

# **Cloud-hosted Banking Data Analytics and Reporting System on AWS**

"CloudBank Analytics" is a comprehensive cloud-based banking data analytics and reporting system designed to streamline financial data processing, analysis, and visualization. The project leverages Amazon Web Services (AWS) for robust and scalable infrastructure, utilizing Amazon RDS for secure and efficient data storage, AWS Elastic Beanstalk for reliable web application hosting, and AWS Lambda for automated data processing and report generation.

The user-friendly web interface, built with Flask, allows bank administrators to log in to their personal accounts, creating a seamless experience for data visualization and report management. Once logged in, users are presented with a dashboard that serves as a central hub for all banking analytics activities. The dashboard prominently features real-time financial data, key performance indicators (KPIs), and options for generating custom reports.

By combining modern cloud technology with an intuitive user interface, "CloudBank Analytics" aims to provide financial institutions with powerful tools for monitoring customer behavior, managing risk, and ensuring regulatory compliance, ultimately improving decision-making and operational efficiency.

## **Scenarios**

### **Scenario 1: Real-time Transaction Monitoring**

Sarah, a bank's fraud detection analyst, logs into CloudBank Analytics during her morning routine. The dashboard immediately alerts her to unusual transaction patterns detected overnight. Using the real-time analytics feature, Sarah quickly investigates the flagged transactions, confirms a potential fraud attempt, and takes immediate action to protect the affected accounts.

### **Scenario 2: Custom Report Generation**

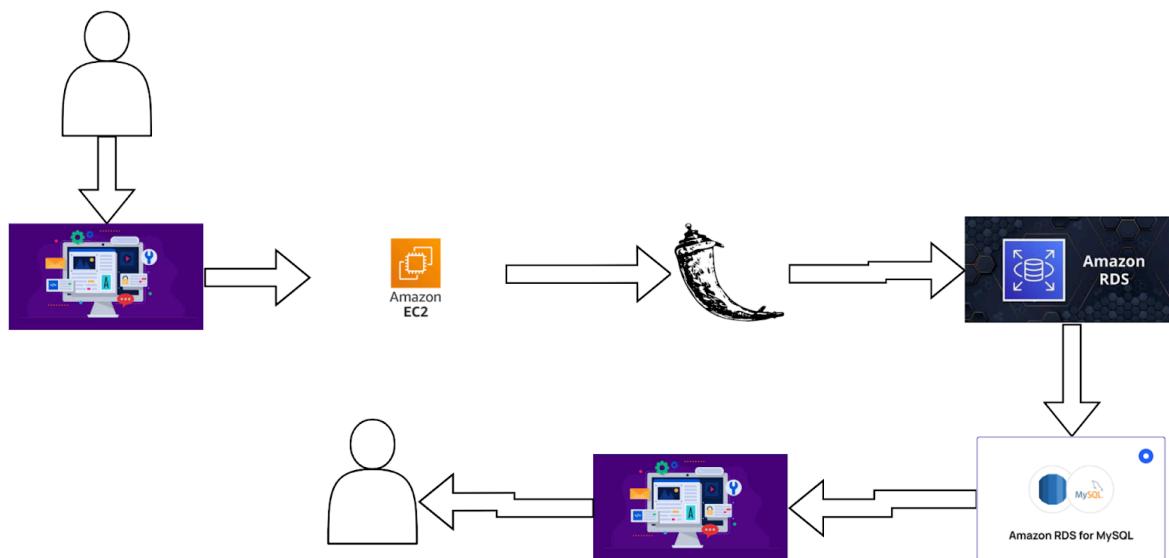
John, a financial manager, needs to prepare a comprehensive quarterly report for the board meeting. He logs into CloudBank Analytics and uses the custom report generation feature. John selects various metrics such as loan performance, deposit growth, and customer acquisition rates. The system, leveraging AWS Lambda, quickly

processes the vast amount of data stored in Amazon RDS and generates a detailed report, which is then stored in Amazon S3 for easy access and sharing.

### Scenario 3: Regulatory Compliance Monitoring

Lisa, a compliance officer, uses CloudBank Analytics to ensure the bank meets all regulatory requirements. She accesses a specialized dashboard that tracks key compliance metrics in real-time. When she notices that a particular metric is approaching a regulatory threshold, she uses the system to drill down into the underlying data, identify the root cause, and initiate corrective actions before any compliance issues arise.

#### Architecture:



#### Prior Knowledge:

1. AWS Account Setup: [https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk\\_M4FfVM-Dh](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)
2. Web Application Stack : [FLask](#) || [MySQL Connector using flask](#) || [HTML/JS/CSS](#)
3. AWS EC2 Instance: [https://www.youtube.com/results?search\\_query=aws+ec2+oneshot](https://www.youtube.com/results?search_query=aws+ec2+oneshot)
4. RDS Database: [https://www.youtube.com/results?search\\_query=rds+oneshot](https://www.youtube.com/results?search_query=rds+oneshot)
5. MySQL: [https://www.youtube.com/results?search\\_query=mysql+tutorial](https://www.youtube.com/results?search_query=mysql+tutorial)
6. RDS to MySQL:[https://www.youtube.com/results?search\\_query=mysql+connector+for+rds](https://www.youtube.com/results?search_query=mysql+connector+for+rds)
7. Clone Git repo: [https://www.youtube.com/results?search\\_query=clone+github+repository](https://www.youtube.com/results?search_query=clone+github+repository)

8. AWS Cost Management: <https://youtu.be/OKYJCHHSWb4?si=aY3DOl1v26CfZxXA>

9. AWS IAM: [https://www.youtube.com/results?search\\_query=aws+iam](https://www.youtube.com/results?search_query=aws+iam)

10. AWS CloudWatch: [https://www.youtube.com/results?search\\_query=aws+cloudwatch](https://www.youtube.com/results?search_query=aws+cloudwatch)

## Project Flow:

### Project Initialization:

- Define objectives, scope, and KPIs; set up the AWS environment.

### EC2 Instance Setup:

- Launch and configure an EC2 instance to host the web application.

### RDS Database Setup:

- Create and configure an RDS instance with MySQL engine.

### Web Application Development:

- Develop the web application with registration, login, and dashboard features.

### Database Integration:

- Connect the web application to the RDS database using appropriate drivers.

### User Interface Implementation:

- Create user-friendly interfaces for registration, login, and blood request management.

### Testing and Optimization:

- Conduct thorough testing of all features and optimize for performance.

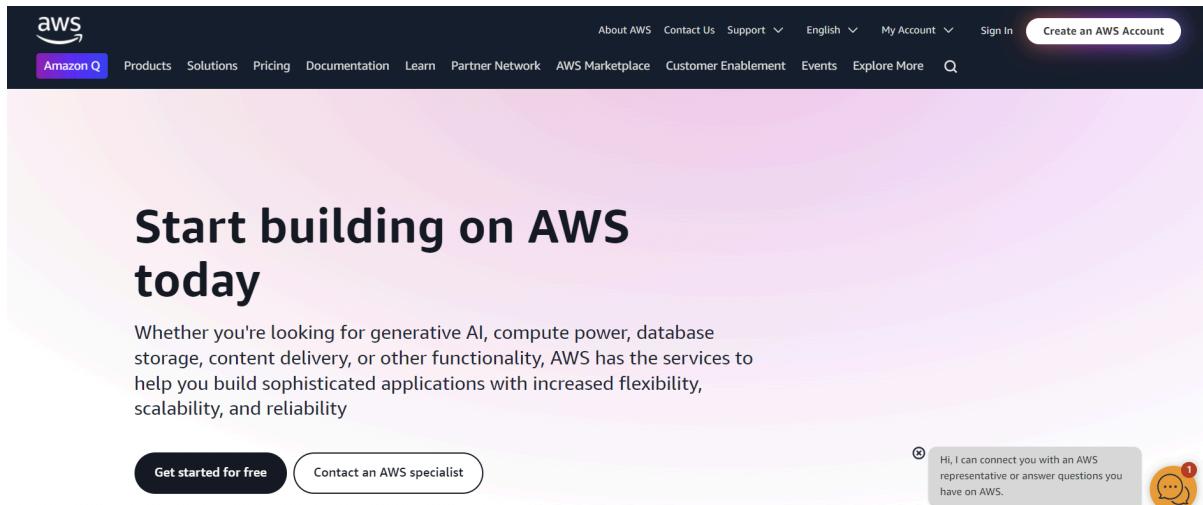
## Milestone 1: AWS Account Creation

In this milestone, we will set up an AWS account to access the necessary services for the BloodBridge project.

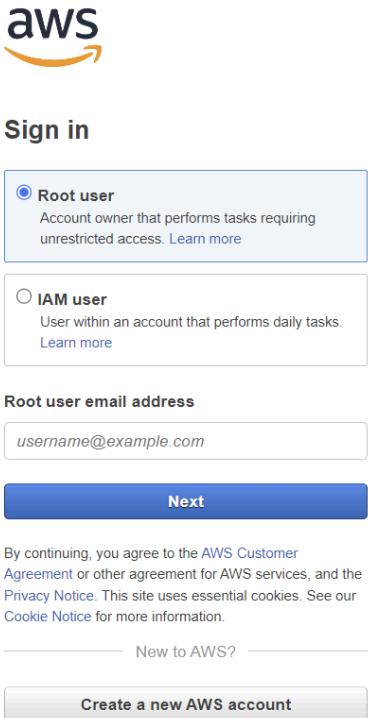
### Activity 1: Create AWS Account

1. Go to the AWS website.
2. Click on the "Create an AWS Account" button.
3. Follow the prompts to enter your email address and choose a password.

4. Provide the required account information, including your name, address, and phone number.
5. Enter your payment information. (Note: While AWS offers a free tier, a credit card is required for verification.)
6. Complete the identity verification process.
7. Choose a support plan (the basic plan is free and sufficient for starting).
8. Sign in to your new AWS account.



The screenshot shows the AWS homepage. At the top, there's a navigation bar with links for About AWS, Contact Us, Support, English, My Account, Sign In, and a search bar. A prominent pink banner in the center says "Start building on AWS today". Below the banner, a subtext reads: "Whether you're looking for generative AI, compute power, database storage, content delivery, or other functionality, AWS has the services to help you build sophisticated applications with increased flexibility, scalability, and reliability". Two buttons are visible: "Get started for free" and "Contact an AWS specialist". To the right, there's a message bubble from an AWS representative and a yellow notification icon with the number "1".

The screenshot shows the AWS sign-in page. It asks the user to choose between being a "Root user" (selected) or an "IAM user". The "Root user" option is described as an "Account owner that performs tasks requiring unrestricted access". Below this is a "Root user email address" input field containing "username@example.com" and a "Next" button. At the bottom, there's a cookie consent notice and a "Create a new AWS account" button.


The screenshot shows the AWS Training and Certification landing page. It features the AWS logo at the top, followed by the text "AWS Training and Certification" and "Propel your career. Get AWS certified.". There's a large hexagonal icon with a checkmark in the center. Below the main text, there's a "Learn more" link.

## Activity 2: Set Up IAM Users and Permissions

1. Access the IAM console from the AWS Management Console.
2. Create a new IAM user for yourself with administrative access.
3. Set up multi-factor authentication (MFA) for added security.
4. Create a group for developers and assign necessary permissions.
5. Generate access keys for programmatic access if needed.

IAM Dashboard

Security recommendations 0 

 Root user has MFA  
Having multi-factor authentication (MFA) for the root user improves security for this account.

 Root user has no active access keys  
Using access keys attached to an IAM user instead of the root user improves security.

IAM resources 

Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
1	3	8	0	1

 Services  [Alt+S]    Stockholm  Giri@1881-2230-9045 

 Console Home     

The screenshot shows the AWS IAM Users page. At the top, there are navigation icons for Home, Notifications, Help, and Settings, followed by the location 'Stockholm' and a user profile 'Giri @ 1881-2230-9045'. Below the header are two buttons: 'Reset to default layout' and '+ Add widgets'. On the right side, there are two small circular icons: one with an 'i' and another with a hexagon.

**Users (3) [Info](#)**

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

<input type="checkbox"/>	User name	Path	Group:	Last activity	MFA	Password age	Console last sign-in	Access I
<input type="checkbox"/>	Giri	/	1	3 minutes ago	-	3 minutes	September 18, 2024, 1...	-

[Create user](#)

## Milestone 2: Local Development Environment Setup

### Activity 1: Set Up Local Development Environment

1. Install Python and pip on your local machine.
2. Install Flask and other necessary Python packages (e.g., `flask-sqlalchemy`, `mysql-connector-python`).
3. Set up a virtual environment for your project.

### Activity 2: Create Flask Application Structure

1. Create a new directory for your Easybank project.
2. Set up a basic Flask application structure:

```
✓ BANK
  ✓ static
    ✓ assets
      ✓ css
        # main.css
        > img
        > js
        > scss
        > vendor
        > forms
      ✓ templates
        <> about.html
        <> balance.html
        <> confirm.html
        <> contact.html
        <> dashboard.html
        <> deposit.html
        <> index.html
        <> login.html
        <> register.html
        <> services.html
        <> statement.html
        <> support.html
      <> statement.html
      <> support.html
      <> transfer.html
      <> update.html
  ⚡ bank.py
```

## Milestone 3: Develop Easybank Web Application

### **Activity 1: Create Web Pages**

1. Design and create HTML templates for your banking application, including:
  - Home page
  - User registration page
  - Login page
  - Account dashboard
  - Transaction page
  - Account creation page
  - Check Balance page
  - Deposit page
  - Services page
  - Contact Page
  - confirm page
  - Account Login page
  - Statement page
2. Place these templates in the `templates/` directory.
3. Create CSS files in the `static/css/` directory for styling.
4. Add any necessary JavaScript files in the `static/js/` directory.

### **Activity 2: Implement Flask Routes and Views**

1. In `app.py`, create Flask routes for each of your web pages.
2. Implement view functions to render the appropriate templates.
3. Add form handling for user input (registration, login, transactions, etc.).

```
⚡ bank.py > ...
1  from flask import Flask, render_template, request, redirect, url_for
2  import mysql.connector
3  from flask import flash, session
4  from datetime import datetime
5
6  app = Flask(__name__)
7  app.secret_key = "your_secret_key" # Needed for flash msgs
8
9  # Database config
10 db_config = [
11     {
12         'host': 'bank.crqmssgockvo.ap-south-1.rds.amazonaws.com',
13         'user': 'admin',
14         'password': 'Surya123456',
15         'database': 'bank'
16     }
17 ]
18
19 cnxpool = mysql.connector.pooling.MySQLConnectionPool(pool_name="mypool",
20                                         pool_size=5,
21                                         **db_config)
22
23 # Function to establish a database connection
24 def get_db_connection():
25     try:
26         return cnxpool.get_connection()
27     except mysql.connector.Error as err:
28         print(f"Error: {err}")
29         return None
30
31 @app.route("/test-db-connection")
32 def test_db_connection():
33     try:
34         conn = get_db_connection()
35         cursor = conn.cursor()
36         cursor.execute("SELECT DATABASE();") # Test query to check connection
37         db_name = cursor.fetchone()
38         cursor.close()
39         conn.close()
40         return f"Connected to the database: {db_name[0]}"
41     except mysql.connector.Error as err:
42         return f"Error: {err}"
43
```

```
44 # Define your routes here
45 @app.route("/")
46 def index():
47     try:
48         conn = get_db_connection()
49         cursor = conn.cursor()
50     except mysql.connector.Error as err:
51         print(f"Error: {err}")
52     finally:
53         if conn:
54             cursor.close()
55             conn.close()
56     return render_template("index.html")
57
58
59 @app.route("/register", methods=['post', 'get'])
60 def register():
61     if request.method == 'POST':
62         # session.init_app(app) # Initialize the session
63         full_name = request.form['full_name']
64         email = request.form['email']
65         password = request.form['password']
66         phone = request.form['phone']
67         address = request.form['address']
68         aadhar_number = request.form['aadhar_number']
69         pan_card = request.form['pan_card']
70
71         conn = get_db_connection()
72         cursor = conn.cursor()
73
74         # Check if the user already exists
75         cursor.execute("SELECT * FROM users WHERE email = %s", (email,))
76         user = cursor.fetchone()
77
78         if user:
79             flash("email already exists! Please log in")
80             return redirect(url_for('login', email= email)) # redirect to login page
81
82         # Validate phone number and Aadhar number
83         if len(phone) != 10:
84             flash("Phone number must be 10 digits")
85             return render_template("register.html")
86         if len(aadhar_number) != 12:
```

```
✉ bank.py > ...
60  def register():
61      return render_template("register.html")
62  if len(aadhar_number) != 12:
63      flash("Aadhar number must be 12 digits")
64  return render_template("register.html")
65
66  # Insert the new user into the database
67  cursor.execute("INSERT INTO users (full_name, email, password, phone, address, aadhar_number, pan_card) VALUES (%s,%s, %s, %s, %s, %s, %s)", 
68      (full_name, email, password, phone, address, aadhar_number, pan_card))
69  conn.commit()
70  cursor.close()
71  conn.close()
72
73  user_data = {
74      'full_name': full_name,
75      'email': email,
76  }
77  session['user'] = user_data
78  flash("Registration successful! Please log in.")
79  return redirect(url_for('confirm',user=user_data))
80
81  return render_template("register.html")
82
83 @app.route("/confirm")
84 def confirm():
85     user = session.get('user')
86     if user:
87         return render_template("confirm.html", user=user)
88     else:
89         return redirect(url_for('login'))
90
91 @app.route("/login", methods=['post','get'])
92 def login():
93     if request.method == 'POST':
94         email = request.form['email']
95         password = request.form['password']
96
97         conn = get_db_connection()
98         cursor = conn.cursor()
99
100        # Verify login credentials
```

```
_bank.py > ...
116  def login():
124
125      # Verify login credentials
126      cursor.execute("SELECT * FROM users WHERE email = %s AND password = %s",
127                  (email, password))
128      user = cursor.fetchone()
129
130      cursor.close()
131      conn.close()
132
133      if user:
134          user_data = {
135              'fullname' : user[4],
136              'email': user[1],
137              'user_id':user[0]
138          }
139          session['user'] = user_data
140          print('this is email', email)
141          return redirect(url_for('dashboard'))
142      else:
143          flash("Invalid login credentials!")
144          return redirect(url_for('login'))
145
146
147      return render_template("login.html")
148
149
150 @app.route("/dashboard")
151 def dashboard():
152     user_data = session.get('user')
153     if user_data:
154         email = user_data['email']
155         conn = get_db_connection()
156         cursor = conn.cursor()
157         cursor.execute("SELECT * FROM users WHERE email = %s", (email,))
158         user = cursor.fetchone()
159         cursor.close()
160         conn.close()
161         return render_template("dashboard.html", user=user)
162     else:
163         return redirect(url_for('login'))
164
165 @app.route("/deposit", methods=['post', 'get'])
```

```
#!/usr/bin/python

#bankpy > ...

165 @app.route("/deposit", methods=['post', 'get'])
166 def deposit():
167     user_data = session.get('user')
168     if user_data:
169         if request.method == 'POST':
170             amount = float(request.form['deposit_amount'])
171             account_type = request.form['account_type']
172             conn = get_db_connection()
173             cursor = conn.cursor()
174             cursor.execute("SELECT 1 FROM accounts WHERE user_id = %s", (user_data['user_id'],))
175             if not cursor.fetchone():
176                 # Insert a new row
177                 cursor.execute("INSERT INTO accounts (user_id, balance, account_type) VALUES (%s, %s, %s)", (user_data['user_id'], amount, account_type))
178                 cursor.execute("INSERT INTO account_statements (user_id, transaction_type, transaction_amount, transaction_date) VALUES (%s, 'Credit', %s, %s)", (user_data['user_id'], amount, datetime.datetime.now()))
179             else:
180                 # Update the existing row
181                 cursor.execute("UPDATE accounts SET balance = balance + %s WHERE user_id = %s", (amount, user_data['user_id']))
182                 cursor.execute("INSERT INTO account_statements (user_id, transaction_type, transaction_amount, transaction_date) VALUES (%s, 'Credit', %s, %s)", (user_data['user_id'], amount, datetime.datetime.now()))
183             conn.commit()
184             cursor.close()
185             conn.close()
186             flash("Funds deposited successfully!")
187             return redirect(url_for('dashboard'))
188     return render_template("deposit.html")
189 else:
190     return redirect(url_for('login'))
191
192 @app.route("/balance", methods=['get'])
193 def check_balance():
194     user_data = session.get('user')
195     if user_data:
196         email = user_data['email']
197         conn = cxnpool.get_connection()
198         cursor = conn.cursor()
199         cursor.execute("SELECT balance FROM accounts WHERE user_id = %s", (user_data['user_id'],))
200         balance = cursor.fetchone()[0]
201         cursor.close()
202         conn.close()
203         return render_template("balance.html", balance=balance)
204     else:
205         return redirect(url_for('login'))
```

```
❸ bank.py > ...
225 def transfer():
249     if sender_balance is None:
250         flash("Sender account not found!")
251         return redirect(url_for('transfer'))
251
252     sender_balance = sender_balance[0]
253
254     if sender_balance >= amount:
255         # Update sender's balance
256         cursor.execute("UPDATE accounts SET balance = balance - %s WHERE user_id = %s", (amount, user_data['user_id']))
257
258         # Update recipient's balance
259         cursor.execute("UPDATE accounts SET balance = balance + %s WHERE user_id = %s", (amount, user_id))
260
261         # Insert a new transaction into the account_statements table
262         cursor.execute("INSERT INTO account_statements (user_id, transaction_type, transaction_amount, transaction_date) VALUES (%s, 'Debit', %s, %s)", (user_data['user_id'], amount, datetime.now()))
263         cursor.execute("INSERT INTO account_statements (user_id, transaction_type, transaction_amount, transaction_date) VALUES (%s, 'Credit', %s, %s)", (user_id, amount, datetime.now()))
264
265         conn.commit()
266         cursor.close()
267         conn.close()
268         flash("Funds transferred successfully!")
269         return redirect(url_for('dashboard'))
270     else:
271         flash("Insufficient balance!")
272         return redirect(url_for('transfer'))
273 except ValueError:
274     flash("Invalid transfer amount!")
275     return redirect(url_for('transfer'))
276 else:
277     flash("Please fill in all fields!")
278     return redirect(url_for('transfer'))
279 return render_template('transfer.html')
280
281 else:
282     return redirect(url_for('login'))
283
284 @app.route("/statement", methods=['GET'])
285 def statements():
286     user_data = session.get('user')
287     if user_data:
288         user_id = user_data['user_id']
289         conn = get_db_connection()
290         conn.cursor(cursor_factory=DictCursor)
```

```

bank.py > ...
284     def statements():
285         user_data = session.get('user')
286         if user_data:
287             user_id = user_data['user_id']
288             conn = get_db_connection()
289             cursor = conn.cursor()
290
291             # Fetch the account statements for the logged-in user
292             cursor.execute("""
293                 SELECT transaction_type, transaction_amount, transaction_date, description
294                 FROM account_statements
295                 WHERE user_id = %s
296                 ORDER BY transaction_date DESC
297             """, (user_id,))
298
299             transactions = cursor.fetchall()
300
301             cursor.close()
302             conn.close()
303
304             return render_template("statement.html", transactions=transactions)
305         else:
306             return redirect(url_for('login'))
307
308
309     @app.route("/customer-support")
310     def customer_support():
311         return render_template("customer_support.html")
312
313
314     @app.route("/services")
315     def services():
316         return render_template("services.html")
317
318     @app.route("/contact")
319     def contact():
320         return render_template("contact.html")
321
322
323     if __name__ == "__main__":
324         app.run(debug=True)

```

## Milestone 3: AWS RDS Setup and MySQL Integration

### Activity 1: Create Amazon RDS Instance

1. Access RDS Console from the AWS Management Console.
2. Create a new RDS instance:
  - o Choose MySQL as the engine type.
  - o Select an appropriate instance size (e.g., db.t3.micro for testing).

- Configure storage, network settings, and security groups.
  - Set up the master username and password.
  - Make sure to allow connections from your local IP for development and testing.
3. Note down the endpoint, port, database name, username, and password.

RDS > Create database

## Create database Info

**Choose a database creation method**

Standard create  
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create  
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

**Engine type Info**

Aurora (MySQL Compatible) 

Aurora (PostgreSQL Compatible) 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

IBM Db2 

**Edition**

MySQL Community

## ▼ Credentials Settings

### Master username [Info](#)

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

1 to 16 alphanumeric characters. The first character must be a letter.

### Credentials management

You can use AWS Secrets Manager or manage your master user credentials.

Managed in AWS Secrets Manager - *most secure*  
RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.

Self managed  
Create your own password or have RDS create a password that you manage.

### Auto generate password

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

### Master password [Info](#)

Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @

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

## 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](#)

## Availability and durability

### Deployment options [Info](#)

The deployment options below are limited to those supported by the engine you selected above.

#### Multi-AZ DB Cluster

Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.

#### Multi-AZ DB instance (not supported for Multi-AZ DB cluster snapshot)

Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.

#### Single DB instance (not supported for Multi-AZ DB cluster snapshot)

Creates a single DB instance with no standby DB instances.

▼ Advanced settings

Baseline IOPS of 3,000 IOPS and storage throughput of 125 MiBps are included for allocated storage less than 400 GiB.

i To provision additional IOPS and throughput, increase the allocated storage to 400 GiB or greater.

Provisioned IOPS [Info](#)

3000

IOPS

Storage throughput [Info](#)

125

MiBps

▼ Storage autoscaling

Storage autoscaling [Info](#)

Provides dynamic scaling support for your database's storage based on your application's needs.

Enable storage autoscaling  
Enabling this feature will allow the storage to increase after the specified threshold is exceeded.

## Activity 2: Connect to RDS using MySQL Client

1. Install MySQL client on your local machine if not already done.
2. Use the following command to connect to your RDS instance:

Copy

```
mysql -h <your-rds-endpoint> -P 3306 -u <your-username> -p
```

3. Enter your RDS master password when prompted.

Connection Name: Bank

Connection   Remote Management   System Profile

Connection Method: Standard (TCP/IP) ▾ Method to use to connect to the RDBMS

Parameters   SSL   Advanced

Hostname:	bank.crmssgcockvo.ap-south	Port:	3306	Name or IP address of the server host - and TCP/IP port.
Username:	admin			Name of the user to connect with.
Password:	Store in Vault ...	Clear	The user's password. Will be requested later if it's not set.	
Default Schema:	The schema to use as default schema. Leave blank to select it later.			

## MySQL Workbench



Successfully made the MySQL connection

Information related to this connection:

Host:

bank.crqmssgockvo.ap-south-1.rds.amazonaws.com

Port: 3306

User: admin

SSL: enabled with TLS\_AES\_128\_GCM\_SHA256

A successful MySQL connection was made with  
the parameters defined for this connection.

OK

## Milestone 5: EC2 Deployment

### Activity 1: Launch EC2 Instance

1. Access EC2 Console from the AWS Management Console.
2. Launch a new EC2 instance:
  - Choose an Amazon Linux 2 AMI.
  - Select an appropriate instance type (e.g., t2.micro for testing).
  - Configure instance details, including VPC and subnet.
  - Add storage as needed.
  - Configure security group to allow SSH (port 22) and HTTP/HTTPS (ports 80/443).
3. Create or select an existing key pair for SSH access.

## Launch an instance Info

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.

### Name and tags Info

Name

[Add additional tags](#)

### ▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

Recents
Quick Start



Amazon Linux



macOS



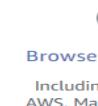
Ubuntu



Windows



Red Hat



SU: >

 [Browse more AMIs](#)  
Including AMIs from AWS, Marketplace and the Community

**Amazon Machine Image (AMI)**

**Amazon Linux 2023 AMI**  
ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)  
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

**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.

<b>Architecture</b>	<b>Boot mode</b>	<b>AMI ID</b>
64-bit (x86)	uefi-preferred	ami-02b49a24cfb95941c

Verified provider

### ▼ Instance type Info | Get advice

**Instance type**

**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

Free tier eligible

All generations
[Compare instance types](#)

## ▼ 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](#)

## ▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-00053d268636862c3  
172.31.0.0/16

(default)



[Create new VPC](#)

Subnet [Info](#)

No preference



[Create new subnet](#)

Auto-assign public IP [Info](#)

Enable



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.

[Create security group](#)

[Select existing security group](#)

Security group name - *required*

launch-wizard-2

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](#)

launch-wizard-2 created 2024-08-29T19:00:44.460Z

### Inbound Security Group Rules

#### ▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

[Remove](#)

Type [Info](#)

ssh

Protocol [Info](#)

TCP

Port range [Info](#)

22

Source type [Info](#)

Anywhere

Source [Info](#)

Add CIDR, prefix list or security group

Description - optional [Info](#)

e.g. SSH for admin desktop

0.0.0.0/0



Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.



[Add security group rule](#)

## **Activity 2: Configure EC2 Instance**

1. Connect to your EC2 instance using SSH.
2. Update the system and install necessary software:  
`sudo yum update -y`  
`sudo yum install python3 python3-pip mysql -y`
3. Install required Python packages:

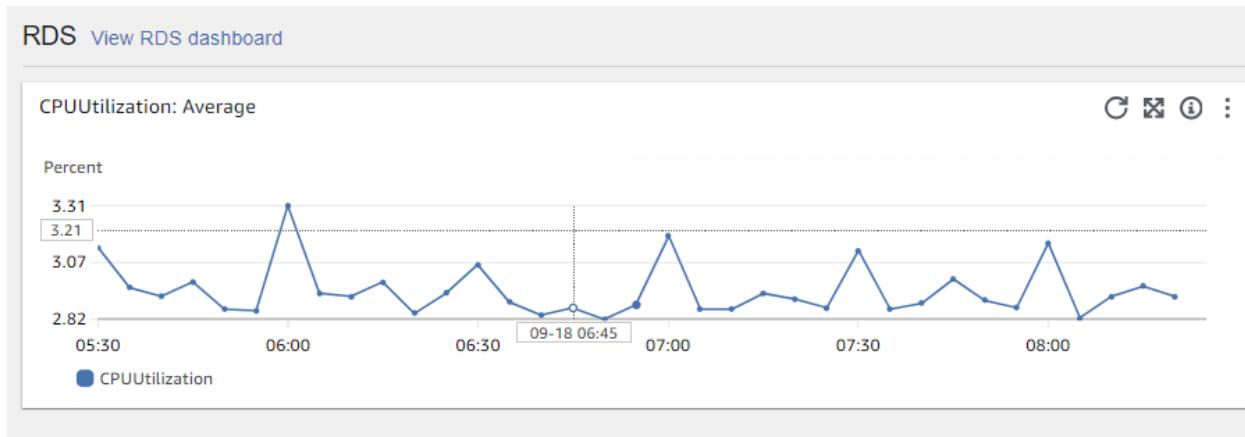
## **Activity 3: Deploy Flask Application to EC2**

1. Transfer your application code to the EC2 instance.
2. Set up any necessary environment variables, including database connection strings.
3. Configure the web server to serve your application.
4. Start your application and ensure it's accessible via the EC2 instance's public IP or domain.
5. Run the below commands on ec2 terminal
6. `sudo yum update -y`
7. `sudo yum install python3 -y`
8. `sudo pip3 install virtualenv`
9. `python3 -m venv venv`
10. `source venv/bin/activate`
11. `pip install flask`
12. `git clone https://github.com/your-repo/your-flask-app.git`
13. `cd your-flask-app`
14. `python3 app.py`

## **Milestone 6: Monitoring and Management**

### **Activity 1: Set Up CloudWatch Monitoring**

1. Access the CloudWatch console from the AWS Management Console.
2. Create a new dashboard for your Easybank application.
3. Add widgets to monitor key metrics:
  - EC2 instance CPU utilization, network traffic, and status checks
  - RDS instance CPU utilization, free storage space, and database connections
  - Application-specific metrics (e.g., number of transactions, active users)
4. Set up CloudWatch Alarms for critical thresholds (e.g., high CPU usage, low free storage).



### Activity 2: Configure CloudWatch Logs

1. Install and configure the CloudWatch Logs agent on your EC2 instance.
2. Set up log groups for your application logs, EC2 system logs, and Nginx logs (if applicable).
3. Create log filters to extract and analyze important log events.

## Milestone 7: Testing, Optimization, and Maintenance

### Activity 1: Conduct Thorough Testing

1. Perform functionality testing of all banking features.
2. Conduct security testing, including penetration testing if possible.
3. Perform load testing to ensure the application can handle expected traffic.

### Activity 2: Optimize Performance

1. Analyze and optimize database queries.
2. Implement caching mechanisms where appropriate (e.g., Flask-Caching).
3. Optimize front-end assets (minify CSS/JS, optimize images).

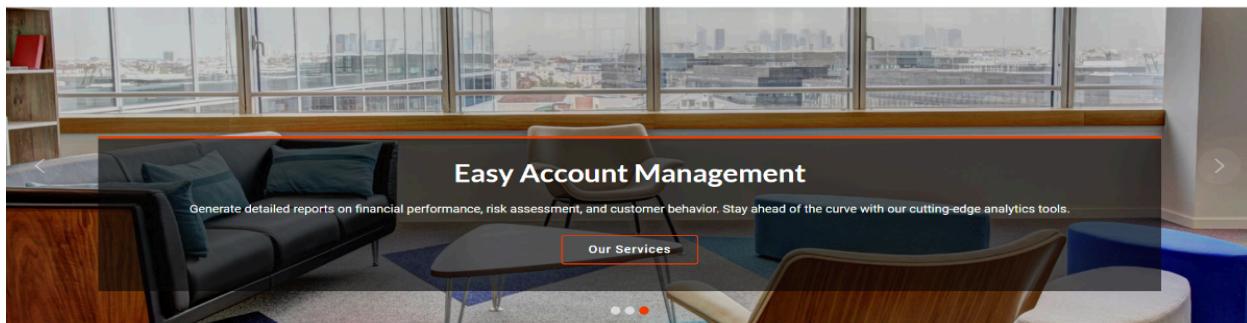
### Activity 3: Implement Backup and Disaster Recovery

1. Configure automated RDS snapshots for database backups.
2. Set up a disaster recovery plan, including steps for failover to a different region if necessary.

### Activity 4: Continuous Improvement

1. Regularly update EC2 instances and RDS with the latest security patches.
2. Monitor CloudWatch metrics and logs to identify areas for improvement.
3. Gather user feedback and iterate on the application features and user experience.
4. Continuously optimize AWS resource usage for cost-effectiveness.

**INDEX.HTML:**

**Financial Performance Analytics**

Gain deep insights into your institution's financial health with comprehensive performance metrics and trend analysis.

Call To Action

**Online Account Opening**

Open a new bank account from the comfort of your home. It's quick, easy, and secure.

**24/7 Account Access**

Log into your account anytime, anywhere to manage your finances.

**View statements**

Access your account statements online. View, download, or print them as needed.

**Analyse your spends**

Analyse your spends and give you insights on your monthly spend data.

**Easy Fund Transfer**

Transfer to anyone who has a valid account number within our bank with no extra fee.

**Easy Support**

Get Easy customer support anytime and from anywhere and get your issues resolved within no time.

**Partnered With Us****Flattern****MyBank**

Your trusted partner for online banking services.

A108 Adam Street  
New York, NY 535022Phone: +91 554466 554466  
Email: info@EasyBank.com**Useful Links**

- [Home](#)
- [About us](#)
- [Services](#)
- [Terms of service](#)
- [Privacy policy](#)



## SERVICE.HTML:

EASYBANK

HOME CONTACT

**Services**

**Personal Banking**  
Our personal banking services are designed to make your life easier. From checking and savings accounts to credit cards and loans, we've got you covered.

**Business Banking**  
Our business banking services are tailored to meet the unique needs of your business. From business checking and savings accounts to loans and credit lines, we're here to help you succeed.

**Investment Services**  
Our investment services are designed to help you achieve your long-term financial goals. From retirement accounts to investment portfolios, our experts are here to guide you.

**Online Banking**  
Our online banking services allow you to manage your accounts from anywhere, at any time. Pay bills, transfer funds, and check your balances with ease.

**Mobile Banking**  
Our mobile banking app allows you to bank on the go. Deposit checks, transfer funds, and check your balances from your mobile device.

**Customer Support**  
Our customer support team is here to help you with any questions or concerns you may have. Contact us by phone, email, or in person.

**Flattern**

A108 Adam Street  
New York, NY 535022  
Phone: +1 5589 55488 55  
Email: info@example.com

© Copyright MyWebsite. All Rights Reserved  
Designed by BootstrapMade

## LOGIN.HTML:

EASYBANK

HOME SERVICES REGISTER LOGIN CONTACT

**Login**  
Enter your credentials to access your account.

Email  Password

**Contact Us**  
A108 Adam Street  
New York, NY 535022  
United States  
Phone: +1 5589 55488 55  
Email: info@example.com

© Copyright EasyBank. All Rights Reserved

## REGISTER.HTML:

EASYBANK

HOME SERVICES REGISTER LOGIN CONTACT



### Register Now

Fill out the form below to create an account.

Full Name

Email

Password

Phone

Address

Aadhar Number

PAN Card

### Contact Us

A108 Adam Street  
New York, NY 535022  
United States

Phone: +1 5589 55488 55  
Email: info@example.com

↑

## BALANCE.HTML:

# EASYBANK

- [Home](#)
- [View Account Statement](#)
- [Transfer Funds](#)
- [Customer Support](#)

### Check Bank Balance

your current bank balance is:**1100.00**

© 2024 EASYBANK. All rights reserved.

## DASHBOARD.HTML:

EASYBANK

HOME SERVICES DASHBOARD LOGOUT CONTACT

Welcome to Easy dashboard

Your trusted online banking partner. You can now access your account, view statements, and manage your finances with ease.

Deposit Funds

Check Bank Balance  
Check your bank account from the comfort of your home. It's quick, easy, and secure.

Fund Transfer  
Send you funds to anyone from anywhere and at anytime with no time.

View statements  
Access your account statements online. View, download, or print them as needed.

Analyse your spends  
Analyse your spends and give you insights on your monthly spend data.

Apply Credit Card  
Apply to your fav credit card and enjoy the deals with it.

Easy Support  
Get Easy customer support anytime and from anywhere and get your issues resolved within no time.

## BANKING DEALS

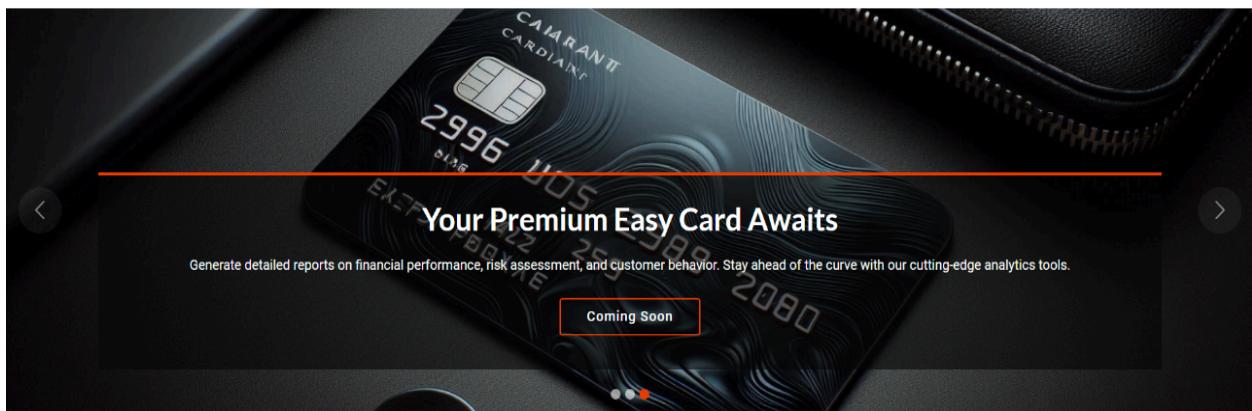

**Flattern**  
**MyBank**  
Your trusted partner for online banking services.

A108 Adam Street  
New York, NY 535022  
Phone: +91 554466 554466  
Email: info@EasyBank.com

**Useful Links**

- [Home](#)
- [About us](#)
- [Services](#)
- [Terms of service](#)
- [Privacy policy](#)

© Copyright  
© 2024 MyBank. All Rights Reserved  
Designed by [BootstrapMade](#)

**STATEMENTS.HTML:**

## Account Statements

Transaction Type	Amount	Date	Description
Credit	100.00	2024-09-18 11:34:39	None
Credit	1000.00	2024-09-18 11:32:27	None

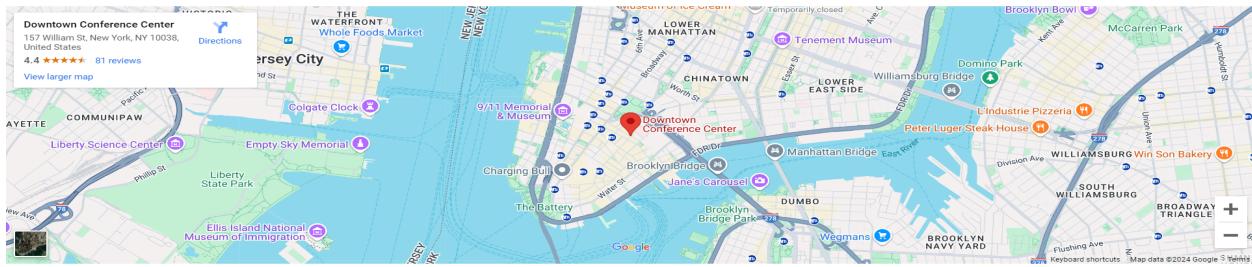
[Back to Dashboard](#)

© 2024 Your Bank. All rights reserved.

## CONTACT.HTML:

### FLATTERN

HOME SERVICES REGISTER LOG-IN CONTACT



**Downtown Conference Center**  
157 William St, New York, NY 10038, United States  
4.4 ★★★★☆ 81 reviews  
[View larger map](#)

**Get in touch**  
Get in touch with our top executives working tirelessly to solve your issues

**Location:**  
A108 Adam Street, New York, NY 535022

**Email:**  
EasyBank@info.com

**Call:**  
+1 5589 55488 55

Your Name  Your Email   
Subject   
Message

**Send Message**

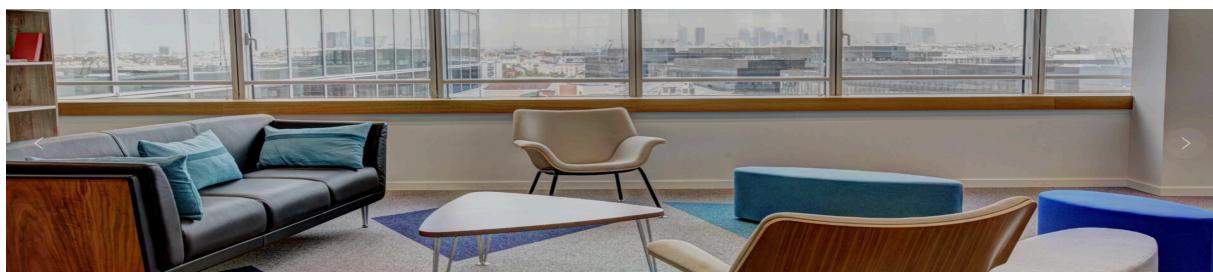
### Support

A108 Adam Street  
New York, NY 535022  
Phone: +1 5589 55488 55  
Email: info@example.com

© Copyright MyWebsite. All Rights Reserved  
Designed by [BootstrapMade](#)

X F I N ↑

## DEPOSIT.HTML:



**Deposit Funds**

Deposit Amount:

Account Type:

**Deposit**

## CONFIRM.HTML:

[EASYBANK](#)

[HOME](#) [SERVICES](#) [REGISTER](#) [LOGIN](#) [CONTACT](#)



## Account Created Successfully!

Welcome, akhil!

Your account has been created successfully. You can now log in to access our services.

[Log in](#)

### Contact Us

A108 Adam Street  
New York, NY 535022  
United States

Phone: +1 5589 55488 55  
Email: [info@example.com](mailto:info@example.com)

### Useful Links

- > [Home](#)
- > [About us](#)
- > [Services](#)
- > [Terms of service](#)
- > [Privacy policy](#)

↑

## TRANSFER.HTML:

### Transfer Funds

Securely transfer funds between accounts with ease.

Funds transferred successfully!

Registration successful! Please log in.

Invalid login credentials!

Recipient's User ID:

Transfer Amount:

[Transfer](#)

© 2024 Your Bank. All rights reserved.

## **CONCLUSION:**

The Easybank project serves as a robust foundation for a cloud-based banking system. By leveraging AWS services, we've created a scalable, secure, and efficient platform that can adapt to the evolving needs of modern banking. This project not only demonstrates technical proficiency in cloud technologies but also highlights the complexities and considerations involved in developing financial applications in the cloud.

As the financial sector continues to embrace digital transformation, projects like Easybank pave the way for more innovative, secure, and user-friendly banking solutions. The lessons learned and the architecture developed here can serve as a valuable reference for future fintech projects and cloud-based applications in sensitive domains.