**BloodBridge: Optimizing Lifesaving Resources using AWS services**

**Project Description:**

"BloodBridge" is a comprehensive web-based blood bank management system designed to streamline the process of blood donation and distribution. The project leverages Amazon Web Services (AWS) for robust and scalable infrastructure, utilizing Amazon RDS for secure and efficient data storage and Amazon EC2 for reliable web hosting. The user-friendly web interface allows individuals to register and log in to their personal accounts, creating a seamless experience for both donors and recipients. Once logged in, users are presented with a dashboard that serves as a central hub for all blood-related activities. The dashboard prominently features current blood requests, allowing users to view real-time needs in their community. Additionally, registered users can easily submit their own blood requests, specifying blood type, quantity, and urgency. This system not only facilitates quick responses to critical blood needs but also fosters a sense of community engagement in the life-saving act of blood donation. By combining modern cloud technology with an intuitive user interface, "BloodBridge" aims to bridge the gap between blood donors and those in need, ultimately saving lives and improving healthcare outcomes.

**Scenario 1:** Emergency Blood Request:

Sarah, a hospital administrator, logs into LifeLink during a critical situation. A patient needs a rare blood type urgently. Using her dashboard, Sarah quickly submits a high-priority blood request, specifying the required blood type and quantity. The system immediately notifies potential donors in the area, significantly reducing the time to find a match and potentially saving the patient's life.

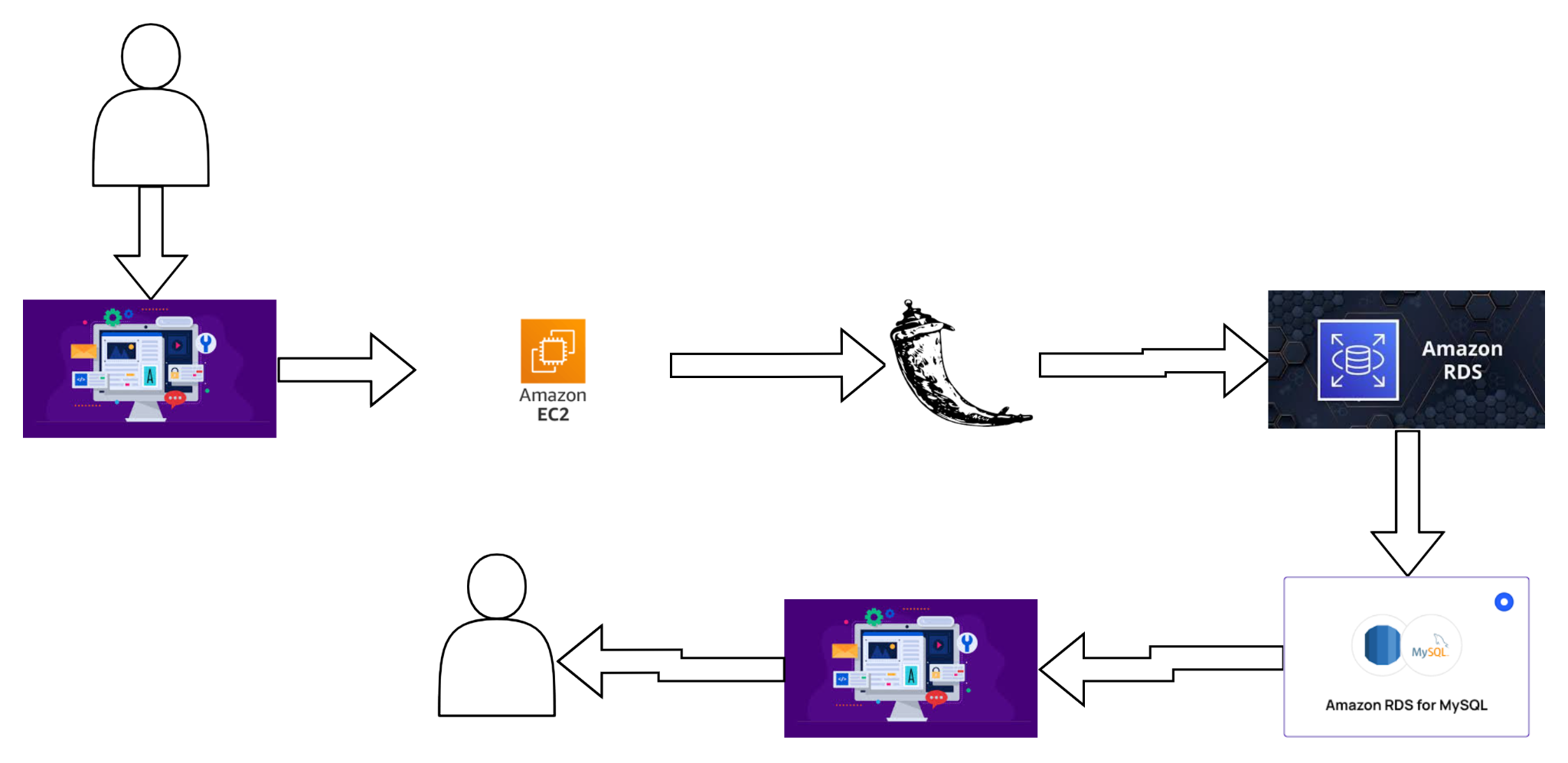
**Scenario 2**: : Regular Donor Management

John, a regular blood donor, uses LifeLink to manage his donations. After logging in, he checks his dashboard to see when he's eligible to donate again. He notices a nearby blood drive event listed in the requests section. John uses the system to schedule his next donation, helping maintain a steady supply of blood for the local hospitals.

**Scenario 3:** Blood Bank Inventory Update:

A blood bank manager, Lisa, uses LifeLink to update the current blood inventory. She logs into her specialized account and accesses a feature to input the latest stock levels for each blood type. The system automatically updates the dashboard for all users, reflecting the current needs. This real-time update helps prioritize requests for blood types that are running low, ensuring efficient distribution of this vital resource.

**Architecture:**



**Prior Knowledge:**

### 1. AWS Account Setup: [https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk\_M4FfVM-D](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)h

### 2.Web Application Stack : [FLask](https://www.youtube.com/results?search_query=flask+tutorial) || [MySQL Connector using flask](https://www.youtube.com/results?search_query=mysql+connector+for+flask) || [HTML/JS/CSS](https://www.youtube.com/results?search_query=html+js+css+tutorial+)

### 3. AWS EC2 Instance: <https://www.youtube.com/results?search_query=aws+ec2+oneshot>

### 4. RDS Database: <https://www.youtube.com/results?search_query=rds+oneshot>

### 5. MySQL: <https://www.youtube.com/results?search_query=mysql+tutorial>

### 6. RDS connects MySQL:<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>

### 8. AWS Cost Management: <https://youtu.be/OKYJCHHSWb4?si=aY3DQl1v26CfZxXA>

**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:**

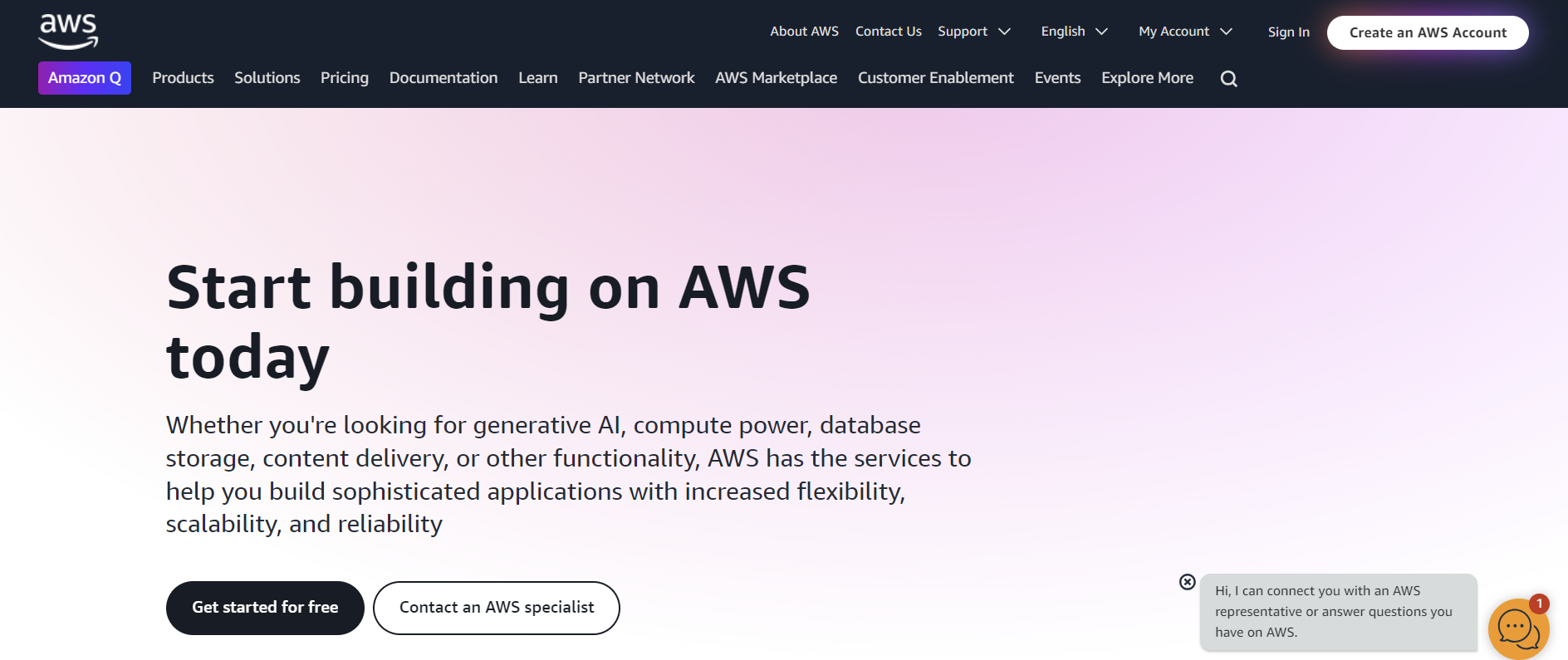
* Conduct thorough testing of all features and optimize for performance.

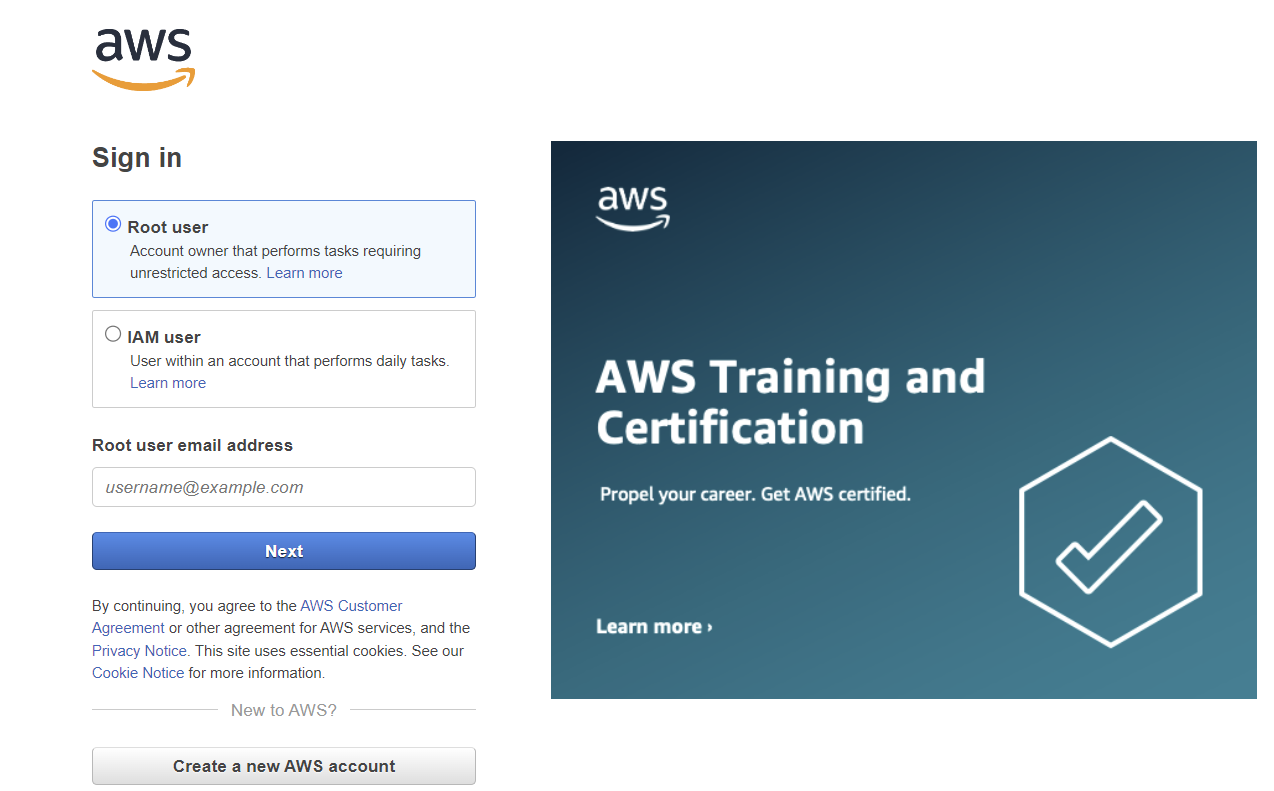
## **Milestone 1: AWS Account Creation**

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 (<https://aws.amazon.com/>).
2. Click on "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 or debit 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. Once verified, you can sign in to your new AWS account.





**Milestone 2: Set Up AWS Environment**

In this milestone, we will create and configure an EC2 instance to host the BloodBridge web application.

**Activity 1.1 Create and Configure an Amazon EC2 Instance**

1. Access EC2 Console: In the AWS Management Console, go to the EC2 service.
2. Launch Instance: Click on "Launch Instance" and follow the wizard:
   * Choose an Amazon Machine Image (AMI) suitable for your web application (e.g., Amazon Linux 2).
   * Select an instance type (e.g., t2.micro for testing).
   * Configure instance details, including network settings.
   * Add storage as needed.
   * Add tags for better resource management.
   * Configure security group to allow HTTP/HTTPS traffic.
3. Review and Launch: Review your instance configuration and launch it, selecting or creating a key pair for SSH access.

**Activity 1.2: Configure Security Groups**

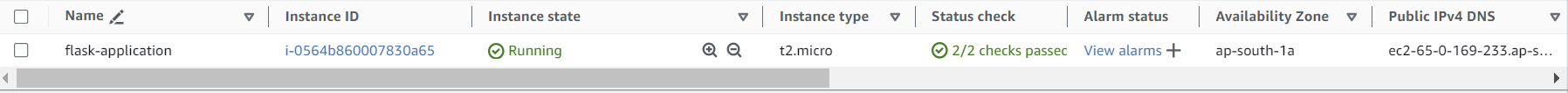
Add a Security Group:

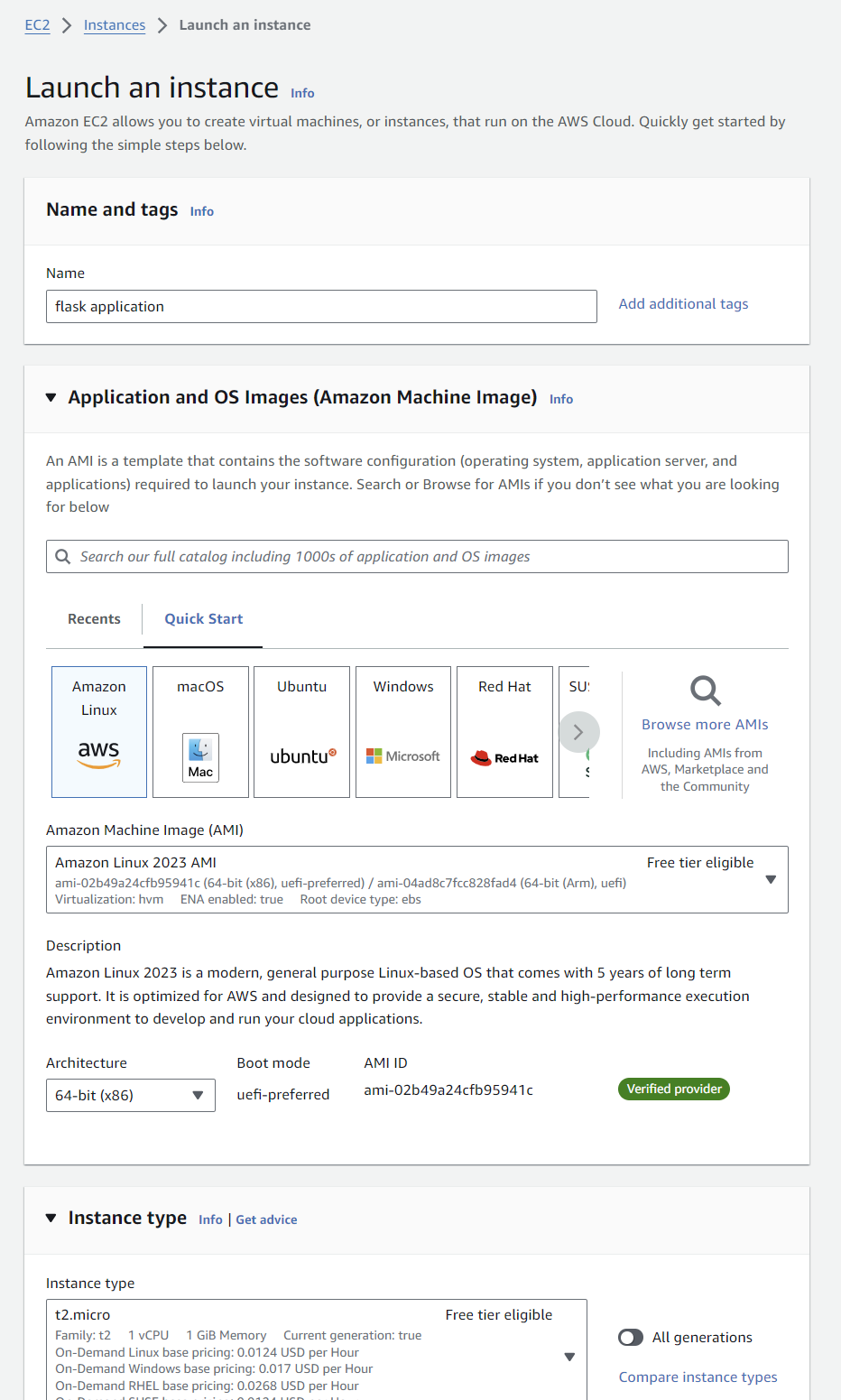
Allow SSH (port 22) from your IP for remote access.

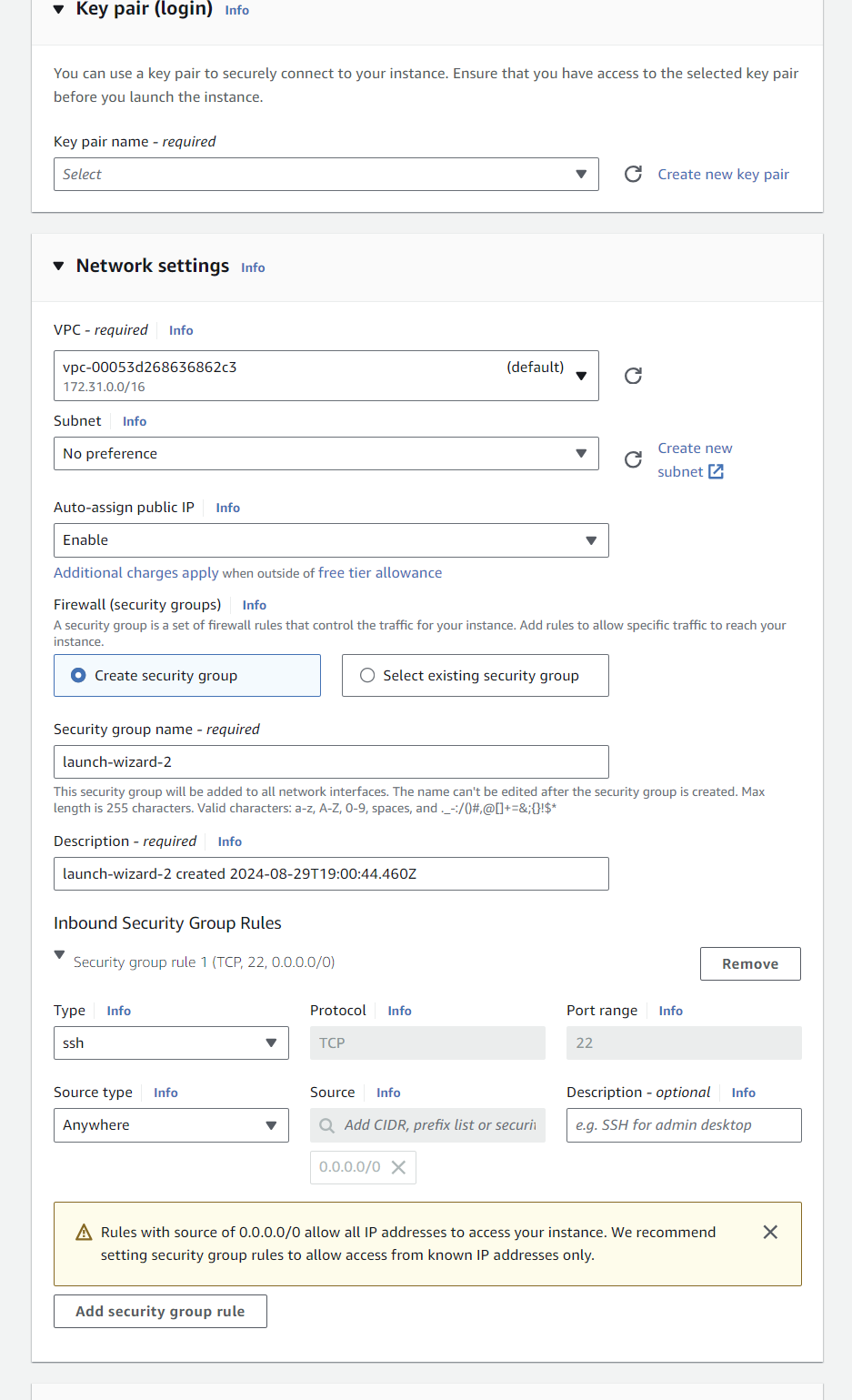
Allow HTTP (port 80) and HTTPS (port 443) to access your web application.

If you have other requirements (e.g., specific port for your Flask application), add those as well.

**Activity 1.3: Launch the Instance**







## **Milestone 3: Setting up RDS Database**

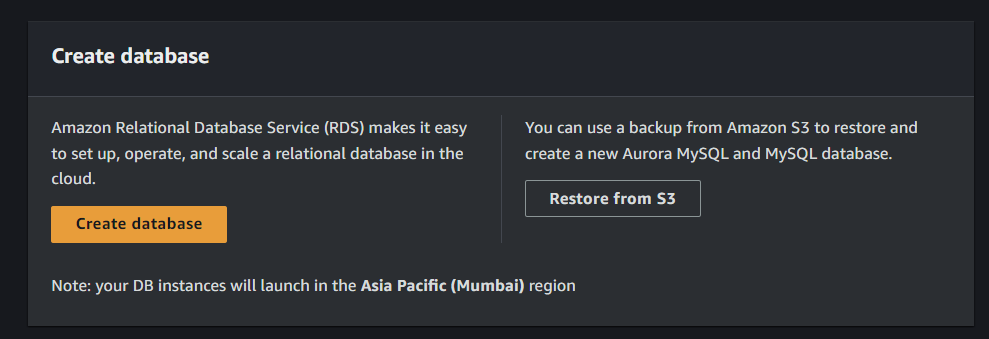
In this milestone, we will create and configure an RDS instance with MySQL to store and manage BloodBridge data.

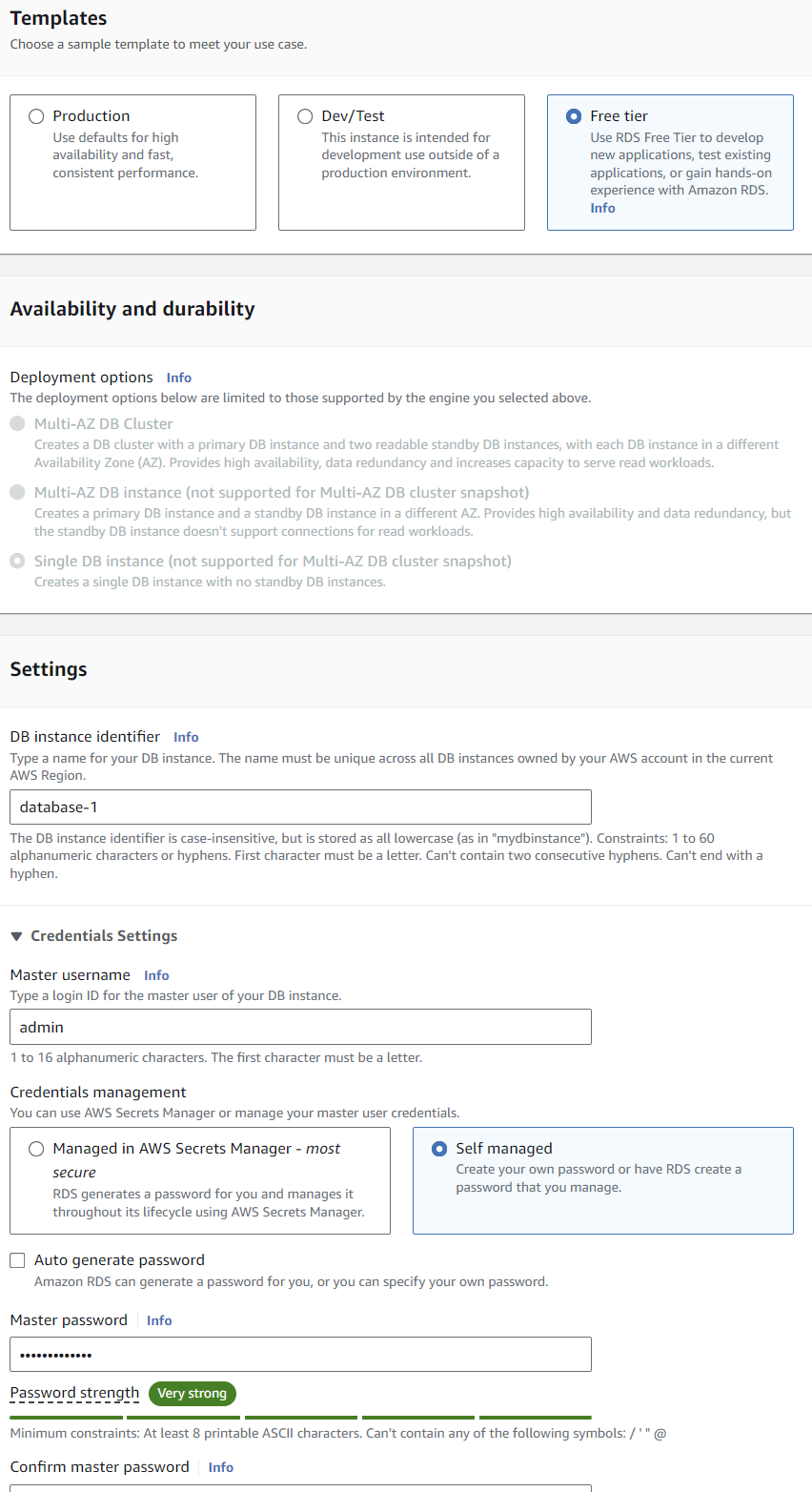
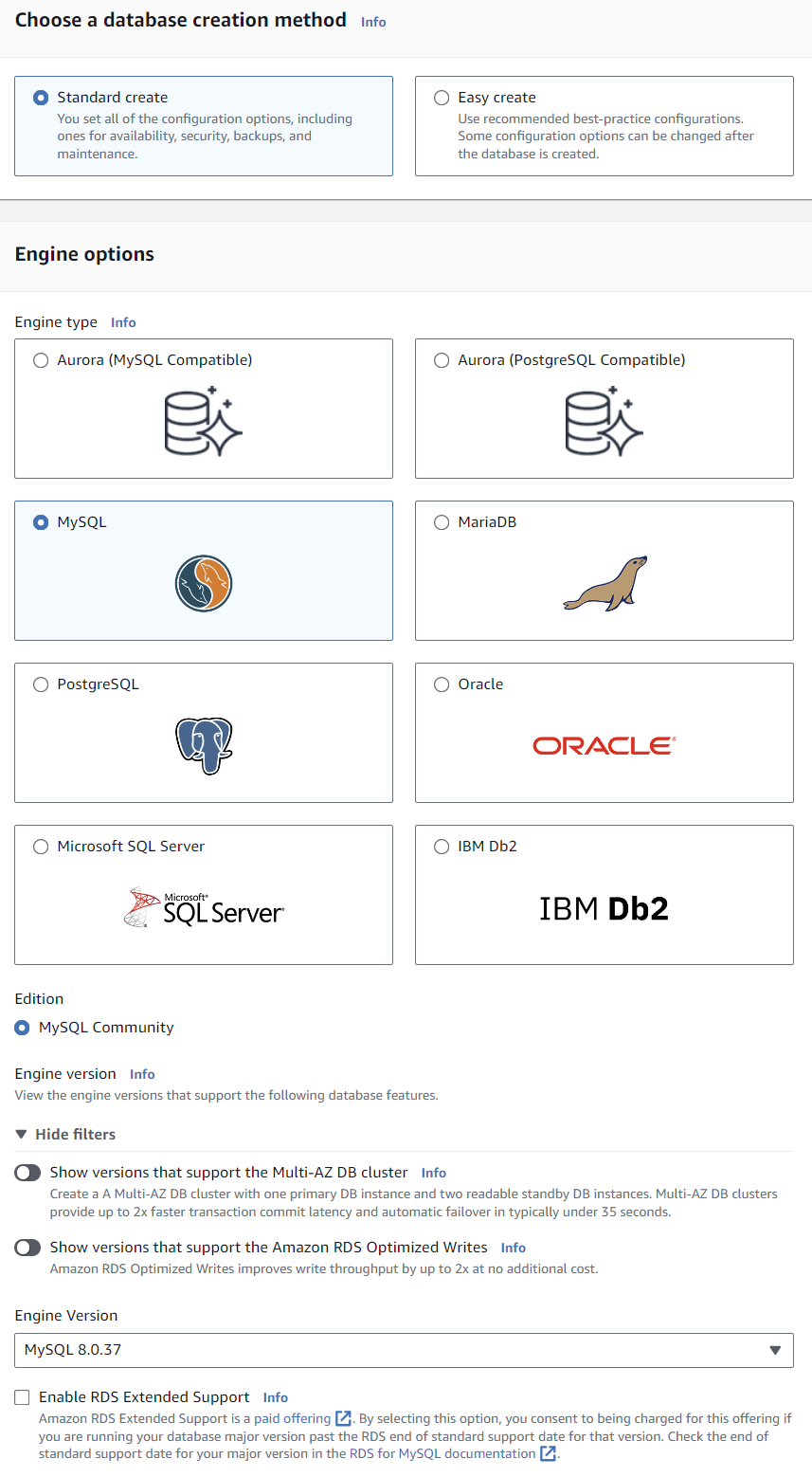
**Activity 1: Create RDS Instance** [**[RDS]**](https://console.aws.amazon.com/rds/)

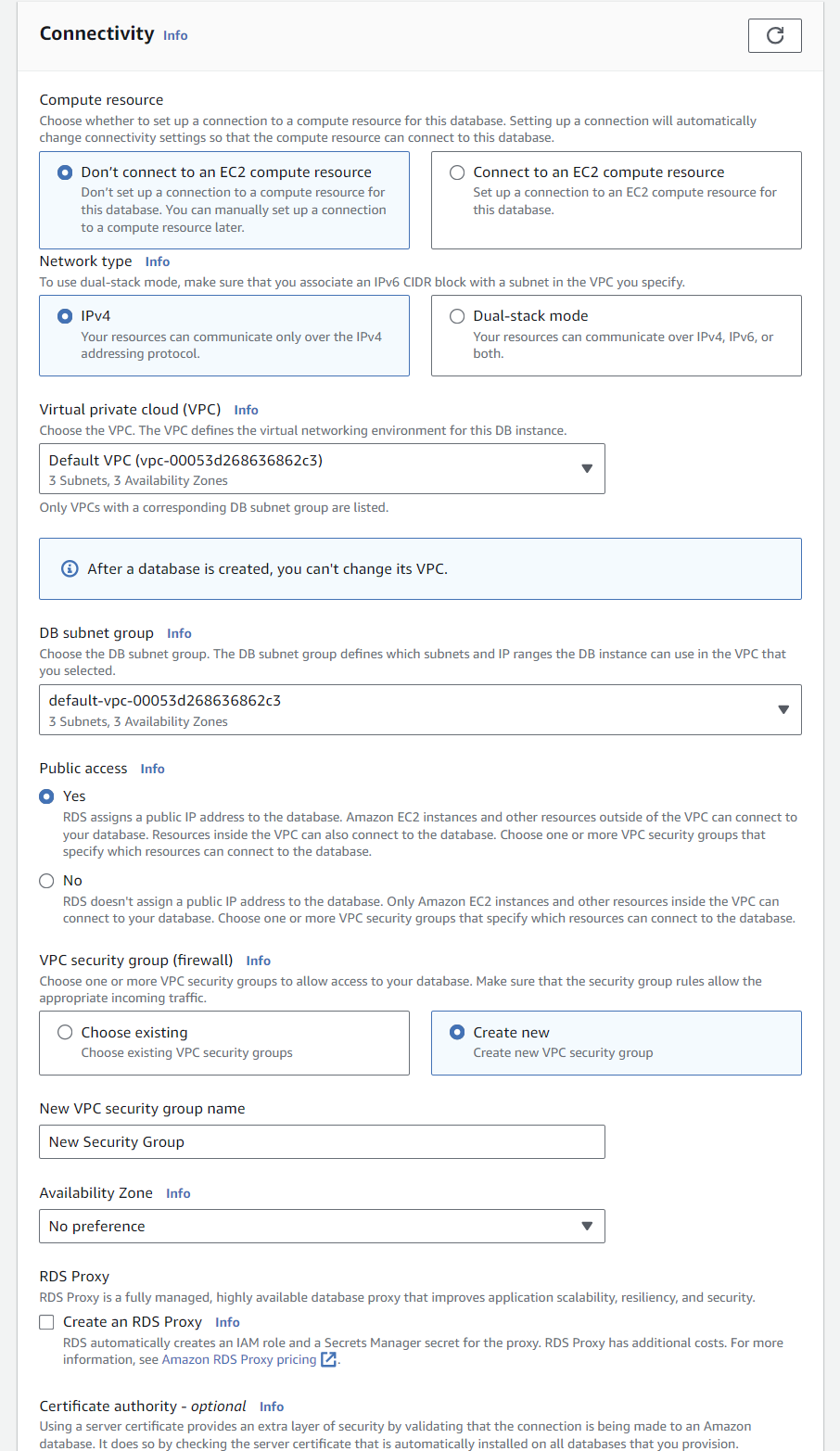
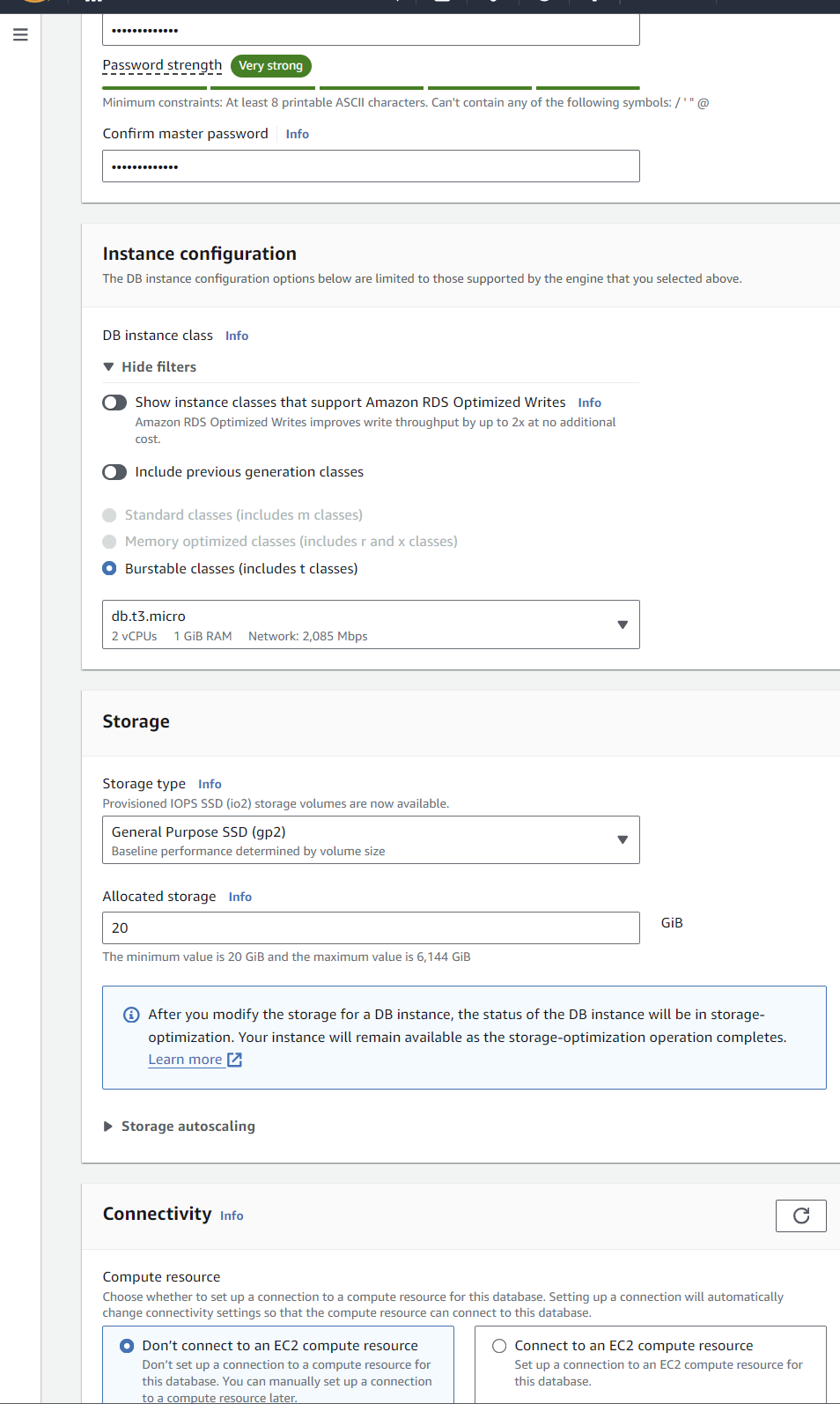
1. Access RDS Console: From the AWS Management Console, go to the RDS service.
2. Create Database: Click on "Create database" and follow the wizard:
   * Choose MySQL as the engine type.
   * Select the appropriate version and instance size.
   * Configure storage, network settings, and security groups.
   * Set up the master username and password.
   * VPC and Subnet: Ensure the RDS instance is in the same VPC as your EC2 instance.
   * Public Accessibility: Enable this option if you need direct access from outside the VPC (not recommended for production).
   * Security Group: Create or use an existing security group that allows MySQL traffic (default port 3306).
   * Set an initial database name (e.g., `bloodbank`).
3. Review and Create: Review your database configuration and create the instance.

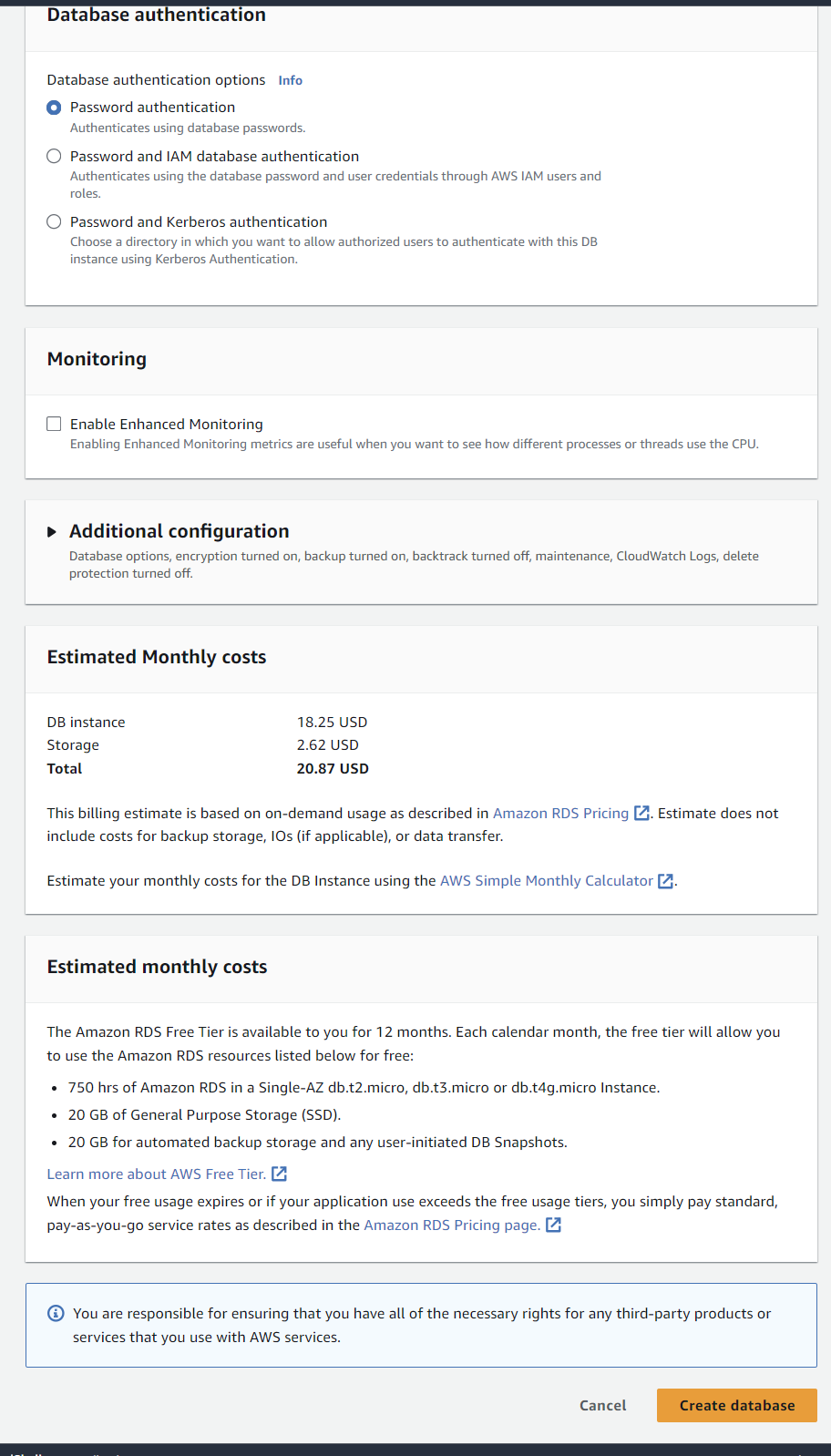
**Activity 2: Configure Security Group**

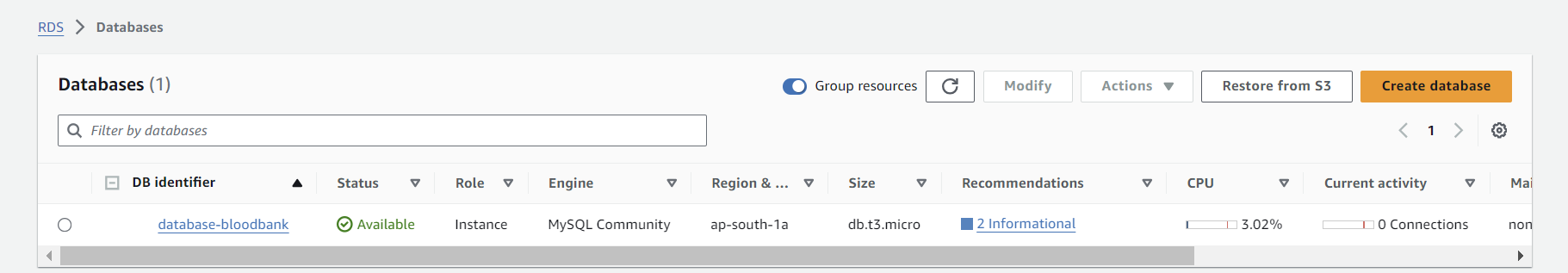
1. Once your RDS instance is created, go to its details page.
2. In the "Connectivity & security" tab, click on the VPC security group.
3. Add an inbound rule to allow MySQL/Aurora traffic (port 3306) from your IP address for now (we'll update this later to only allow traffic from the EC2 instance). And add rule to allow inbound traffic from everywhere.

****

****

****

****

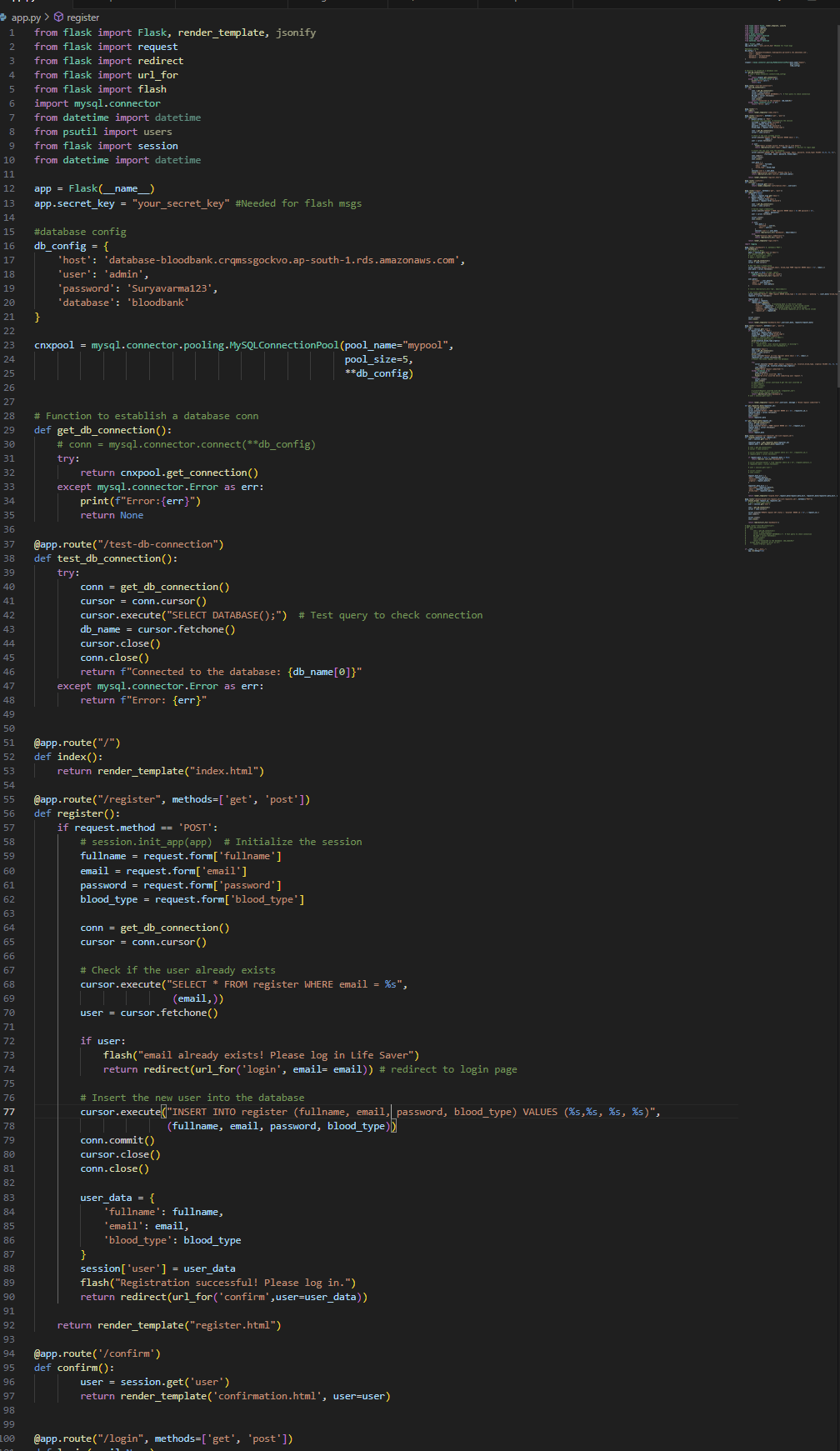
****

## **Milestone 4: Develop Web Application**

In this milestone, we will develop the BloodBridge web application with user registration, login, and dashboard features.

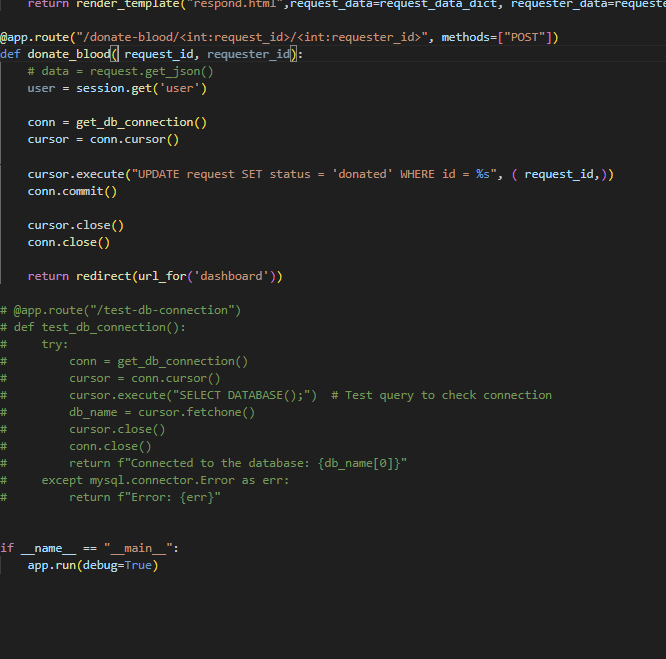
**Activity 1: Set Up Development Environment**

1. Choose your preferred backend framework (e.g., Express.js for Node.js or Flask for Python).
2. Set up the project structure and install necessary dependencies.









**Activity 2: Implement User Authentication**

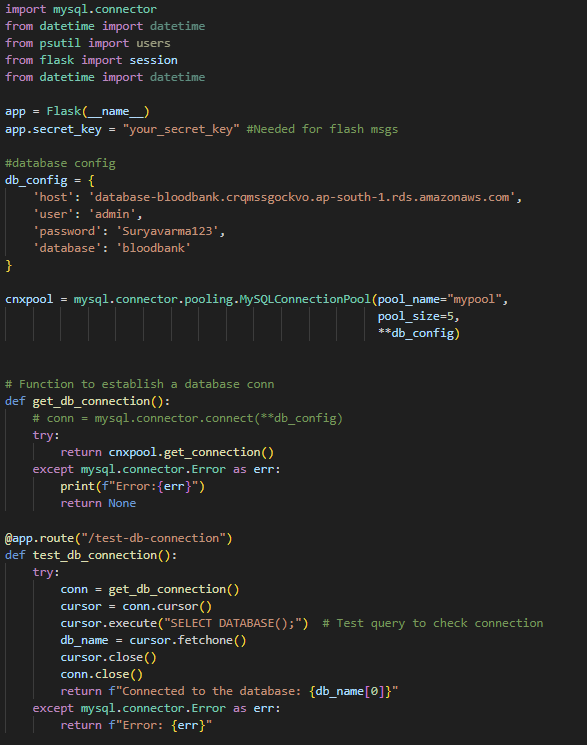
1. Create registration and login routes and forms.
2. Implement user authentication logic, including password hashing and session management.

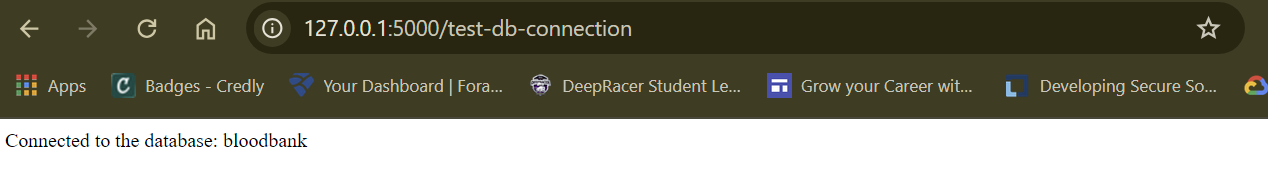
**Activity 3: Develop User Dashboard**

1. Create a dashboard interface displaying user information and blood request options.
2. Implement blood request submission and tracking functionality.

**Activity 4: Integrate with RDS Database**

1. Set up database connection using appropriate drivers (e.g., mysql2 for Node.js).
2. Implement database queries for user management and blood request handling.

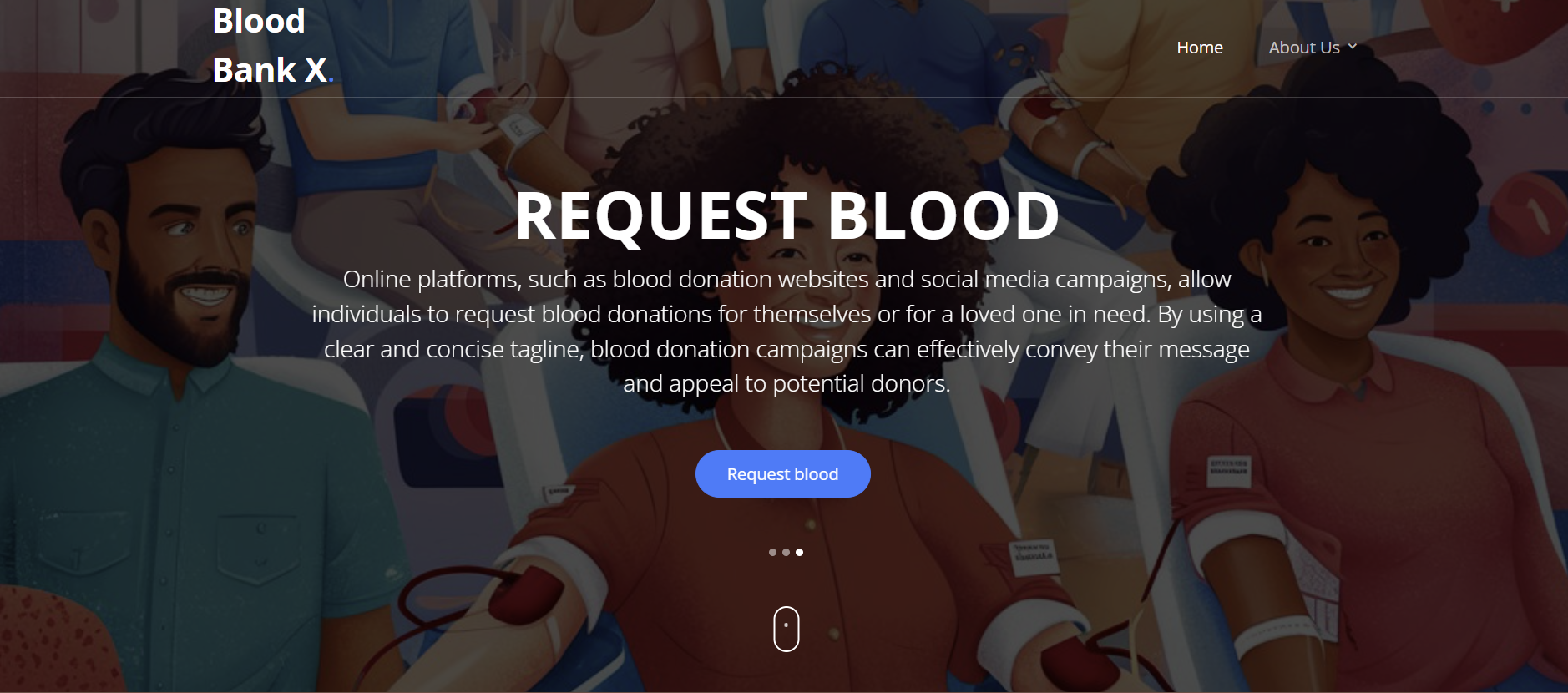


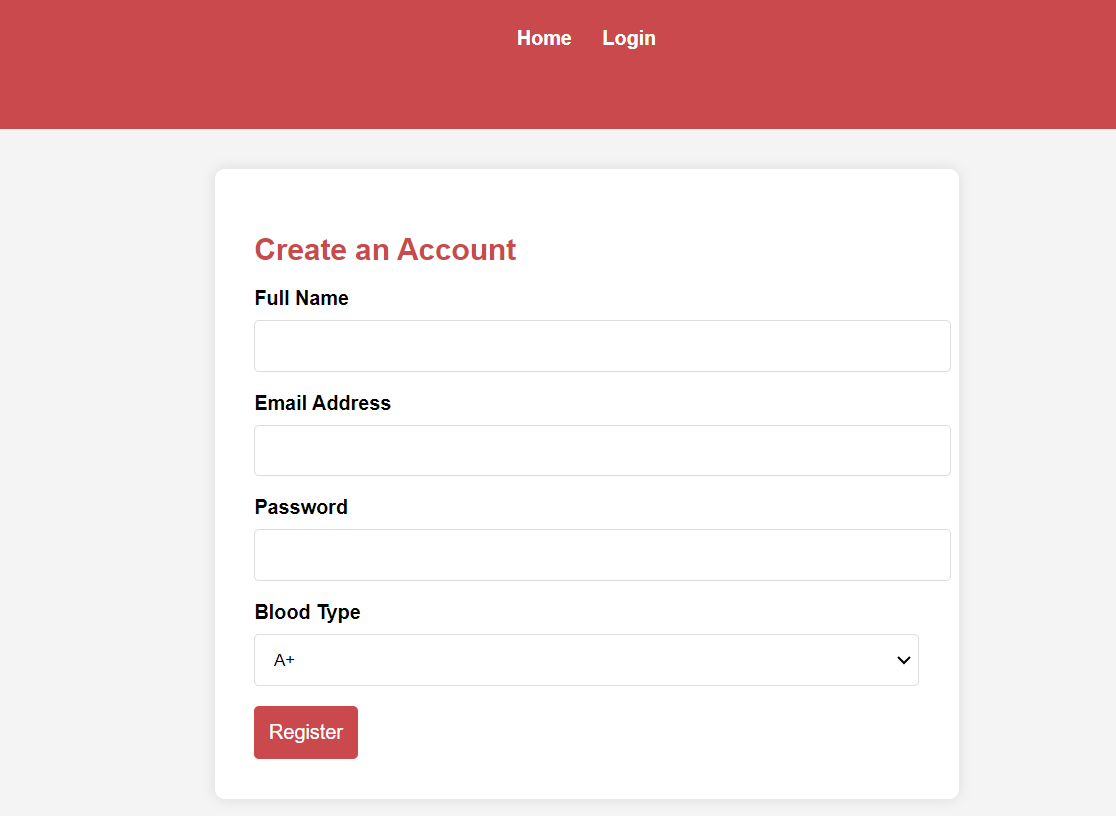


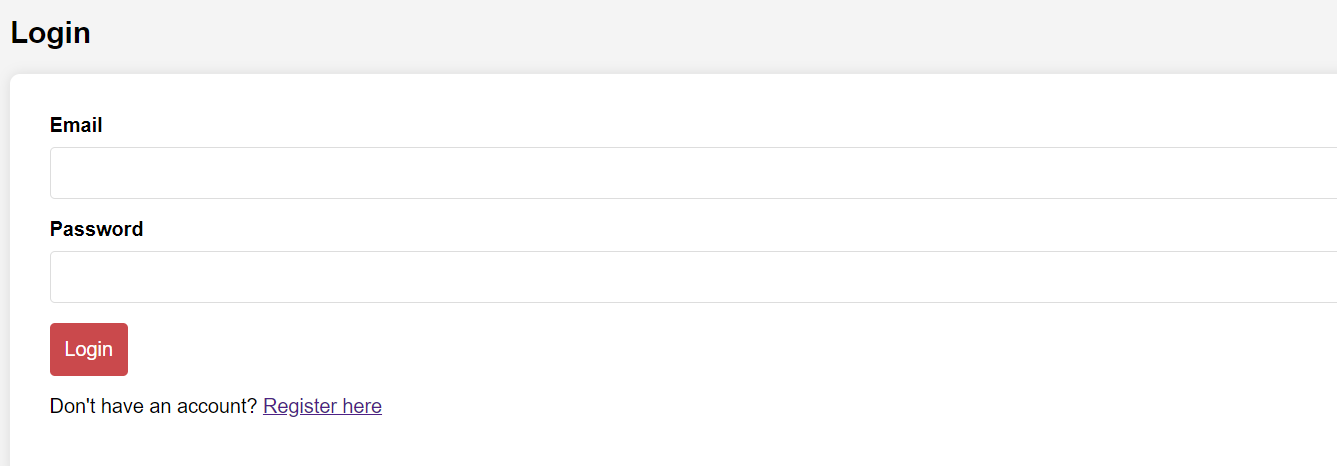
## **Milestone 4: Testing and Deployment**

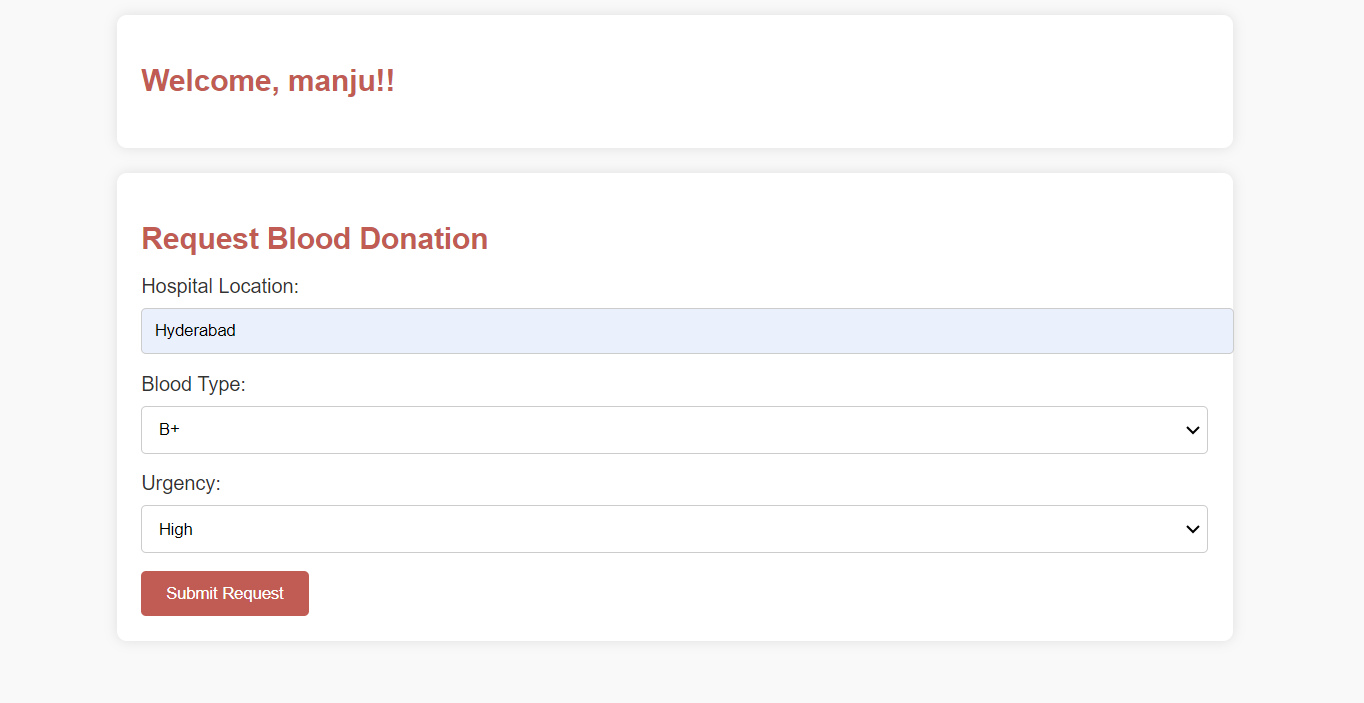
**Activity 1: Deploy to EC2**

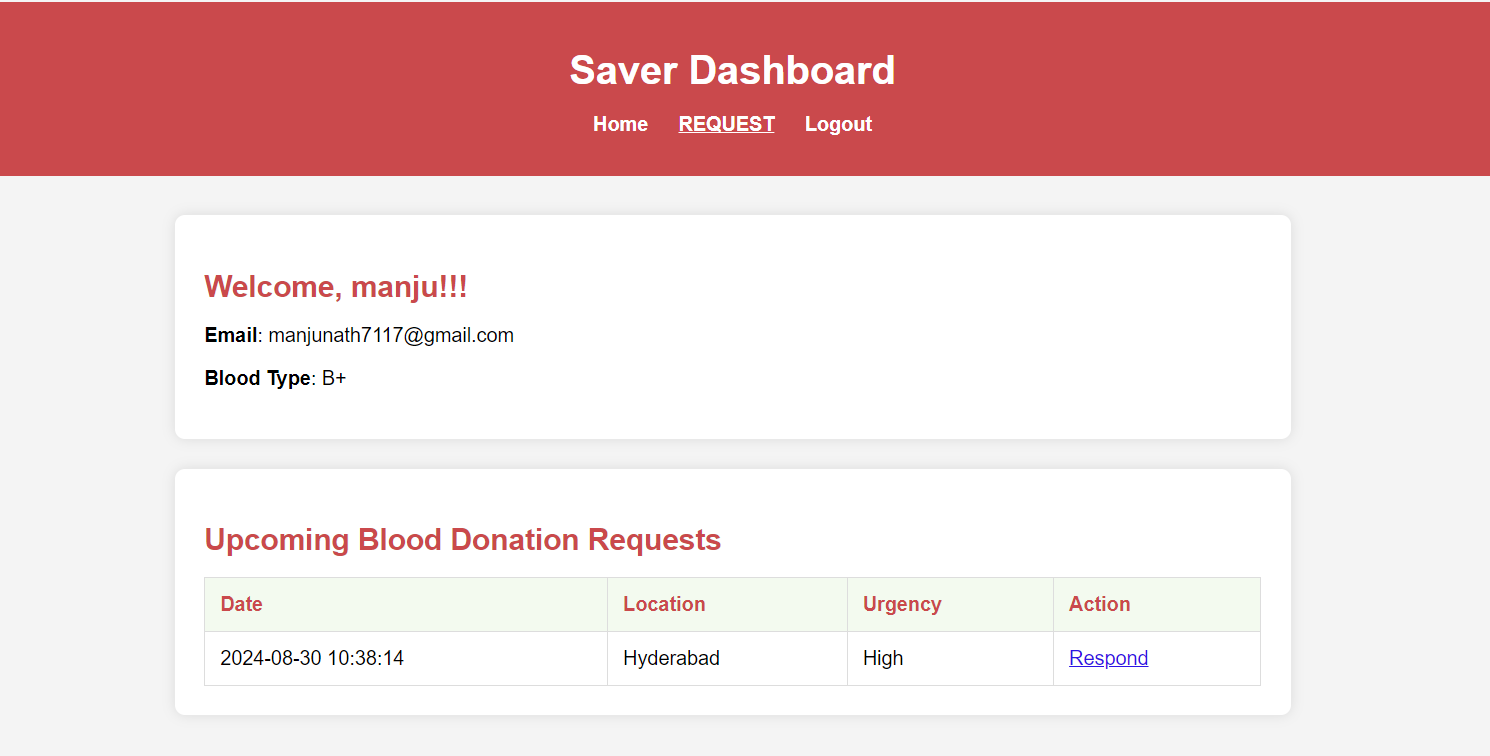
1. Transfer your application code to the EC2 instance.
2. Set up any necessary environment variables, including database connection strings.
3. Configure the web server to serve your application.
4. Start your application and ensure it's accessible via the EC2 instance's public IP or domain.
5. Run the 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

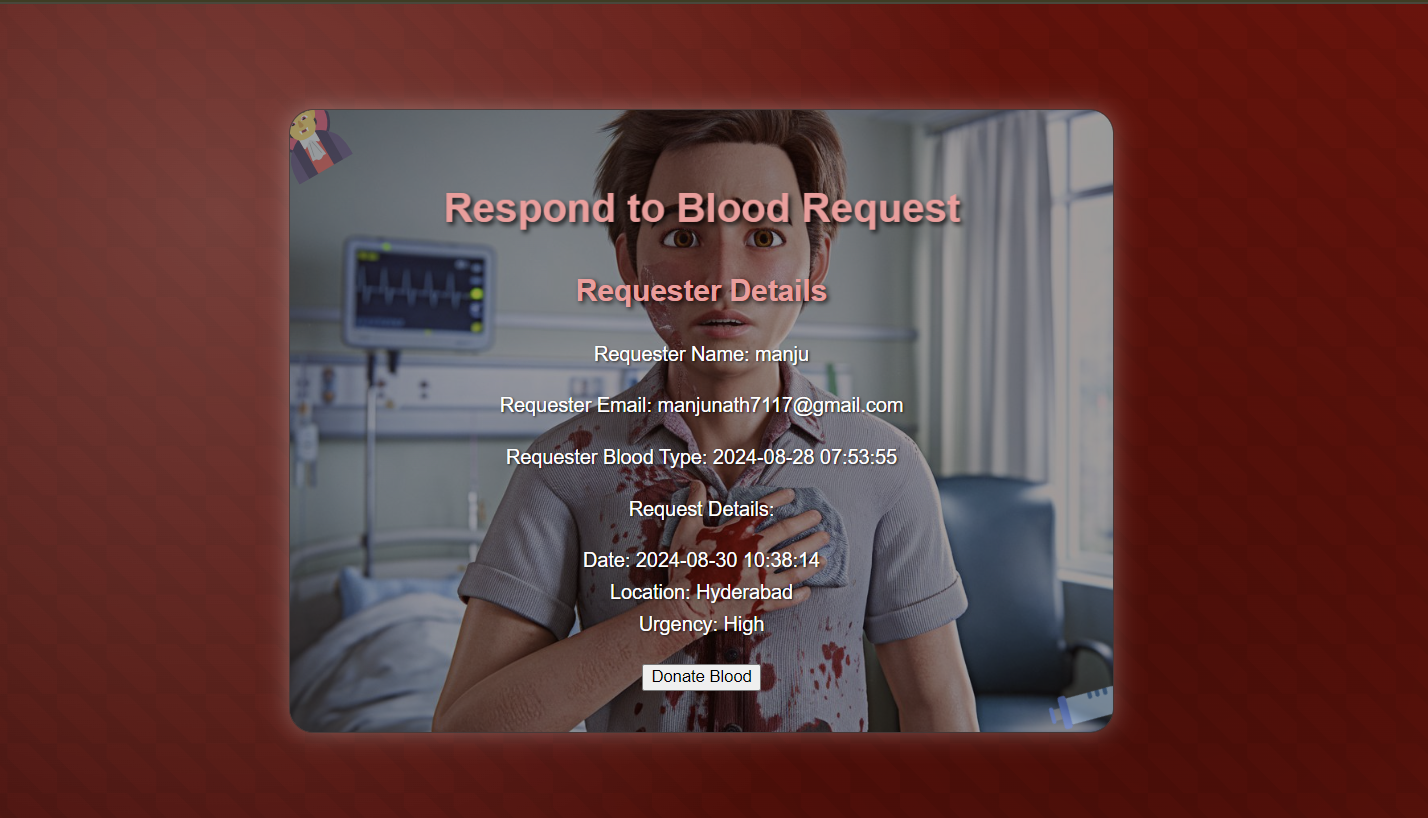












**Conclusion**:

This document provides a comprehensive guide for setting up the BloodBridge platform using AWS services, emphasizing the key steps involved in creating an RDS database, developing a Flask application, and deploying it on an EC2 instance. By adhering to these milestones and activities, you can establish a robust, scalable web application tailored to efficiently manage blood donation requests. Leveraging EC2 for web hosting and RDS for database management ensures a high-performance, reliable infrastructure that can grow with the needs of your application.