Assignment: Deploy a Two-Instance vLLM Chat Service with User Login

DA 410/510

Due Date: **11:59 PM, May 1, 2026**

Overview

In this assignment, you will deploy a small LLM-backed chat application on AWS. The project has two critical elements:

1. deploy `vLLM` on a free-tier eligible EC2 instance and run it as a model server
2. deploy a separate REST API/web interface with user account creation and password-based login before users can chat with the model

The goal is not high-performance inference. Free-tier EC2 instances are CPU-only and limited, so this assignment uses `HuggingFaceTB/SmolLM2-135M-Instruct`. Expect slow responses. The goal is to show that you can deploy an LLM model server on one machine, deploy the REST API/web interface on another machine, protect the model behind the application layer, and require account-based access before a user can send prompts.

Learning Objectives

By completing this assignment, you should be able to:

1. Launch and configure separate EC2 instances for a model server and a REST API/web interface.
2. Install and run `vLLM` as an OpenAI-compatible HTTP server.
3. Build a REST API/web application that calls the private `vLLM` server over the VPC network.
4. Implement account creation and password-based login.
5. Store users, password hashes, sessions, and chat history in a persistent datastore.
6. Restrict unauthenticated users from sending prompts to the model.
7. Document the security boundary between the public web app and the private model server.

Required Architecture

Your deployment must include the following components:

- one free-tier eligible EC2 instance for the REST API/web interface
- one separate free-tier eligible EC2 instance for the `vLLM` model server
- both instances in the same VPC so the REST API/web instance can reach the model server by private IP address
- a running `vLLM` server on the model server instance
- a REST API/web application on the separate interface instance
- `nginx` on the REST API/web instance as the public reverse proxy in front of FastAPI
- a password-based login workflow for user account creation and authentication
- persistent storage for user accounts, active sessions, and chat history

The public entry point must be `nginx` on the REST API/web interface instance. FastAPI should listen on a local or private application port behind `nginx`. The raw `vLLM` server must run on the separate model server instance and must not be directly reachable from the public internet. The REST API/web application should call `vLLM` using the model server instance's private IP address or private DNS name.

EC2 and vLLM Requirements

You must:

- launch a REST API/web interface EC2 instance
- launch a separate `vLLM` model server EC2 instance
- use free-tier eligible instance types for both EC2 instances when possible
- use an EC2 instance type for the model server that is marked free-tier eligible for your AWS account and region
- use free-tier eligible Linux AMIs
- install `vLLM` on the model server instance
- serve `HuggingFaceTB/SmolLM2-135M-Instruct` through `vLLM`'s OpenAI-compatible server
- configure `vLLM` with an API key or equivalent access control
- keep the `vLLM` service off the public internet
- show that the REST API/web application can successfully send a prompt to `vLLM` over the private network and receive a response

Password Login Requirements

Your application must support account creation and password-based login.

The workflow must satisfy the following requirements:

1. A visitor can create an account with an email address and password.
2. Your application stores a strong password hash, not the raw password.
3. A returning user can log in with the email address and password.
4. When login succeeds, your application creates an authenticated session.
5. When login fails, your application rejects the request without revealing whether the email address or password was the specific problem.
6. Users can log out and end the current session.
7. Only authenticated users can access the chat page or send prompts to the model.

Use a standard password hashing approach such as `bcrypt`, `argon2`, or a framework-provided password hashing utility. Do not store plaintext passwords.

AI Assistance Note

You are encouraged to use AI tools to help create the HTML templates, CSS, and basic visual styling for your registration, login, and chat pages. The user interface should be clear enough to test the workflow, but the main focus of this assignment is the cloud architecture, private model-server integration, authentication, sessions, security groups, and deployment.

If you use AI-generated frontend code, you are still responsible for understanding how it fits into your FastAPI application and for verifying that authentication and authorization checks happen on the server side.

Chat Requirements

Your authenticated chat interface must:

- show a prompt input box only after login
- send user prompts from your REST API/web application instance to the separate `vLLM` server instance
- display model responses to the authenticated user
- store each chat message with the associated user account
- prevent one user from viewing another user's chat history
- provide a logout control that ends the current session

Unauthenticated requests to chat routes must be redirected to login or rejected with an appropriate HTTP status code.

Security Expectations

Your deployment must satisfy all of the following:

- expose only the public `nginx` HTTP port publicly
- expose public HTTP through `nginx`, not by opening the internal FastAPI application port directly
- do not expose the raw `vLLM` server port publicly
- do not configure `nginx` to proxy public traffic directly to `vLLM`
- allow inbound `vLLM` traffic on the model server only from the REST API/web interface instance or its security group
- use the model server's private IP address or private DNS name from the REST API/web application
- store password hashes rather than plaintext passwords
- use a password hashing library designed for password storage
- reject login attempts with invalid credentials
- protect session cookies with reasonable settings for your framework
- keep API keys and other secrets out of your source code
- use environment variables or a local configuration file excluded from version control for secrets

Required Implementation Stack

Use the following stack for this assignment:

- Python with FastAPI for the REST API/web application
- SQLite for users, password hashes, sessions, and chat messages
- `vLLM` running as an OpenAI-compatible server on the separate model server instance
- the OpenAI Python client configured with a custom `base_url` pointing to the model server's private IP address or private DNS name
- `nginx` as the required public reverse proxy in front of FastAPI

Task Sequence

Part 1: Prepare the EC2 Instances

- launch a REST API/web interface EC2 instance
- launch a separate free-tier eligible EC2 instance for the `vLLM` model server
- place both instances in the same VPC
- record the model server's private IP address or private DNS name
- configure SSH access securely
- configure security groups so the public internet can reach only the REST API/web interface

Part 2: Run vLLM

- install system dependencies on the model server instance
- install Python and create a virtual environment on the model server instance
- install `vLLM` on the model server instance
- start `vLLM` as an OpenAI-compatible server using `HuggingFaceTB/SmolLM2-135M-Instruct`
- bind the service so it can receive private-network requests from the REST API/web instance rather than only `localhost`
- configure an API key or equivalent access control
- allow the `vLLM` port only from the REST API/web instance or its security group
- test the model server locally from the model instance
- test the model server from the REST API/web instance using the model server's private IP address or private DNS name

Part 3: Build the REST API/Web Application

- install system dependencies on the REST API/web instance
- create the registration page
- create the login page
- create users with email addresses and hashed passwords
- validate login attempts against stored password hashes
- create sessions after successful login
- implement logout

Part 4: Add Authenticated Chat

- create a protected chat page
- reject unauthenticated chat requests
- send authenticated prompts from the REST API/web instance to the private `vLLM` instance
- display responses in the web interface
- store chat messages by user
- confirm that users cannot see each other's chat history

Part 5: Deploy and Validate

- run the REST API/web application on the interface instance
- install and configure `nginx` as the public reverse proxy in front of FastAPI
- configure FastAPI to listen on a local or private application port behind `nginx`
- configure the REST API/web security group so only the public `nginx` HTTP port and SSH are reachable from approved sources
- configure the model server security group so the `vLLM` port is reachable only from the REST API/web instance or its security group
- verify that the public can reach the REST API/web app through `nginx`
- verify that the internal FastAPI application port is not directly exposed publicly
- verify that the REST API/web app can reach `vLLM` over the private network
- verify that the public cannot directly reach `vLLM`
- verify that registration, login, and logout work end to end

Deliverables

Submit the following:

1. vLLM Deployment Evidence

Provide screenshots or terminal output showing:

- both EC2 instances running
- the model server EC2 instance type and free-tier eligibility
- `vLLM` installed on the model server instance
- the model server running `HuggingFaceTB/SmolLM2-135M-Instruct`
- a successful local request to the `vLLM` server from the model server instance

- a successful private-network request to the `vLLM` server from the REST API/web instance
- evidence that the raw `vLLM` port is not publicly reachable

2. Login Evidence

Provide screenshots or logs showing:

- account registration
- successful login with valid credentials
- rejected login with invalid credentials
- logout

3. Authenticated Chat Evidence

Provide screenshots or logs showing:

- an unauthenticated user blocked from the chat page
- an authenticated user sending a prompt
- a response returned from the model
- chat history associated with the correct user
- a second user unable to view the first user's chat history

Submission Checklist

Before submitting, verify that you have included all of the following:

- evidence for both EC2 instances
- `vLLM` server evidence
- successful private-network prompt-response test from the REST API/web instance to the model server
- proof that direct public access to `vLLM` is blocked
- `nginx` reverse proxy configuration evidence
- proof that the public reaches the app through `nginx`
- account registration evidence
- successful and failed login evidence
- authenticated chat evidence
- per-user chat history evidence

References

- vLLM OpenAI-compatible server documentation
- vLLM CPU installation documentation
- AWS EC2 Free Tier usage documentation