

# Project 2 - SES

---

This project is going to build off what I've learned so far this semester. At the end, it will utilize Serverless, NOSQL databases, IAM, CloudWatch, and a few new services to create a system that will automatically send a “stat of the day” email.

## Section 1: Simple Email Service (SES) Introduction setup

Source: <https://aws.amazon.com/ses/>

We'll use SES with Lambda to send out a notification email when an event is triggered. SES (Simple Email Service) is a cloud-based email service provider that can integrate into any application for high-volume email automation.

## Amazon Simple Email Service

Optimize your email communication with reliable, scalable, and secure solutions that ensure compliance and efficiency at competitive prices

Get started with Amazon SES

Contact sales

## Section 2: IAM Permissions and setup

1. Before we can use AWS SES, we need to give our IAM account permissions to do so.
2. Sign into AWS console using your **root** account.
3. Add permissions “**AmazonSESFullAccess**” and “**AmazonEventBridgeFullAccess**” to your EC2 IAM user.
  - Be sure to “attach policies directly”

It should look like this:

## Permissions summary (2)

Name 

[AmazonSESFullAccess](#)

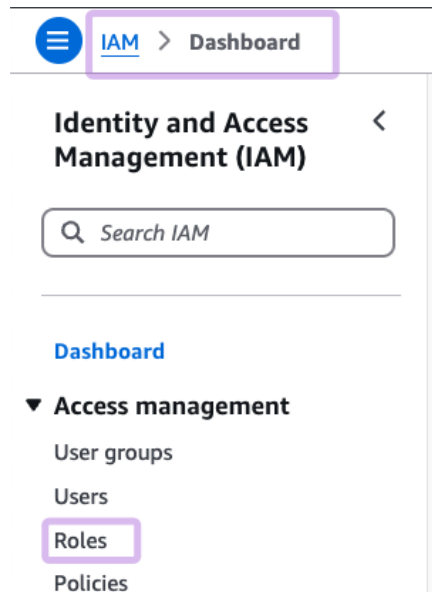
[AmazonEventBridgeFullAccess](#)

## Section 3: IAM roles – EC2

We'll be testing our code in VS Code, and then eventually applying it to Lambda, which will also use DynamoDB and SES. So, we need to establish **Roles** to permit these services to communicate with EC2.

So, we'll need to create new IAM **roles** with the appropriate permissions to allow this to happen.

1. In the AWS console (still with your Root account), go to IAM (Identity and Management)
2. Select Roles



Create role

3. Create a new role
4. Select "EC2" as the common use case

**Trusted entity type**

☒ **AWS service**  
 Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ **AWS account**  
 Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☐ **Web identity**  
 Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

☐ **SAML 2.0 federation**  
 Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ **Custom trust policy**  
 Create a custom trust policy to enable others to perform actions in this account.

**Use case**

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

**Service or use case**

EC2

Choose a use case for the specified service.

**Use case**


☒ **EC2**  
 Allows EC2 instances to call AWS services on your behalf.

- Click on the Next button

**Next**


- Add permission for **AmazonDynamoDBFullAccess** and **SESFullAccess**

Policy name

☒ ☐  [AmazonDynamoDBFullAccess](#)

Before you click a button. Also add permissions for “SESFullAccess”

Policy name

☒ ☐  [AmazonSESFullAccess](#) AWS ma

- Then click on “Next”

Next

#### 8. Call the Role: "Project2EC2Role"

##### Role name

Enter a meaningful name to identify this role.

Project2EC2Role

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

#### 9. Click on "Create role"

Create role

### Section 4: IAM roles -- Lambda

Repeat Section 3. Except this time in step 4, select "Lambda" as the common use case. Repeat all of the other steps the same except name this role "Project2LambdaRole"

##### Role name

Enter a meaningful name to identify this role.

Project2LambdaRole

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

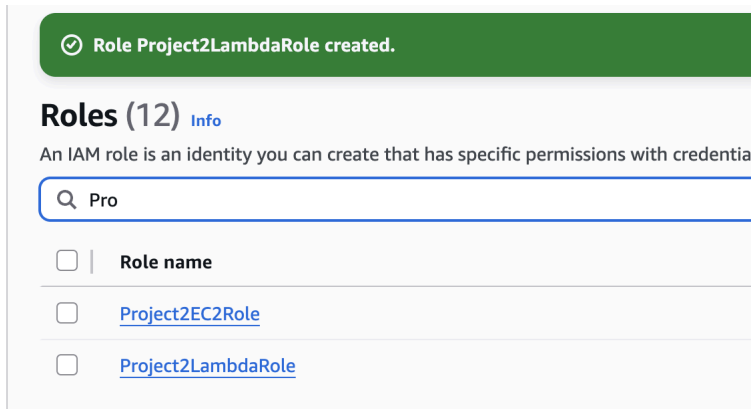
#### Permissions policy summary

Policy name 

[AmazonDynamoDBFullAccess](#)

[AmazonSESFullAccess](#)

Create role



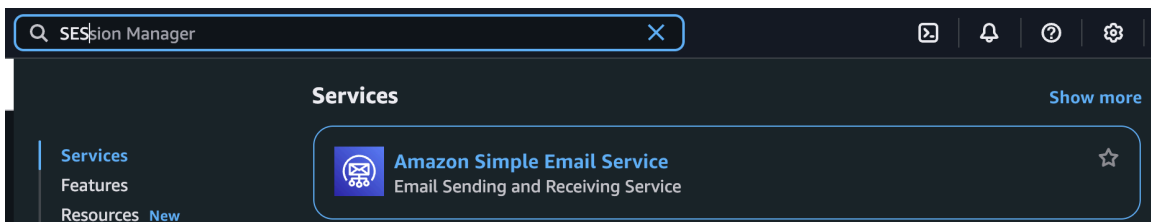
## Section 5: Update your EC2 IAM

1. You'll need to update the Role for the EC2 instance to the new role created in Section 3.
2. Go to the AWS EC2 dashboard
3. Select your instance
4. Select Actions → Security → Modify IAM role
5. Select the Project2EC2Role
6. Press "Update IAM role"
7. Log out of your Root Account
8. Log in to your EC2 Account

## Section 6: SES

In this section, you'll use **Amazon SES (Simple Email Service)** to send your stat of the day via email. SES lets you send messages to a "topic," and anyone subscribed to that topic will receive the messages by email, SMS, etc.

1. In the AWS Console, go to SES



1. Click on **Identities** on the left-hand side

## Identities

The **Identities** pane lists your domains, subdomains, and email address identities. All identities must be verified before you use them to send email in Amazon SES. [Learn more](#) [?]. The **Recommendations** pane lists high-impact email authentication issues found for the identities you select and check for recommendations. [Learn more](#) [?]

**Identities (0)** [Info](#) Last updated 1 minute ago [Check for recommendations](#) [Send test email](#) [Delete](#) [Create identity](#)

Identity

Identity type

Identity status

**No identities**  
No identities to display.

[Create identity](#)

2. Click Create Identity
  - click "Next"
3. Choose Email address and enter your email address

**Identity details** [Info](#)

**Identity type**  
☐ Domain  
To verify ownership of a domain, you must have access to its DNS settings to add the necessary records.

☒ Email address  
To verify ownership of an email address, you must have access to its inbox to open the verification email.

① Sending email from an email address identity without having the domain identity verified will result in your message being quarantined or rejected depending on the domain's DMARC policy. [Learn more about DMARC and how to look up a domain's DMARC policy.](#) [?]

**Email address**  
  
Email address can contain up to 320 characters, including plus signs (+), equals signs (=) and underscores (\_).

☐ Assign a default configuration set  
Enabling this option ensures that the assigned configuration set is applied to messages sent from this identity by default whenever a configuration set isn't specified at the time of sending.

4. Click Create Identity

**Action required**  
To verify ownership of this identity, check your inbox for a verification request email and click the link provided. [Resend](#)

**joseph.joswiak@drake.edu** [Refresh](#) [Delete](#) [Send test email](#)

5. Verify your identity by checking your inbox and clicking the confirmation link

### Amazon Web Services – Email Address Verification Request in region US East (N. Virginia)

AS

Amazon Web Services<no-reply-aws@amazon.com>  
To: ☺ Joseph Joswiak  
Wed 5/14/2025 10:52 AM

☺ ↶ ↷ ↘ ↙ 📎 📎 📎 ⋮

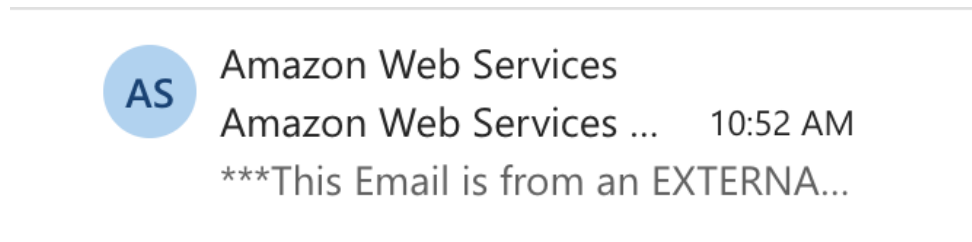
\*\*\*This Email is from an EXTERNAL source. Ensure you trust this sender before clicking on any links or attachments.\*\*\*

Dear Amazon Web Services Customer,

We have received a request to authorize this email address for use with Amazon SES and Amazon Pinpoint in region US East (N. Virginia). If you requested this verification, please go to the following URL to confirm that you are authorized to use this email address:

6. Confirm the subscription:

- a. Go to your email inbox, open the confirmation email from “Amazon Web Services”



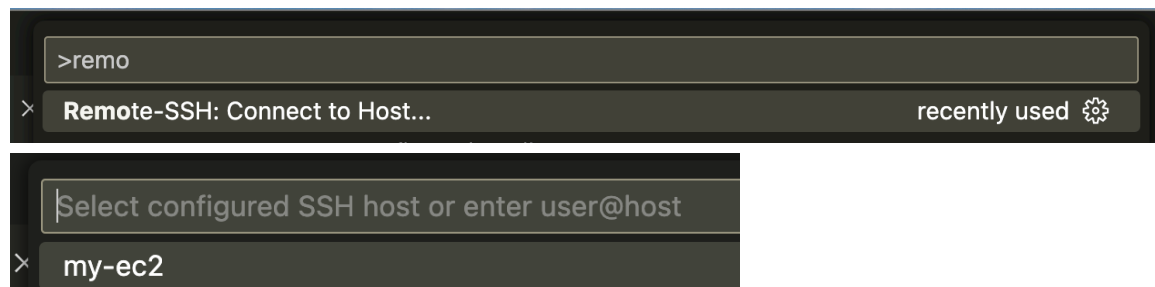
- b. Click on the link to confirm
- c. Your subscription status should now say “Congratulations” in the AWS console.

# Congratulations!

You have successfully verified an email address. You can now start sending email from this address.

## Section 7: Python for Sending Email

1. Connect to your EC2 instance:
  - a. Use the Remote-SSH extension, and connect to your EC2 instance via the extension.



2. Open a Terminal:
3. Now, go ahead and clone the starter repository to your EC2 instance:  
<https://github.com/merriekay/cs178-lab16-starter> with the following command in your EC2 Terminal:  
`git clone https://github.com/merriekay/cs178-lab16-starter.git`
  - This will download the starter code files `sendJoke.py` and `emailTest.py` to your EC2 instance.

```
[ec2-user@ip-172-31-20-48 ~]$ git clone https://github.com/merriekay/cs178-lab16-starter.git
Cloning into 'cs178-lab16-starter'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 18 (delta 7), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (18/18), 5.89 KiB | 1.96 MiB/s, done.
Resolving deltas: 100% (7/7), done.
[ec2-user@ip-172-31-20-48 ~]$
```

- You can skip this step if you have it from Lab16Lambda2
- 4. Now, if you type **ls** into the terminal, you should see a new folder on your EC2 instance called **cs178-lab16-starter**.
  - You can skip this step if you have it from Lab16Lambda2

```
[ec2-user@ip-172-31-80-102 ~]$ ls
CS178_Project1  FlaskApp  cs178-lab16-starter  emailTest.py  exit  movies.sql  mysql80-community-release-el9-1.noarch.rpm
[ec2-user@ip-172-31-80-102 ~]$
```

5. Let's change directory into the **cs178-lab16-starter** by running:
  - **cd cs178-lab16-starter**
  - You can skip this step if you have it from Lab16Lambda2

```
[ec2-user@ip-172-31-80-102 ~]$ cd cs178-lab16-starter
[ec2-user@ip-172-31-80-102 cs178-lab16-starter]$
```

6. Now, run the command **ls** to list the contents of the folder. You should see something like this:

```
[ec2-user@ip-172-31-80-102 cs178-lab16-starter]$ ls
README.md  emailTest.py  sendJoke.py
[ec2-user@ip-172-31-80-102 cs178-lab16-starter]$
```

7. We'll be using the built in file editor **nano** for the Terminal to make a couple of small updates to these files.
  - a. Run the command: **nano emailTest.py**

```
[ec2-user@ip-172-31-20-48 cs178-lab16-starter]$ nano emailTest.py
```

8. Replace everything in **emailTest.py** with this SES + DynamoDB version:

```
import boto3
import random
from botocore.exceptions import ClientError

def lambda_handler(event, context):
    # DynamoDB setup
    dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
    table = dynamodb.Table('Project2Stats')

    # Pull random stat
    response = table.scan()
    stat = random.choice(response['Items'])['stat']

    # SES setup
```



```

ses = boto3.client('ses', region_name='us-east-1')

SENDER = "your_verified_email@example.com"
RECIPIENT = "your_verified_email@example.com"

try:
    ses.send_email(
        Source=SENDER,
        Destination={'ToAddresses': [RECIPIENT]},
        Message={
            'Subject': {'Data': 'Stat of the Day'},
            'Body': {'Text': {'Data': stat}}
        }
    )
    print("Email sent!")
except ClientError as e:
    print("Error:", e.response['Error']['Message'])

```

- a. update "your\_verified\_email@example.com" with your email address
- b. To save, **ctrl + O**, press **Enter** to confirm the filename, and then **ctrl + X** to exit.

## Section 8: Introduction to UUIDs

A **universally unique identifier (UUID)** is a label used for information in computer systems. You may also hear them referred to as a **globally unique identifier (GUID)**. UUIDs are, for practical purposes, unique, and are often used as unique keys for databases. Here are a few examples:

90d1e9fe-b59d-11ec-b909-0242ac120002

c42937c6-b59d-11ec-b909-0242ac120002

77cd8b61-11b9-4b83-9291-bf99df236676

Take a look at the following website that will generate UUIDs:

<https://www.uuidgenerator.net/>

## Section 9: Create a DynamoDB table

1. Go to DynamoDB
2. Create a new table called "Project2Stats" and call the Partition key "uuid" (which will be stored as a String)

## Create table

**Table details** [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

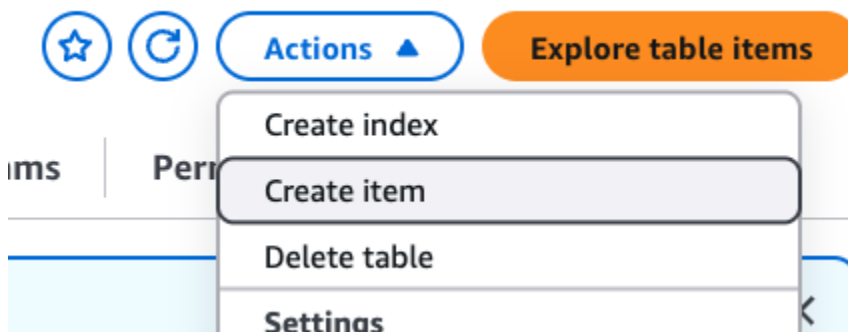
Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

3. Keep the rest of the default values and press “Create table”

4. Once created, select the “Project2Stats” table and select Actions → Create item



5. Go to <https://www.uuidgenerator.net/> and copy a newly-generated UUID

6. paste the UUID in for the uuid Partition key

7. Click on “Add new attribute” → String. Call the attribute “stat”

8. Enter in a stat for the value.

**Attributes** [Add new attribute](#)

Attribute name	Value	Type
uuid - Partition key	<input type="text" value="3bb17e55-8576-4dd0-9053-e6d9fafa2730"/>	String
<input type="text" value="stat"/>	<input type="text" value="Adaptability in Playing Time: Peebles has shown adaptability in his role on the team. Despite variations in playing time—averaging between 16.6 to 21.3 minutes per game over four seasons—he has consistently contributed on the scoreboard. This flexibility indicates his ability"/>	String <a href="#">Remove</a>

1. Add at least three items to your table. Each item should have a newly-generated UUID for the partition key and an attribute called “stat”. The stat can be anything you want. Make sure that the attribute name is always “stat” (exactly spelling; exact capitalization).

At the end, your “Project2Stats” table should look something like this:

**Table: Project2Stats - Items returned (3)**

Actions

Create item

Scan started on May 14, 2025, 11:31:51

<

1

>

⚙

<input type="checkbox"/>	uuid (String)	stat
<input type="checkbox"/>	<a href="#">3bb17e55-8576-4dd...</a>	Adaptability in Playing Time: Peebles has shown adaptability in his role o...
<input type="checkbox"/>	<a href="#">d9df9361-c113-416c...</a>	Impressive Shooting Efficiency: Over his collegiate tenure, Peebles has de...
<input type="checkbox"/>	<a href="#">c86c3605-a4f8-453e...</a>	Consistent Scoring Across Seasons: Peebles has maintained a steady scori...

Section 10: Python for getting stats

- 1. Go back to VS Code
- 2. Create a new file in your Project2 Folder called **databaseTest.py**, you can create this one locally (rather than on your EC2 instance).
- 3. Enter the following code:

```
import boto3
import random

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table = dynamodb.Table('Project2Stats')

response = table.scan()
stat = random.choice(response['Items'])['stat']
print(stat)
```

- 4. Run it to verify that you get a stat printed. Run it several times to verify that you get a random joke selected from your database.

```
databaseTest.py
(.venv) josephjoswiak@Josephs-MacBook-Pro CS 178 % "/Users/josephjoswiak/Desktop/Spring 2025 Classes/CS 178/.venv/bin/python" "/Users/josephjoswiak/Desktop/Spring 2025 Classes/CS 178/Project2 Folder/databaseTest.py"
Adaptability in Playing Time: Peebles has shown adaptability in his role on the team. Despite variations in playing time—averaging between 16.6 to 21.3 minutes per game over four seasons—he has consistently contributed on the scoreboard. This flexibility indicates his ability to make an impact regardless of his time on the court.
(.venv) josephjoswiak@Josephs-MacBook-Pro CS 178 %
```

- If you get a **python not found** error, try using the run button instead of typing python3 databaseTest.py.
- 5. Copy this code into your **emailTest.py** file (back on your EC2 instance). Update the email code so that the body of the email includes your random stat.
  - Note that you may need to make sure the indentation of the code is consistent (e.g. using tabs vs. spaces) in emailTest.py.

## Section 11: Lambda testing

1. Go to AWS Lambda.
2. Create a new Lambda function. Call the function “Project2Lambda”. Change the runtime to Python 3.9.

**Create function** [Info](#)  
Choose one of the following options to create your function.


☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Continue with template**  
Select a template to create your function.

**Basic information**


**Function name**  
Enter a name that describes the purpose of your function.  
  
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-)

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
 



**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

Also scroll down and change the default execution role to “Project2Lambda” that you created in Section 4.

**▼ 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](#) .

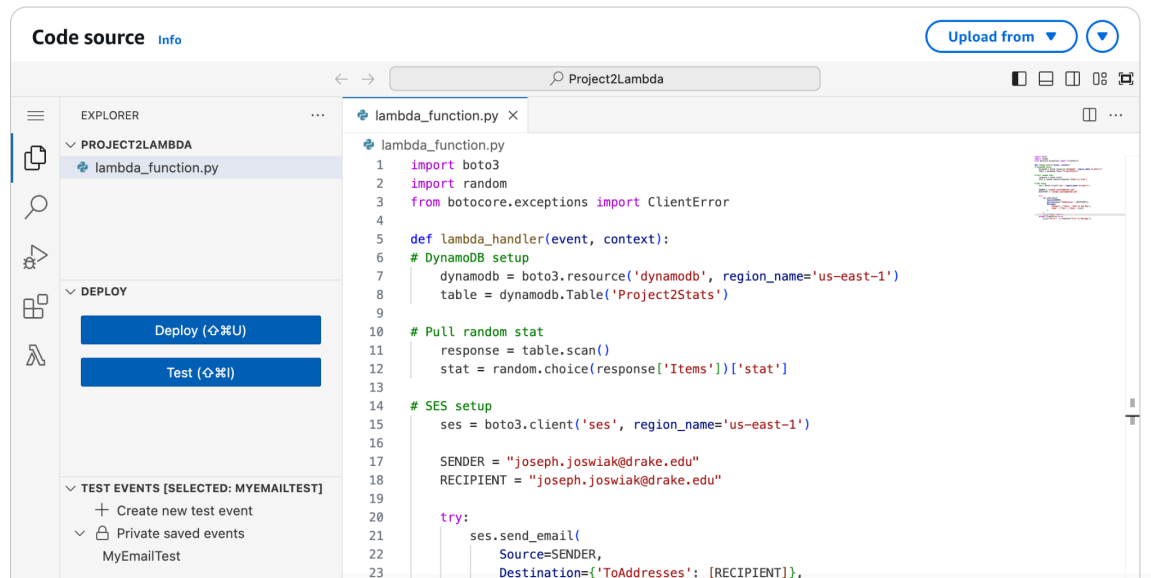
☐ Create a new role with basic Lambda permissions  
☒ Use an existing role  
☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch.  
   
[View the Project2LambdaRole role](#)  on the IAM console.

Then press “Create function”

3. Remove all of the code in the default **lambda\_function.py** file.

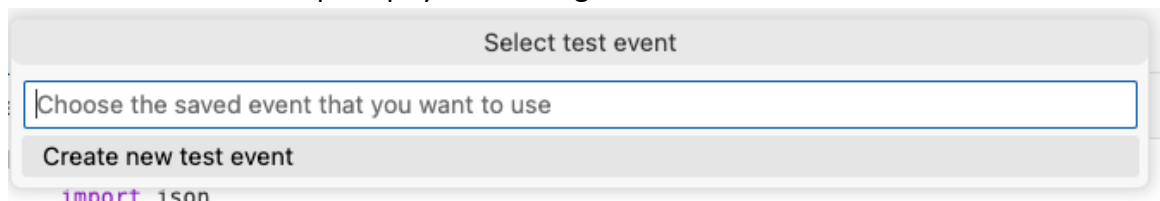
- Copy the code from **emailTest.py** file to inside of your lambda function, update the code to pull a random joke from your dynamodb table (the code from databaseTesting.py should come in handy here).



- Save. Then deploy your code



- Click on "Test". This will prompt you to configure a test event. Name the event



7. “MyEmailTest”, keep the default Event json and click on “Save”

Create new test event

Create new test event

Invoke

Save

Event Name

MyEmailTest

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 ten. [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 }
```

8. Go back to the lambda console and click on info about Project2Lambda, Scroll down and select “Configuration”

Code

Test

Monitor

Configuration

Aliases

Versions

General configuration

Triggers

Permissions

Destinations

General configuration

Description

-

Timeout

0 min 3 sec

Memory

128 MB

SnapStart

None

Info

Info

Then, click “Edit”



and change the timeout to 20 seconds:

**Timeout**

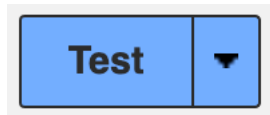
0 min 20 sec

and click “Save”

9. Make sure any changes are deployed



10. Click on “Test”



- 11.
12. This will generate a test run of the code in the lambda function. If all goes well, you should receive an email as a result. The logs will look like this:

```
Status: Succeeded
Test Event Name: MyEmailTest

Response:
null
```

- 13.

## Section 12: Event Trigger


Next, Let’s add a trigger so this lambda function executes at certain times during the day. At first, for testing, we’ll have it fire every minute. After we’re sure that it works, we’ll have it fire only at 7:00am every morning.


1. In you Lambda Function, click on “Add Trigger”

## Project2Lambda

▼ **Function overview** [Info](#) [Exp](#)

[Diagram](#) | [Template](#)

 **Project2Lambda**

 Layers (0)

[+ Add trigger](#) [+ Add destination](#)


2. Select EventBridge

## Trigger configuration [Info](#)

Select a source

event

**Batch/bulk data processing**

 **EventBridge (CloudWatch Events)**  
aws asynchronous schedule management-tools

- Click on “Create a new rule”
- Call the Rule name “Project2Trigger”
- Under the “Schedule expression” enter rate(1 minute)
  - This will make the event trigger every minute – great for testing.
- Click “Add” (see below)



Pick an existing rule, or create a new one.

☒ Create a new rule  
☐ Existing rules

**Rule name**  
Enter a name to uniquely identify your rule.

Project2Trigger

**Rule description**  
Provide an optional description for your rule.

**Rule type**  
Trigger your target based on an event pattern, or based on an automated schedule.

☐ Event pattern  
☒ Schedule expression

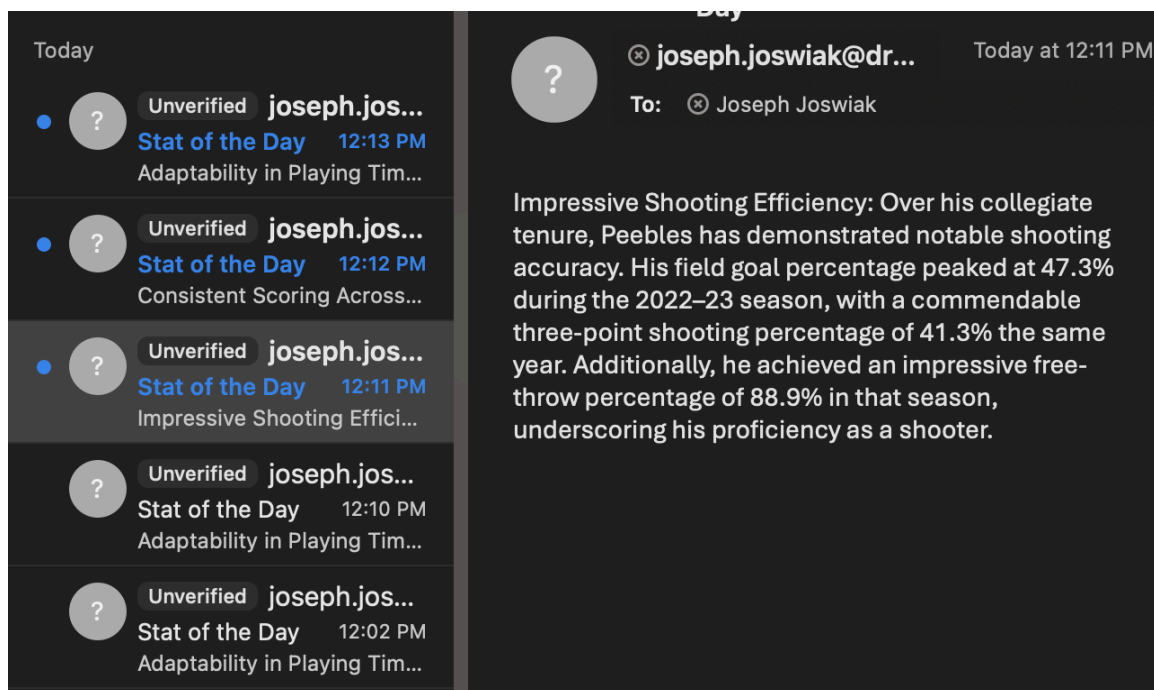
**Schedule expression**  
Self-trigger your target on an automated schedule using [Cron or rate expressions](#). Cron expressions are in UTC.

rate(1 minute)

e.g. rate(1 day), cron(0 17 ? \* MON-FRI \*)

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

- Wait a minute and check your email. You should be receiving new joke emails every minute.



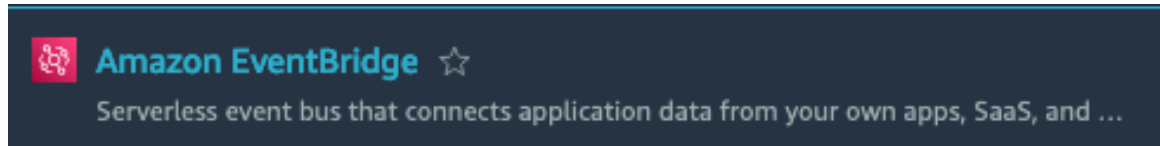
- Note: You're seeing "Unverified" because Amazon SES has not authorized your email domain (drake.edu) to send on its behalf. Even though your individual

email (joseph.joswiak@drake.edu) is verified, the domain itself (drake.edu) is not, which is why recipients see this warning.

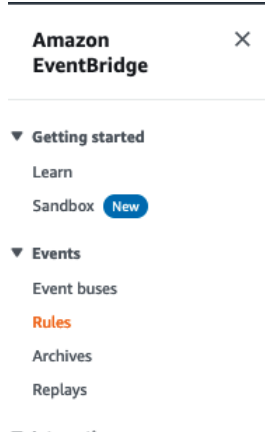
- Ignore the warning (Unverified). This is totally fine for labs, testing, and demos.

## Section 13: Event Trigger

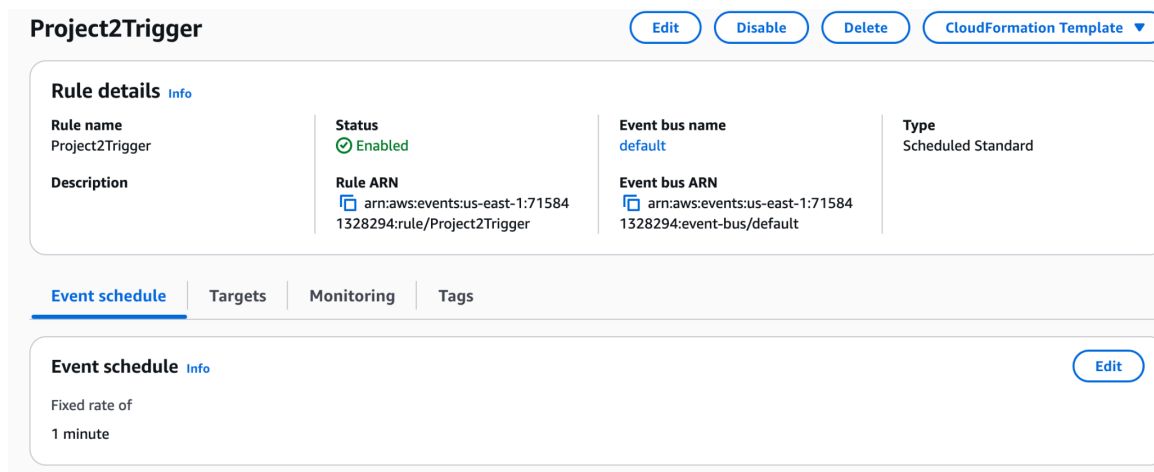
1. Let's change the trigger so it happens at 7:00am instead of every minute.
2. In AWS search for "EventBridge"



3. On the left hand side, click on "Rules"



4. You should see your Project2Trigger here. Click on it.



5. Under the "Event schedule", click on "Edit"
6. **Cron** is a utility that can specify a specific time occurrence during a day (e.g. 7:00am every weekday).
7. Change the Schedule pattern to look like this:

## Schedule pattern

### Schedule pattern


Choose the schedule type that best meets your needs.

☒ A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.

☐ A schedule that runs at a regular rate, such as every 10 minutes.

### Cron expression [Info](#)

Define the cron expression for the schedule

 cron (       )

Minutes

Hours

Day of month

Month

Day of week

Year

Next 10 trigger date(s)

Local time zone



Thu, May 15, 2025, 07:00 AM CDT  
Fri, May 16, 2025, 07:00 AM CDT  
Sat, May 17, 2025, 07:00 AM CDT  
Sun, May 18, 2025, 07:00 AM CDT  
Mon, May 19, 2025, 07:00 AM CDT  
Tue, May 20, 2025, 07:00 AM CDT  
Wed, May 21, 2025, 07:00 AM CDT  
Thu, May 22, 2025, 07:00 AM CDT  
Fri, May 23, 2025, 07:00 AM CDT  
Sat, May 24, 2025, 07:00 AM CDT

- Click Next. And Next. And Next And finally, "Update Rule"  
With the rule update, the trigger should only fire at 7:00am every morning.