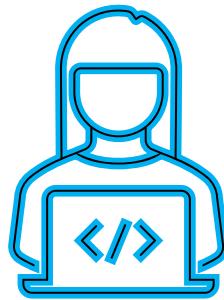


---

# AWS JUMPSTART: AN INTRODUCTORY GUIDE

---



JANET YU



# Table of Contents

|                                                             |           |
|-------------------------------------------------------------|-----------|
| Preface.....                                                | i         |
| <b>Chapter 1. Virtual Machines.....</b>                     | <b>1</b>  |
| 1.1 Linux Virtual Machine.....                              | 2         |
| 1.2 Windows Virtual Machine.....                            | 9         |
| <b>Chapter 2. Storage and Web Services.....</b>             | <b>13</b> |
| 2.1 S3 and Static Web Hosting .....                         | 14        |
| 2.2 Access S3 with CLI .....                                | 22        |
| <b>Chapter 3. Lambda Functions .....</b>                    | <b>25</b> |
| 3.1 Role, Policy, and Lambda Function.....                  | 26        |
| 3.2 API Gateway.....                                        | 34        |
| 3.3 Trigger and Destination .....                           | 41        |
| <b>Chapter 4. Database Services.....</b>                    | <b>58</b> |
| 4.1 Relational Database .....                               | 59        |
| 4.2 DocumentDB .....                                        | 66        |
| 4.3 DynamoDB .....                                          | 79        |
| <b>Chapter 5. Integrations .....</b>                        | <b>84</b> |
| 5.1 Write data to DynamoDB using Lambda Function .....      | 85        |
| 5.2 Retrieve data from DynamoDB using Lambda Function ..... | 89        |
| 5.3 Git, GitHub, and Amplify .....                          | 91        |



# Preface

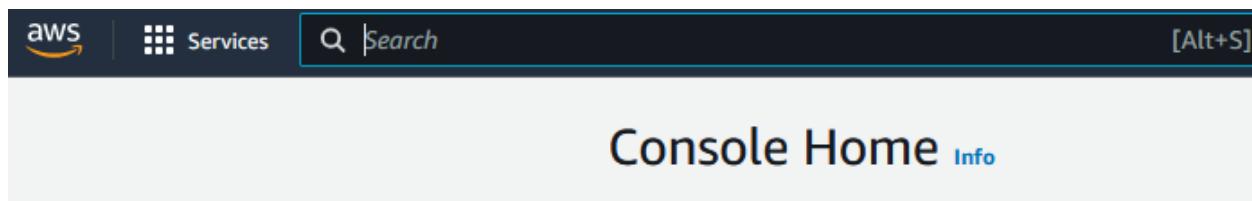
AWS (Amazon Web Services) is Amazon's offer of cloud computing solutions. It is now the world's largest cloud computing service provider. Its services include infrastructure (such as EC2, ECS, EKS), storage (such as S3, EBS, EFS), database (such as DocumentDB, DynamoDB, Aurora), analytics, machine learning, 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.

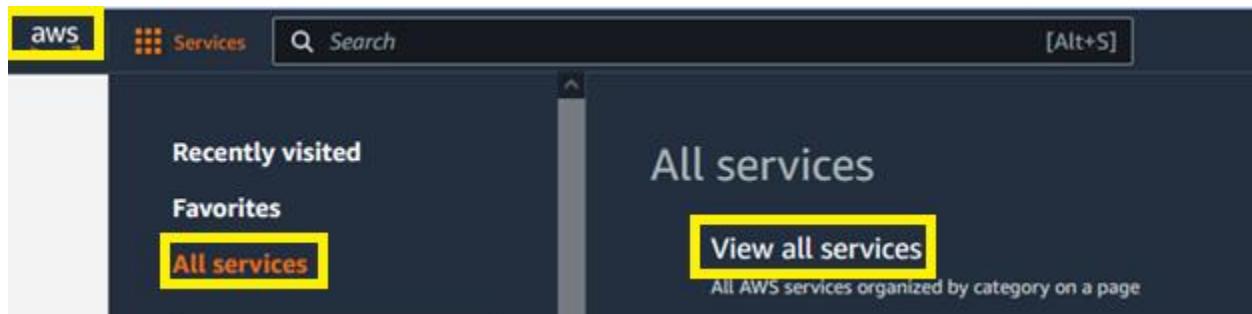
In Chapter 1, we will cover AWS infrastructure services. In Chapter 2, we will practice using AWS storage services and web services. In Chapter 3, we will learn AWS lambda functions. In Chapter 4, we will use AWS database services.

## 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.

## All services

| Services by category                                                                         |                                                                                                   |                                                                                                    |                                                                                                             |  |  |
|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|--|--|
|  Compute    |  Developer Tools |  Machine Learning |  AWS Cost Management     |  |  |
| EC2                                                                                          | CodeStar                                                                                          | Amazon SageMaker                                                                                   | AWS Cost Explorer                                                                                           |  |  |
| Lightsail                                                                                    | CodeCommit                                                                                        | Amazon Augmented AI                                                                                | AWS Budgets                                                                                                 |  |  |
| Lambda                                                                                       | CodeBuild                                                                                         | Amazon CodeGuru                                                                                    | AWS Marketplace                                                                                             |  |  |
| Batch                                                                                        | CodeDeploy                                                                                        | Amazon DevOps Guru                                                                                 | Subscriptions                                                                                               |  |  |
| Elastic Beanstalk                                                                            | CodePipeline                                                                                      | Amazon Comprehend                                                                                  | AWS Application Cost                                                                                        |  |  |
| Serverless Application                                                                       | Cloud9                                                                                            | Amazon Forecast                                                                                    | Profiler                                                                                                    |  |  |
| Repository                                                                                   | CloudShell                                                                                        | Amazon Fraud Detector                                                                              | AWS Billing Conductor                                                                                       |  |  |
| AWS Outposts                                                                                 | X-Ray                                                                                             | Amazon Kendra                                                                                      |                                                                                                             |  |  |
| EC2 Image Builder                                                                            | AWS FIS                                                                                           | Amazon Personalize                                                                                 |                                                                                                             |  |  |
| AWS App Runner                                                                               | CodeArtifact                                                                                      | Amazon Polly                                                                                       |                                                                                                             |  |  |
| AWS SimSpace Weaver                                                                          | Amazon CodeCatalyst                                                                               | Amazon Rekognition                                                                                 |                                                                                                             |  |  |
|  Containers | AWS AppConfig                                                                                     | Amazon Textract                                                                                    |                                                                                                             |  |  |
| Elastic Container Registry                                                                   | Amazon CodeWhisperer                                                                              | Amazon Transcribe                                                                                  |                                                                                                             |  |  |
| Elastic Container Service                                                                    | Application Composer                                                                              | Amazon Translate                                                                                   |                                                                                                             |  |  |
| Elastic Kubernetes Service                                                                   |                                                                                                   | AWS DeepComposer                                                                                   |                                                                                                             |  |  |
| Red Hat OpenShift Service on AWS                                                             | AWS IQ                                                                                            | AWS DeepLens                                                                                       |                                                                                                             |  |  |
|  Storage    | Managed Services                                                                                  | AWS DeepRacer                                                                                      |                                                                                                             |  |  |
| S3                                                                                           | Activate for Startups                                                                             | AWS Panorama                                                                                       |                                                                                                             |  |  |
| EFS                                                                                          | Support                                                                                           | Amazon Monitron                                                                                    |                                                                                                             |  |  |
| FSx                                                                                          |                                                                                                   | Amazon HealthLake                                                                                  |                                                                                                             |  |  |
| S3 Glacier                                                                                   |  Robotics      | Amazon Lookout for Vision                                                                          |                                                                                                             |  |  |
| Storage Gateway                                                                              | AWS RoboMaker                                                                                     | Amazon Lookout for Equipment                                                                       |                                                                                                             |  |  |
| AWS Backup                                                                                   |                                                                                                   | Amazon Lookout for Metrics                                                                         |                                                                                                             |  |  |
| <small>AWS Firewall Manager</small>                                                          |  Blockchain    | Amazon Lex                                                                                         |                                                                                                             |  |  |
|                                                                                              | Amazon Managed                                                                                    | Amazon Comprehend                                                                                  |                                                                                                             |  |  |
|                                                                                              |                                                                                                   | Medical                                                                                            |                                                                                                             |  |  |
|                                                                                              |                                                                                                   |                                                                                                    |  Front-end Web & Mobile  |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | AWS Amplify                                                                                                 |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | AWS AppSync                                                                                                 |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Device Farm                                                                                                 |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Amazon Location Service                                                                                     |  |  |
|                                                                                              |                                                                                                   |                                                                                                    |  Application Integration |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Step Functions                                                                                              |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Amazon AppFlow                                                                                              |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Amazon EventBridge                                                                                          |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Amazon MQ                                                                                                   |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Simple Notification Service                                                                                 |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Simple Queue Service                                                                                        |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | SWF                                                                                                         |  |  |
|                                                                                              |                                                                                                   |                                                                                                    | Managed Apache Airflow                                                                                      |  |  |
|                                                                                              |                                                                                                   |                                                                                                    |  Business Applications |  |  |

## Downloading the example code

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

## Chapter 1. Virtual Machines

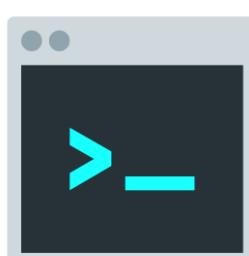
AWS's Elastic Compute Cloud (EC2) is Amazon's solution to cloud-based virtual machines. It provides a wide selection of instance types with customizing CPU, memory, storage, network, and operating systems to meet different business needs. In this chapter, we will practice creating a Linux virtual machine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



ssh



aws ec2

### Part 1. Create a Linux virtual machine

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

#### Find Services

You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

#### ▼ All services



Blockchain  
Amazon Managed Blockchain

4. Click “Launch instance”.

#### Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

**Launch instance ▼**

5. Assign a name.

Name and tags [Info](#)

Name

AWSbookC1S1

6. Select “Red Hat”.



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

Create key pair X

Key pair name  
Key pairs allow you to connect to your instance securely.

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

RSA  
RSA encrypted private and public key pair

ED25519  
ED25519 encrypted private and public key pair (Not supported for Windows instances)

Private key file format

.pem  
For use with OpenSSH

.ppk  
For use with PuTTY

**⚠️** When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

[Cancel](#) 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”.

[View all instances](#)

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

**Instances (1) [Info](#)**

*Find instance by attribute or tag (case-sensitive)*

| <input type="checkbox"/> | Name        | Instance ID         | Instance state | Instance type |
|--------------------------|-------------|---------------------|----------------|---------------|
| <input type="checkbox"/> | AWSbookC1S1 | i-01b57615a73e80af2 | Running        | t2.micro      |

## 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**.

[EC2 Instance Connect](#)    [Session Manager](#)    **SSH client**    [EC2 Serial Console](#)

Instance ID  
 i-0404f4e4a88ddd6c6

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is iusbaws.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.  
 chmod 400 iusbaws.pem
4. Connect to your instance using its Public DNS:  
 ec2-54-167-255-224.compute-1.amazonaws.com

Example:  
 ssh -i "iusbaws.pem" ec2-user@ec2-54-167-255-224.compute-1.amazonaws.com

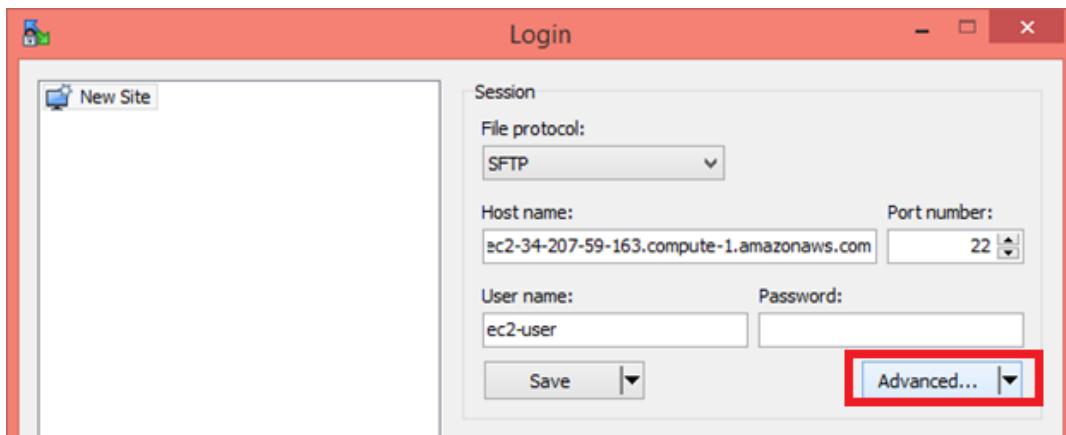
14. Start WinSCP. On Hostname, paste the one you see on Connection Page (Step 4).

4. Connect to your instance using its Public DNS:  
ec2-54-167-255-224.compute-1.amazonaws.com

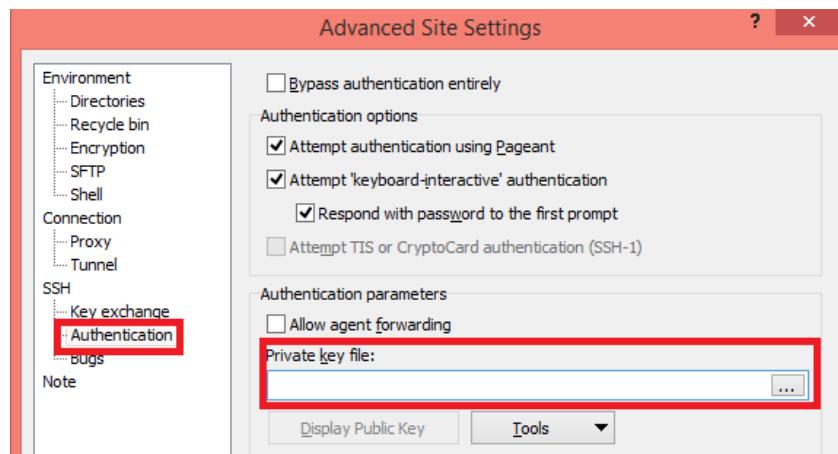
15. Username should be “**ec2-user**”.

ssh -i "iusbaws.pem" ec2-user@ec2-54-167-255-224.compute-1.amazonaws.com

16. Click “Advanced”.

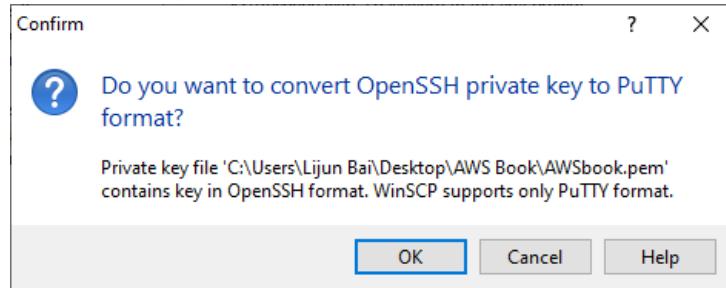


17. Click “Authentication”. Browse to Private key file you downloaded earlier.

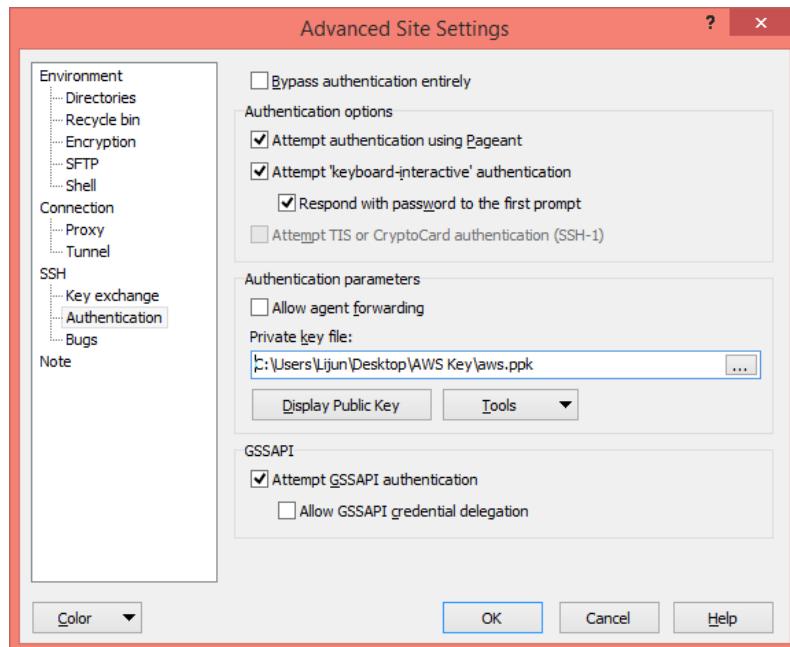


18. To show **.pem** file, select all file types. Then select the private key file (**.pem**) you saved at Step 8.

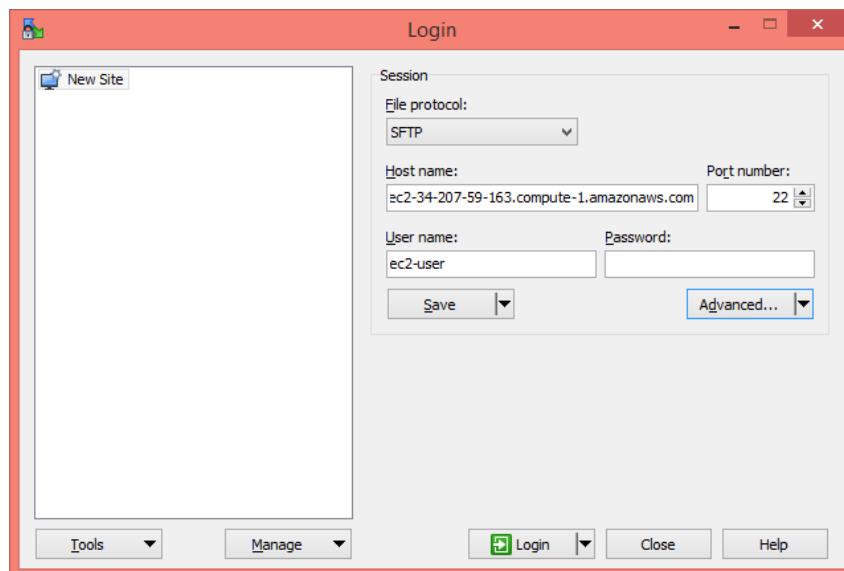
19. You will be asked to convert **.pem** to **.ppk** format. Click “OK”.



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).

4. Connect to your instance using its Public DNS:  
 ec2-54-167-255-224.compute-1.amazonaws.com

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.

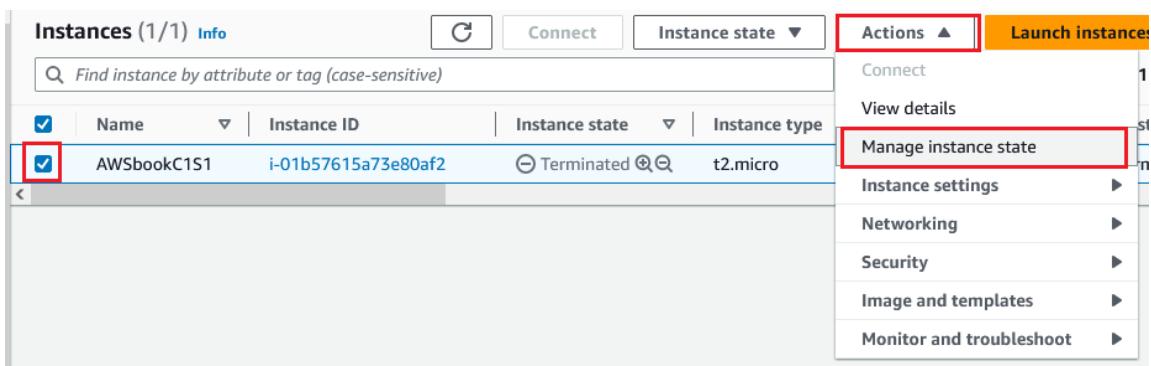


28. Click Open, you will be asked for username (**ec2-user**), which can be found here.

ec2-34-207-59-163.compute-1.amazonaws.com  
nple:  
ssh -i "aws.pem" **ec2-user**@ec2-34-207-59-163.compute-1.amazonaws.com

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”.



## 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

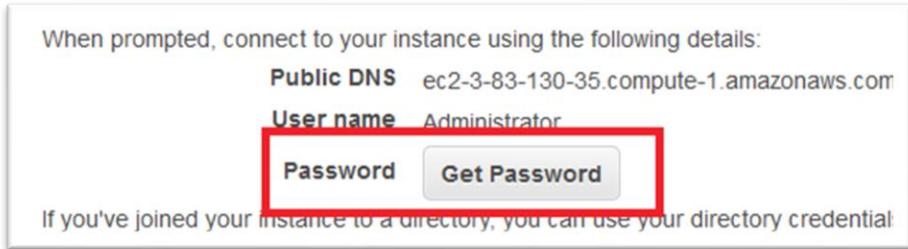
1. Repeat Steps 1-5 in Section 1.1. Instead of choosing Red Hat, you should select a windows Free Tier machine.
2. Select your **existing** key pair, you do not need to create a new key pair.
3. Click “Launch instance”.



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

| Instances (1/2) <a href="#">Info</a>                                    |             | <a href="#">C</a>   | <a href="#">Connect</a>                             | <a href="#">Instance state ▾</a> |
|-------------------------------------------------------------------------|-------------|---------------------|-----------------------------------------------------|----------------------------------|
| <input type="text"/> Find instance by attribute or tag (case-sensitive) |             |                     |                                                     |                                  |
|                                                                         | Name        | Instance ID         | Instance state                                      | Instance type                    |
| <input type="checkbox"/>                                                | AWSbookC1S1 | i-01b57615a73e80af2 | <span>Terminated</span> <a href="#">Get Details</a> | t2.micro                         |
| <input checked="" type="checkbox"/>                                     | AWSbookC1S2 | i-08157de6f22e94fda | <span>Running</span> <a href="#">Get Details</a>    | t2.micro                         |

6. Click “RDP client” then click “Get Password”.



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

## Get Windows password Info

Use your private key to retrieve and decrypt the initial Windows administrator password for this instance.

Instance ID  
i-03dd0e913893afc88 (AWSbookC1S2)

Key pair associated with this instance  
AWSbook

Private key  
Either upload your private key file or copy and paste its contents into the field below.

**AWSbook.pem**  
1.678KB

Private key contents - *optional*

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAtWmEGh5VpfNMvGBuLs/nFCiZcTbXM0kO4NfSZHdvMQ628aAa
w65DibycubE2bvi0y3tNbM7skkp5HQFd7JFaClgm/w4+RpN2KSXkKP8/M4wJKBhc
1S6AFaVPKUTEjeXH/FgKK8qDFzmb7WayEqT0F4Z9r04g2zuNEPoUxGiMXRaYMjq
KNcjvhOPVNeJ71SlyoILUOnQTCFUfLwuN5qFKLxNuSYtoiNevCBq2JFJZ/DYHRM7
IhG7hXZtUIOD1x/db0e31Bbb5/0aeuDP+cxOqQT8SJ/g+ZGJiLKmiBV6TMrgfCho
JFYzAg3MBtNxOGg8cNATU6W2PIU3Sj2QtrmJ9wIDAQABAoIBAQCGXaHfqoorgTp
6FyTxZ9U+gE0rrbUll+pj5Egoqmzvz9XVguc1gzmDtJSY2R66zEV6bzeE6F7dTeh
eSq1Fq+F1Dp1/Xm40m3ZalxE1VFdf6IkIVujBG8QGUG5wlqG8yOdzyOxbayM3jmU
```

**Decrypt password**

- Copy the **password**. Close the window if needed.

## Connect to your instance

**Connection method**  A standalone RDP client [\(i\)](#)  
 Session Manager [\(i\)](#)

You can connect to your Windows instance using a remote desktop client of your choice, and download and running the RDP shortcut file below:

[Download Remote Desktop File](#)

When prompted, connect to your instance using the following details:

**Public DNS** ec2-3-83-130-35.compute-1.amazonaws.com

**User name** Administrator

**Password** XbKiengN&TKKd\$@T\$g1Sycw=uZHHuYn!

If you've joined your instance to a directory, you can use your directory credentials to connect

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

**Connection method**  A standalone RDP client [\(i\)](#)  
 Session Manager [\(i\)](#)

You can connect to your Windows instance using a remote desktop client of your choice, and download and running the RDP shortcut file below:

[Download Remote Desktop File](#)

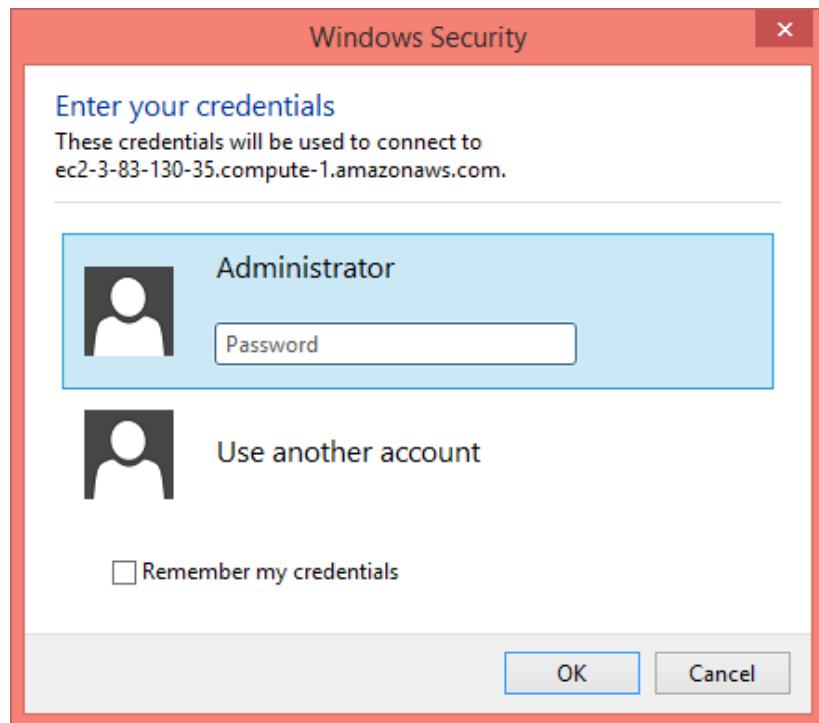
When prompted, connect to your instance using the following details:

**Public DNS** ec2-3-83-130-35.compute-1.amazonaws.com

**User name** Administrator

**Password** [Get Password](#)

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”.

A screenshot of the AWS EC2 Instances page. It shows one instance named "AWSbookC1S2" with an ID of "i-03dd0e913893afc88" in a "Running" state (indicated by a green checkmark). The "Actions" menu is open for this instance, with "Instance settings" highlighted. Other options in the menu include "Connect", "View details", "Manage instance state", "Networking", "Security", "Image and templates", and "Monitor and troubleshoot". A red box highlights the "Actions" button and the "Instance settings" option.

## Chapter 2. Storage and Web Services

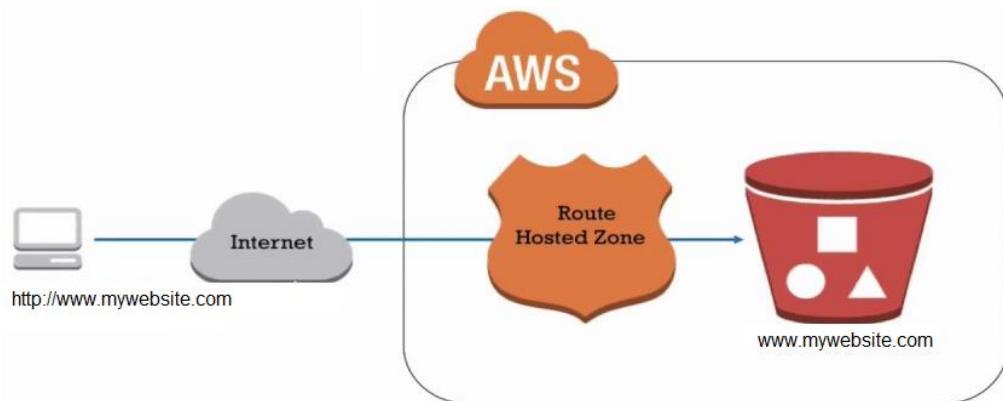
Amazon's cloud-based storage includes Simple Storage Service (S3), Elastic File System (EFS), Elastic Block Store (EBS), etc. In this chapter, we will practice using S3 service to host a static website.



## 2.1 S3 and Static Web Hosting

### Objectives:

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

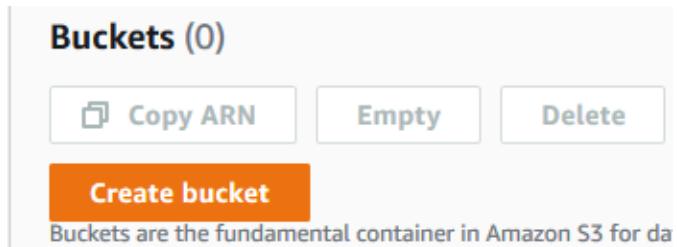


### Part 1. Create and test an AWS storage S3 instance (container)

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.

## General configuration

Bucket name

awsbookc2s1

Bucket name must be globally unique and must not contain spaces or uppercase letters. See [rules for bucket naming](#)

AWS Region

US East (N. Virginia) us-east-1



**Copy settings from existing bucket - optional**

Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

5. Choose “ACLs enabled”.

### Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

**ACLs disabled (recommended)**

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

**ACLs enabled**

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

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

### Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)



#### **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

**Block public access to buckets and objects granted through new access control lists (ACLs)**

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

**Block public access to buckets and objects granted through any access control lists (ACLs)**

S3 will ignore all ACLs that grant public access to buckets and objects.

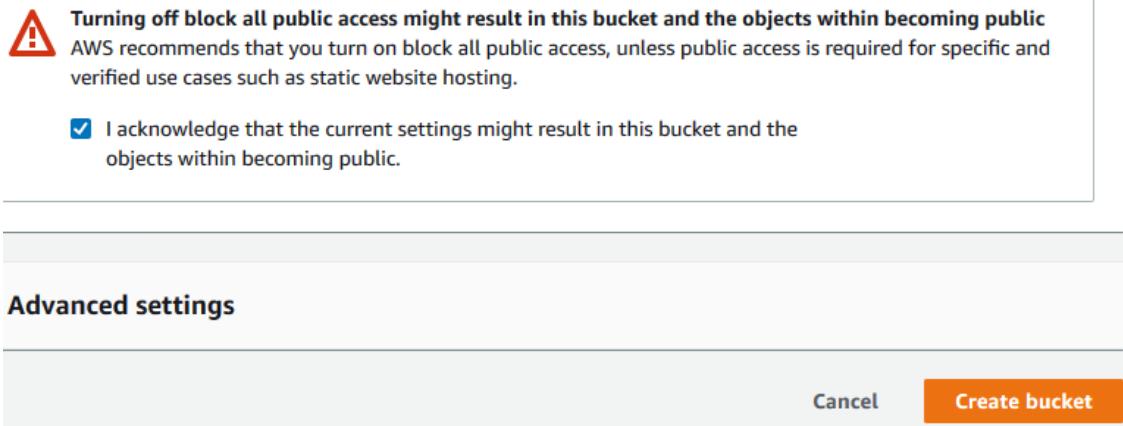
**Block public access to buckets and objects granted through new public bucket or access point policies**

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

**Block public and cross-account access to buckets and objects through any public bucket or access point policies**

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

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



8. Click the bucket you just created.

A screenshot of the AWS Buckets list page. The title is "Buckets (1)" with an "Info" link. A sub-header says "Buckets are containers for data stored in S3. [Learn more](#)". There is a search bar with placeholder text "Find buckets by name". The main table has columns "Name" and "AWS Region". One row is shown: "awsbookc2s1" under "Name" and "US East (N. Virginia) us-east-1" under "AWS Region".

9. Click “Create folder”.

A screenshot of the AWS Objects list page. The title is "Objects (0)". A sub-header says "Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permission." Below are buttons for "Copy S3 URI", "Copy URL", "Download", "Open", "Delete", "Actions", and "Create folder" (which is highlighted with a red box). There is also a "Upload" button. A search bar with placeholder text "Find objects by prefix" is present. The main table has columns "Name", "Type", "Last modified", and "Size".

10. Create a new folder.

**Folder**

Folder name  /  
Folder names can't contain "/". See rules for naming [\[?\]](#)

**Server-side encryption**  
Server-side encryption protects data at rest.

i The following encryption settings apply only to the folder object and not to sub-folder objects.

Encryption key type [Info](#)  
 Amazon S3 managed keys (SSE-S3)  
 AWS Key Management Service key (SSE-KMS)

[Cancel](#) Create folder

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

| Files and folders                                 |        | Configuration |
|---------------------------------------------------|--------|---------------|
| <b>Files and folders (1 Total, 32.4 KB)</b>       |        |               |
| <input type="text"/> <a href="#">Find by name</a> |        |               |
| Name                                              | Folder | Type          |
| Math.png                                          | -      | image/png     |

12. Click the file and copy “Object URL”.

Object URL

<https://awsbookc2s1.s3.amazonaws.com/images/Math.png>

13. Paste the URL into a web browser. Your access will be denied.

14. Click “Permissions” for this file, then “Edit”

Properties Permissions Versions

**Access control list (ACL)**  
Grant basic read/write permissions to AWS accounts [Learn more \[?\]](#)

Edit

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

Object owner       Read       Read  
Canonical ID:  6023b3756f  
1c3abbe665b683c22f87f7e3e1  
a6b06dd937fd87fbea0d5bf074  
fb

Everyone (public access)       ⚠️ Read       Read  
Group:  http://acs.amazonaws.com/groups/global/AllUsers

Authenticated users group       Read       Read  
(anyone with an AWS account)  
Group:  http://acs.amazonaws.com/groups/global/AuthenticatedUsers

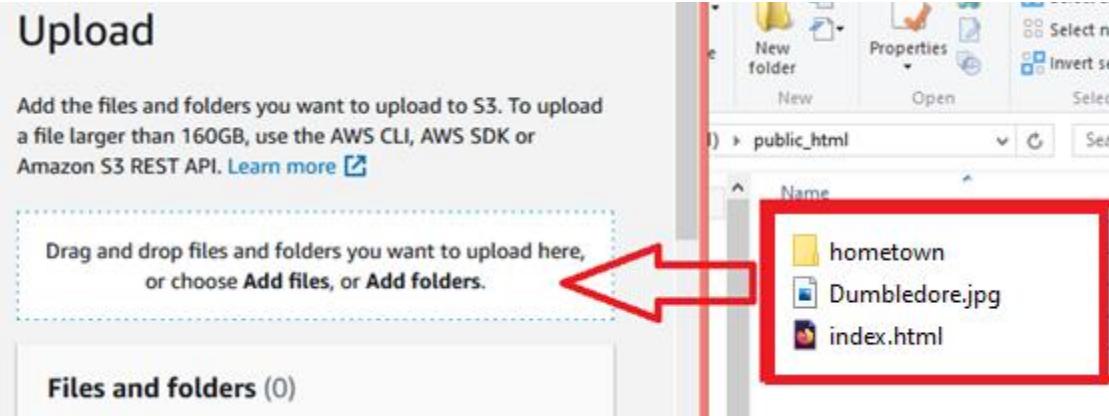
⚠️ When you grant access to the Everyone or Authenticated users group grant access this object.  
[Learn more ↗](#)

I understand the effects of these changes on this object.

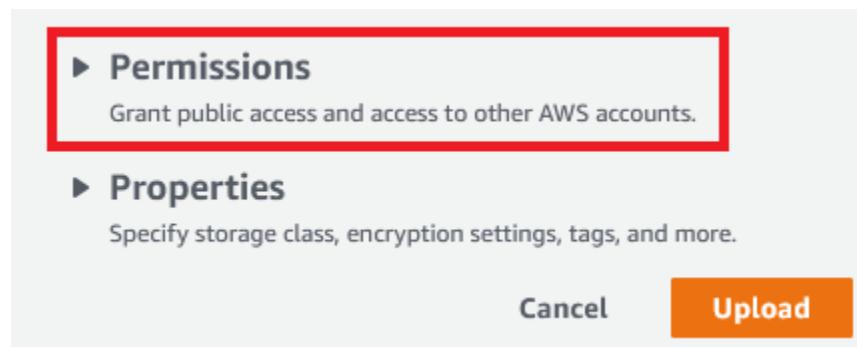
16. Copy and paste your URL into a web browser. You should be able to see the file now.

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

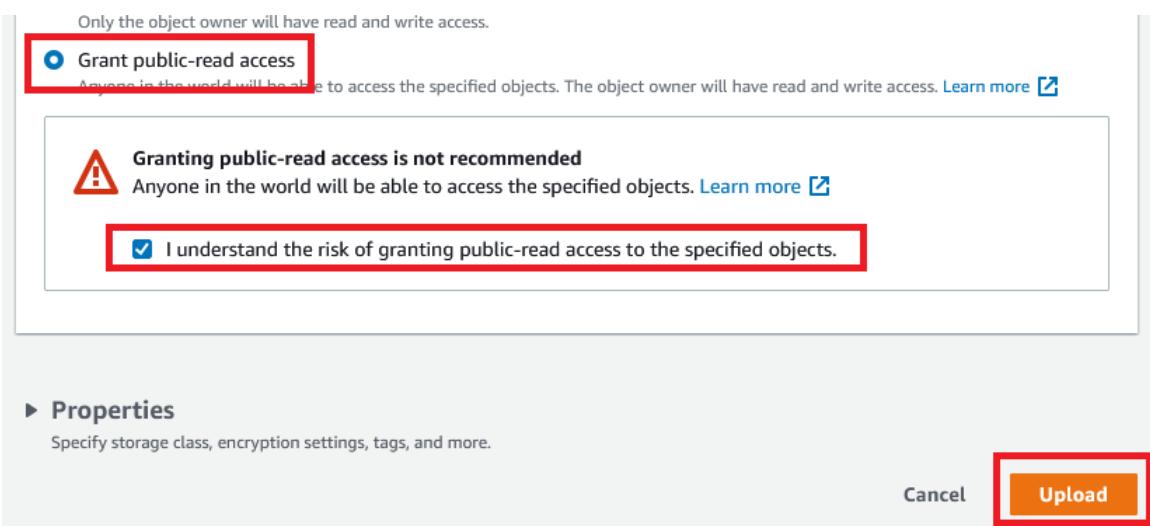
17. Download the zip file provided online (<https://github.com/awsbookresource/files>). Unzip it.
18. Go back to your Amazon S3 home. Click on the bucket you created.
19. Now you are inside the bucket (outside of the folder you created). Click “**Upload**”.
20. Upload **all the unzipped files** (without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

| Objects (4)                                                                                                                                                                                                             |  |                             |                          |                          |                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|-----------------------------|--------------------------|--------------------------|----------------------|
| Objects are the fundamental entities stored in Amazon S3. You can use <a href="#">Amazon S3 inventory</a> to get a list of all objects in your bucket. To explicitly grant them permissions, <a href="#">Learn more</a> |  |                             |                          |                          |                      |
|                                                                                                                                                                                                                         |  | <a href="#">Copy S3 URI</a> | <a href="#">Copy URL</a> | <a href="#">Download</a> | <a href="#">Open</a> |
| <a href="#"><b>Upload</b></a>                                                                                                                                                                                           |  |                             |                          |                          |                      |
| <input type="text"/> <i>Find objects by prefix</i>                                                                                                                                                                      |  |                             |                          |                          |                      |
| <input type="checkbox"/> <b>Name</b> <span style="float: right;">▲ ▼</span> <b>Type</b> <span style="float: right;">▼</span> <b>Last modified</b>                                                                       |  |                             |                          |                          |                      |
| <input type="checkbox"/> <a href="#">Dumbledore.jpg</a> jpg May 23, 2023, 13:07:18 (UTC-04:00)                                                                                                                          |  |                             |                          |                          |                      |
| <input type="checkbox"/> <a href="#">hometown/</a> Folder -                                                                                                                                                             |  |                             |                          |                          |                      |
| <input type="checkbox"/> <a href="#">images/</a> Folder -                                                                                                                                                               |  |                             |                          |                          |                      |
| <input type="checkbox"/> <a href="#">index.html</a> html May 23, 2023, 13:07:19 (UTC-04:00)                                                                                                                             |  |                             |                          |                          |                      |

24. On your bucket page, click “Properties” tab, move all the way down. Click Edit “Static website hosting”.

**Static website hosting**

Use this bucket to host a website or redirect requests. [Learn more](#)

[Edit](#)

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

**Static website hosting**

Disable

Enable

**Hosting type**

Host a static website

Use the bucket endpoint as the web address. [Learn more](#)

Redirect requests for an object

Redirect requests to another bucket or domain. [Learn more](#)

**Index document**

Specify the home or default page of the website.

**index.html**

26. Copy the website URL as shown below.

**Static website hosting**

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting  
**Enabled**

Hosting type  
**Bucket hosting**

Bucket website endpoint  
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)

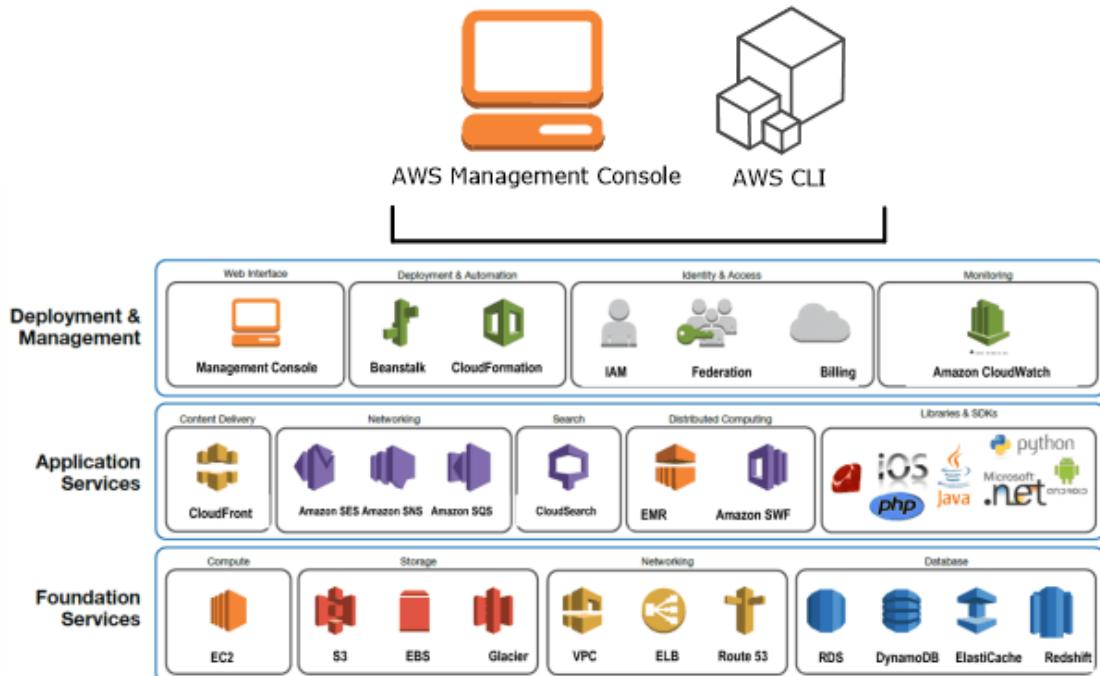
 <http://awsbookc2s1.s3-website-us-east-1.amazonaws.com>

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

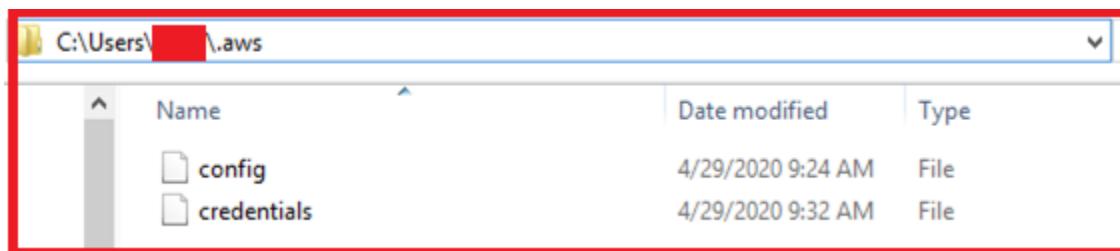
6. Execute the following command to configure AWS CLI.

## aws configure

7. AWS Access KEY ID and AWS Secret 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/bPxRfijCYEXAMPLEKEY
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.

```
Credentials
AWS Access
Session started at: 2020-04-29T12:13:42-0700
Session to end at: 2020-04-29T15:13:42-0700
Remaining session time: 2h50m35s

AWS Starter account
Term: 363 days 01:11:20

AWS CLI:
Copy and paste the following into ~/.aws/credentials
[default]
aws_access_key_id=ASIAY7IFGTLAXK5QVMXX
aws_secret_access_key=yih5+K3XjvHb2cgevXIGLsyfJmZxb8nGCz8+Sc9Z
aws_session_token=FwoGZXIxVYXdzEN3//////////wEaDNFHSMGmONlyDbXQnCK7AbwOvTlscdtm9CD2i6IXM17J/AT5r7gB1ApnkTqv/chsnMnqH3h5XlZlpT3zMABCdvT4ggu6
j9uI0WlhSUGsh2ykj0+Q18CCGeEwmWdQRIuMdSkivxKkjUKNy+ayd5m+fYhROXA80qOsZCn17pUfPhbhYNrx+rgVtzdTMrAgCHLvoPh+c4B+9RL0DbPw+QihbGof7lys8rag0xvQg!t
xo9Saqbo8THZDvMupU6dKfIZtzv8dNpYjXT+1M74wo560n9QuyLfBW2UsEqo1dvKO7R0flyJeuUtEoYD470dFJN3eMrqCFHBwKkOkwy1ceDAL6qg==
```

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.

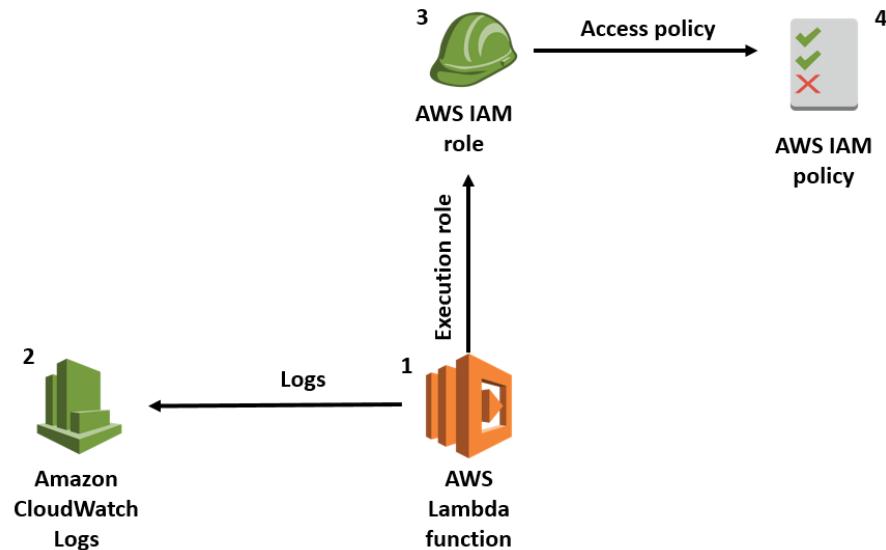


**AWS Lambda**

## 3.1 Role, Policy, and Lambda Function

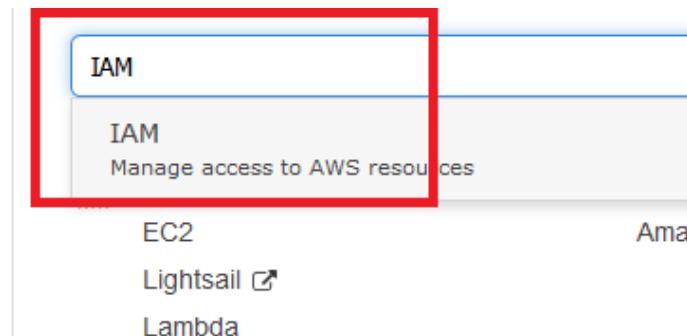
### Objectives:

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



### Part 1. Create an AWS role

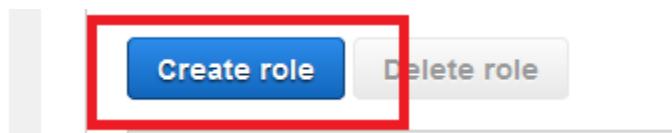
1. Log onto Aws Console.
2. At your AWS Console home. Search “IAM”, then click “IAM”. You can also find it under “Security, Identity, & Compliance”.



3. Click Roles.

The screenshot shows the AWS IAM Dashboard. On the left, there's a sidebar with 'Dashboard' at the top, followed by a '▼ Access management' section containing 'Groups', 'Users', and 'Roles'. The 'Roles' option is highlighted with a red box. To the right, there's a list of users with one entry: 'User: arn:aws:sts::616875596'. Below that is a section titled 'IAM Resources' with 'Users: 0' and 'Groups: 0'.

4. Click "Create role".



5. Under "AWS service", click "Lambda".

The screenshot shows the 'Choose a use case' section. It has two main options: 'AWS service' (EC2, Lambda and others) and 'Another AWS account' (Belonging to you or 3rd party). Below these, under 'Common use cases', are 'EC2' (Allows EC2 instances to call AWS services on your behalf.) and 'Lambda' (Allows Lambda functions to call AWS services on your behalf.). The 'Lambda' option is highlighted with a red box.

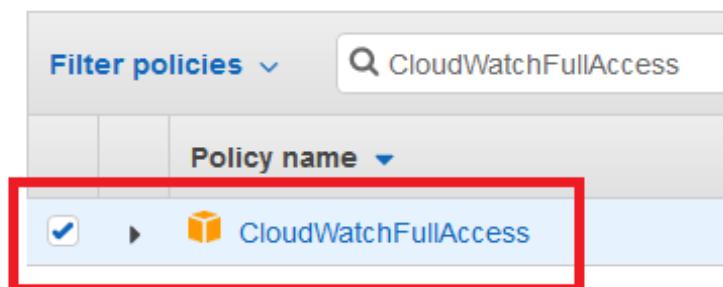
6. Click "Next".



7. In the search box, search "AWSLambdaExecute" policy. Select the policy.



8. Search “CloudWatchFullAccess” and select it.



9. Click “Next”.

[Cancel](#)    [Previous](#)    **Next**

10. Click “Next: Review”.

11. Assign your **aws role a name**, **remember it**. We are going to use it many times.

### Role details

#### Role name

Enter a meaningful name to identify this role.

LabRole

Maximum 64 characters. Use alphanumeric and '+,-,.,@-' characters.

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

## Step 2: Add permissions

Edit

Permissions policy summary

| Policy name          | Type        | Attached as        |
|----------------------|-------------|--------------------|
| CloudWatchFullAccess | AWS managed | Permissions policy |
| AWSLambdaExecute     | AWS managed | Permissions policy |

## Tags

### Add tags - optional Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add tag

You can add up to 50 more tags.

Cancel

Previous

Create role

## Part 2. Create a Lambda function

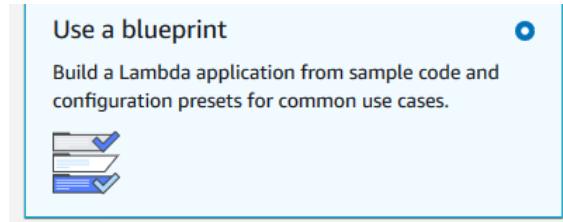
13. Go back to your AWS console page, click **Lambda** under Compute.



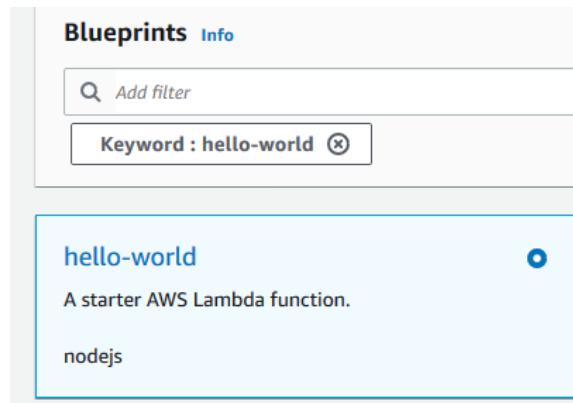
14. Click “Create function”.



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.

**Function name**  
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime**

**Architecture**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
 Create a new role with basic Lambda permissions  
 Use an existing role  
 Create a new role from AWS policy templates

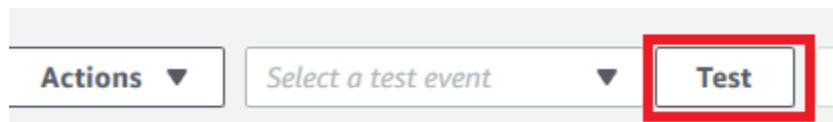
**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to  
  
[View the LabRole role](#) on the IAM console.

18. Click “Create function”.

19. Examine your code `index.js`. It takes three parameters from an HTTP call.

```
index.js
1 console.log('Loading function');
2
3 exports.handler = async (event, context) => {
4     //console.log('Received event:', JSON.stringify(event, null, 2));
5     console.log('value1 =', event.key1);
6     console.log('value2 =', event.key2);
7     console.log('value3 =', event.key3);
8     return event.key1; // Echo back the first key value
9     // throw new Error('Something went wrong');
10};
```

20. Click “Test”.



21. Assign a test Event name.

Create new test event

Edit saved test events

Event template

Hello World

Event name

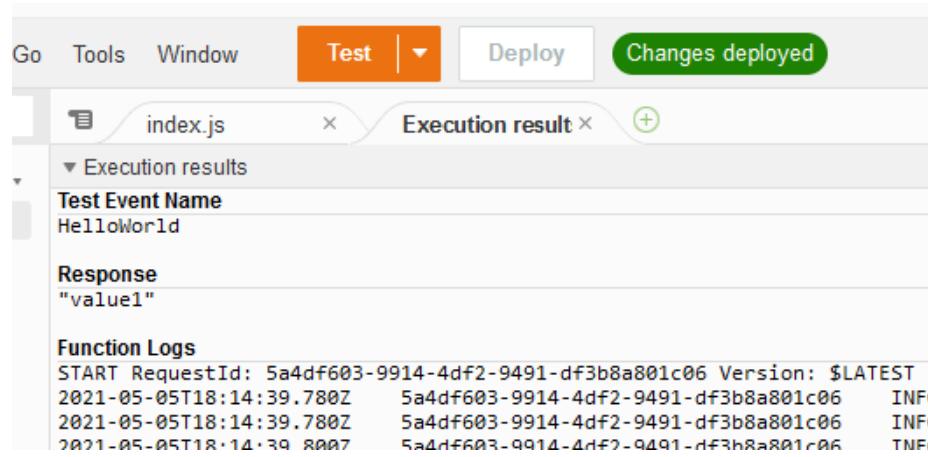
test1

```
1 [ ]
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 ]
```

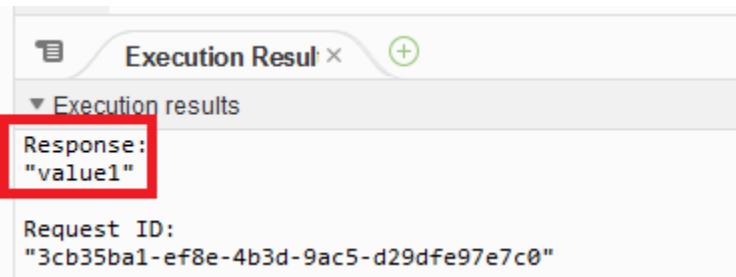
22. Click “Save”.



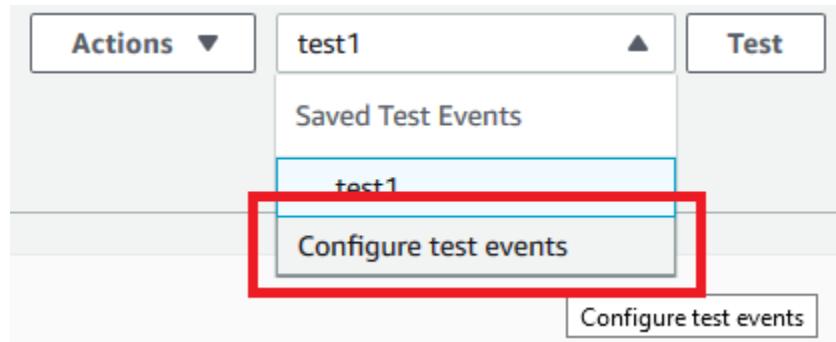
23. When your test event is selected, Click “Test”.



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.

## Event JSON

```
1 {  
2   "key1": "myFirstname",  
3   "key2": "myLastname",  
4   "key3": "myLocation"  
5 }
```

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>**



The screenshot shows the AWS Lambda function editor. The left pane displays the code for **index.js**:

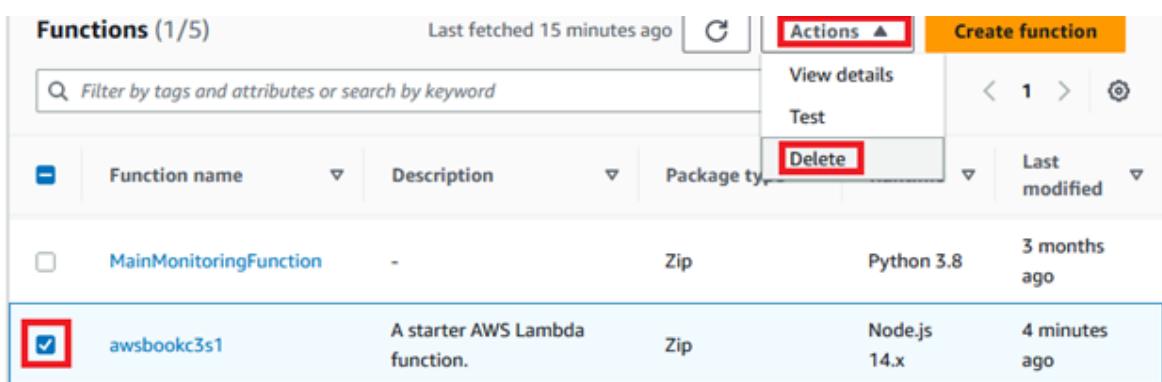
```
1 console.log('Loading function');  
2  
3 exports.handler = async (event, context) => {  
4     //console.log('Received event:', JSON.stringify(event, null, 2));  
5     console.log('value1 =', event.key1);  
6     console.log('value2 =', event.key2);  
7     console.log('value3 =', event.key3);  
8     return "Hello " + event.key1 + " " + event.key2 + " from " + event.key3;  
9     // throw new Error('Something went wrong');  
10};
```

The right pane shows the **Execution results** tab, which is currently empty.

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”.



The screenshot shows the AWS Lambda **Functions** page. It lists two functions:

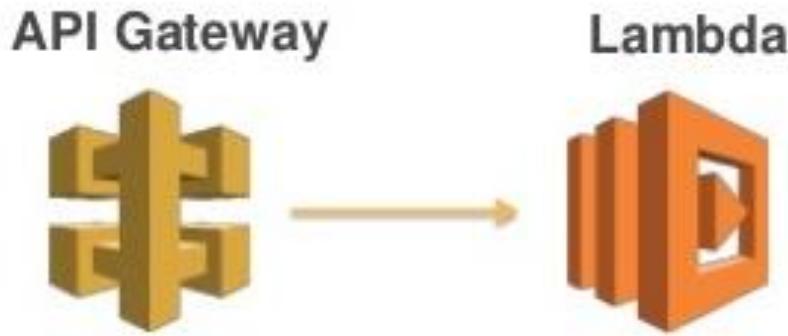
| Function name                                   | Description                    | Package type | Last modified                 |
|-------------------------------------------------|--------------------------------|--------------|-------------------------------|
| MainMonitoringFunction                          | -                              | Zip          | Python 3.8<br>3 months ago    |
| <input checked="" type="checkbox"/> awsbookc3s1 | A starter AWS Lambda function. | Zip          | Node.js 14.x<br>4 minutes ago |

The **awsbookc3s1** row has a red box around its checkbox. The **Actions** dropdown menu for this row is open, with the **Delete** button highlighted by a red box.

## 3.2 API Gateway

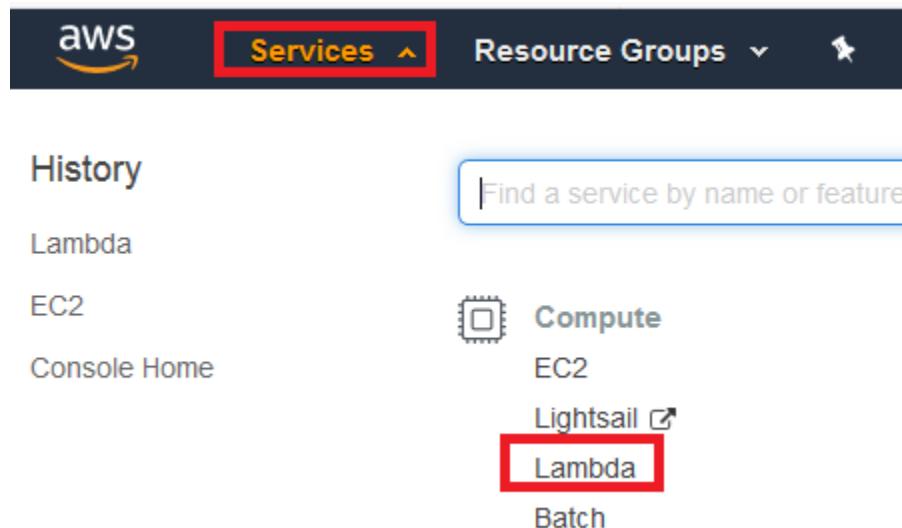
### Objectives:

- 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**”.

# Create function Info

AWS Serverless Application Repository applications have moved to [Create application](#).

## Author from scratch

Start with a simple Hello World example.

4. Assign a name.

### Function name

Enter a name that describes the purpose of your function.

azurebookc3s2

Use only letters, numbers, hyphens, or underscores with no spaces.

### Runtime Info

Choose the language to use to write your function. Note that the console code editor supports Node.js, Python, Java, C#, Go, and Docker.

Node.js 18.x

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

### Change default execution role

#### Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- Create a new role with basic Lambda permissions
- Use an existing role
- Create a new role from AWS policy templates

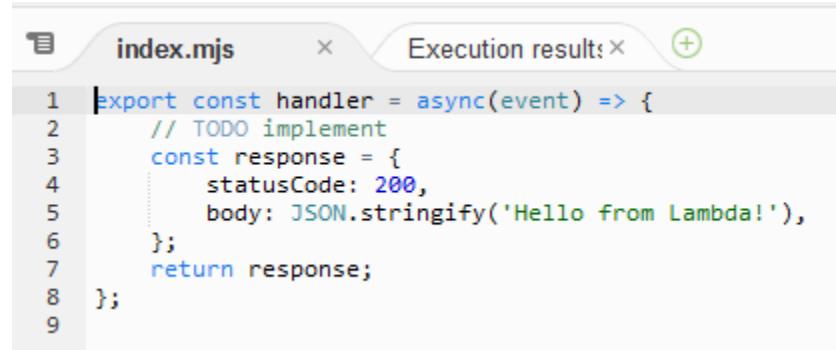
#### Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission

LabRole

[View the LabRole role](#) on the IAM console.

6. Click “Create function”.
7. Examine your code [index.mjs](#).



```
index.mjs
1 export const handler = async(event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Lambda!'),
6     };
7     return response;
8 }
9
```

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

Test event action

Create new event       Edit saved event

Event name

test1

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

Private  
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

Shareable  
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

```
1 [{}]
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 ]
```

Format JSON

Cancel      **Save**

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

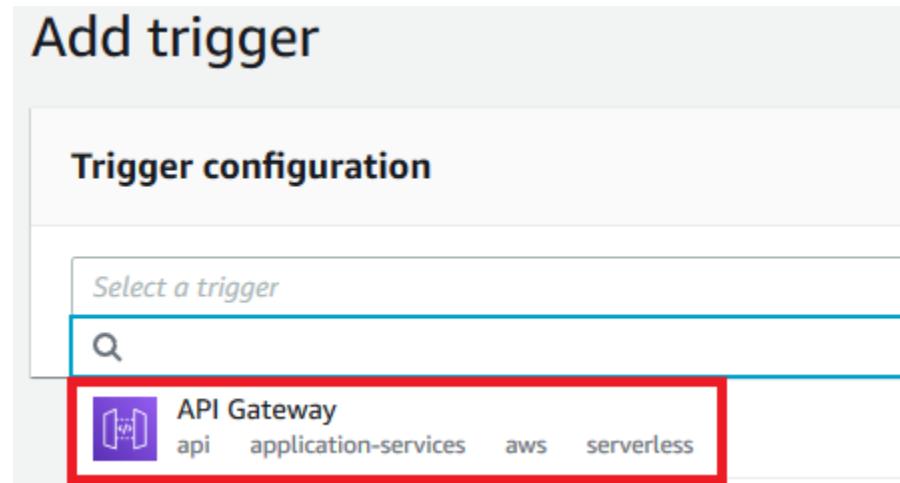
The screenshot shows the AWS Lambda execution results interface. At the top, there's a tab bar with 'index.mjs' and 'Execution result'. The 'Execution result' tab is active. Below the tabs, it says 'Test Event Name: test1'. Under 'Response', there is a JSON object:

```
{  "statusCode": 200,  "body": "\"Hello from Lambda!\""}
```

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”.

API  
Create a new API or attach an existing one.

[Create an API](#)

API type

**HTTP API**  
Create an HTTP API.

**REST API**  
Create a REST API.

Security  
Configure the security mechanism for your API endpoint.

[Open](#)

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

**Triggers (1)** [Info](#)

Trigger

 **API Gateway: awsbookc3s2-API**  
arn:aws:execute-api:us-east-1:614860265610:ok0aeu5n3a/\*/\*/awsbookc3s2  
API endpoint: <https://ok0aeu5n3a.execute-api.us-east-1.amazonaws.com/default/awsbookc3s2>

[▶ Details](#)

15. You should see the following message.

"Hello from Lambda!"

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

```

1  export const handler = async(event) => {
2      // TODO implement
3      let result = {
4          'First name': event.queryStringParameters.key1,
5          'Last name': event.queryStringParameters.key2,
6          'Location': event.queryStringParameters.key3
7      };
8
9      const response = {
10         statusCode: 200,
11         body: JSON.stringify(result),
12     };
13     return response;
14 };

```

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.

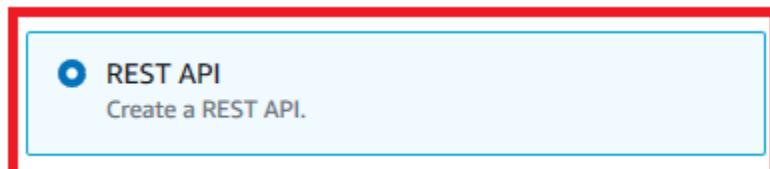
```
{"First name": "FirstName", "Last name": "Lastname", "Location": "South Bend"}
```

## Part 2. Create another API Gateway trigger

20. Delete your HTTP API Gateway.

The screenshot shows the AWS Lambda Triggers page. At the top, there is a header with the text 'Triggers (1/1)' and several buttons: 'Info' (blue), a refresh icon, 'Fix errors', 'Edit', 'Delete' (which is highlighted with a red box), and 'Add trigger'. Below the header is a search bar with the placeholder 'Find triggers' and a page navigation section with arrows and the number '1'. A checkbox labeled 'Trigger' is checked. Underneath, there is a list item with a purple icon containing a white 'API' symbol. The item is titled 'API Gateway: awsbookc3s2-API' and includes the ARN 'arn:aws:execute-api:us-east-1:614860265610:ok0aeu5n3a/\*/\*awsbookc3s2'. It also shows the API endpoint 'https://ok0aeu5n3a.execute-api.us-east-1.amazonaws.com/default /awsbookc3s2'. A 'Details' link is present at the bottom of this item.

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.

```
1 export const handler = async(event) => {
2     // TODO implement
3     let result = {
4         'First name': event.queryStringParameters.key1,
5         'Last name': event.queryStringParameters.key2,
6         'Location': event.queryStringParameters.key3
7     };
8
9     const response = {
10        statusCode: 200,
11        body: JSON.stringify(result),
12    };
13    return response;
14};
```

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.

| JSON        | Raw Data     | Headers      |
|-------------|--------------|--------------|
|             |              |              |
| Save        | Copy         | Collapse All |
|             |              | Expand All   |
|             |              | Filter JSON  |
| First name: | "FirstName"  |              |
| Last name:  | "Lastname"   |              |
| Location:   | "South Bend" |              |

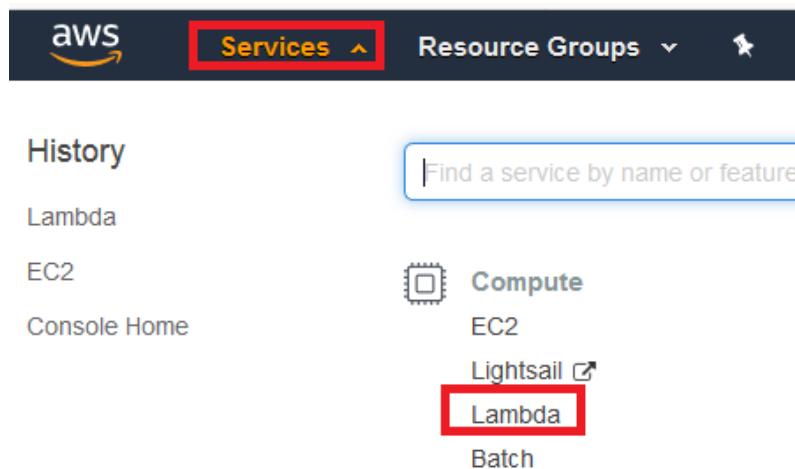
### 3.3 Trigger and Destination

#### 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

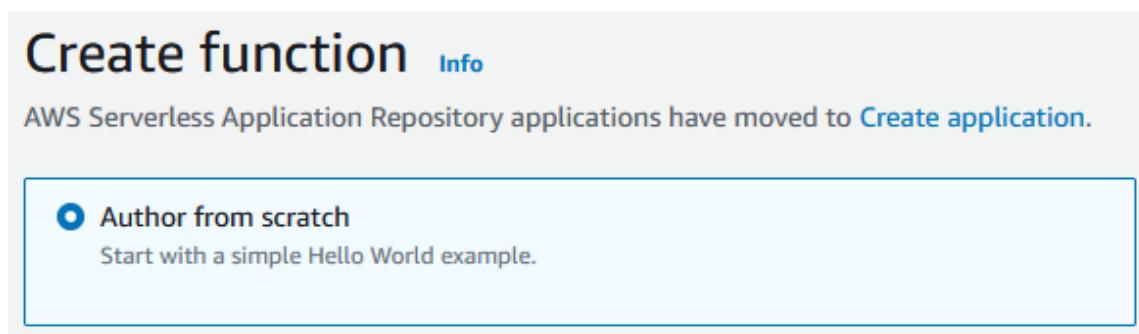
1. Log onto AWS Console home. Click **Services**, then “**Lambda**”.



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**.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- Create a new role with basic Lambda permissions
- Use an existing role
- Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission

LabRole

[View the LabRole role](#) on the IAM console.

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

The screenshot shows the AWS Lambda function editor interface. At the top, there are tabs for "index.mjs" and "Execution results". The "index.mjs" tab is active, displaying the following code:

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

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

The screenshot shows the "Event JSON" configuration dialog. It contains a text area with the following JSON code:

```
1 {  
2     "key1": "value1",  
3     "key2": "value2",  
4     "key3": "value3"  
5 }
```

At the top right of the dialog are "Format JSON" and "Save" buttons. At the bottom right are "Cancel" and "Save" buttons.

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

```

index.mjs      Execution result × +
Execution results
Test Event Name
test1

Response
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}

```

## 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.

Buckets (2) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

[C](#) [Copy ARN](#) [Empty](#) [Delete](#) [Create bucket](#)

Find buckets by name

< 1 > | [⚙️](#)

| Name        | AWS Region                      |
|-------------|---------------------------------|
| awsbookc2s1 | US East (N. Virginia) us-east-1 |

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

Overview Properties Permissions

Type a prefix and press Enter to search. Press ESC to clear

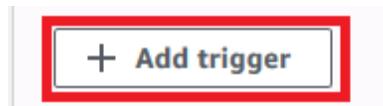
[Upload](#) [Create folder](#) [Download](#) [Actions](#)

Name ▾

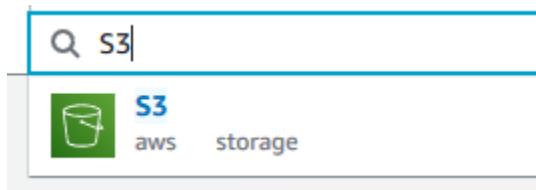
myimages

### 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.

The screenshot shows the "Add trigger" configuration page for an S3 bucket:

- Bucket:** s3/awssbookc2s1 (selected)
- Event types:** All object create events (selected), PUT (highlighted with a red box)
- Prefix - optional:** myimages/ (highlighted with a red box)
- Suffix - optional:** jpg (highlighted with a red box)
- Recursive invocation:** I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs. (checkbox checked)
- Permissions:** Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. (Learn more)
- Buttons:** Cancel, Add (highlighted with a red box)

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.

The screenshot shows the AWS Lambda Test Execution interface. At the top, there are tabs for 'index.mjs' and 'Execution result'. The 'Execution result' tab is active, showing the following details:

- Test Event Name:** test1
- Response:**

```
{ "statusCode": 200, "body": "\"Hello from lambda!\""} 
```
- Function Logs:**

```
START RequestId: c6b97d47-374d-4600-9111-f16b96c73bdf Version: $LATEST
2023-05-26T13:15:09.811Z      c6b97d47-374d-4600-9111-f16b96c73bdf      INFO      I am running!
END RequestId: c6b97d47-374d-4600-9111-f16b96c73bdf
REPORT RequestId: c6b97d47-374d-4600-9111-f16b96c73bdf Duration: 9.73 ms      Billed Duration: 
```
- Request ID:** c6b97d47-374d-4600-9111-f16b96c73bdf

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.

**CloudWatch Logs Insights** Info

Lambda logs all requests handled by your function and automatically stores logs generated by your code through Amazon CloudWatch Logs. To view custom logging statements. The following tables list the most recent and most expensive function invocations across all function activity. To view log alias, visit the **Monitor** section at that level.

Add to dashboard    1h 3h 12h 1d

| Recent invocations |                          |                                      |                                                       |
|--------------------|--------------------------|--------------------------------------|-------------------------------------------------------|
| #                  | : Timestamp              | : RequestID                          | : LogStream                                           |
| ▶ 1                | 2021-05-07T13:12:08.017Z | e8a07227-1f68-4f5c-b631-adaa4af084fe | 2021/05/07/[\$LATEST]7409c3432fc94dd48bcd2756d92ceb67 |

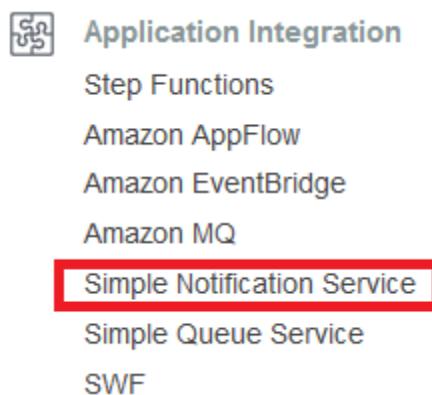
25. You can see your message is logged. (**It might take several minutes to see the result**).

|                                             | Time (UTC +00:00) | Message                                                                            |
|---------------------------------------------|-------------------|------------------------------------------------------------------------------------|
| 2020-05-04                                  |                   |                                                                                    |
|                                             |                   | No older events found at the moment. Retry.                                        |
| ▶ 17:26:58                                  |                   | START RequestId: f8a425b6-5063-4bdb-8bff-abfa93d918cf Version: \$LATEST            |
| ▼ 17:26:58                                  |                   | 2020-05-04T17:26:58.429Z f8a425b6-5063-4bdb-8bff-abfa93d918cf INFO I am running!   |
| 2020-05-04T17:26:58.429Z                    |                   |                                                                                    |
|                                             |                   | f8a425b6-5063-4bdb-8bff-abfa93d918cf INFO I am running!                            |
| ▶ 17:26:58                                  |                   | END RequestId: f8a425b6-5063-4bdb-8bff-abfa93d918cf                                |
| ▶ 17:26:58                                  |                   | REPORT RequestId: f8a425b6-5063-4bdb-8bff-abfa93d918cf Duration: 34.72 ms Billed D |
| No newer events found at the moment. Retry. |                   |                                                                                    |

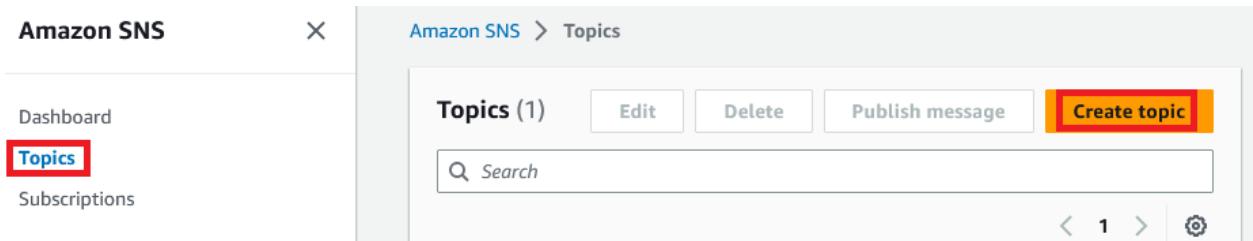
## Part 5. Add a destination to Lambda function

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”.



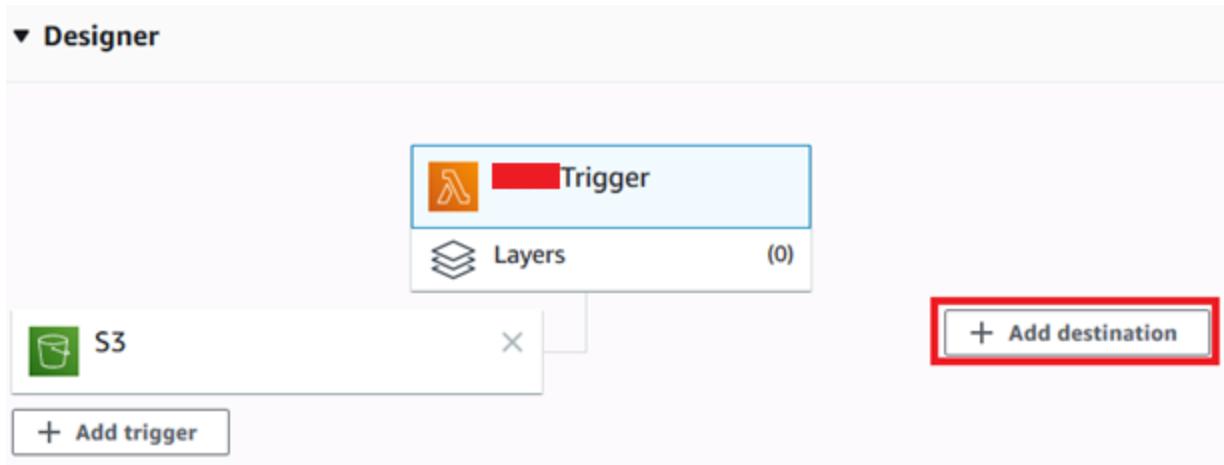
29. Assign a topic name.

A screenshot of the 'Create topic' dialog. It has a title 'Create topic' and a section for 'Topic name'. A text input field contains the value 'imageupload'. Below the input field is a large orange 'Next step' button.

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

A screenshot of the 'Create topic' dialog. Under 'Type', 'Standard' is selected (indicated by a blue circle). In the 'Name' field, 'imageupload' is entered. In the 'Display name - optional' field, 'img loaded' is entered. A red box highlights the 'Display name - optional' field. Below the fields, there's a note: 'To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.' An 'Info' link is also present.

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”.

The screenshot shows the "Trigger" configuration dialog. It has sections for "Source", "Condition", "Destination type", and a "Destination" dropdown.

- Source:** A radio button for "Asynchronous invocation" is selected and highlighted with a red box.
- Condition:** A radio button for "On success" is selected and highlighted with a red box.
- Destination type:** A dropdown menu is open, showing "SNS topic" which is highlighted with a red box.
- Destination:** A dropdown menu contains "imageupload", which is highlighted with a red box.
- Info Message:** A message box states: "Your function's execution role doesn't have permission to send result to the destination. By clicking save we'll attempt to add permission to the role for you."
- Buttons:** At the bottom are "Cancel" and "Save" buttons, with "Save" being orange.

## 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**”.

Author from scratch

Start with a simple Hello World example.

Basic information

Function name  
Enter a name that describes the purpose of your function.  
listener

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

**Create function**

39. This is the code we have now.

```
index.js
```

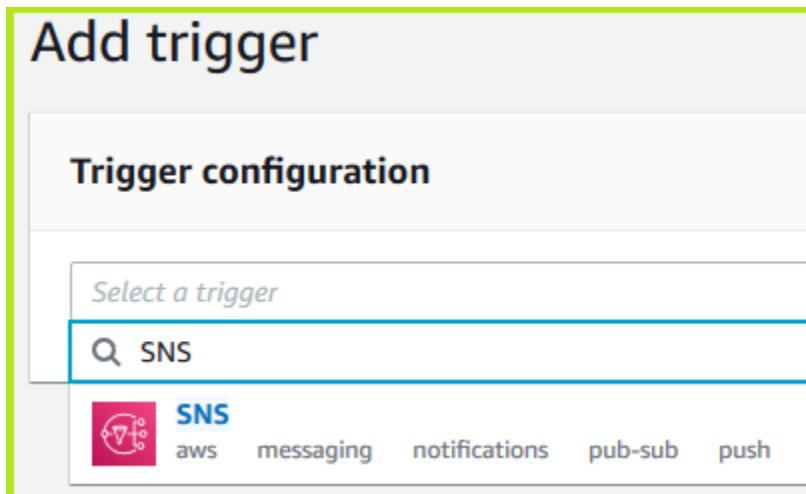
```
1 exports.handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Lambda!'),
6     };
7     return response;
8 };
```

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

```
index.mjs
```

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

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



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

SNS topic  
Select the SNS topic to subscribe to.  
 X C

Lambda will add the necessary permissions for Amazon SNS to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Enable trigger  
Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel 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”.

Code | Test | **Monitor** | Configuration | Aliases | Versions

Metrics **Logs** Traces

View CloudWatch logs

45. Click on the top log entry.

The screenshot shows the AWS CloudWatch Log Stream interface. At the top, there are tabs: 'Log streams' (which is selected), 'Metric filters', 'Subscription filters', and 'Contributors'. Below the tabs, it says 'Log streams (1)'. There is a search bar with the placeholder 'Filter log streams or try prefix search'. A table follows, with columns for a checkbox, 'Log stream', and the log entry itself. The log entry is: '2021/05/07/[\$LATEST]e75249eebf24447eab2b8f13e7d...'. The entire log entry row is highlighted with a red box.

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

The screenshot shows the AWS CloudWatch Log Events interface. It has a 'Log events' header and a note: 'You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)'. Below is a table with columns: a chevron icon, 'Timestamp', and 'Message'. The 'Message' column contains log entries. One entry from May 26, 2023, at 10:48:34 is highlighted with a red box and contains the text 'I heard an event!'. The log entries are:

- 2023-05-26T10:48:34.043-04:00 INIT\_START Runtime Version: nodejs:18.v6 Runtime Version ARN: arn:aws:lambda:us-east-1::r...
- 2023-05-26T10:48:34.223-04:00 START RequestId: 4cb29cc4-2a16-4c86-95cf-f5b52902af7f Version: \$LATEST
- 2023-05-26T10:48:34.224-04:00 2023-05-26T14:48:34.224Z 4cb29cc4-2a16-4c86-95cf-f5b52902af7f INFO I heard an event!
- 2023-05-26T10:48:34.232-04:00 END RequestId: 4cb29cc4-2a16-4c86-95cf-f5b52902af7f
- 2023-05-26T10:48:34.232-04:00 REPORT RequestId: 4cb29cc4-2a16-4c86-95cf-f5b52902af7f Duration: 8.96 ms Billed Duration:

## 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”.

```

2020-05-04T23:14:57.673Z      ccb965da-d1bf-40e3-a08f-7c9a6214a840      INFO
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "2020-05-04T23:14:54.573Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AROAY7IFGTLATGGBVSP0V:user744983=ligyu@iu.edu"
      },
      "requestParameters": {
        "sourceIPAddress": "104.13.16.202"
      },
      "responseElements": {
        "x-amz-request-id": "778EAC4309F0C491",
        "x-amz-id-2": "/Sb+oHjJQp0mrdbF8ABov+i6IfD1/n/f7Xus8q9pqDD0vh0zNawTV"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "f636872b-2a1e-4682-adc7-53dabd32abe6",
        "bucket": {
          "name": "c490",
          "ownerIdentity": {
            "principalId": "A3AGMQ3K8Z291A"
          },
          "arn": "arn:aws:s3:::c490"
        },
        "object": {
          "key": "myimages/dog3.jpg",
          "size": 105464,
          "eTag": "a86c865bfad364efd5f22518a253364b",
          "sequencer": "005EB0A1F05C4EE493"
        }
      }
    }
  ]
}

```

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

```

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    console.log(event.Records[0].s3.object.key);
14    return response;
15 }

```

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”.

```

2020-05-04T23:30:28.060Z 5f16d3eb-2f8b-404e-9de5-52af87cb3bf4 INFO myimages/dog3.jpg
5f16d3eb-2f8b-404e-9de5-52af87cb3bf4 INFO myimages/dog3.jpg
END RequestId: 5f16d3eb-2f8b-404e-9de5-52af87cb3bf4

```

## 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.

```

1 exports.handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Listener!'),
6     };
7     console.log("I headed an event!");
8     console.log(JSON.stringify(event));
9     return response;
10 }

```

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”.

Metrics    **Logs**    Traces    Related links ▲

**CloudWatch Logs Insights** Info

Lambda logs all requests handled by your function and automatically stores logs generated in CloudWatch Logs. To validate your code, instrument it with custom logging statements. The following tables list the most recent and most expensive function invocations across all function activity. To view logs for a specific function version or alias, visit the **Monitor** section at that level.

Add to dashboard    1h 3h 12h 1d 3d 1w custom ▾

Recent invocations

| #   | : Timestamp              | : RequestID                         | : LogStream                    |
|-----|--------------------------|-------------------------------------|--------------------------------|
| ▶ 1 | 2021-05-07T15:09:33.261Z | fb3683a-5b67-4034-9008-00c024cec0a6 | 2021/05/07[\$LATEST]2742910b04 |

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\": \"OW1ZqSJftmKfNfgJfPzDOC2zdNHMh06t0q+zE0zHMMnxiOGQ4tJq4PnGGZUJ\",
          \"SigningCertUrl\": \"https://sns.us-east-1.amazonaws.com/SimpleNotificationService-616875596481.pem\",
          \"UnsubscribeUrl\": \"https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:616875596481:bucketEvent:ec867d\",
          \"MessageAttributes\": {}"
      }
    }
  ]
}
```

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

```

1 exports.handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Listener!'),
6     };
7     console.log("I headed an event!");
8     console.log(JSON.stringify(event));
9     console.log(event.Records[0].Sns);
10    return response;
11 };

```

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.

```

Message: {"version":"1.0","timestamp":"2020-05-04T23:30:28.263Z","requestContext":{"requestId":"5f16d3eb-2f8b-404e-9de5-52af87cb3bf4","functionArn":"arn:aws:lambda:us-east-1:616875596481:function:C490Trigger:$LATEST","condition":"Success","approximateInvokeCount":1}, "requestPayload":{"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us-east-1", "eventTime": "2020-05-04T23:30:23.563Z", "eventName": "ObjectCreated", "principalId": "AWS:AROAY7IFGTLATGBVSPOV:user744983=ligyu@iu.edu"}, {"requestParameters": {"sourceIPAddress": "104.13.16.202"}, "responseElements": {"requestId": "509957E66FC7B3AA", "x-amz-id-2": "X51e6Z93YFjn4+FMFa57DrF0c2WMTg/xp+hJ8v0D6aeihvciaElRy4tKPhAABVmRpJJj81uESGFPPDxwloPpGFYZlu"}, {"s3SchemaVersion": "1.0", "configurationId": "f636872b-2a1e-4682-adc7-53dabd32abe6", "bucket": {"name": "c490", "ownerIdentity": {"principalId": "A3AGMQ3K8Z291A"}, "arn": "arn:aws:s3:::c490"}, "object": {"key": "myimages/dog3.jpg", "size": 105464, "eTag": "a86c865bfad364efdf5f22518a253364b"}, "sequencer": "005EBAA5923856CD97"}]}}, "responseContext": null}

```

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.

## Add destination

**Destination configuration**  
Send invocation records to a destination when your function is invoked asynchronously, or if your function processes records from a stream.

**Source**  
The type of invocation that maps to the destination.  
 Asynchronous invocation  
 Stream invocation

**Condition**  
The condition for using the destination.  
 On failure  
 On success

**Destination type**  
An SQS queue, SNS topic, Lambda function, or EventBridge event bus.  
Lambda function ▾

**Destination**  
arn:aws:lambda:us-east-1:614860265610:function:newdestination

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 };
```

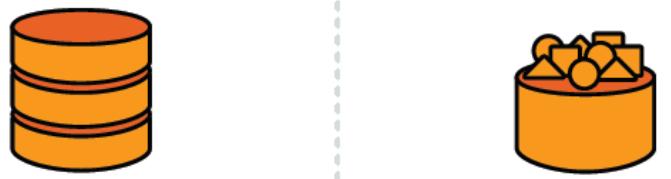
66. Modify its code (Line #8) to retrieve “event” parameter information.

```
1 exports.handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from real handler!'),
6     };
7     console.log(JSON.stringify(event));
8     return response;
9 }
10 };
```

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.

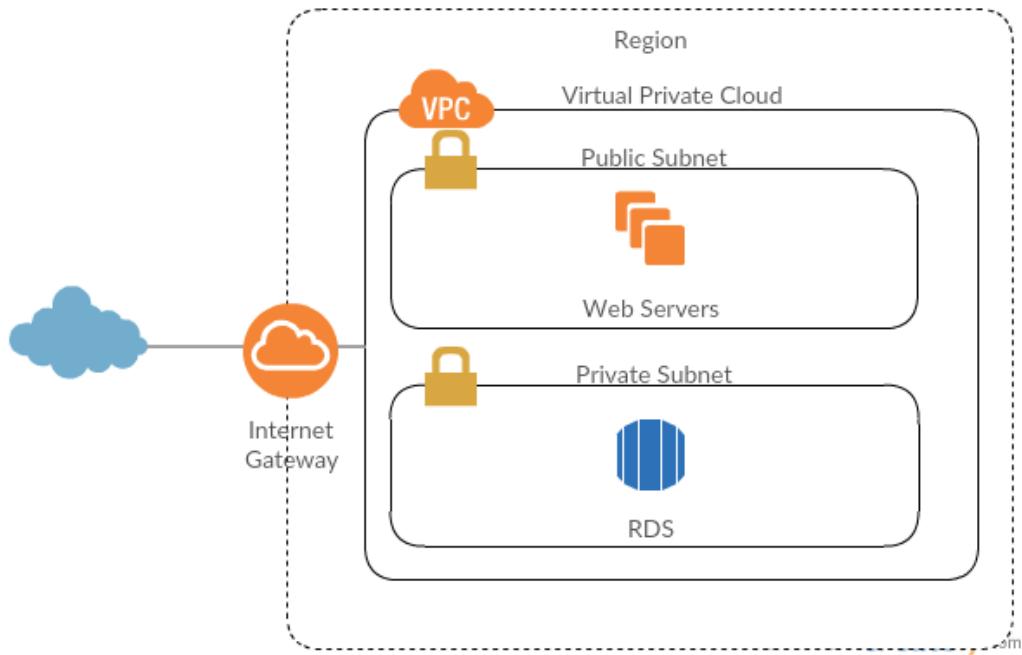


**Amazon Database Services**

## 4.1 Relational Database

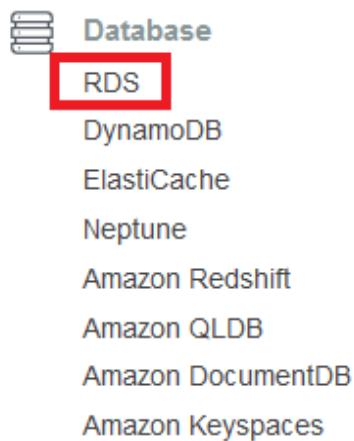
### Objectives:

- 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”.

## Create database

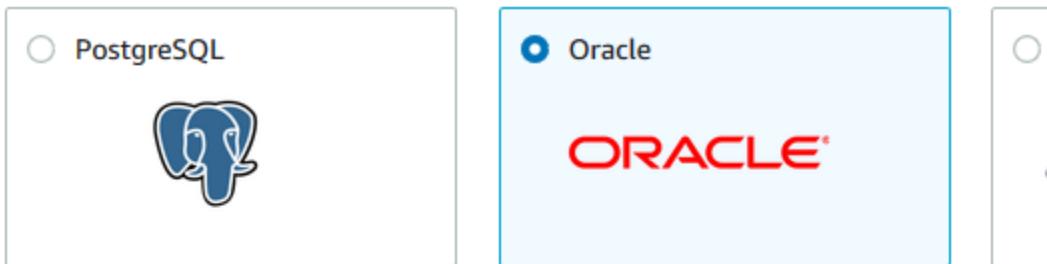
Amazon Relational Database Service (RDS) makes it easy to set up, operate, and manage relational database in the cloud.

[Restore from S3](#)

[Create database](#)

Note: your DB instances will launch in the US East (N. Virginia) region

3. Select Oracle, Standard Edition One.



### Edition

- Oracle Enterprise Edition

Efficient, reliable, and secure database management system that delivers comprehensive high-end capabilities for mission-critical applications and demanding database workloads.

- Oracle Standard Edition

Affordable and full-featured database management system supporting up to 32 vCPUs.

- Oracle Standard Edition One

Affordable and full-featured database management system supporting up to 16 vCPUs.

- Oracle Standard Edition Two

Affordable and full-featured database management system supporting up to 16 vCPUs. Oracle Database Standard Edition Two is a replacement for Standard Edition and Standard Edition One.

4. Select “Free tier” for Templates.

### Templates

Choose a sample template to meet your use case.

- Production

Use defaults for high availability and fast, consistent performance.

- Dev/Test

This instance is intended for development use outside of a production environment.

- Free tier

Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.  
[Info](#)

5. Enter DB name, user ID and password.

**DB instance identifier** [Info](#)

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

awsbookc4s1

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

**▼ Credentials Settings**

**Master username** [Info](#)

Type a login ID for the master user of your DB instance.

awsuser

1 to 16 alphanumeric characters. First character must be a letter.

**Manage master credentials in AWS Secrets Manager**

Manage master user credentials in Secrets Manager. RDS can generate a password for you and manage it throughout its lifecycle.

 If you manage the master user credentials in Secrets Manager, some RDS features aren't supported.

[Learn more](#) 

**Auto generate a password**

Amazon RDS can generate a password for you, or you can specify your own password.

**Master password** [Info](#)

\*\*\*\*\*

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote)', "(double quote) and @ (at sign).

**Confirm master password** [Info](#)

\*\*\*\*\*

6. Allow public access.

**Publicly accessible** [Info](#)

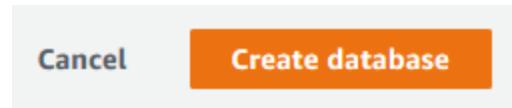
**Yes**

Amazon EC2 instances and devices outside the VPC can connect or more VPC security groups that specify which EC2 instances can connect to the database.

**No**

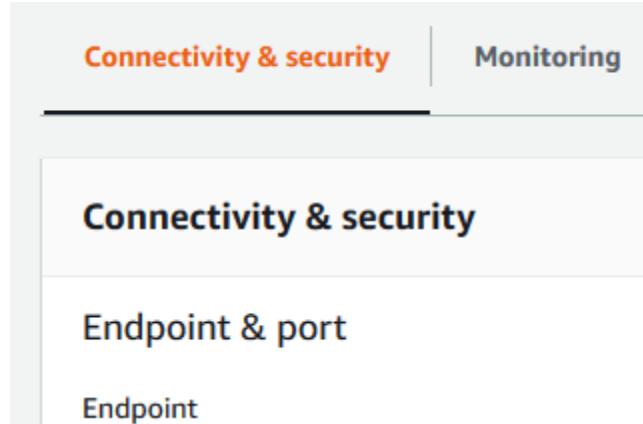
RDS will not assign a public IP address to the database. Only Am

7. Click “Create database”.



## Part 2. Update inbound and outbound traffic rules

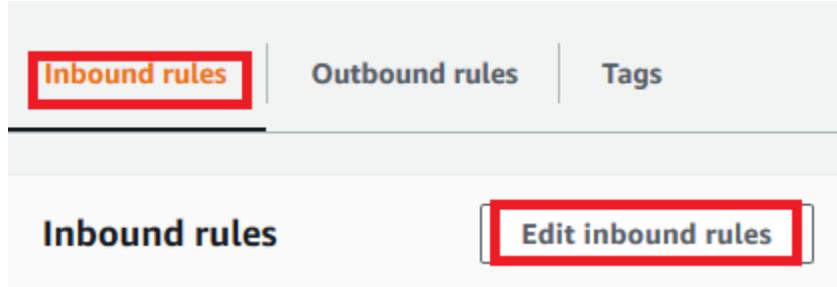
8. After creation is completed, click on your oracle database instance and select **Connectivity & security**.



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



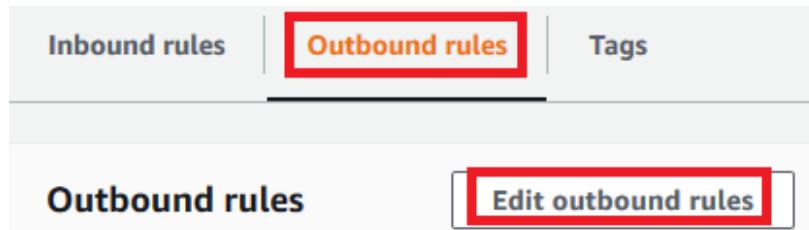
10. Click "Inbound rules", "Edit inbound rules".



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".



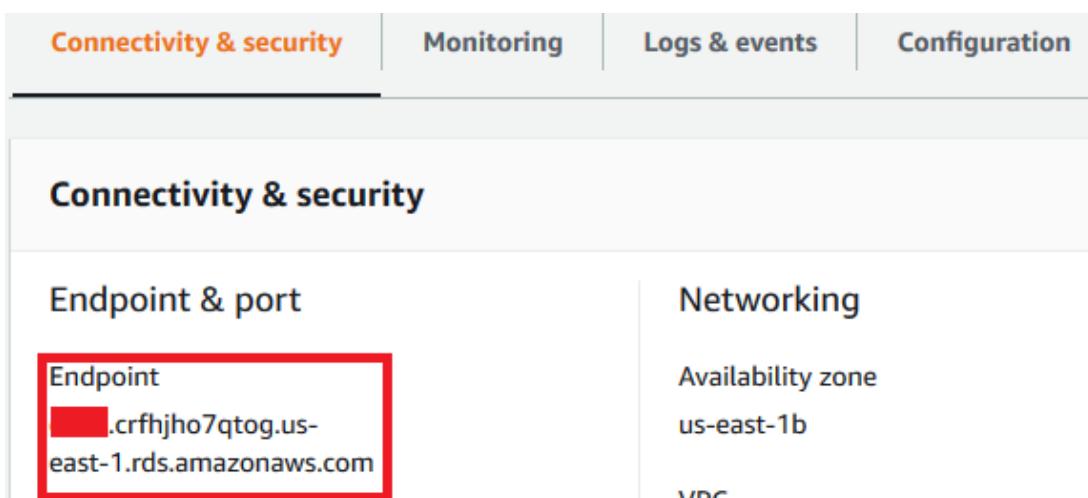
12. Click “Outbound rules”, “Edit outbound rules”.



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.

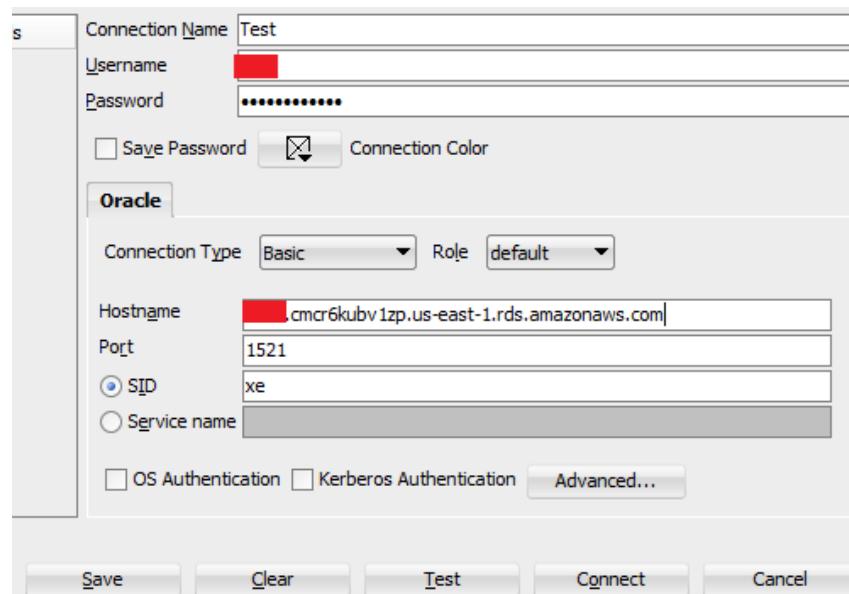


15. Click on “Configuration” and find out “DB name”.

| Connectivity & security        | Monitoring                    | Logs & events | Configuration                   |
|--------------------------------|-------------------------------|---------------|---------------------------------|
| Instance                       |                               |               |                                 |
| Configuration                  | Instance class                |               | Storage                         |
| DB instance id<br>[REDACTED]   | Instance class<br>db.t3.micro |               | Encryption<br>Enabled           |
| Engine version<br>11.2.0.4.v23 | vCPU<br>2                     |               | KMS key<br>aws/rds [?]          |
| DB name<br>ORCL                | RAM<br>1 GB                   |               | Storage type<br>General Purpose |

### 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).

Schema: LIGYU  
 Name: ACCOUNT

Table DDL

Columns: name

| PK | Name      | Data Type | Size | Not Null                            |
|----|-----------|-----------|------|-------------------------------------|
|    | ACTNUM    | NUMBER    |      | <input checked="" type="checkbox"/> |
|    | FIRSTNAME | VARCHAR2  | 20   | <input type="checkbox"/>            |
|    | LASTNAME  | VARCHAR2  | 20   | <input type="checkbox"/>            |
|    | BALANCE   | NUMBER    |      | <input type="checkbox"/>            |

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

Test > ACCOUNT >

Columns Data Model Constraints Grants Statistics Triggers Fl

Sort.. Filter:

|   | ACTNUM | FIRSTNAME | LASTNAME | BALANCE |
|---|--------|-----------|----------|---------|
| 1 | 1001   | John      | Smith    | 98      |
| 2 | 3003   | Joe       | Boy      | 106     |
| 3 | 4004   | Lisa      | Dew      | 786     |

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

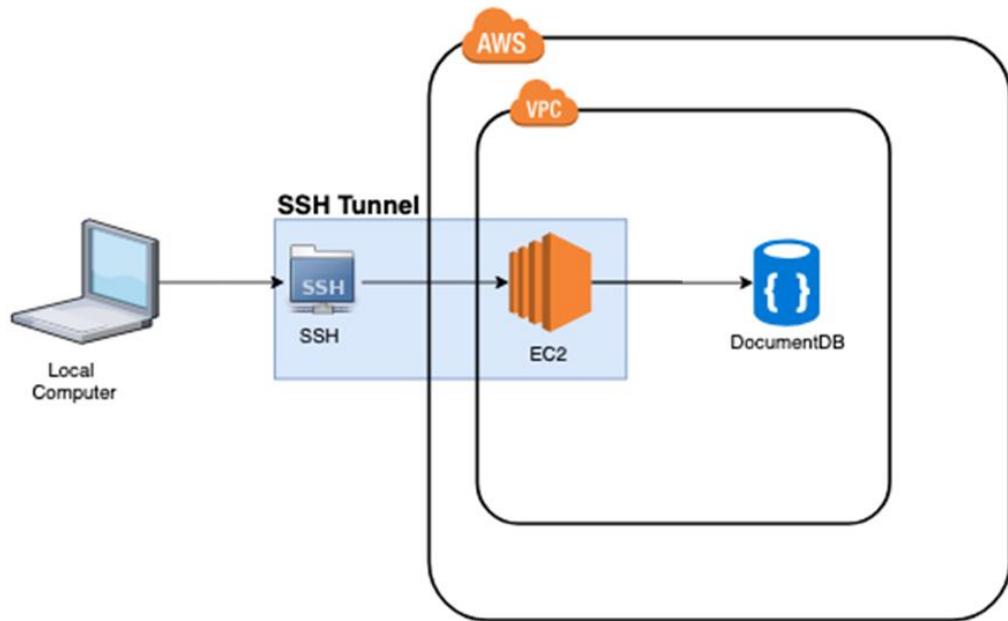
```
SELECT * FROM ACCOUNT WHERE balance < 200;
```

25. Delete the database instance.

## 4.2 DocumentDB

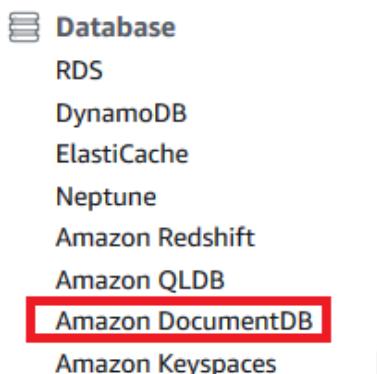
### Objectives:

- 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".

## Document Database

Create an Amazon DocumentDB cluster

**Launch Amazon DocumentDB**

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

**Configuration**

Cluster identifier [Info](#)  
Specify a unique cluster identifier.

Instance class [Info](#)  
  
2 vCPUs 16GiB RAM

Number of instances [Info](#)

5. Specify username and password.

## Authentication

### Username [Info](#)

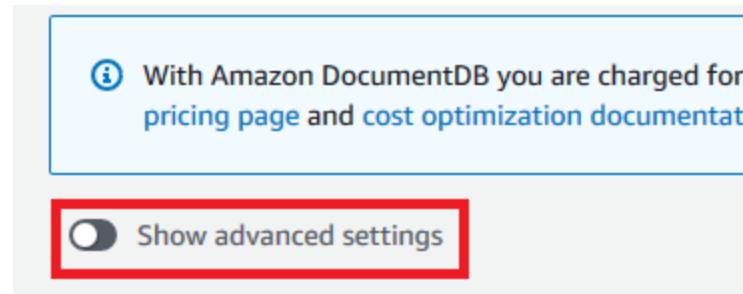
Specify an alphanumeric string that defines the login ID for the user.

Username must start with a letter and contain 1 to 63 characters

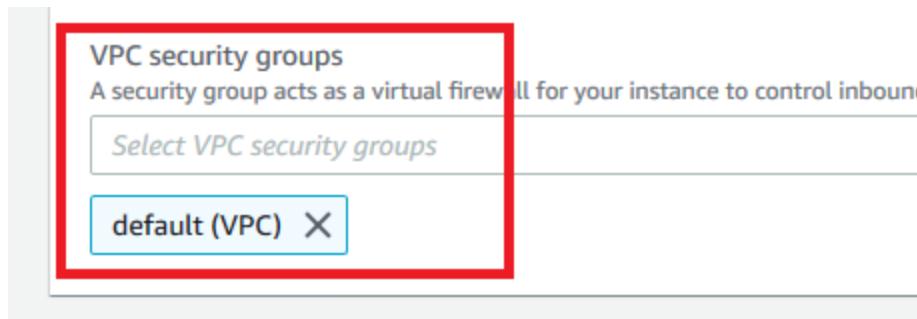
### Password [Info](#)

### Confirm password [Info](#)

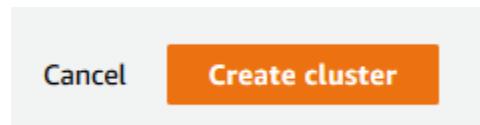
6. Enable “Show advanced settings”.



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.

| Cluster identifier        | ▲ | Engine version | ▼ | Status                 |
|---------------------------|---|----------------|---|------------------------|
| docdb-2020-05-02-00-16-16 |   | docdb 3.6.0    |   | <span>available</span> |

11. Move all the way down and click on “**Security group**”.

| Security Groups (1)                        |                       |
|--------------------------------------------|-----------------------|
| <input type="text"/> Filter security group |                       |
| Security group name (ID)                   | ▲                     |
| default (sg-03a37820d7ea00bb1)             | vpc-06b386755b75bee02 |

12. Click on the Security Group ID again.

The screenshot shows the AWS Management Console interface for security groups. At the top, it says "Security Groups (1/1)" and "Info". Below that is a search bar with the placeholder "Filter security groups" and a search input field containing "search: sg-03a37820d7ea00bb1" with a clear button. There are two dropdown filters: "Security group ID" and "Security group name", both currently set to "sg-03a37820d7ea00bb1". The main table lists one security group: "sg-03a37820d7ea00bb1" with a "default" status. The "Security group ID" column for this row is highlighted with a red border.

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

The screenshot shows the "Inbound rules" tab selected. It has three tabs: "Inbound rules", "Outbound rules", and "Tags". The "Inbound rules" section contains a table with four columns: Type, Protocol, Port range, and Source. A single row is shown: "All traffic", "All", "All", and "0.0.0.0/0". This row is highlighted with a red border.

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

The screenshot shows the "Outbound rules" tab selected. It has three tabs: "Inbound rules", "Outbound rules", and "Tags". The "Outbound rules" section contains a table with four columns: Type, Protocol, Port range, and Destination. A single row is shown: "All traffic", "All", "All", and "0.0.0.0/0". This row is highlighted with a red border.

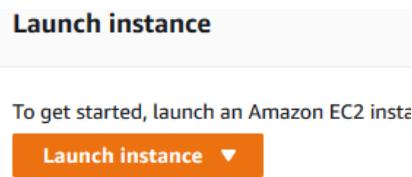
15. Save all the change.

## 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.

Amazon Linux 2 AMI (HVM), SSD Volume Type

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).

## Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. Amazon EC2 security groups.

Assign a security group:  Create a new security group  Select an existing security group

| Security Group ID                                        | Name            | Description                                  |
|----------------------------------------------------------|-----------------|----------------------------------------------|
| <input checked="" type="checkbox"/> sg-03a37820d7ea00bb1 | default         | default VPC security group                   |
| <input type="checkbox"/> sg-020ba08d9c0ec6254            | launch-wizard-1 | launch-wizard-1 created 2020-04-14T18:45:40Z |

24. Click “Review and Launch”.

[Previous](#) [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.

4. Connect to your instance using its Public DNS:

[ec2-54-164-202-238.compute-1.amazonaws.com](http://ec2-54-164-202-238.compute-1.amazonaws.com)

example:

`ssh -i "iusbaws.pem" ec2-user@ec2-54-164-202-238.compute-1.amazonaws.com`

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.

The screenshot shows the 'Clusters (1)' section of the AWS CloudFormation console. A search bar at the top says 'Filter clusters'. Below it, a table has a single row with a blue circular icon and the identifier 'docdb-2020-05-02-00-16-16'. The table header is 'Cluster identifier'.

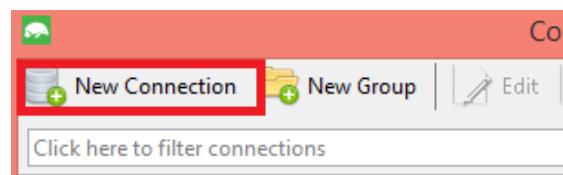
32. Copy the connection string. Replace <insertYourPassword> with your password. Note here:  
 <> should also be replaced.

The screenshot shows the 'Connectivity & security' tab of a cluster's details page. It includes sections for 'Getting Started Guide', 'Enabling/Disabling TLS', and 'Connecting programmatically'. The 'mongo' command for the mongo shell is shown. Below it, a red box highlights the 'Connect to this cluster with an application' section, which contains a 'Copy' button and a URI template: 'mongodb://<redacted>:<insertYourPassword>@docdb-2020-05-02-00-16-16.cluster-cuoflgs9g900.us-east-1.docdb.amazonaws.com:27017/?ssl=true&ssl\_ca\_certs=rds-combined-ca-bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'.

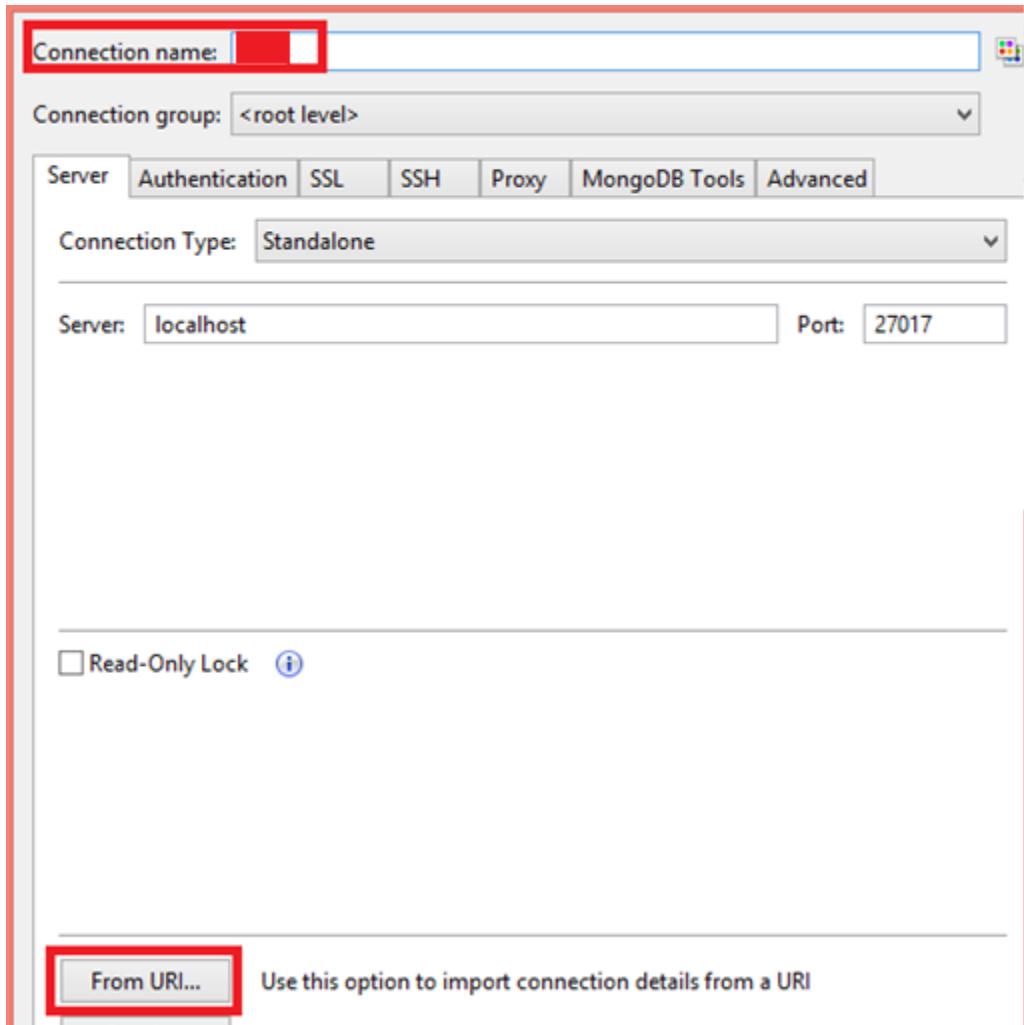
33. Start Studio 3T. Click “Connect”.



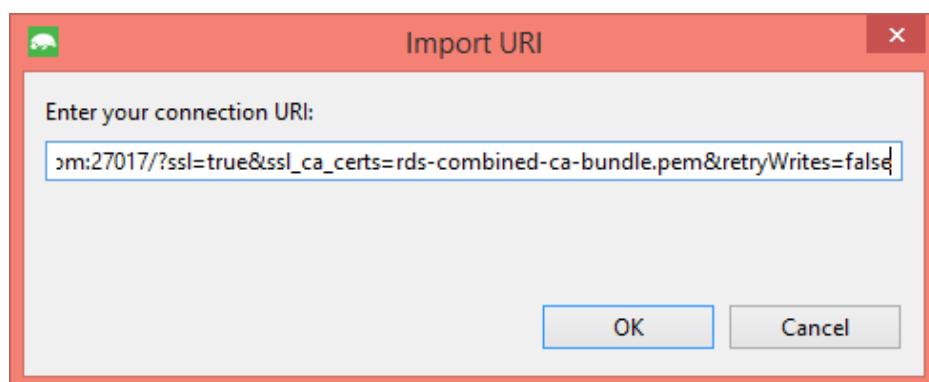
34. Click “New Connection”.



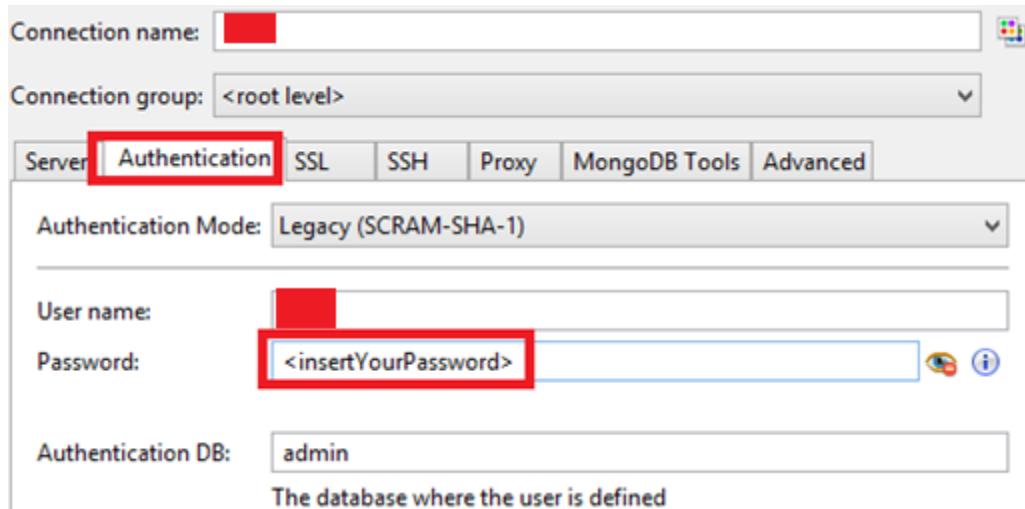
35. Assign a name to this connection, then click “From URI”.



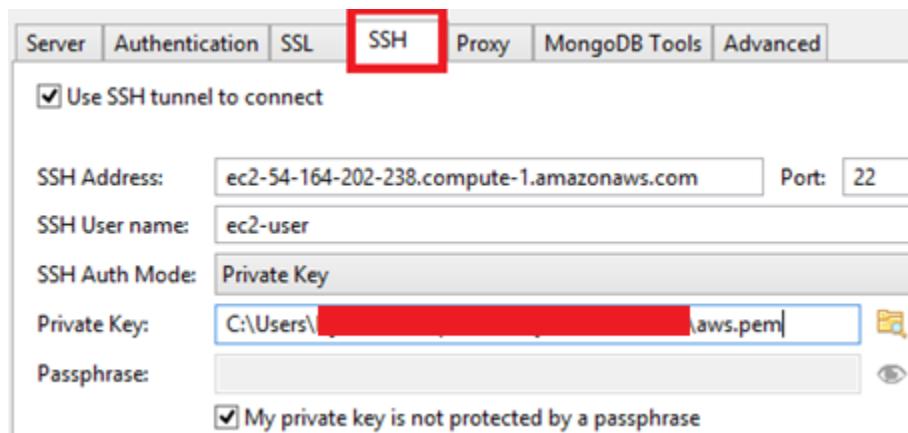
36. Paste the Connection String you copied at Step 32. Click “OK” (be **careful** here: make sure the **whole** URL is pasted here).



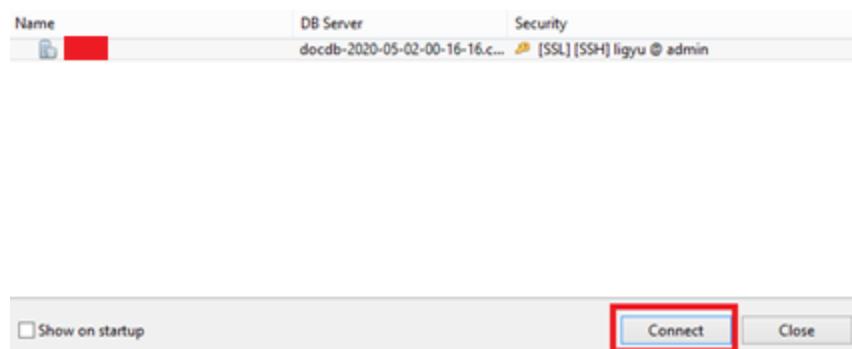
37. Click “Authentication”. Replace <insertYourPassword> with your password if you have not done it.



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.

The screenshot shows the MongoDB Compass interface. On the left, the database structure is displayed with a tree view. Under the 'admin' database, there is a 'Collections (1)' folder containing a single collection named 'Sales'. This collection has 0 views, 0 GridFS buckets, and 0 system documents. On the right, the 'Result' tab of the 'Sales' collection is selected, showing a table with the following data:

| _id               | Transaction_date | Product  |
|-------------------|------------------|----------|
| Seaccc7a5514c5... | 1/2/09 6:17      | Product1 |
| Seaccc7a5514c5... | 1/2/09 4:53      | Product1 |
| Seaccc7a5514c5... | 1/2/09 13:08     | Product1 |
| Seaccc7a5514c5... | 1/3/09 14:44     | Product1 |
| Seaccc7a5514c5... | 1/4/09 12:56     | Product2 |

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.

The screenshot shows the MongoDB Compass interface. On the left, the database structure is displayed with a tree view. Under the 'admin' database, there is a 'Collections (2)' folder containing two collections: 'Donuts' and 'Sales'. There are also 0 views, 0 GridFS buckets, and 0 system documents. On the right, the 'Result' tab of the 'Donuts' collection is selected, showing a table with the following data:

| _id               | id   | type |
|-------------------|------|------|
| Seacce905514c5... | 0001 |      |
| Seacce905514c5... | 0002 |      |
| Seacce905514c5... | 0003 |      |

## 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.

|                               |                               |        |                |           |
|-------------------------------|-------------------------------|--------|----------------|-----------|
| docdb-<br>2020-05-02-00-16-16 | docdb-<br>2020-05-02-00-16-16 | Writer | docdb<br>3.6.0 | available |
|-------------------------------|-------------------------------|--------|----------------|-----------|

2. Click "Configuration".

The screenshot shows the AWS RDS Configuration tab selected. Below it, the Cluster details section is visible. A red box highlights the 'Modify' button in the top right corner of the cluster details area.

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

The screenshot shows the 'Cluster 'docdb-2020-05-02-00-16-16' details' page. A red box highlights the 'Modify' button in the top right corner of the main content area.

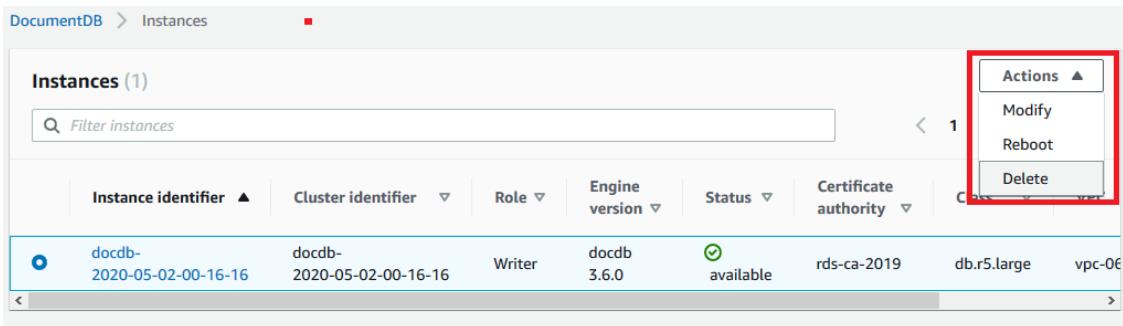
4. Move all the way down, uncheck “Deletion Protection”. Click “Continue”.

The screenshot shows the 'Maintenance' and 'Deletion protection' sections of the modify cluster wizard. In the 'Deletion protection' section, a red box highlights the checkbox labeled 'Enable deletion protection'. At the bottom right, there are 'Cancel' and 'Continue' buttons, with 'Continue' being highlighted by a red box.

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

The screenshot shows the 'Apply immediately' and 'Potential unexpected downtime' sections of the modify cluster wizard. In the 'Apply immediately' section, a red box highlights the radio button for 'Apply immediately'. In the 'Potential unexpected downtime' section, a red box highlights the warning icon and text. At the bottom right, there are 'Cancel', 'Back', and 'Modify cluster' buttons, with 'Modify cluster' being highlighted by a red box.

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



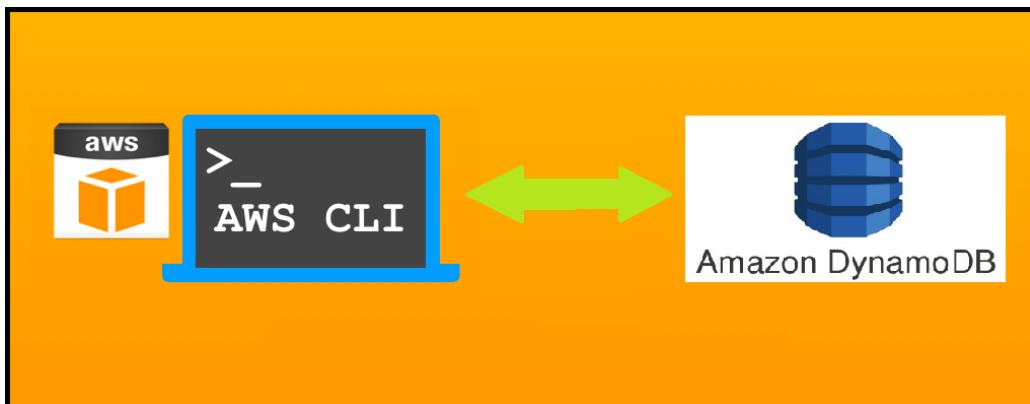
The screenshot shows the AWS DocumentDB Instances page. At the top, there's a breadcrumb navigation: DocumentDB > Instances. Below the header, a table titled "Instances (1)" displays one row of data. The columns are: Instance identifier, Cluster identifier, Role, Engine version, Status, Certificate authority, and Class. The instance details are: docdb-2020-05-02-00-16-16, docdb-2020-05-02-00-16-16, Writer, docdb 3.6.0, available, rds-ca-2019, db.r5.large, vpc-0e. To the right of the table, a context menu is open with a red border, containing the following options: Actions (with arrows), Modify, Reboot, and Delete. The "Delete" option is highlighted with a red background.

| Instance identifier       | Cluster identifier        | Role   | Engine version | Status    | Certificate authority | Class       |        |
|---------------------------|---------------------------|--------|----------------|-----------|-----------------------|-------------|--------|
| docdb-2020-05-02-00-16-16 | docdb-2020-05-02-00-16-16 | Writer | docdb 3.6.0    | available | rds-ca-2019           | db.r5.large | vpc-0e |

## 4.3 DynamoDB

### Objectives:

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



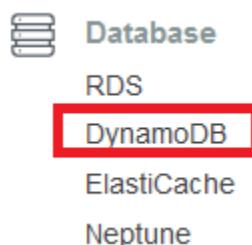
### Problem description:

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”.

Table name\*  i

Primary key\* Partition key

String i

Add sort key

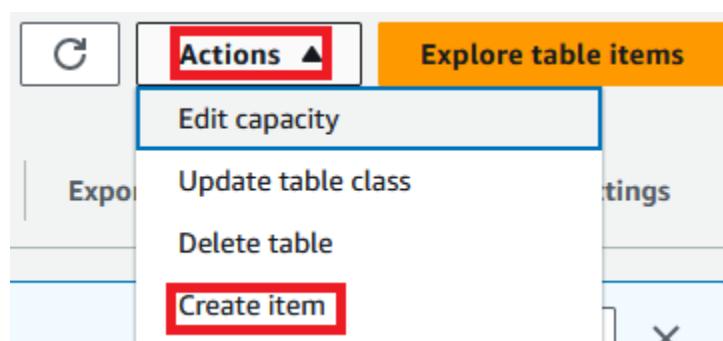
Number i

7. Click on table name.

**Tables (1) i**

| <input type="checkbox"/> | Name          | Status              | Partition key | Sort key  | Indexes |
|--------------------------|---------------|---------------------|---------------|-----------|---------|
| <input type="checkbox"/> | <b>Events</b> | <span>Active</span> | Location (S)  | Month (N) | 0       |

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



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

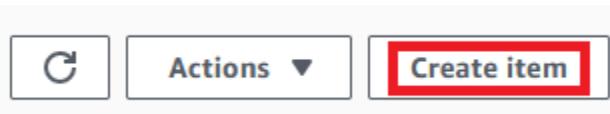
Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

| Attributes               |         | Add new attribute ▾ |
|--------------------------|---------|---------------------|
| Attribute name           | Value   | Type                |
| Location - Partition key | Chicago | String              |
| Month - Sort key         | 8       | Number              |
| Members                  | 35      | Number              |
| EventName                | Cruise  | String              |

Cancel Create item

11. Click “Create item”.



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

| <input type="checkbox"/> | Location ⓘ    | Month | Members | Successful | EventName |
|--------------------------|---------------|-------|---------|------------|-----------|
| <input type="checkbox"/> | Chicago       | 8     | 35      |            | Cruise    |
| <input type="checkbox"/> | Mishawaka     | 7     |         | true       | Fireworks |
| <input type="checkbox"/> | New York City | 11    | 39      |            | Parade    |
| <input type="checkbox"/> | 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 json 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>).

- Log onto AWS account
- Copy credential and paste it into your credential file
- 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.

Create item Actions ▾

Scan: [Table] Events: Location, Month ▾

|                          | Location ⓘ    | Month | EventName      | Members | Successful | Comment            |
|--------------------------|---------------|-------|----------------|---------|------------|--------------------|
| <input type="checkbox"/> | Chicago       | 8     | Cruise         | 35      |            |                    |
| <input type="checkbox"/> | Elkhart       | 6     | 4H Fair        | 17      |            |                    |
| <input type="checkbox"/> | Holland       | 5     | Tulip Festival |         |            | canceled this year |
| <input type="checkbox"/> | Indianapolis  | 5     | Indy 500       | 125     |            |                    |
| <input type="checkbox"/> | Mishawaka     | 7     | Fireworks      |         | true       |                    |
| <input type="checkbox"/> | New York City | 11    | Parade         | 39      |            |                    |
| <input type="checkbox"/> | 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

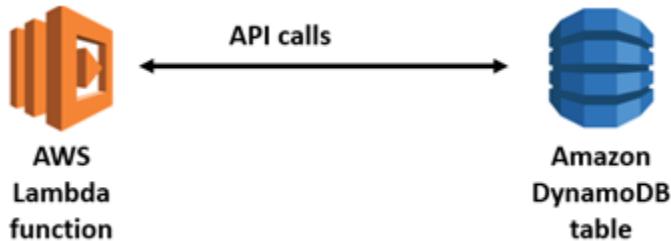
To solve a real-world business problem, multiple AWS services must be used together. From infrastructure to storage, to web hosting, AWS services can be integrated seamlessly to provide reliable and on demand business solutions. Often, AWS services are also used together with third party services, such 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.

This screenshot shows the AWS DynamoDB Tables page. On the left, a sidebar menu includes "Dashboard", "Tables" (which is selected and highlighted with a red box), "Backups", "Reserved capacity", and "Preferences". The main area has a "Create table" button and a "Delete table" button. Below that is a search bar labeled "Filter by table name". A table lists tables with columns "Name" and "Status". The "Events" table is listed, with its "Name" column value "Events" highlighted with a red box and its "Status" column value "Active" in green.

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

This screenshot shows the results of a "Scan" operation on the "Events" table. At the top, there are "Create item" and "Actions" buttons. Below that, the scan results are displayed with the header "Scan: [Table] Events: Location, Month". The results table has a "Scan" button and a dropdown menu. The table body shows a single row with the key "Location" and "Month". The entire row is highlighted with a red box, and the key values "Location" and "Month" are also highlighted with a red box.

### 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.

```

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 }

```

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



```

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     };
}

```

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.



```

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.

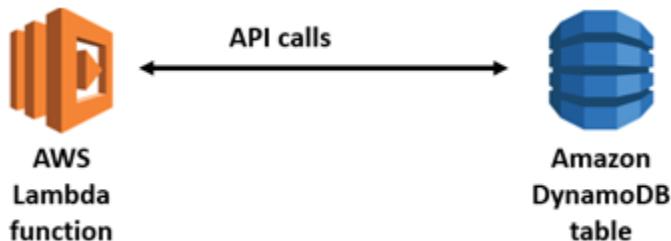
| Location   | Month | EventName  | Notes              | Budget |
|------------|-------|------------|--------------------|--------|
| NYU        | 1     | New Year   |                    | \$90k  |
| Purdue     | 9     | Corn Grill | Open to the public |        |
| Notre Dame | 11    | Football   | 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

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.

This screenshot shows the AWS DynamoDB console's "Tables" page. The sidebar on the left includes links for Dashboard, Tables (which is selected and highlighted with a red box), Backups, Reserved capacity, and Preferences. The main area features a "Create table" button and a "Delete table" button. Below these are search and filter options, including a "Filter by table name" search bar and columns for "Name" and "Status". A table lists existing tables, with the row for "Events" highlighted with a red box. The "Events" table is marked as "Active".

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

This screenshot shows the AWS DynamoDB console's "Events" table details page. At the top, there are "Create item" and "Actions" buttons. Below is a section titled "Scan: [Table] Events: Location, Month". A dropdown menu shows "Scan" and "[Table] Events: Location, Month", with the second option highlighted with a red box. This indicates the composite primary key for the table.

### 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.

```

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 }

```

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



```

index.mjs      Execution results × +
```

```

1 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
2 import {
3   DynamoDBDocumentClient,
4   ScanCommand,
5 } from "@aws-sdk/lib-dynamodb";
6
7 const client = new DynamoDBClient({});
8 const dynamo = DynamoDBDocumentClient.from(client);
9
10 export const handler = async (event, context) => {
11   let body;
12   let statusCode = 200;
13   const headers = {
14     "Content-Type": "application/json",
15   };
16
17   body = await dynamo.send(
18     new ScanCommand({ TableName: 'Events' })
19   );
20   body = body.Items;
21
22   return {
23     statusCode,
24     body,
25     headers,
26   };
27 }

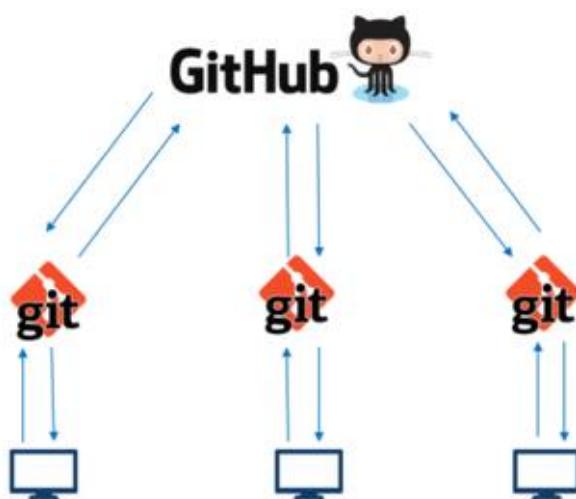
```

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

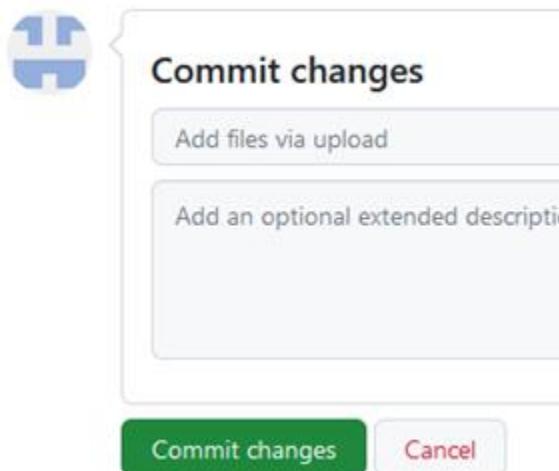
1. Download the zip file public\_html.zip given online (<https://github.com/awsbookresource/files>). Unzip it.
2. Log onto GitHub.
3. Create a new repository.
4. Click uploading an existing file.

## Quick setup — if you've done this kind of thing before

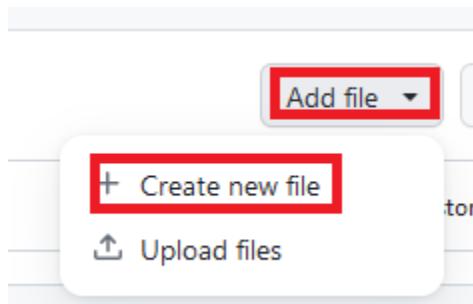
[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/yliguo/awsweb.git>

Get started by creating a new file or [uploading an existing file](#). We recommend every repository include a `README`, `LICENSE`, and `.gitignore`.

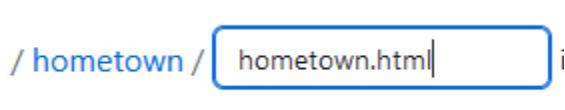
5. Choose the two files (`index.html` and `Dumbledore.jpg`) and click “Commit changes”.



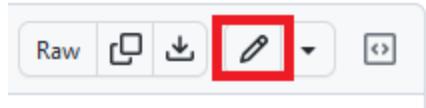
6. Click on “Add file”, then “Create new file”.



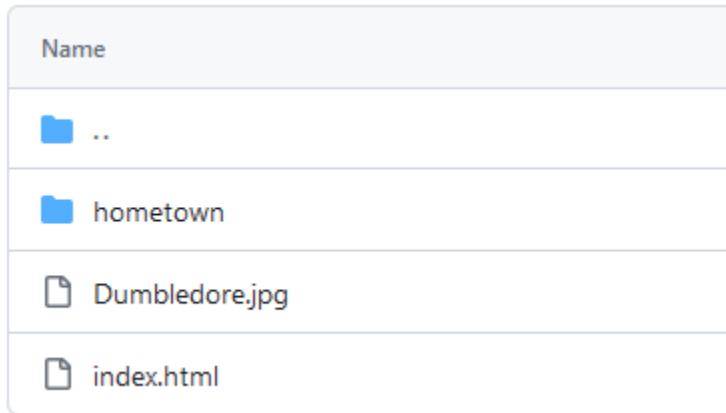
7. Type “`hometown/hometown.html`”, then click “**Commit changes**”



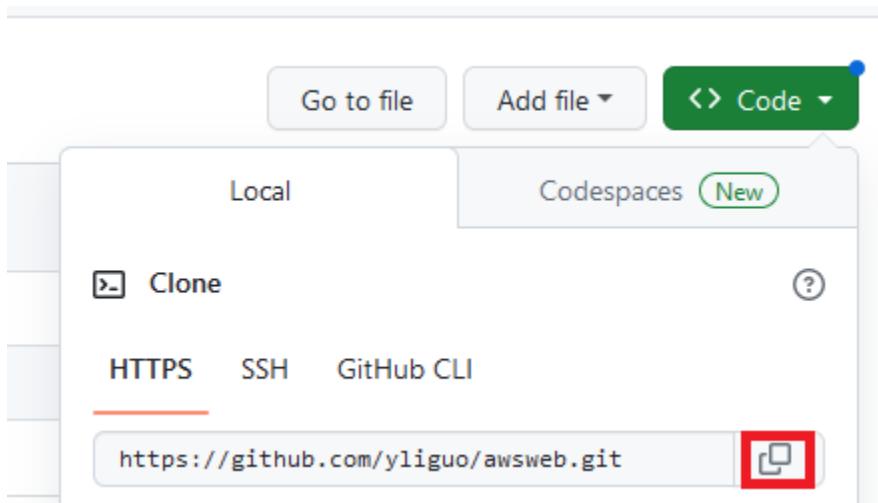
8. Click on file `hometown.html`, which is empty.
9. Click on “edit” symbol.



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

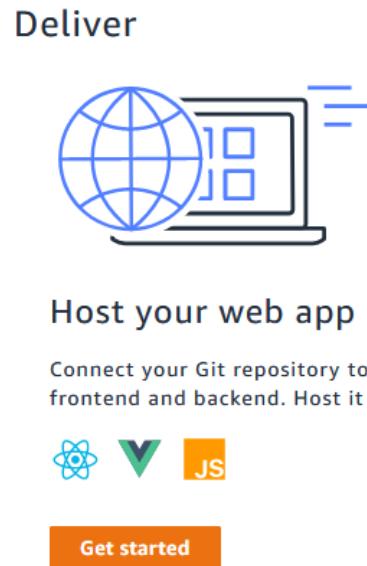
23. Execute the following command to push the change again.

**\$ 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”.

**Build and test settings**  
We've auto-detected your app's build settings. Please ensure your build command and output folder (baseDirectory) are correctly detected.

```

1 |version: 1
2 |frontend:
3 |  phases:
4 |    # IMPORTANT - Please verify your build commands
5 |    build:
6 |      commands: []
7 |
8 |artifacts:
9 |  # IMPORTANT - Please verify your build output directory
10 |  baseDirectory: /
11 |  files:
12 |    - '**/*'
13 |  cache:
14 |    paths: []

```

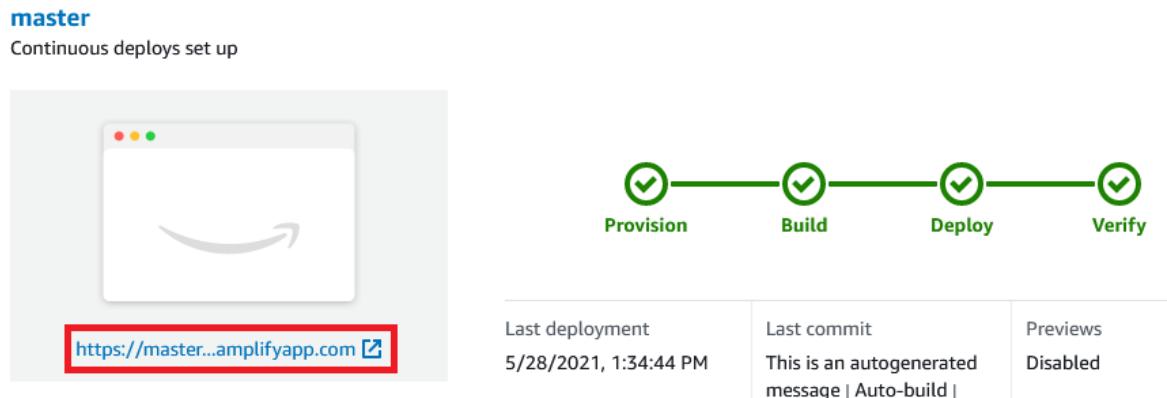
Build and test settings [Download](#) [Edit](#)

Allow AWS Amplify to automatically deploy all files hosted in your project root directory

► Advanced settings

[Cancel](#) [Previous](#) **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.