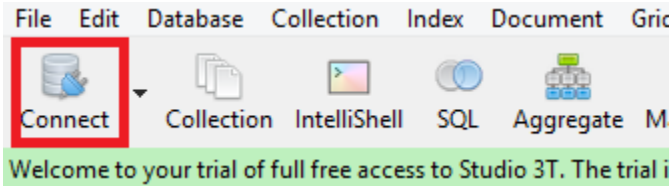
29. Download Studio 3T (https://studio3t.com/) and install it.
30. Go back to your AWS DocumentDB instance page.
31. Click on your DocumentDB instance.



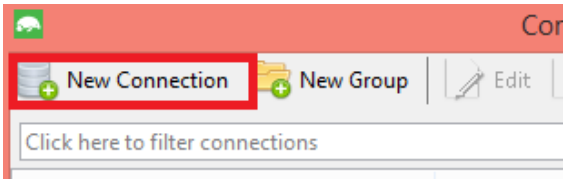
32. Copy the connection string. Replace <insertYourPassword> with your password. Note here: <> should also be replaced.



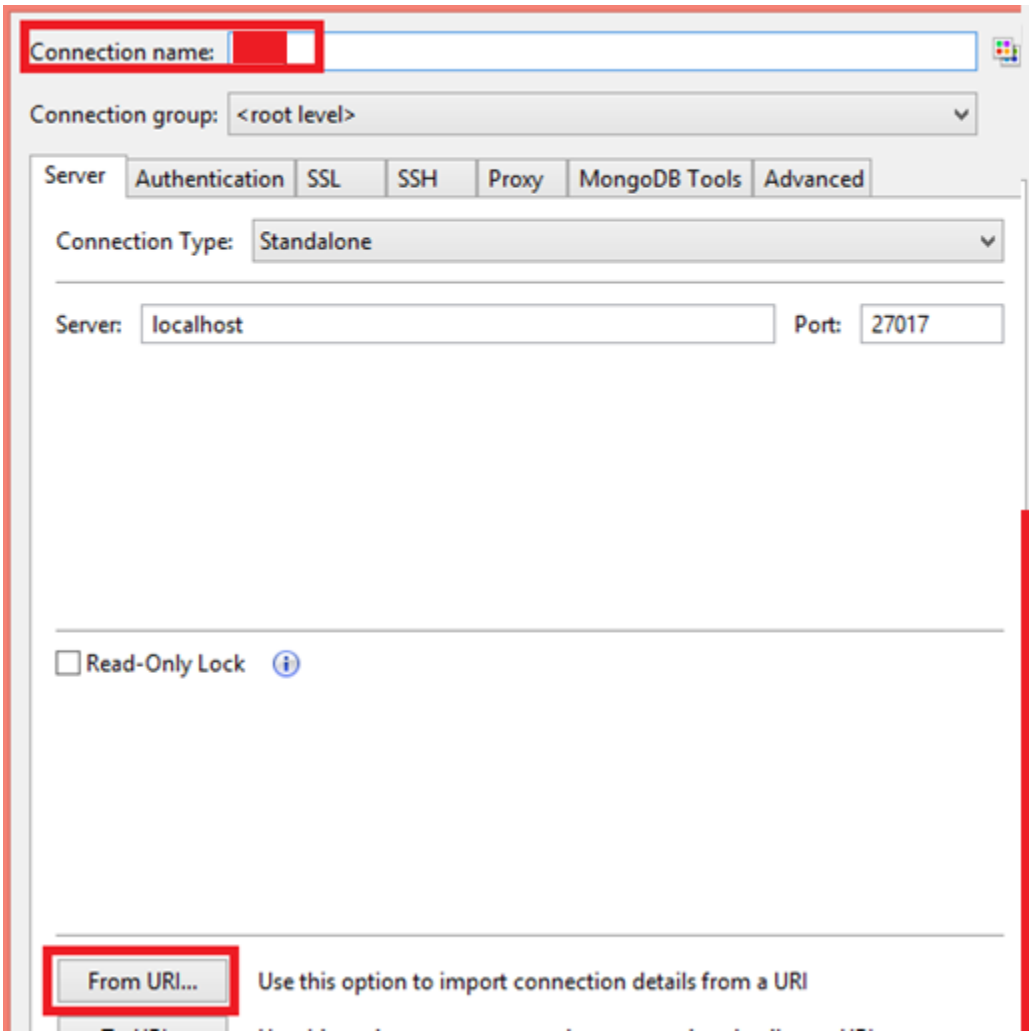
33. Start Studio 3T. Click “Connect”.



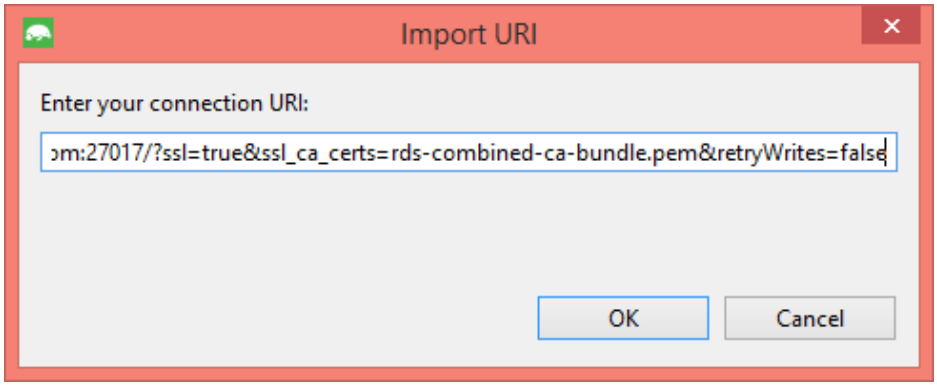
34. Click “New Connection”.



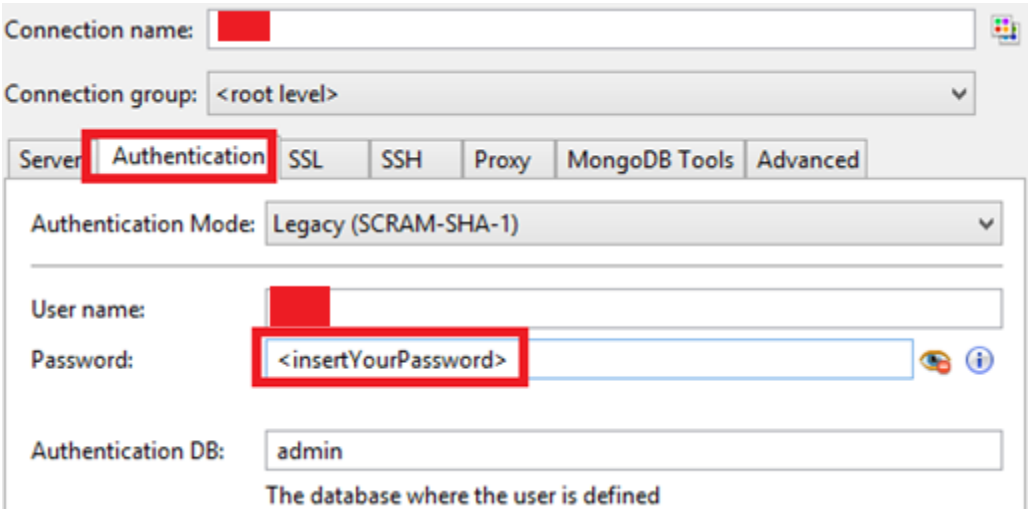
35. Assign a name to this connection, then click “From URI”.



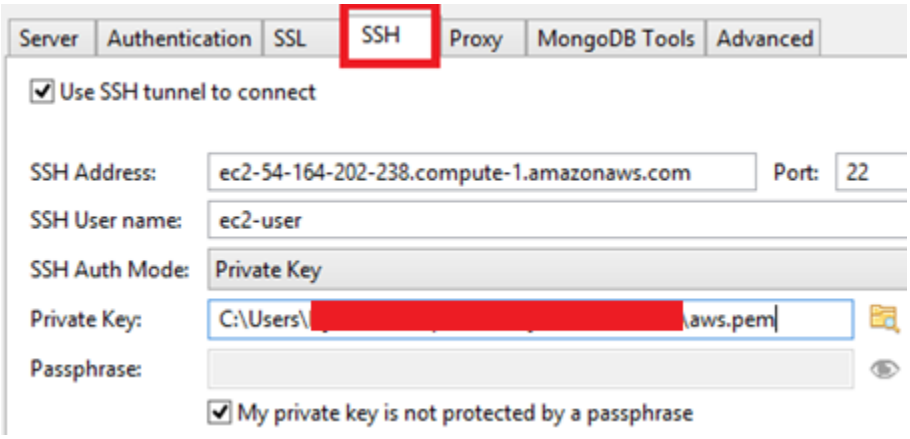
36. Paste the Connection String you copied at Step 32. Click "OK" (be **careful** here: make sure the **whole** URL is pasted here).



37. Click "Authentication". Replace <insertYourPassword> with your password if you have not done it.



38. Click SSH, check "Use SSH tunnel to connect".
39. Paste your EC2 instance address and user name you copied at Step 27. Locate your private key (**.pem**).
40. Click Save.

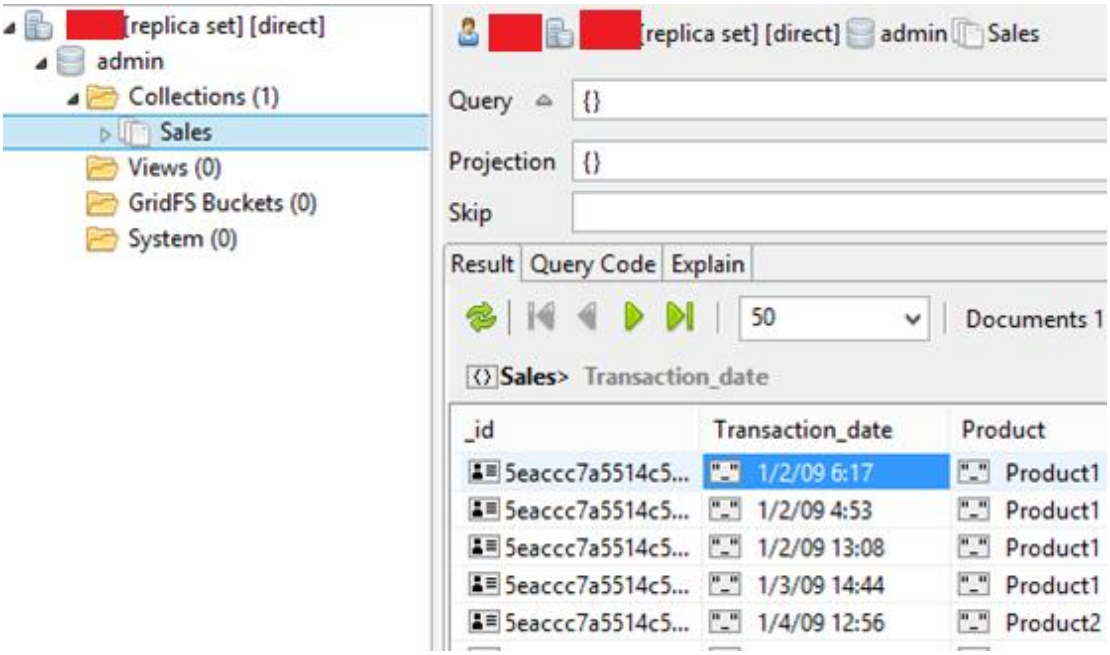


41. Click "Connect".

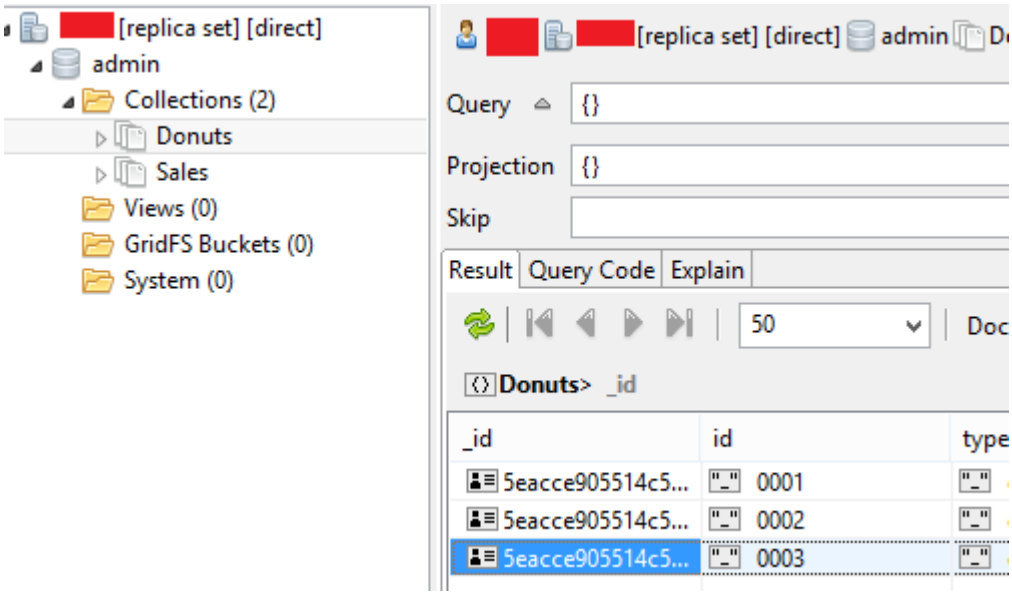
42. You should be connected to your AWS DocumentDB.

Part 4. Create collections and upload data

43. Add a new Collection (Table), let's call it "Sales".
44. Import data from file "Sales.csv" provided online (https://github.com/awsbookresource/files).
45. The result might look like the following.



46. Add a new Collection (Table), let's call it "Donuts".
47. Import data from "Donuts.json" given online (https://github.com/awsbookresource/files).
48. Your result might look like the following.



Part 5. Manipulate data, create query, and export data

49. Practice each of the following operations (Lock/Unlock documents, Add document, View document, Edit document, Update document, Remove document, Find document) through manipulate data (documents) in Collection Donuts.

50. Export all documents in Collection Donuts to your desktop in JSON format. For your reference, my exported file is given online (https://github.com/awsbookresource/files).
51. Examine the exported data with your original data, could you see any difference?
52. Validate the JSON Syntax of exported data using an online tool (https://jsonlint.com/).
53. Create a query for Collection Sales which returns all the documents whose City field has value “Ottawa".

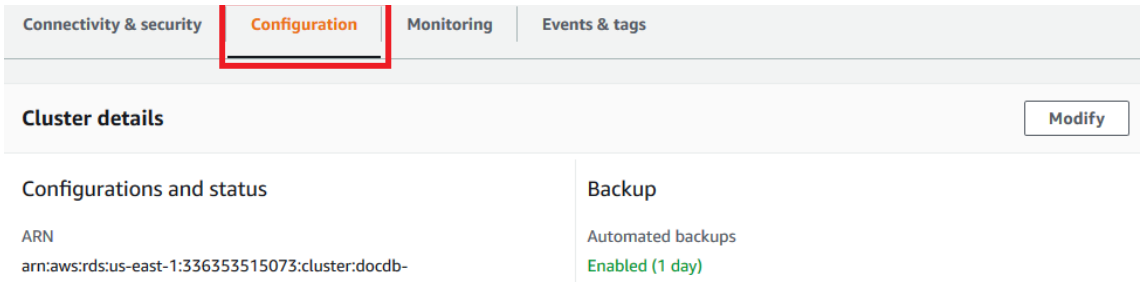
```
select *
from Sales
where City = "Ottawa";
```

54. Export the query result to your desktop in JSON format. For your reference, my query result given online (https://github.com/awsbookresource/files).
55. Validate the JSON Syntax of exported data using an online tool (https://jsonlint.com/).
56. Delete your DocumentDB instance and your EC2 instance.

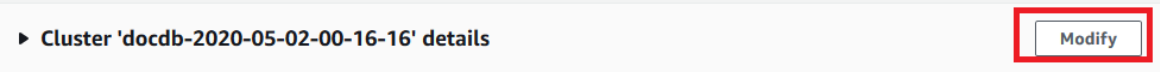
For your reference, if you could not figure out how to delete AWS DocumentDB, here are the instructions.

1. Click on the instance.

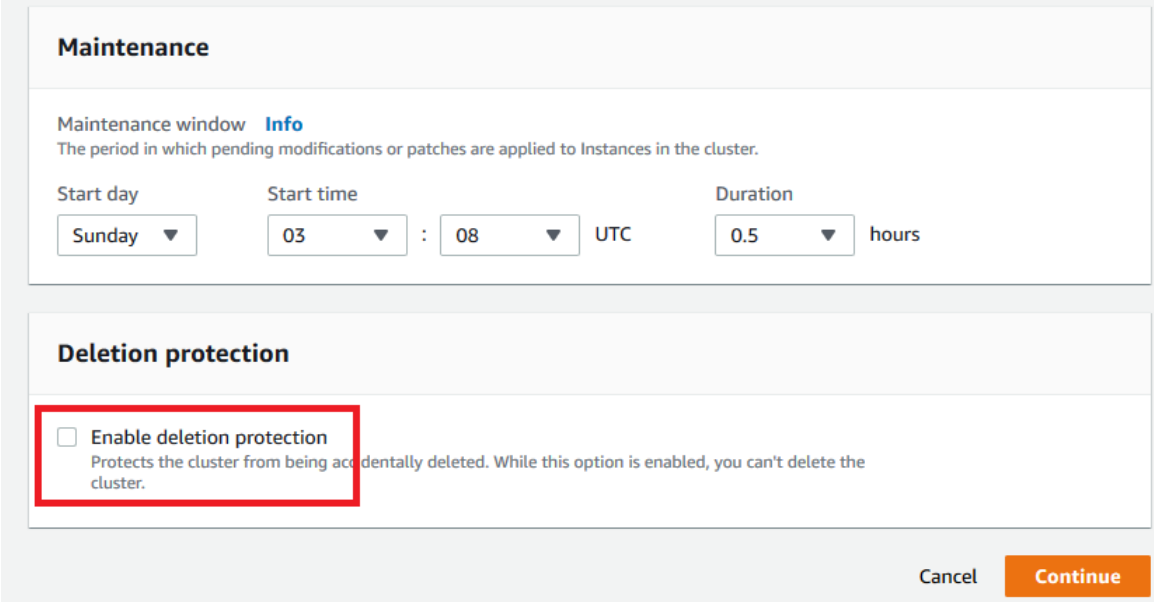
2. Click “Configuration”.



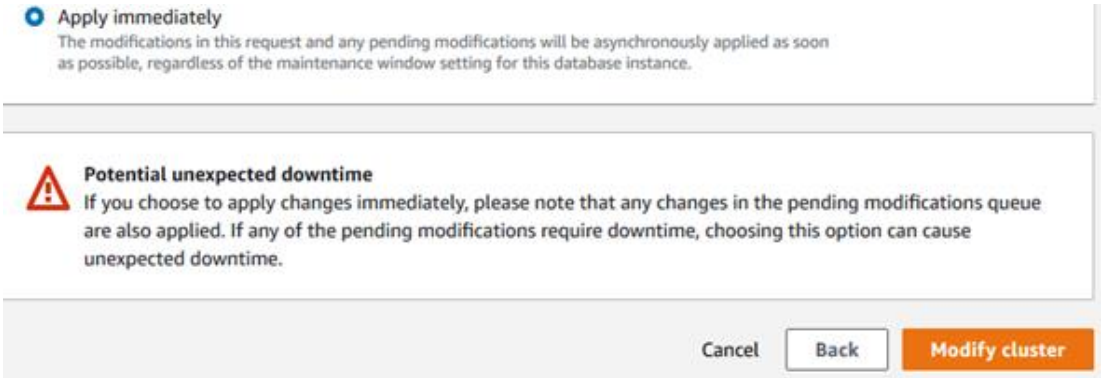
3. Move down to Cluster details, Click “Modify”.



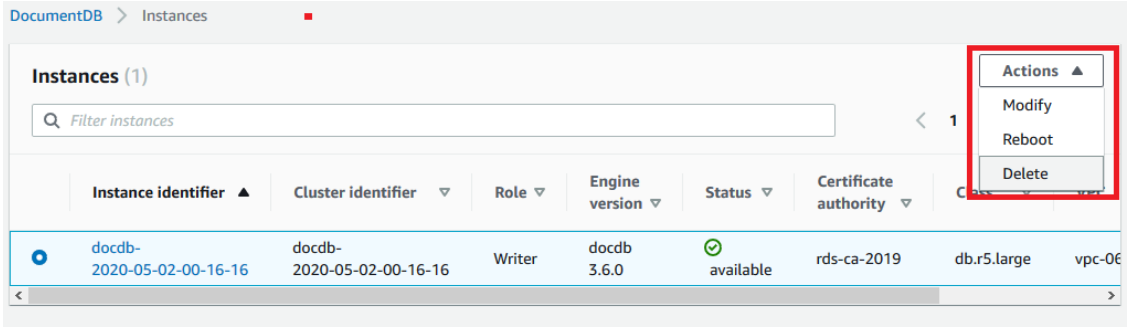
4. Move all the way down, uncheck “Deletion Protection”. Click “Continue”.



5. Select “Apply immediately”. Click “Modify cluster”.



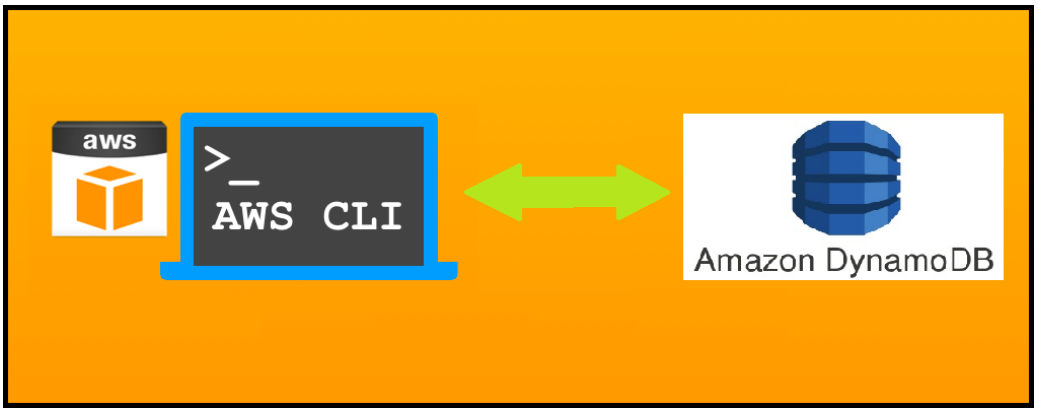
6. Go back to your instance page, delete it.



4.3 DynamoDB

Objectives:

- Create a DynamoDB
- Upload data using AWS CLI.
- Query a DynamoDB



Problem description:

We would like to build a database table of city events that our travel club would like to attend this year. In any month, we will attend at most one event for any city. We decided to choose DynamoDB to build the database because we have various information to store. This database will be used by clients all over the world. They can enter any kind of data.

Composite primary key will be used in this database. They are City Name (including state and country information if needed) and Month of the event. City name is the partition key and Month is the sort key. So events happening at the same city will have the same hash value and will be stored physically together.

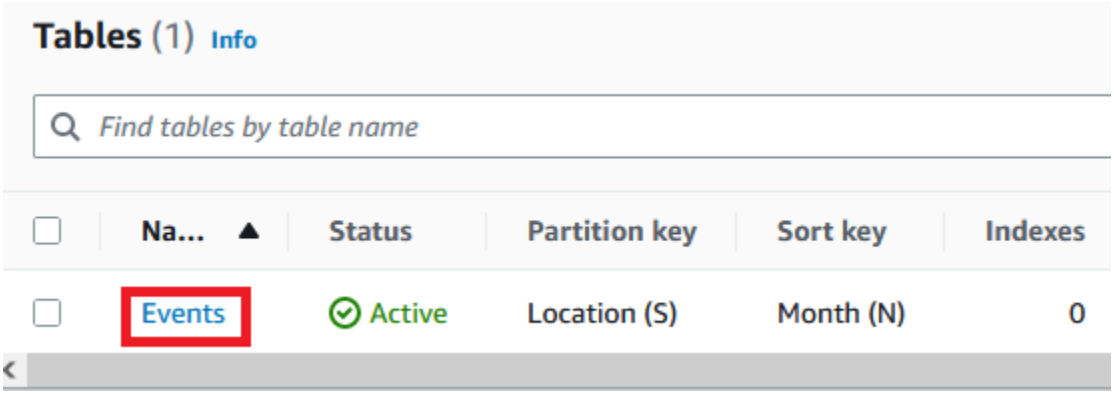
Part 1. Create a DynamoDB

1. From AWS Console, click on Services at the top.
2. Click on DynamoDB under Database services.

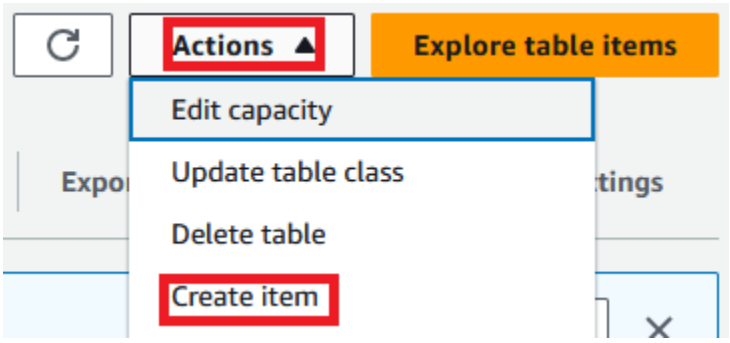
3. Click on Create table.
4. Table name: “Events”.

5. Partition Key: “Location”.
6. Check “Add sort key”: “Month”, data type: **Number**. Click on “Create”.

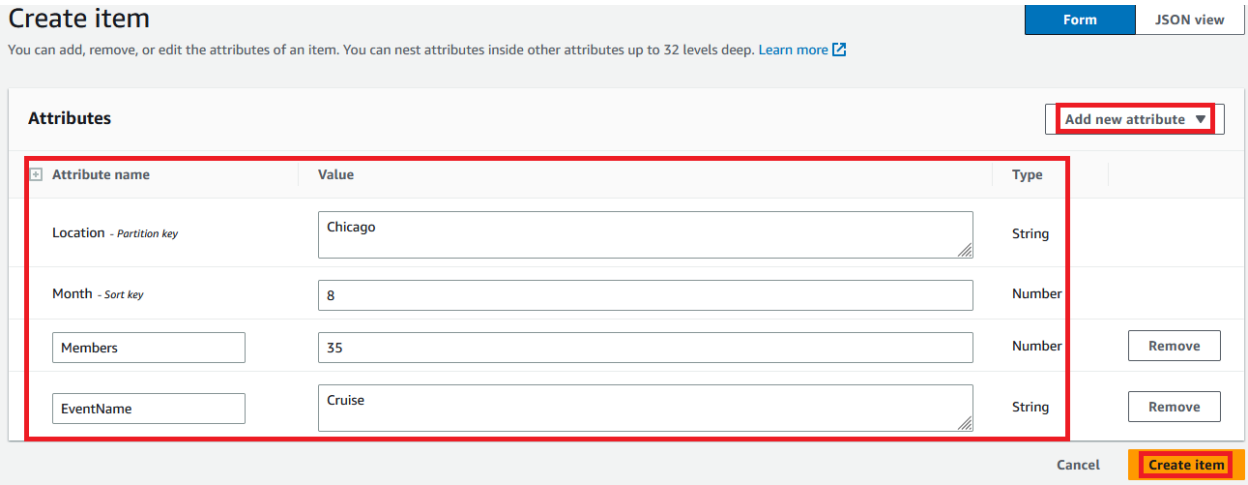
7. Click on table name.



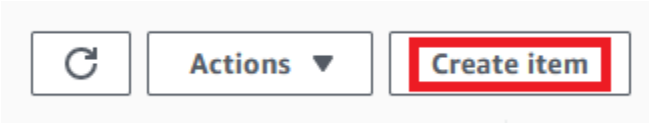
8. Click on “**Actions**”.
9. Click on “**Create item**”.



10. Add attributes to this item as shown below and click “**Create item**”.



11. Click “Create item”.



12. Repeat Steps 10-11 and enter the following data items.

Location	Month	Members	Successful	EventName
Chicago	8	35		Cruise
Mishawaka	7		true	Fireworks
New York City	11	39		Parade
South Bend	9		true	Football

Part 2. Upload data through AWS CLI

13. We would like to upload the following data items into the “Events” DynamoDB table.

Location	Month	EventName	Members	Comment
Indianapolis	5	Indy 500	125	
Holland	5	Tulip Festival		Canceled this year
Elkhart	6	4H Fair	17	

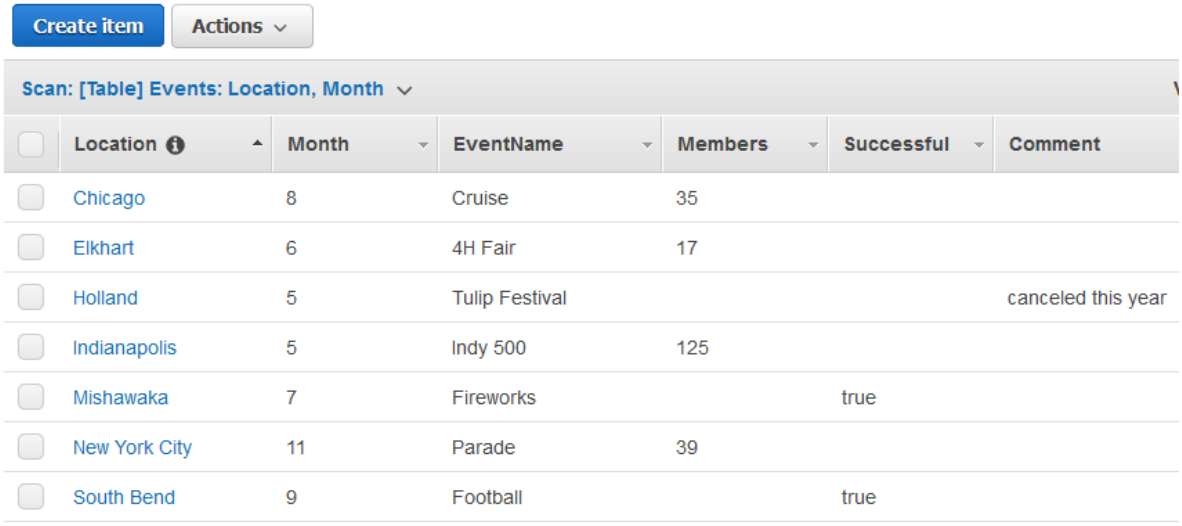
14. A josn file (Events.json) with these data items has been created for you. Download it from canvas. The file looks like the following.

```
{
    "Events": [
        {
            "PutRequest": {
                "Item": {
                    "Location": {"S":"Indianapolis"},
                    "Month": {"N":"5"},
                    "EventName": {"S":"Indy 500"},
                    "Members": {"N":"125"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Location": {"S":"Holland"},
                    "Month": {"N":"5"},
                    "EventName": {"S":"Tulip Festival"},
                    "Comment": {"S":"canceled this year"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Location": {"S":"Elkhart"},
                    "Month": {"N":"6"},
                    "EventName": {"S":"4H Fair"},
                    "Members": {"S":"17"}
                }
            }
        }
    ]
}
```

15. Understand its structure and format (reference: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html) .
16. Upload data to DynamoDB using AWS CLI (reference: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SampleData.LoadData.html).
    a. Log onto AWS account
    b. Copy credential and paste it into your credential file
    c. Execute the following command

```
aws dynamodb batch-write-item --request-items file://Events.json
```

17. Go to your DynamoDB table, refresh the table, you can see that your table has seven items now.



Location	Month	EventName	Members	Successful	Comment
Chicago	8	Cruise	35		
Elkhart	6	4H Fair	17		
Holland	5	Tulip Festival			canceled this year
Indianapolis	5	Indy 500	125		
Mishawaka	7	Fireworks		true	
New York City	11	Parade	39		
South Bend	9	Football		true	

18. Execute the following AWS CLI command to download Table Events from DynamoDB.

```
aws dynamodb scan --table-name Events > export.json
```

19. Examine the file you downloaded.
20. Do **NOT** delete Table Events. We will use it in Chapter 5.

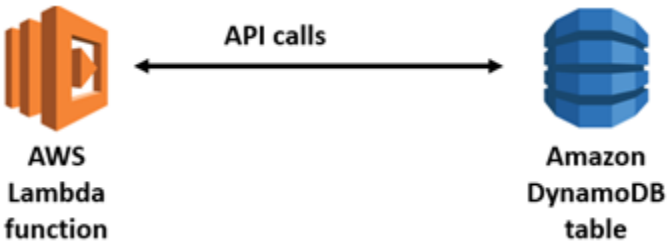
Chapter 5. Integrations

To solve a real-world business problem, multiple AWS services must be used together. From infrastructure to storage, to web hosting, AWS services can be integrated seamlessly to provide reliable and on demand business solutions. Often, AWS services are also used together with third party services, such as GitHub. In this chapter, we will create three applications using multiple AWS services.

5.1 Write data to DynamoDB using Lambda Function

Objectives:

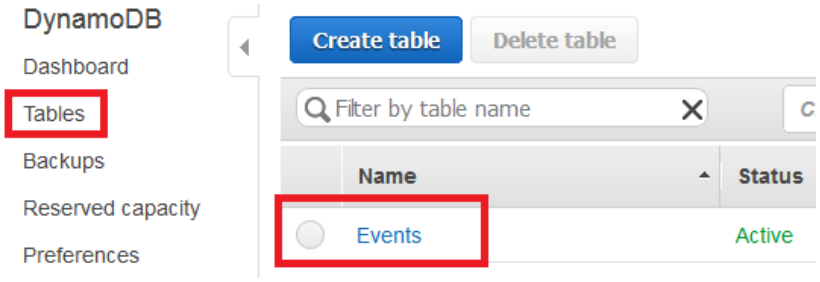
- Create a lambda function
- Write data to a DynamoDB table with the lambda function



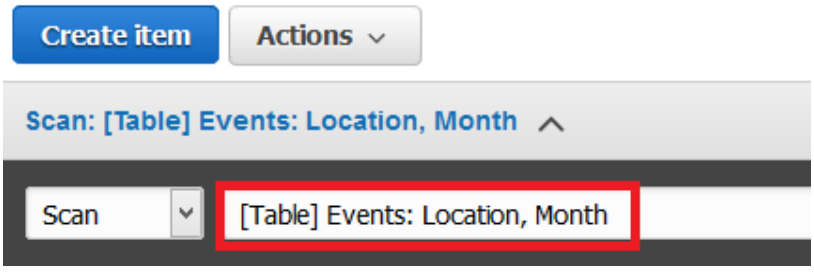
Note: This practice is created under an AWS Academy student account "LabRole". There is no need to set up policies. For regular AWS account users, a DynamoDB policy needs to be created and assigned to a role.

Part 1. Check your DynamoDB table

1. Log onto AWS Console.
2. Make sure your Events Table is there. If it is deleted, you should create a new one following the instruction given in Section 4.3.



3. Remember, your Events Table has a composite primary key (Location and Month).



Part 2. Create a Lambda function

4. Create a new Lambda function.
5. Let's choose "Author from scratch". Assign a name to your function.

6. Expand “Change default existing role”.
7. Pick “Use an existing role”.
8. Choose “LabRole”.
9. Click on “Create function”.
10. This is the code we have now.

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

11. Replace the code (**index.mjs**) with the one provided online (https://github.com/awsbookresource/files).

index.mjs | Execution results

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  PutCommand,
  GetCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const dynamo = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };
      body = {
        "Action": "Put an item",
      };

  await dynamo.send(
    new PutCommand({
        TableName: 'Events',
        Item: {
            Location: 'South Bend',
            Month: 7,
            EventName: 'Cool Summer',
            Notes: 'It is fun!'
        },
    })
 );

  return {
  statusCode,
  body,
  headers,
  };
```

12. Deploy the changes.
13. Create a testing case and run it. You should see the following result.

Test Event Name

test1

Response

```
{
  "statusCode": 200,
  "body": {
    "Action": "Put an item"
  },
  "headers": {
    "Content-Type": "application/json"
  }
}
```

14. Check the DynamoDB Events table to see if a new item is added.

Part 3. Insert more items

15. Modify the circled code part.

index.mjs | Execution result

```
1  import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
2  import {
3    DynamoDBDocumentClient,
4    PutCommand,
5    GetCommand,
6  } from "@aws-sdk/lib-dynamodb";
7
8  const client = new DynamoDBClient({});
9  const dynamo = DynamoDBDocumentClient.from(client);
10
11 export const handler = async (event, context) => {
12   let body;
13   let statusCode = 200;
14   const headers = {
15     "Content-Type": "application/json",
16   };
17       body = {
18         "Action": "Put an item",
19       };
20
21   await dynamo.send(
22     new PutCommand({
23         TableName: 'Events',
24         Item: {
25             Location: 'South Bend',
26             Month: 7,
27             EventName: 'Cool Summer',
28             Notes: 'It is fun!'
29         },
30     })
31 );
32
33   return {
34   statusCode,
35   body,
36   headers,
37   };
```

16. Execute the lambda function three more times to insert the following items to the **Events** table.

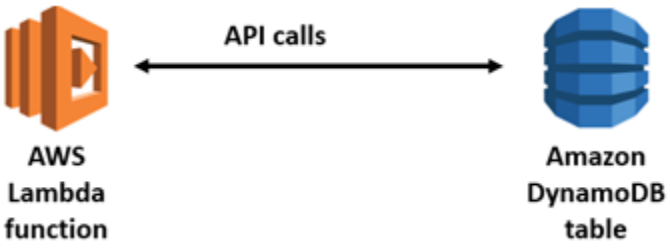
Location	Month	EventName	Notes	Budget
NYU	1	New Year		$90k
Purdue	9	Corn Grill	Open to the public	
Notre Dame	11	Football		

17. Check the DynamoDB Events table to see if the new items are added.

5.2 Retrieve data from DynamoDB using Lambda Function

Objectives:

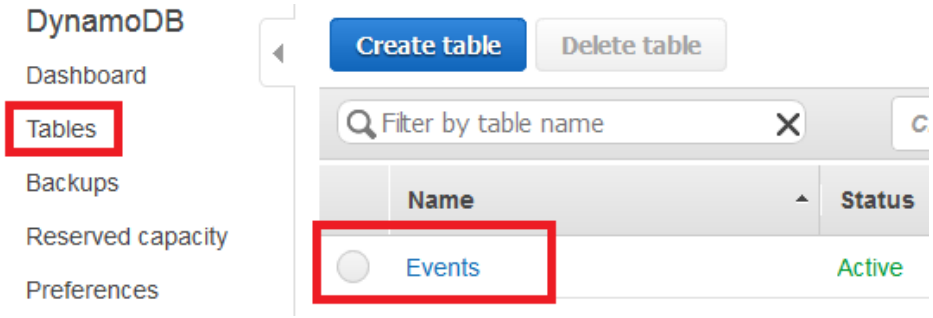
- Create a lambda function
- Retrieve data from a DynamoDB table with the lambda function



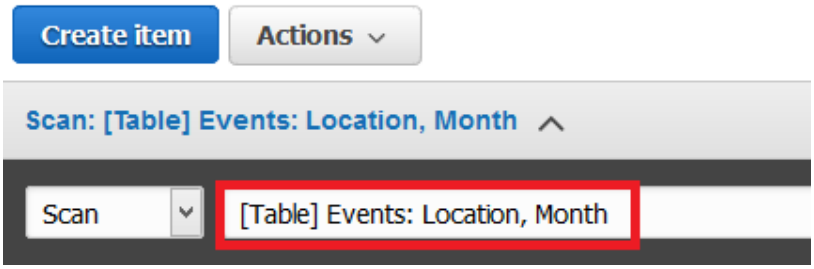
Note: This practice is created under an AWS Academy student account "LabRole". There is no need to set up policies. For regular AWS account users, a DynamoDB policy needs to be created and assigned to a role.

Part 1. Create a DynamoDB table if needed

1. Log onto AWS Console.
2. Make sure your Events Table is there. If it is deleted, you should create a new one following the instruction given in Section 4.3.



3. Remember, your Events Table has a composite primary key (Location and Month).



Part 2. Create a Lambda function

4. Create a new Lambda function.

5. Let's choose "Author from scratch". Assign a name to your function.
6. Expand "Change default existing role".
7. Pick "Use an existing role".
8. Choose "LabRole".
9. Click on "Create function".
10. This is the code we have now.

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

11. Replace the code (**index.mjs**) with the one given online (https://github.com/awsbookresource/files).

index.mjs | Execution results

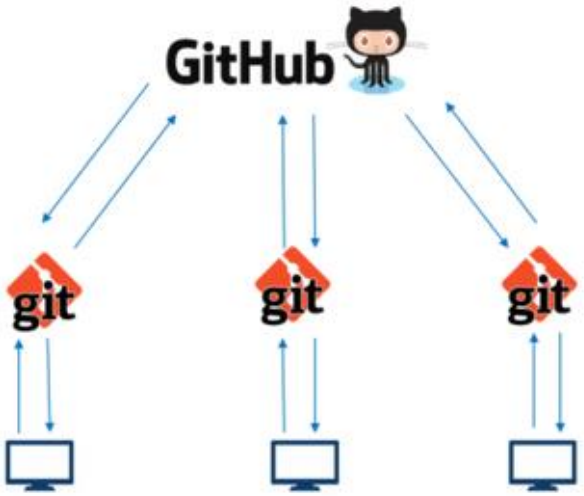
```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ScanCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const dynamo = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {
  let body;
  let statusCode = 200;
  const headers = {
    "Content-Type": "application/json",
  };

    body = await dynamo.send(
    new ScanCommand({ TableName: 'Events' })
    );
    body = body.Items;

  return {
  statusCode,
  body,
  headers,
  };
};
```

12. Deploy the changes.
13. Create a testing case and run it. You should see the result with retrieved items.
14. Terminate all services.

5.3 Git, GitHub, and Amplify

Objectives:

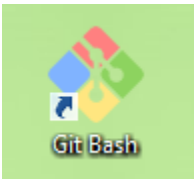
- Install Git Bash
- Create a GitHub account
- Upload source code file from a local repository to GitHub
- Deploy a static web site stored in GitHub with AWS Amplify



Part 1: Install Git and Create a GitHub account

Ignore Part 1 if you have installed Git and created GitHub account before.

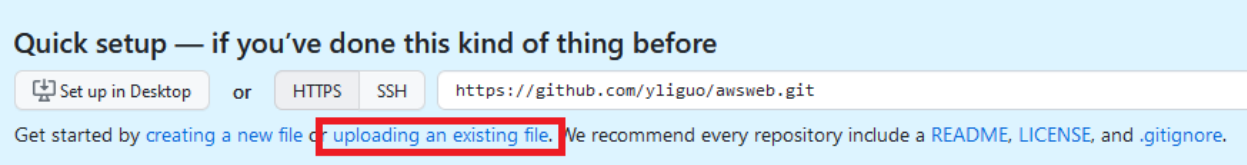
1. Download and install Git: https://git-scm.com/downloads.
2. Create a shortcut of Git Bash on your desktop (optional, for easy access).



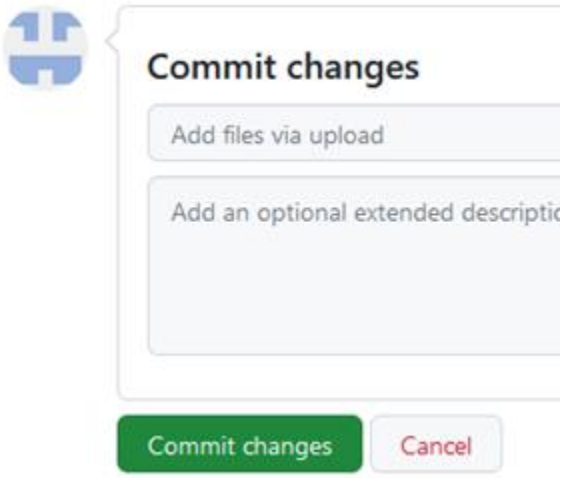
3. Go to https://github.com/
4. Create a GitHub account if you do not have one.

Part 2. Create a Git Repository

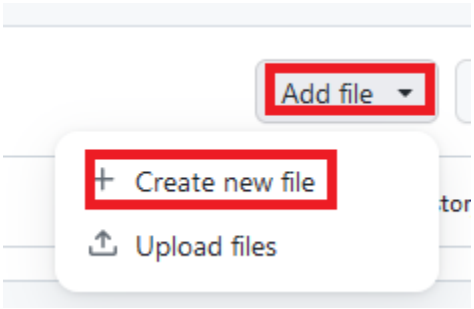
1. Download the zip file public_html.zip given online (https://github.com/awsbookresource/files). Unzip it.
2. Log onto GitHub.
3. Create a new repository.
4. Click uploading an existing file.



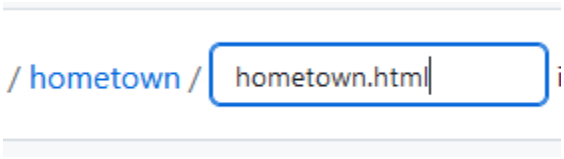
5. Choose the two files (index.html and Dumbledore.jpg) and click "Commit changes".



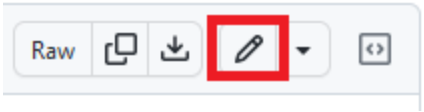
6. Click on "Add file", then "Create new file".



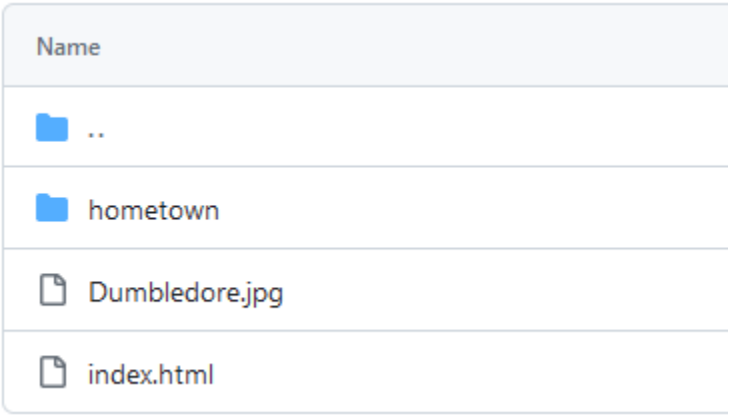
7. Type "**hometown/hometown.html**", then click "**Commit changes**"



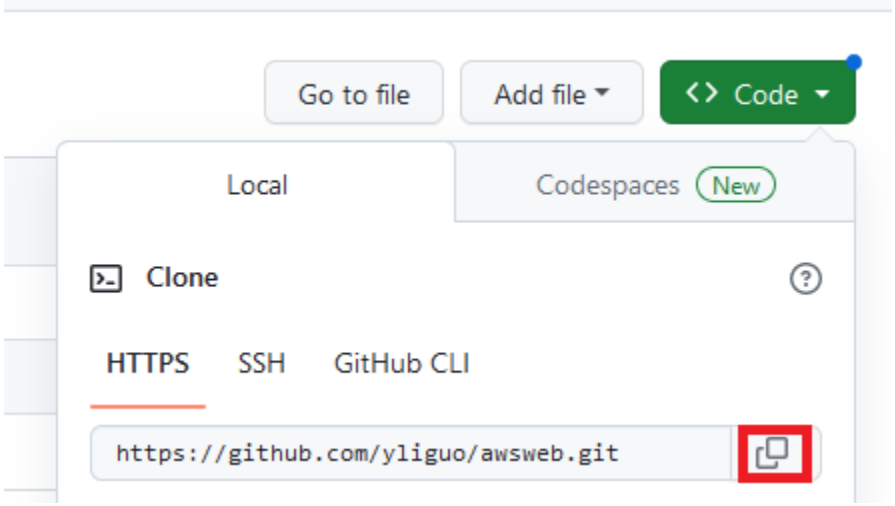
8. Click on file hometown.html, which is empty.
9. Click on "edit" symbol.



10. Copy the code of hometown.html you just downloaded and paste it into GitHub.
11. Commit the Changes.
12. Go back to your project directory, it looks like the following.



13. Copy its URL.



Part 2. Clone the remote repository to your local computer

13. Start **Git Bash** on your local computer.
14. Execute the following commands.

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

15. Execute the following command (replace url with the URL of your GitHub repository):

    $ Git clone url

16. Go inside the repository from your local computer.
17. Add a blank line to index.html.
18. From Git Bash, switch to this directory and execute the following commands to push the changes to GitHub:
    $git add -A
    $git commit -m "change a file"
    $git push

19. You might be asked to sign into GitHub or provide username and password.
20. If authentication fails, please do Steps 21-23. Otherwise go to Step 24.
21. Create Access Token on GitHub.
    - Click on your **GitHub profile icon** in the top right corner.
    - Click **Settings.**
    - From the menu shown on the left, click **Developer Settings.**
    - Click **Personal access tokens.**
    - Click **Generate new token.**
    - Add a note that will help you identify the scope of the access token to be generated.
    - **Choose the Expiration period** from the drop down menu.
    - Finally, **select the scopes you want to grant the corresponding access to the generated access token**. I will check **all of them**.
    - Finally click **Generate Token.**
    - Copy the token and save it on your desktop computer.

22. Execute the following command to reset authentication method.

    $ git remote set-url origin https://<token>@github.com/<user>/<repository>.git

    In the above command make sure to replace:
    - **<token>** with the personal access token you have copied in the previous step
    - **<user>** with your GitHub username
    - **<repository>** with the name of your GitHub repository

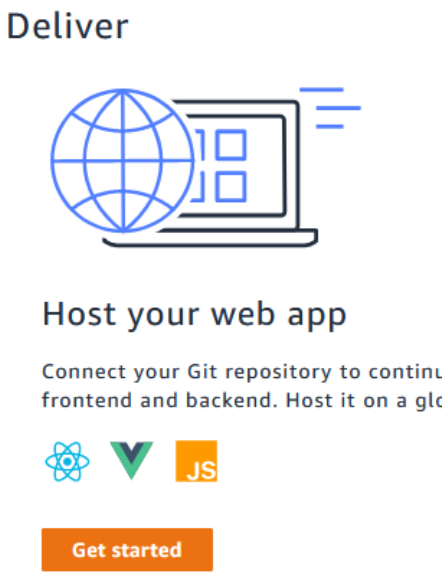
23. Execute the following command to push the change again.

    $ git push

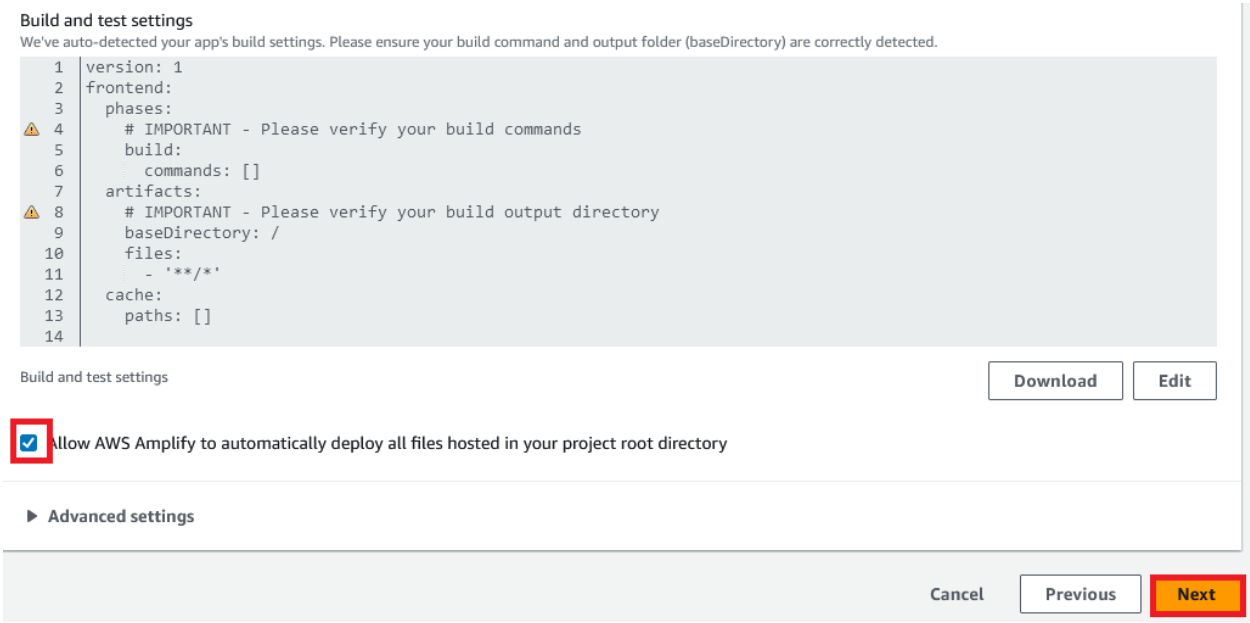
24. Check your repository on GitHub to see if it is updated.

Part 3. Enable web hosting with Amplify Console

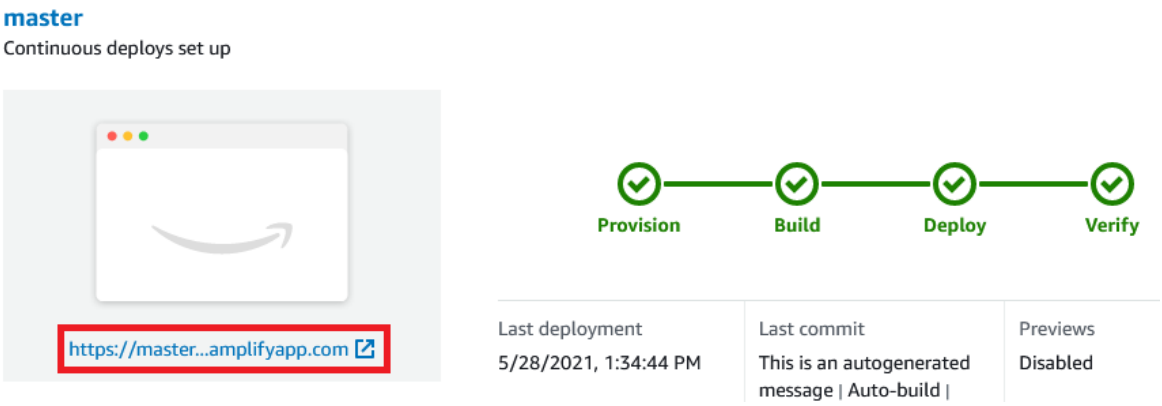
25. From AWS services page, launch the Amplify Console.
26. Click **Get Started** under Deploy with Amplify Console.
27. Select "Host web app". Click on "**Get started**".



28. Select the "**GitHub**" and click on "Continue".
29. Log onto GitHub.
30. Assign Amplify the permission to access all repositories.
31. Select the repository and branch to host your web pages.
32. Check "all auto deploy" and click "Next".



33. Click "Save and deploy".
34. When deployment completes, click on the web site URL.



Part 4: Modify the source code and push the changes to GitHub

35. Modify index.html, delete the blank line you added at Step 17.
36. Save the changes, push the changes to GitHub.
37. Refresh the web page a few minutes later.
38. Terminate all services.