

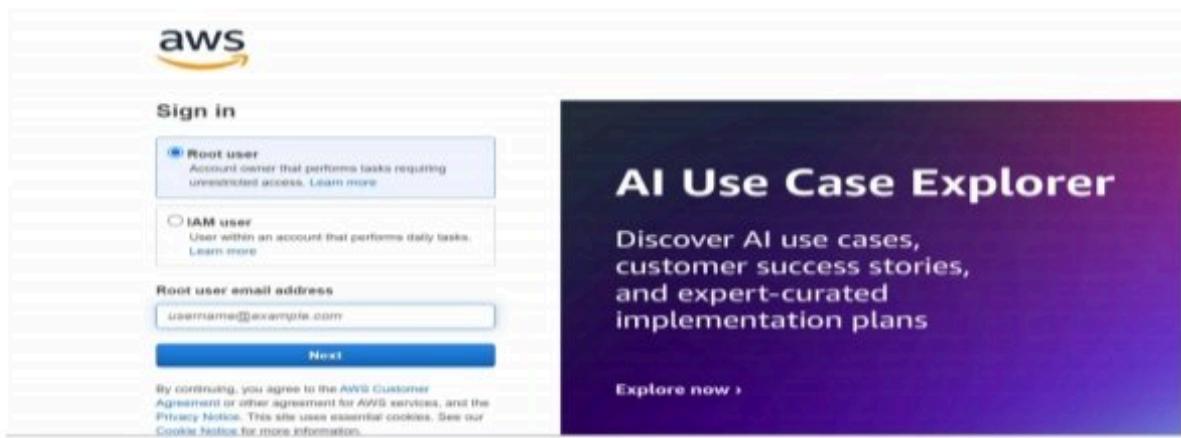
## Milestone 1: AWS Account Setup and Login

- **Activity 1.1: Set up an AWS account if not already done.**
  - Sign up for an AWS account and configure billing settings.



- **Activity 1.2: Log in to the AWS Management Console**

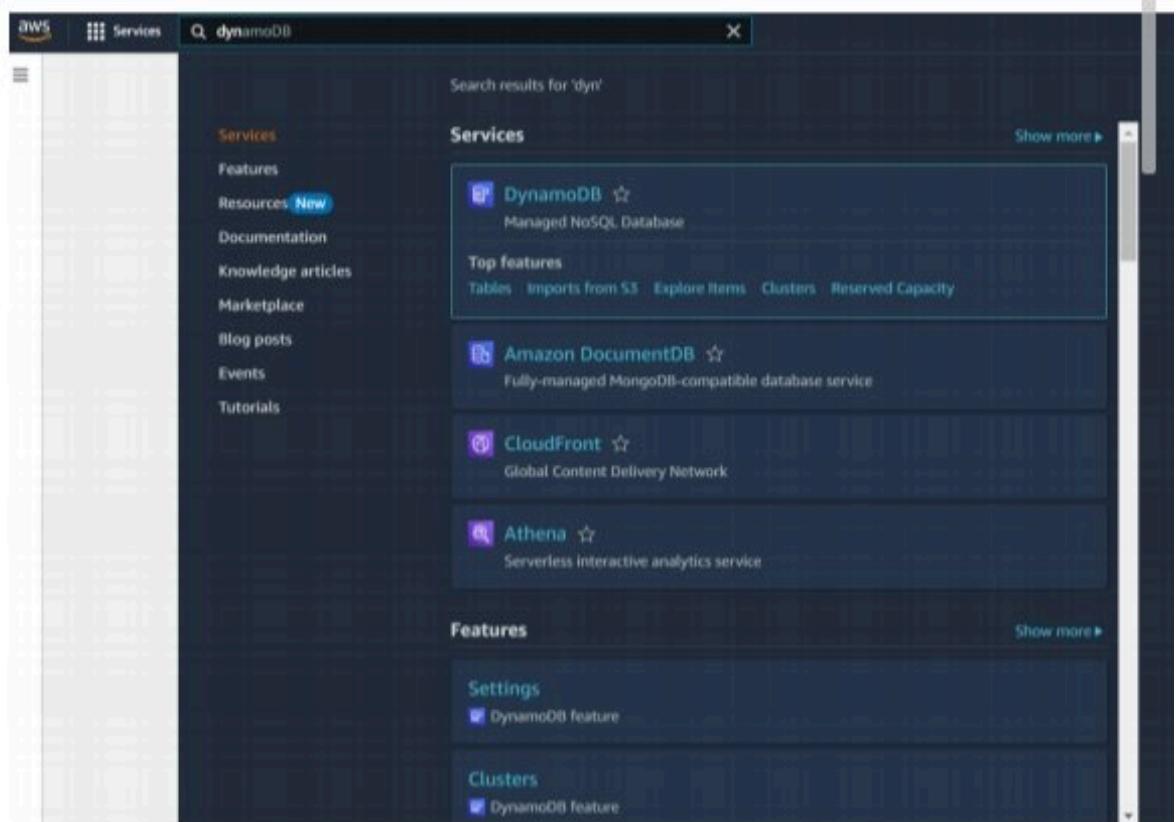
- After setting up your account, log in to the [AWS Management Console](#).



## Milestone 2: DynamoDB Database Creation and Setup

- **Activity 2.1: Navigate to the DynamoDB**

- In the AWS Console, navigate to DynamoDB and click on create tables.

A screenshot of the DynamoDB Dashboard. The left sidebar has sections for "Dashboard", "Tables", "Exports from S3", "Imports to S3", "Integrations (New)", "Reserved capacity", and "Settings". Under "DAX", it shows "Clusters" and "Events". The main dashboard has two main sections:

- Alarms (0) info:** A table with columns for "Alarm name" and "Status". It says "No custom alarms".
- DAX clusters (0) info:** A table with columns for "Cluster name" and "Status". It says "No clusters to display".

On the right, there's a "Create resources" section with a "Create table" button and a brief description of DAX. Below it is a "What's new" section with a single item from September 19.

- **Activity 2.2:Create a DynamoDB table for storing registration details and book requests.**
  - Create Users table with partition key "Email" with type String and click on create tables.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The path in the top navigation bar is 'DynamoDB > Tables > Create table'. The main title is 'Create table'. Under 'Table details', there is a note: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' field is set to 'Users', which is described as being between 3 and 255 characters, containing letters, numbers, underscores, hyphens, and periods. The 'Partition key' section shows 'email' as the key name, defined as a String type. A note states that the partition key is part of the primary key and is used to retrieve items from the table. The 'Sort key - optional' section shows an empty field for the sort key name and a String type dropdown, with a note that it is optional and allows sorting or searching among items sharing the same partition key.

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

## Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#)

[Create table](#)

The screenshot shows the AWS DynamoDB Tables page. On the left, there's a sidebar with links like Dashboard, Tables, Explore items, Partitions editor, Backups, Exports to S3, Imports from S3, and Integrations. The main area has a green header bar with the message "The Users table was created successfully". Below this, there's a table titled "Tables (1) info". The table has columns: Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. One row is shown for the "Users" table, which is "Active", uses "email" as the partition key, and has 0 sort keys. The "Read capacity mode" is set to "Provisioned (0)" and "Write capacity mode" is also "Provisioned (0)". The "Total size" is listed as "0 bytes". At the top right of the main area, there are buttons for "Actions", "Delete", and "Create table".

- Follow the same steps to create a requests table with Email as the primary key for book requests data.

## 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.

#### Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.



1 to 255 characters and case sensitive.

### Table settings

#### Default settings

This feature creates a standard table. Many more available.

#### Customize settings

Our team advanced features for more DynamoDB needs.

Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

### Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

[Add new tag](#)

You can add 50 more tags.

[Cancel](#)

[Create table](#)

The Requests table was created successfully.

Tables (2) <a href="#">Info</a>								
	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode
<input type="checkbox"/>	Requests	<span>Active</span>	email (\$1)	-	0	<input checked="" type="radio"/> Off	Provisioned (5)	Provisioned (5)
<input type="checkbox"/>	Users	<span>Active</span>	email (\$1)	-	0	<input checked="" type="radio"/> Off	Provisioned (5)	Provisioned (5)

## Milestone 3: SNS Notification Setup

- **Activity 3.1: Create SNS topics for sending email notifications to users and library staff.**

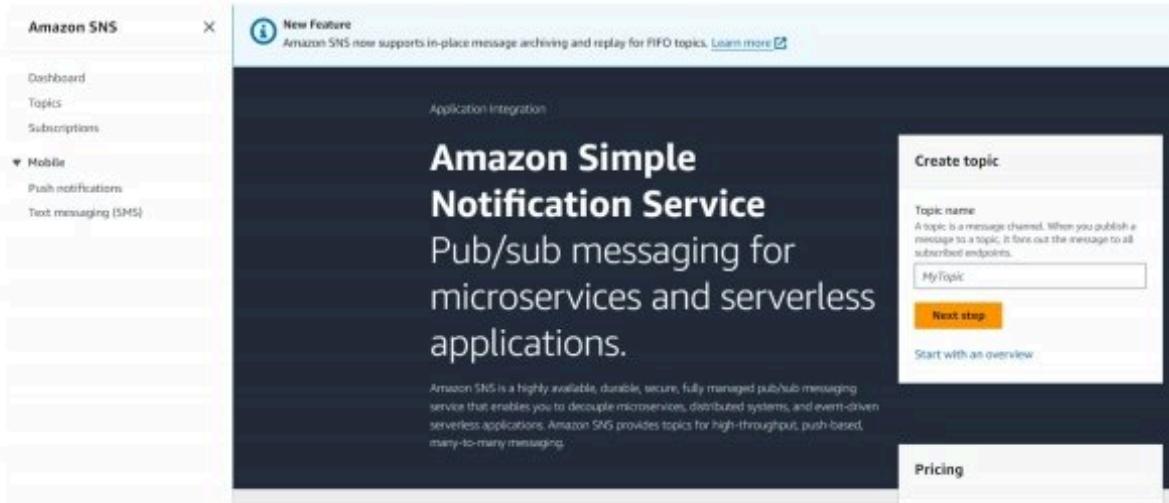
- In the AWS Console, search for SNS and navigate to the SNS Dashboard.

The screenshot shows the AWS search interface with the query 'sns' entered in the search bar. Below the search bar, it says 'Search results for 'sns''. On the left, there's a sidebar with links: Services, Features, Resources (New), Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main content area is titled 'Services' and shows the following results:

- Simple Notification Service** (star icon)  
SNS managed message topics for Pub/Sub
- Route 53 Resolver**  
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53** (star icon)  
Scalable DNS and Domain Name Registration
- AWS End User Messaging** (star icon)  
Engage your customers across multiple communication channels

Below this, under 'Features', there are three sections:

- Events**  
ElastiCache feature
- SMS**  
AWS End User Messaging feature
- Hosted zones**  
Route 53 feature



- Click on **Create Topic** and choose a name for the topic.



- Choose Standard type for general notification use cases and Click on Create Topic.

## Create topic

### Details

Type [Info](#)

Topic type cannot be modified after topic is created

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

BookRequestNotifications

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

Display name - optional [Info](#)

To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

► **Access policy - optional** Info  
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic.

► **Data protection policy - optional** Info  
This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic.

► **Delivery policy (HTTP/S) - optional** Info  
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section.

► **Delivery status logging - optional** Info  
These settings configure the logging of message delivery status to CloudWatch Logs.

► **Tags - optional**  
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. [Learn more](#)

► **Active tracing - optional** Info  
Use AWS X-Ray active tracing for this topic, to view its traces and service map in Amazon CloudWatch. Additional costs apply.

[Cancel](#)

[Create topic](#)

- Configure the SNS topic and note down the **Topic ARN**.

Amazon SNS

New Feature: Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

**Topic BookRequestNotifications created successfully**  
You can create subscriptions and send messages to them from this topic.

Topics > BookRequestNotifications

**BookRequestNotifications**

[Edit](#) [Delete](#) [Publish message](#)

**Details**

Name	BookRequestNotifications	Display name	
ARN	arn:aws:sns:ap-south-1:123456789012:BookRequestNotifications	Topic owner	SAI/Utility Tools
Type	Standard		

**Subscriptions (1)**

ID	Endpoint	Status	Protocol
1	test@example.com	Subscribed	Protocol

No subscriptions found  
You don't have any subscriptions for this topic.

[Create subscription](#)

- **Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications when a book request is made.**
  - Subscribe users (or admin staff) to this topic via Email. When a book request is made, notifications will be sent to the subscribed emails.

Amazon SNS > Subscriptions > Create subscription

### Create subscription

**Details**

Topic ARN  
arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications

Protocol  
The type of endpoint to subscribe  
Email

Endpoint  
An email address that can receive notifications from Amazon SNS.  
instantlibrary2@gmail.com

ⓘ After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)  
This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)  
Send undeliverable messages to a dead-letter queue.

[Cancel](#) [Create subscription](#)

The screenshot shows the AWS SNS console with a success message: "Subscription to BookRequestNotifications created successfully". The message details are: Subscription: d7Be0371-9235-404d-952c-85c2743607c4. The subscription status is "Pending confirmation". The endpoint is "mailto:firehawkray2@gmail.com" and the protocol is "EMAIL". The topic is "BookRequestNotifications".

- o After subscription request for the mail confirmation

The screenshot shows the AWS SNS console for the "BookRequestNotifications" topic. It displays the topic details: Name: BookRequestNotifications, Arn: arn:aws:sns:us-east-1:123456789012:BookRequestNotifications, Topic owner: SA Firehawk Ray, and Type: Standard.

The "Subscriptions" tab is selected, showing one pending confirmation subscription: Endpoint: "mailto:firehawkray2@gmail.com" and Protocol: "EMAIL". There are buttons for "Delete", "Request confirmation", "Confirm subscription", and "Delete subscription".

- o Navigate to the subscribed Email account and Click on the confirm subscription in the AWS Notification- Subscription Confirmation mail.

## AWS Notification - Subscription Confirmation Inbox X

AWS Notifications <no-reply@sns.amazonaws.com>  
to me ▾

9

You have chosen to subscribe to the topic:  
**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):  
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

AWS Notifications <no-reply@sns.amazonaws.com>  
to me ▾

You have chosen to subscribe to the topic:  
**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):  
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)



Simple Notification Service

### Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

**arn:aws:sns:ap-south-1:557690616836:BookRequestNotifications:d78e0371-9235-404d-952c-85c2743607c4**

If it was not your intention to subscribe, [click here to unsubscribe](#).

- o Successfully done with the SNS mail subscription and setup, now store the ARN link.

- File Explorer Structure

The screenshot shows a file explorer window with the following structure:

- HOME... (with icons for back, forward, search, and refresh)
- > static\images
- templates
  - <> about.html
  - <> cart.html
  - <> checkout.html
  - <> contact us.html
  - <> home.html
  - <> index.html
  - <> login.html
  - <> non\_veg\_pickles.html
  - <> signup.html
  - <> snacks.html
  - <> sucess.html
  - <> veg\_pickles.html
- app.py

Description of the code :

? Flask App Initialization

```
from flask import Flask, render_template, request, redirect, url_for, session
from werkzeug.security import generate_password_hash, check_password_hash
import boto3
from datetime import datetime
import json,uuid
```

```
app = Flask(__name__)
```



- Use boto3 to connect to DynamoDB for handling user registration, Order details database operations and also mention region\_name where Dynamodb tables are created.

```
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1') # e.g., 'us-east-1'
table = dynamodb.Table('Users')
```

- Use boto3 to connect to DynamoDB for handling user registration, Order details database operations and also mention region\_name where Dynamodb tables are created.

```
dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')
users_table = dynamodb.Table('Users')
orders_table = dynamodb.Table('Orders')
```

```
products = {
    'non_veg_pickles': [
        {'id': 1, 'name': 'Chicken Pickle', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 2, 'name': 'Fish Pickle', 'weights': {'250': 200, '500': 400, '1000': 800}},
        {'id': 3, 'name': 'Gongura Mutton', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 4, 'name': 'Mutton Pickle', 'weights': {'250': 400, '500': 800, '1000': 1600}},
        {'id': 5, 'name': 'Gongura Prawns', 'weights': {'250': 600, '500': 1200, '1000': 1800}},
        {'id': 6, 'name': 'Chicken Pickle (Gongura)', 'weights': {'250': 350, '500': 700, '1000': 1050}}
    ],
    'veg_pickles': [
        {'id': 7, 'name': 'Traditional Mango Pickle', 'weights': {'250': 150, '500': 280, '1000': 500}},
        {'id': 8, 'name': 'Zesty Lemon Pickle', 'weights': {'250': 120, '500': 220, '1000': 400}},
        {'id': 9, 'name': 'Tomato Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 10, 'name': 'Kakarakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 11, 'name': 'Chintakaya Pickle', 'weights': {'250': 130, '500': 240, '1000': 450}},
        {'id': 12, 'name': 'Spicy Pandu Mirchi', 'weights': {'250': 130, '500': 240, '1000': 450}}
    ], # Add your veg pickle products here
    'snacks': [
        {'id': 7, 'name': 'Banana Chips', 'weights': {'250': 300, '500': 600, '1000': 800}},
        {'id': 8, 'name': 'Crispy Aam-Papad', 'weights': {'250': 150, '500': 300, '1000': 600}},
        {'id': 9, 'name': 'Crispy Chekka Pakodi', 'weights': {'250': 50, '500': 100, '1000': 200}},
        {'id': 10, 'name': 'Boondhi Acchu', 'weights': {'250': 300, '500': 600, '1000': 900}},
        {'id': 11, 'name': 'Chekkalu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 12, 'name': 'Ragi Laddu', 'weights': {'250': 350, '500': 700, '1000': 1000}},
        {'id': 13, 'name': 'Dry Fruit Laddu', 'weights': {'250': 500, '500': 1000, '1000': 1500}},
        {'id': 14, 'name': 'Kara Boondi', 'weights': {'250': 250, '500': 500, '1000': 750}},
        {'id': 15, 'name': 'Gavvalu', 'weights': {'250': 250, '500': 500, '1000': 750}}
    ]
}
```

- Routes for Web Pages
- Login Route (GET/POST): Verifies user credentials, increments login count, and redirects to the dashboard on success.

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

    try:
        # Fetch user from DynamoDB
        response = users_table.get_item(Key={'username': username})

        if 'Item' not in response:
            return render_template('login.html', error='User not found')

        user = response['Item']

        # Ensure password field exists in the DB
        if 'password' not in user:
            return render_template('login.html', error='Password not found in database')

        # Verify password
        if check_password_hash(user['password'], password):
            session['logged_in'] = True
            session['username'] = username
            session.setdefault('cart', []) # Initialize cart if not set
            # session['cart'].append({'id': 1, 'name': 'apple', 'quantity': 1, 'price': 100})
```

• User registration data is stored in the database, hashed the password, and stores user

- SignUp route: Collecting registration data, hashes the password, and stores user details in the database.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username'].strip()
        email = request.form['email'].strip()
        password = request.form['password']

    try:
        # Check if username exists
        response = users_table.get_item(Key={'username': username})
        if 'Item' in response:
            return render_template('signup.html', error='Username already exists')

        # Hash password before storing
        hashed_password = generate_password_hash(password)

        # Store new user in DynamoDB
        users_table.put_item(
            Item={
                'username': username,
                'email': email,
                'password': hashed_password # Store hashed password
            }
        )
        return redirect(url_for('login'))

    except Exception as e:
        app.logger.error(f"Signup error: {str(e)}")
```

```
    hashed_password = generate_hashed_password(password)

    if User.query.filter_by(email=email).first():
        return redirect(url_for('login'))

    except Exception as e:
        app.logger.error(f"Signup error: {str(e)}")
        return render_template('signup.html', error='Registration failed. Please try again.')
```

```
except Exception as e:
    app.logger.error(f"Signup error: {str(e)}")
    return render_template('signup.html', error='Registration failed. Please try again.')
return render_template('signup.html')

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))
```

- Logout route: The user can Logout so that the user can get back to the Login Page

```
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))
```

- Home Route: Home page contains the routing for different categories which are Veg\_pickles, Non\_Veg\_pickles, Snacks.

- Home Route: Home page contains the routing for different categories which are Veg\_pickles,Non\_Veg\_pickles,Snacks.

```
@app.route('/home')
def home():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    return render_template('home.html')

@app.route('/non_veg_pickles')
def non_veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('non_veg_pickles.html', products=products['non_veg_pickles'])

@app.route('/veg_pickles')
def veg_pickles():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    # Simply pass all products without filtering
    return render_template('veg_pickles.html', products=products['veg_pickles'])

@app.route('/snacks')
def snacks():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    return render_template('snacks.html', products=products['snacks'])
```

(i) Restart Visual Studio

- Check out Route:

```
app.route('/checkout', methods=['GET', 'POST'])
def checkout():
    if not session.get('logged_in'):
        return redirect(url_for('login'))

    error_message = None # Variable to hold error messages

    if request.method == 'POST':
        try:
            # Extract form data safely
            name = request.form.get('name', '').strip()
            address = request.form.get('address', '').strip()
            phone = request.form.get('phone', '').strip()
            payment_method = request.form.get('payment', '').strip()

            # Validate inputs
            if not all([name, address, phone, payment_method]):
                return render_template('checkout.html', error="All fields are required.")

            if not phone.isdigit() or len(phone) != 10:
                return render_template('checkout.html', error="Phone number must be exactly 10 digits.")

            # Get cart data from hidden inputs
            cart_data = request.form.get('cart_data', '[]')
            total_amount = request.form.get('total_amount', '0')
```

(1) Restart Visual Studio Code to apply the latest changes.

```
try:
    cart_items = json.loads(cart_data)
    total_amount = float(total_amount)
except (json.JSONDecodeError, ValueError):
    return render_template('checkout.html', error="Invalid cart data format.")

# Ensure cart is not empty
if not cart_items:
```

ENG 14:52 05-07-2025

```
    cart_items = json.loads(cart_data)
    total_amount = float(total_amount)
    response = {"order":order, "cart_items": cart_items}
    return render_template('checkout.html', error="Failed save order. Please try again.")

# Checkout logic
if not cart_items:
    return render_template('checkout.html', error="Your cart is empty.")

# Store order in DynamoDB
try:
    orders_table.put_item(
        Item={
            'order_id': str(uuid.uuid4()),
            'username': session.get('username', 'Guest'),
            'name': name,
            'address': address,
            'phone': phone,
            'items': cart_items,
            'total_amount': total_amount,
            'payment_method': payment_method,
            'timestamp': datetime.now().isoformat()
        }
    )
except Exception as db_error:
    print(f"DynamoDB Error: {db_error}")
    return render_template('checkout.html', error="Failed to save order. Please try again later.")
```

```
# Redirect to success page with success message
return redirect(url_for('sucess', message="Your order has been placed successfully!"))

except Exception as e:
    print(f"Checkout error: {str(e)}")
    return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

return render_template('checkout.html') # Render checkout page for GET request
```



```
    if request.method == 'POST':
        print("POST method")
        print("Order details: ", request.form)
        result = place_order(request.form)
        if result['status']:
            return redirect(url_for('success', message="Your order has been placed successfully!"))
        else:
            return render_template('checkout.html', error="An unexpected error occurred. Please try again.")

    return render_template('checkout.html') # Render checkout page for GET request

@app.route('/success')
def success():
    return render_template('success.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True) # Add debug=True temporarily
```

## Milestone 5: IAM Role Setup

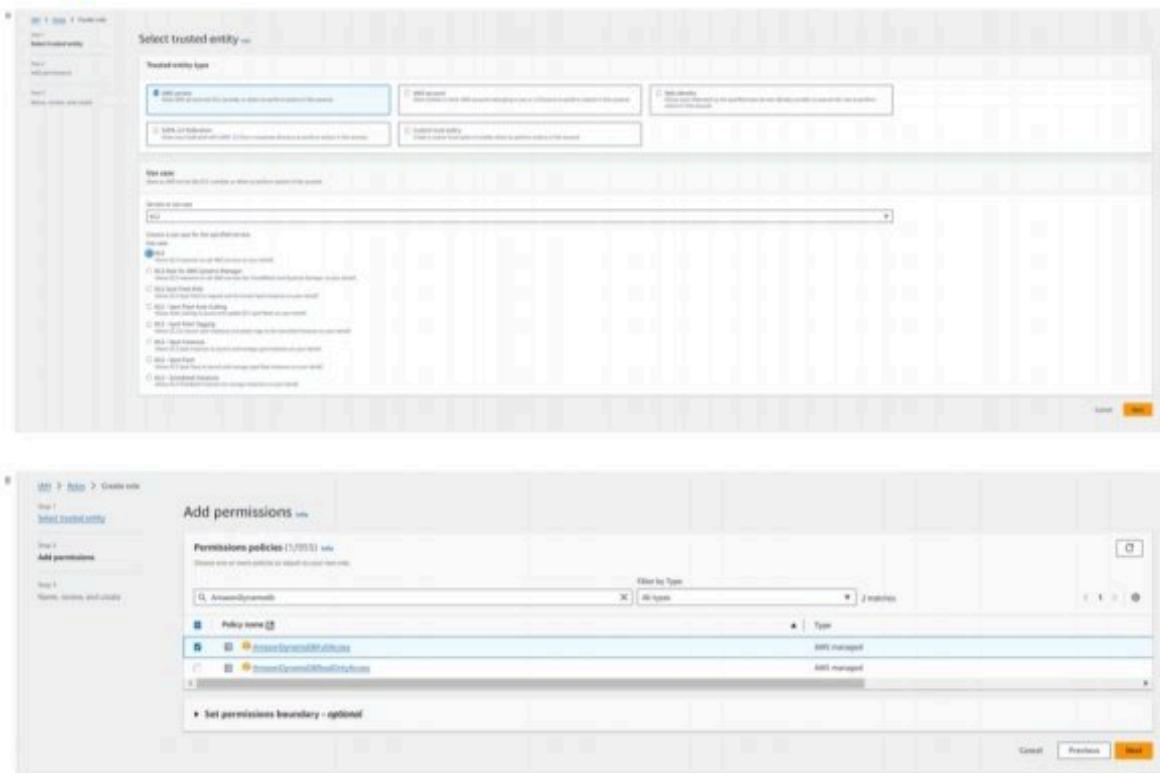
- **Activity 5.1: Create IAM Role.**

- In the AWS Console, go to IAM and create a new IAM Role for EC2 to interact with DynamoDB and SNS.

The screenshot shows the AWS Services search results for 'iam'. The search bar at the top contains 'Q iam'. Below it, a sidebar on the left lists various AWS services and features. The main area displays four service cards:

- IAM** Manage access to AWS resources
- IAM Identity Center** Manage workforce user access to multiple AWS accounts and cloud applications
- Resource Access Manager** Share AWS resources with other accounts or AWS Organizations
- AWS App Mesh** Easily monitor and control microservices

The screenshot shows the 'Roles' page in the IAM section of the AWS console. The title bar says 'Identity and Access Management (IAM) > Roles'. The page has a search bar and a table with columns for 'Role name', 'Trusted entities', and 'Last activity'. A single row is visible in the table.



The screenshot shows two sequential steps in the AWS IAM console for creating a new policy:

**Step 1: Select trusted entity**

- Trusted entity type:**
  - AWS Lambda: Allows AWS Lambda to invoke your Lambda function or invoke another Lambda function on behalf of the Lambda function.
  - AWS account: Grants AWS Lambda permission to invoke Lambda functions in a different AWS account.
  - AWS Lambda role: Grants AWS Lambda permission to assume the role.
  - IAM role: Grants AWS Lambda permission to assume the role.
  - Lambda role boundary: Grants AWS Lambda permission to assume the role with boundary conditions.
- Role name:**
- Permissions boundary:**

**Step 2: Add permissions**

- Permissions policies (0/1000) -**
- Policy name:**
- Permissions:**
  - AmazonDynamoDBFullAccess
  - AmazonSNSFullAccess
  - AmazonDynamoDBStreams
- Type:** AWS managed
- Set permissions boundary - optional**

- **Activity 5.2: Attach Policies.**

Attach the following policies to the role:

- **AmazonDynamoDBFullAccess:** Allows EC2 to perform read/write operations on DynamoDB.
- **AmazonSNSFullAccess:** Grants EC2 the ability to send notifications via SNS.

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

### Add permissions

Permissions policies (1/555) info

Choose one or more policies to attach to your new role.

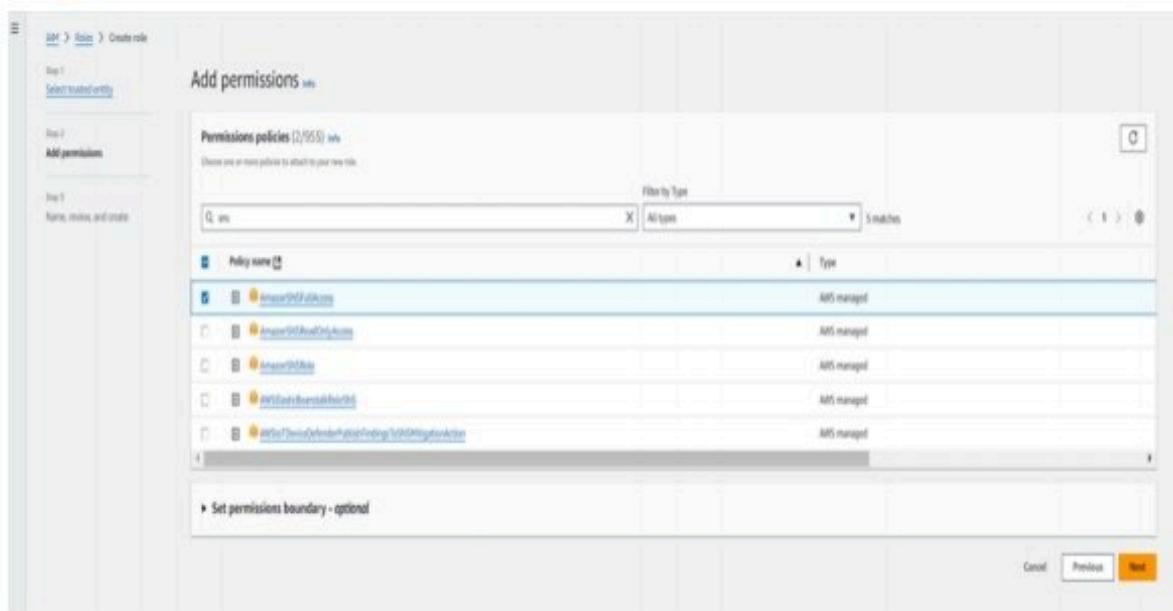
Filter by Type: All types | Snatched

Policy name:

- AmazonS3FullAccess (AWS managed)
- AmazonS3ReadWriteAccess (AWS managed)
- AmazonS3ReadOnlyAccess (AWS managed)
- AmazonVPCFullAccess (AWS managed)
- AmazonDynamoDBFullAccess (AWS managed)

Set permissions boundary - optional

Cancel | Preview | Next



Step 1 Select trusted entities

Step 2 Name, review, and create

Step 3 Name, review, and create

### Name, review, and create

New details

Name:

Description:

Permissions boundary:

Step 4 Select trusted entities

Trust policy:

```
Version: 2012-10-17
Statement: [
    {
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sts:AssumeRole"
    }
]
```

Step 5 Add permissions

Permissions policy summary:

Policy name	Type	Action
AmazonS3FullAccess	AWS managed	AmazonS3FullAccess
AmazonVPCFullAccess	AWS managed	AmazonVPCFullAccess

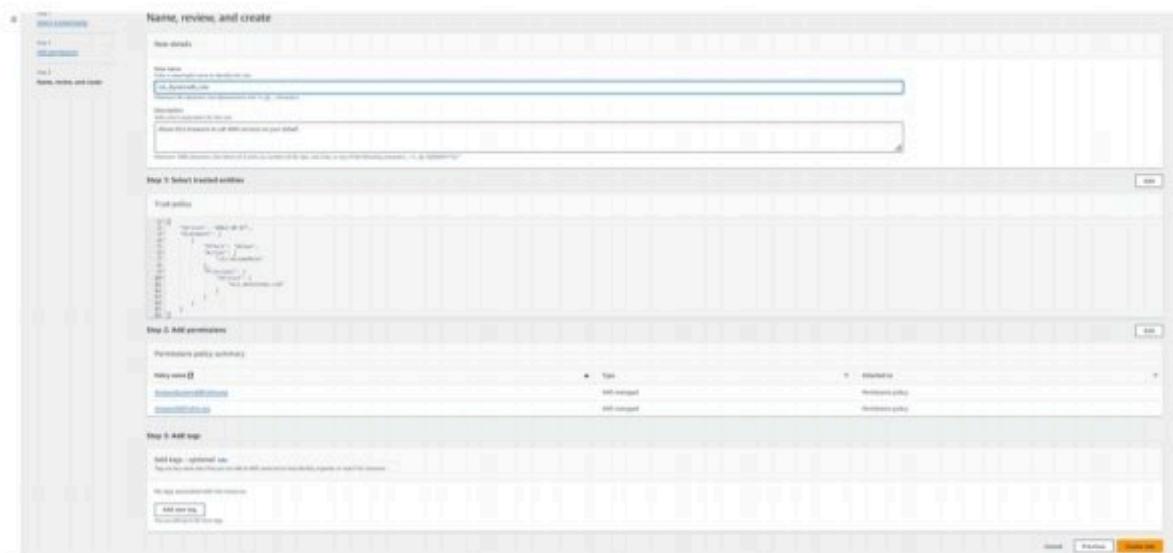
Step 6 Add tags

Add tags - optional? Yes

Add tag associated with this resource

Add new tag

Cancel | Previous | Create role



The screenshot shows the AWS IAM Role configuration page for the role 'sns\_Dynamodb\_role'. The 'Summary' tab is selected, displaying details such as creation date (October 18, 2024), ARN (arn:aws:iam::017000768336:role/sns\_Dynamodb\_role), and maximum session duration (1 hour). The 'Permissions' tab is active, showing two managed policies attached: 'AmazonDynamoDBFullAccess' and 'AmazonSQSFullAccess'. There are tabs for 'Trust relationships', 'Tags', 'Last Accessed', and 'AWS CloudTrail'.

## Milestone 6: EC2 Instance Setup

- **Note: Load your Flask app and Html files into GitHub repository.**

	static	Initial commit
	templates	Update statistics.html
	app.py	Update app.py

 Clone



HTTPS    SSH    GitHub CLI

<https://github.com/AlekhyaPenubakula/InstantLil>



Clone using the web URL.

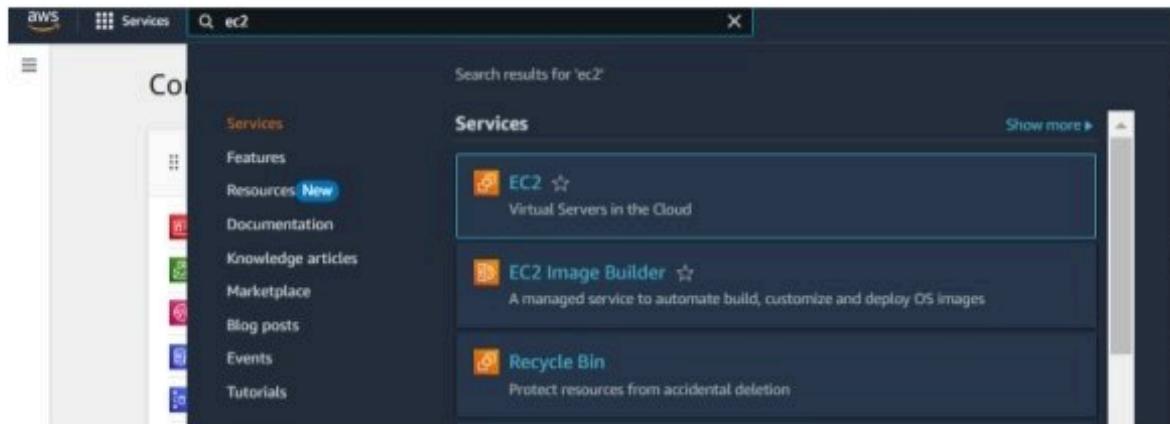
 Open with GitHub Desktop

 Download ZIP

- **Activity 6.1: Launch an EC2 instance to host the Flask application.**

- **Launch EC2 Instance**

- In the AWS Console, navigate to EC2 and launch a new instance.



- Click on Launch instance to launch EC2 instance

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances, Instance Types, Launch Templates, Spot Requests, and Savings Plans. The main area has a search bar at the top with the placeholder 'Find instance by attribute or tag (case-sensitive)'. Below it is a table header with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. A message in the center says 'No instances' and 'You do not have any instances in this region'. At the bottom right of the table area is a large orange 'Launch Instances' button.

This screenshot shows the first step of the 'Launch an instance' wizard. It has a title 'Launch an instance' with a 'Info' link. Below it is a sub-section titled 'Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.' The main form is titled 'Name and tags' with a 'Info' link. It has a 'Name' field containing 'InstantLibraryApp' and a 'Add additional tags' link. To the right, there's a 'Summary' section with a table:

Number of instances	Info
1	

Below the summary is a 'Software Image (AMI)' section with 'Amazon Linux 2023 AMI 2023.5.2...read more' and 'ami-078264b8ba71bc45e'. There are also sections for 'Virtual server type (instance type)' (set to 't2.micro') and 'Firewall (security group)'.

- Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible).

This screenshot shows the details page for the 'Amazon Linux 2023 AMI'. At the top, there's a grid of icons for different AMIs: Amazon Linux, macOS, Ubuntu, Windows, and Red Hat. To the right is a search icon and a link 'Browse more AMIs'. Below the grid, there's a section titled 'Amazon Machine Image (AMI)' with a sub-section for 'Amazon Linux 2023 AMI'. This section includes the AMI ID 'ami-02b49a24cfb95941c', its architecture ('64-bit (x86)'), boot mode ('uefi-preferred'), and AMI ID ('ami-02b49a24cfb95941c'). It also indicates that it's 'Free tier eligible'. Further down, there's a 'Description' section with a detailed paragraph about Amazon Linux 2023. At the bottom, there are filters for 'Architecture' (set to '64-bit (x86)'), 'Boot mode' (set to 'uefi-preferred'), and 'AMI ID' (set to 'ami-02b49a24cfb95941c'). A green 'Verified provider' badge is visible.

▼ Create and download the key pair for server access.

▼ Instance type [Info](#) | [Get advice](#)

Instance type	Free tier eligible
t2.micro Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Linux base pricing: 0.0124 USD per Hour On-Demand Windows base pricing: 0.017 USD per Hour On-Demand RHEL base pricing: 0.0268 USD per Hour On-Demand SUSE base pricing: 0.0124 USD per Hour	<input checked="" type="checkbox"/> All generations <a href="#">Compare instance types</a>

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select [Create new 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

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



InstantLibrary.pem

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture: 64-bit (x86) Boot mode: uefi-preferred AMI ID: ami-07e264b6ba71bc45e Username: ec2-user Verified provider

▼ Summary

Number of Instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.5.2.. read more ami-07e264b6ba71bc45e

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GB

▼ Instance type

t2.micro Family: t2 1 vCPU 1 GB Memory Current generation: true Free tier eligible On-Demand Linux base pricing: \$0.04 USD per Hour On-Demand Windows base pricing: \$0.05 USD per Hour On-Demand RHEL base pricing: \$0.058 USD per Hour On-Demand SUSE base pricing: \$0.034 USD per Hour

All generations Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required InstantLibrary Create new key pair

Can cancel Preview code Launch instance

ⓘ Free tier: In your first year includes 790 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.



**▼ Network settings** [Info](#)

VPC - required [Info](#)

vpc-03cdc7b6f19dd7211	(default) <a href="#">▼</a>	<a href="#">Create new VPC</a>
172.31.0.0/16		

Subnet [Info](#)

No preference	<a href="#">▼</a>	<a href="#">Create new subnet</a> <a href="#">Edit</a>
---------------	-------------------	--

Auto-assign public IP [Info](#)

Enable	<a href="#">▼</a>
--------	-------------------

Additional charges apply when outside of free tier allowance.

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

<input checked="" type="radio"/> <a href="#">Create security group</a>	<input type="radio"/> <a href="#">Select existing security group</a>
--	--

Security group name - required

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and \_-./()#,@[]+=&,:;%^\$\*.

Description - required [Info](#)

Inbound Security Group Rules					
<span style="float: right;"><a href="#">Remove</a></span> ▼ Security group rule 1 (TCP; 22, 0.0.0.0/0)					
Type	Info	Protocol	Info	Port range	Info
ssh		TCP		22	
Source type	Info	Source	Info	Description - optional	Info
Anywhere		Add CIDR, prefix list or security		e.g. SSH for admin desktop	
		0.0.0.0/0	X		
<span style="float: right;"><a href="#">Remove</a></span> ▼ Security group rule 2 (TCP; 80, 0.0.0.0/0)					
Type	Info	Protocol	Info	Port range	Info
HTTP		TCP		80	
Source type	Info	Source	Info	Description - optional	Info
Custom		Add CIDR, prefix list or security		e.g. SSH for admin desktop	
		0.0.0.0/0	X		
<span style="float: right;"><a href="#">Remove</a></span> ▼ Security group rule 3 (TCP; 5000, 0.0.0.0/0)					
Type	Info	Protocol	Info	Port range	Info
Custom TCP		TCP		5000	
Source type	Info	Source	Info	Description - optional	Info
Custom		Add CIDR, prefix list or security		e.g. SSH for admin desktop	
		0.0.0.0/0	X		
<a href="#">Add security group rule</a>					

**Next Steps**

What would you like to do next with this instance? (For example: "Create alarm" or "Create budget")

- Create billing and free tier usage alerts
- Connect to your instance
- Connect an RDS database
- Create EBS snapshot policy
- Manage detailed monitoring
- Create Load Balancer
- Create AWS budget
- Manage CloudWatch alarms
- Disaster recovery for your instances
- Monitor for suspicious runtime activities
- Get instance screenshot
- Get system log

Show all services

- To connect to EC2 using **EC2 Instance Connect**, start by ensuring that an **IAM role** is attached to your EC2 instance. You can do this by selecting your instance, clicking on **Actions**, then navigating to **Security** and selecting **Modify IAM Role** to attach the appropriate role. After the IAM role is connected, navigate to the **EC2** section in the **AWS Management Console**. Select the **EC2 instance** you wish to connect to. At the top of the **EC2 Dashboard**, click the **Connect** button. From the connection methods presented, choose **EC2 Instance Connect**. Finally, click **Connect** again, and a new browser-based terminal will open, allowing you to access your EC2 instance directly from your browser.

Instances (1/1) info

Last updated less than a minute ago

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

All states ▾

C Connect

InstanceState ▾

Actions ▾

Launch instance ▾

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4	Private IP	Monitoring	Security
instantLibrary...	i-001861022fbaec290	Stopped	t2.micro	-	Vin alarms +	ap-south-1b	-	-	-	disabled	launched

**Instance summary for i-001861022fbaec290 (instantLibraryApp)**

InstanceState: Stopped

Public IPv4 address: -

Private IPv4 DNS: -

Private IP: 10.0.0.1022

Private IP: 10.0.0.1022

Instance type: t2.micro

Public IPv4 DNS: -

Private IP: 10.0.0.1022

Subnet ID: subnet-001861022fbaec290

VPC ID: vpc-050d76d91967231

Auto assigned IP address: -

AMI Role: arn:aws:iam::role/lambdaBasicExecutionRole

IP/ENCL Required

AWS Compute Optimizer Enabled: Opt-in to AWS Compute Optimizer for recommendations. Learn more

Auto Scaling Group name: -

Details | Metrics and alarms | Monitoring | Security | Networking | Storage | Tags



EC2 > Instances > i-001861022fbcac290 > Modify IAM role

## Modify IAM role Info

Attach an IAM role to your instance.

Instance ID

i-001861022fbcac290 (InstantLibraryApp)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

sns\_Dynamodb\_role

Cancel

Update IAM role

- Now connect the EC2 with the files

**Connect to instance** Info

Connect to your instance i-001861022fbcac290 (InstantLibraryApp) using any of these options

<b>EC2 Instance Connect</b>	Session Manager	SSH client	EC2 serial console
-----------------------------	-----------------	------------	--------------------

**⚠ Port 22 (SSH) is open to all IPv4 addresses**

Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 15.255.177.0/29. [Learn more](#).

Instance ID  
i-001861022fbcac290 (InstantLibraryApp)

Connection Type

- Connect using EC2 Instance Connect  
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
- Connect using EC2 Instance Connect Endpoint  
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

- Public IPv4 address  
15.200.229.59
- IPv6 address

Username  
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

**Note:** In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

[Cancel](#) [Connect](#)

```
A newer release of "Amazon Linux" is available.
Version 2023.6.20241010
Run "/usr/bin/dnf check-release-update" for full release and version update info
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Tue Oct 16 04:17:59 2024 from 15.255.177.5
[ec2-user@ip-172-31-3-5 ~]
```

i-001861022fbcac290 (InstantLibraryApp)  
PublicIPs: 15.201.74.42 PrivateIPs: 172.31.3.5



## Milestone 7: Deployment on EC2

### Activity 7.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y  
sudo yum install python3 git  
sudo pip3 install flask boto3
```