Deployment Guide For campus companion.

Introduction

Welcome to the Deployment Guide for the Campus Companion application. This guide is designed to facilitate a smooth transition from development to a production-ready state for the Campus Companion application, which is essential for enhancing campus life through streamlined services and integrated educational tools. The application architecture is distributed across three primary components and a program that is used to populate a database for the chat system functionality. Apart from the single program both the front-end and back-end part of the application is encapsulated within its own Docker container.

Application Architecture

The deployment architecture of Campus Companion is comprised of the following Docker containers and a program:

1. API Server Container: This container handles all backend logic and data processing requests. It serves as the central node for fetching and managing data.
2. Front-end UI Container: This container is responsible for delivering the user interface. It is designed to serve static content and handle user interactions efficiently.
3. Shibboleth Authentication Container: Dedicated to managing user authentication and security, this container ensures that access to the application is secure and compliant with NCSU standards.
4. generate.py: This is a Python program that is in the back-end file structure which is used to populate the data in a database.

Overview of the Application structure in the file system.

```
campuscompanion/
├── api/
│   ├── .env
│   ├── Dockerfile
│   ├── main.py
│   ├── api/
│   ├── engine/
│       ├── __init__.py
│       ├── constants.py
│       ├── context.py
│       ├── generate.py
│       ├── index.py
│       ├── loader.py
└── app/
    ├── .env
    ├── .gitignore
    ├── Dockerfile
    ├── app/
```

Requirements for Setting Up Deployment

To prepare for the deployment of Campus Companion, the following requirements must be met to ensure a successful setup and launch. Following these requirements will pave the way for a seamless deployment process.

1. Virtual Computing Environment:
    - You will need access to a virtual computing environment (VCE), such as a Virtual Computing Lab (VCL) or a cloud-based instance with similar capabilities. This environment will serve as the host for your application deployment.
2. Access to Source Code Repository:
    - Deployment requires access to the version-controlled source code hosted on GitHub. Ensure that you have the proper credentials or deployment keys to clone the repository. It is vital that you use the latest stable version of the application for deployment unless instructed otherwise.
3. MongoDB Database Setup with MONGO_URI:
    - Establish a MongoDB database that will be used by the application. Following the setup, acquire the MONGO_URI connection string, which facilitates the communication between the application's API and the MongoDB database. This MONGO_URI will be part of the .env file configuration, ensuring secure and correct database access
4. Environment Configuration File:
    - A .env file is essential for defining environment-specific variables, including database connections and other configuration fields. Before beginning the deployment process, ensure that you have created a .env file with all the following information
5. Web Server and Reverse Proxy:
    - Nginx will be used as the web server and reverse proxy to manage and reroute incoming HTTP and HTTPS requests to the appropriate containerized services. You will need to install Nginx on your deployment machine and configure it according to the requirements of the Campus Companion application.
6. Docker and Docker Compose:
    - Docker must be installed on your VCE as it is required to build and run the containerized services of the application. Alongside Docker, Docker Compose is necessary for managing multi-container Docker applications, allowing you to define and run all services with a single command.

7. Running the Docker Compose File:
    - Finally, a docker-compose.yml file is provided to define the services, networks, and volumes. You will execute this file using Docker Compose to build and start the entire application stack. Ensure that you understand each service's role as defined in the Docker Compose file and how they interact within the deployment ecosystem.

Deployment Process

1. Cloud Instance Preparation
   Objective: Set up and log into a cloud computing instance that will host the application.

   For this step of the deployment, you can choose any cloud provider you desire as long as the following requirements are met to handle large traffic load from the application.
    - CPU: Multi-core processors (e.g., 8 cores or more) to handle concurrent processing.
    - Memory: 16 GB RAM or higher to ensure enough memory for application processes and caching.
    - Storage:
        - SSD (Solid State Drive) for faster I/O operations. Size depends on the application's needs, but starting with at least 100 GB is advisable.
        - Additional storage for databases or file storage, depending on the application data management.
    - Network Bandwidth: Provision for high network throughput, especially if you expect a lot of data transfer or if your application is content-heavy.

   Once you have chosen your machine set up your ssh login and stay logged into the system.

2. Source Code Repository Cloning
   Objective: To retrieve the latest stable release of the Campus Companion application code from the source code repository.

   For this step of the deployment process follow these steps to get the code in the cloud machine.

1. Installing Git on the Cloud InstanceIf Git is not installed on your cloud instance, you'll need to install it. Here's how to install Git based on your cloud instance's operating system:
    - If the system is Ubuntu/Debian-based systems you can use the commands below to install git. If your cloud instance has a different Operating system then use this install git manual.

```
sudo apt update
sudo apt install git
```

2. Navigate to the /var/www directory, or create it if it does not already exist using the following commands
    - To create the /var/www directory use this command

```
sudo mkdir -p /var/www
```

    - To navigate use this command

```
cd /var/www
```

3. Once you're in the /var/www directory, clone the GitHub repository. Once you execute the command listed below you might be asked to authenticate and you should do so accordingly:

```
git clone 'your github repository'
```

4. After cloning the repository, navigate into the newly created project directory:

```
cd var/www/2024SpringTeam30-Entrepreneurs
```

3. MongoDB Database Setup
   Objective: Establish a MongoDB database and obtain the connection URI.

   For this step, you can follow these steps which are links to setup MongoDB on the web and gain the URI connection link
    - For the initial setup of MongoDB

- For Getting the URI Connection

Here are the specifications for the MongoDB instance

- CPU: 8+ cores.
- Memory: Minimum 32 GB RAM, ideally 64 GB or more for larger datasets.
- Storage: SSDs with at least 1 TB capacity.

4. Environment Configuration

Objective: Configure environment variables using a .env file required by the application and place it in the correct file in the campus companion file.

For this step of the deployment process, you need to obtain an OpenAI API key. For this process, you can follow OpenAI official Instruction

In your cloud instance Navigate to the Directory using the following command:

```
cd var/www/2024SpringTeam30-Entrepreneurs/campuscompanion/api
```

Once you are in the correct directory follow these instructions to setup your.env file:

1. From the terminal open up any editor you would like or are familiar with
2. In the opened editor enter the following environment variables and save it.
    - MODEL=gpt-4-turbo-preview
    - OPENAI_API_KEY= “Put Your OpenAI API key”
    - MONGO_URI= “Your MongoDB URI”
    - MONGODB_DATABASE=campusCompanion_DATABASE
    - MONGODB_VECTORS= MONGODB_VECTORS
    - MONGODB_VECTOR_INDEX= MONGODB_VECTOR_INDEX
    - MONGODB_USERS= MONGODB_USERS

5. Data Generation and Preparation

Objective: Execute the generate.py script to populate the database for the chat system utilization

Before we can generate any data we need to prepare the environment. For these steps for a Ubuntu system:

1. To Update the package list enter the following command line

```
sudo apt update
```

2. Install Python by executing the following command in the command line

```
sudo apt install python3
```

Once you have installed Python you can use the following command to check if the installation is successful. The two commands should give an output of the version they are.

```
python3 --version
pip3 --version
```

Follow these steps to generate the Database and create the Vector Index:

1. In the cloud instance navigate to the directory where you can run generate.py as a module in the following command.

```
cd var/www/2024SpringTeam30-Entrepreneurs/campuscompanion/api/

poetry install

python app.engine.generate
```

6. Create a Vector Search Index

Objective: Make the populated database readable.

For this step you will need to follow the steps:

1. Log into your MongoDB database in your web
2. Navigate to the collections tab where there is a bar for "Atlas"
3. Follow this official page for creating a Vector Search Index.

Once you have created the Vector Search Index your data in the database will be ready for the chat system to query.

7. Web Server and Reverse Proxy Setup with Nginx

Objective: Install and configure Nginx to handle web requests and route them accordingly.

Before you configure Nginx you need to have a domain name ready to be used in the configuration.

1. Installing Nginx
    - Update your package list:

```
sudo apt update
```

    - Install Nginx:

```
sudo apt install nginx
```

    - Start and enable Nginx to run on boot:

```
sudo systemctl start nginx
sudo systemctl enable nginx
```

2. Configuring Nginx as a Reverse Proxy
    - Edit Nginx configuration: Create or edit a configuration file in the /etc/nginx/sites-available/ directory by using the text editor you would like. Name this file after your domain or application, such as campuscompanion.conf.
    - Add the following configuration in the file

Unset

```
server {
    listen 80;
    server_name yourdomain.com;  # Replace with your actual domain
    location / {
        proxy_pass http://localhost:8000;  # Assuming your app runs on port 8000
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

    - Enable the configuration by linking it to the sites-enabled directory:

```
sudo ln -s /etc/nginx/sites-available/campuscompanion.conf /etc/nginx/sites-enabled/
```

    - Test Nginx configuration for syntax errors:

```
sudo nginx -t
```

- Reload Nginx to apply the changes

```
sudo systemctl reload nginx
```

3. Securing Nginx with SSL/TLS Certificates
   - Install Certbot and its Nginx plugin (use this for Debian/Ubuntu):

```
sudo apt install certbot python3-certbot-nginx
```

   - Run Certbot to automatically obtain and install an SSL certificate and configure Nginx to use it:

```
sudo certbot --nginx -d yourdomain.com  # Replace with your actual domain
```

   - Automatic Renewal of Certificates:

```
sudo certbot renew --dry-run
```

8. Docker Compose Execution

Objective: Launch the application using Docker Compose.

Installing Docker

1. Update your package manager (assuming you're using a Debian-based system like Ubuntu):

```
sudo apt update
```

2. Install required packages for apt to use a repository over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3. Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

4. Set up the stable Docker repository:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

5. Install Docker CE (Community Edition):

```
sudo apt install docker-ce
```

6. Start Docker and enable it to run at startup:

```
sudo systemctl start docker
sudo systemctl enable docker
```

Installing Docker Compose

1. Download the latest version of Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Make the Docker Compose binary executable:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Running Docker Compose YAML file for campus Companion

1. Navigate to your the projects directory where your docker-compose.yml file is located:

```
cd var/www/2024SpringTeam30-Entrepreneurs/campuscompanion
```

2. Execute Docker Compose to build and start your services:

```
docker compose up --build
```

Once the application will give you a message saying that the application has started in the terminal once the application build the docker image and run them all together.

Troubleshooting Guide for Campus Companion Deployment

This Troubleshooting guide aims to address common issues that may arise during the deployment of the Campus Companion application, focusing on solutions and best

practices to troubleshoot effectively. During the deployment process, various problems can occur due to system configuration errors, network issues, or dependencies among other factors. This guide will help you identify and resolve these issues to ensure a successful deployment of Campus Companion.

Common Problems and Troubleshooting Steps

1. Docker Container Fails to Start
   Problem:
   - Docker containers exit immediately after start.
   - Error messages related to Docker in the terminal or logs.

   Possible Causes:
   - Misconfiguration in docker-compose.yml.
   - Required environment variables are missing or incorrect.
   - Insufficient system resources.

   Troubleshooting Steps:
   - Verify all configurations in docker-compose.yml are correct.
   - Ensure all required environment variables are defined in .env file.
   - Check system resources (CPU, memory, disk space) to ensure they meet the application's requirements.
2. Database Connectivity Issues
   Problem:
   - The application cannot connect to MongoDB.
   - Error logs contain messages about failing to connect to the database.

   Possible Causes:
   - Incorrect MongoDB URI in the .env file.
   - Network issues preventing access to the database server.

   Troubleshooting Steps:
   - Double-check the MongoDB URI and credentials in the .env file.
   - Ensure the database server is up and accepting connections from your application's host.
   - Check network settings and firewall rules that might block database connections.
3. Authentication Errors with Shibboleth Container
   Problem:
   - Users are unable to log in.
   - Error messages related to authentication failures.

   Possible Causes:

- Misconfiguration in the Shibboleth service.
- Incorrect or expired credentials.

Troubleshooting Steps:

- Verify the Shibboleth configuration settings.
- Check logs for detailed error messages and fix any identified issues with credentials or configurations.
- Ensure that the Shibboleth container can communicate with other necessary services.

4. Web Server (Nginx) Misconfigurations

Problem:

- 502 Bad Gateway or 404 Not Found errors.
- Static assets not loading.

Possible Causes:

- Incorrect Nginx configuration.
- Nginx is not properly routing requests to backend services.

Troubleshooting Steps:

- Review Nginx configuration files for errors.
- Ensure that Nginx is set up as a reverse proxy correctly and is pointing to the right ports for backend services.
- Check Nginx and application logs for further insights into the routing issues.

5. SSL/TLS Certificate Issues

Problem:

- Browser warns about insecure connections.
- SSL handshake errors.

Possible Causes:

- SSL certificates are not installed or misconfigured.
- Expired SSL certificates.

Troubleshooting Steps:

- Use tools like SSL Labs' SSL Test to check for common vulnerabilities and configuration issues.
- Verify that Certbot has properly installed and configured SSL certificates for Nginx.
- Check for certificate renewal issues and ensure automatic renewal is set up.

6. Script Failures During Data Generation

Problem:

- generate.py script exits with errors.
- Incomplete or incorrect data in the database.

Possible Causes:

- Python environment issues.
- Script bugs or incorrect assumptions about the data format.

Troubleshooting Steps:

- Ensure the Python environment has all required dependencies installed.
- Run the script in debug mode to capture detailed error logs.
- Validate input data formats and adjust scripts accordingly if they fail to handle current data correctly.

Verification Plan for Campus Companion Deployment

This plan outlines the strategies and methods to verify that each part of the system is installed correctly and operating properly. Here's how you can structure this verification plan:

Web access and functionality Verification Steps

1. Check the Website's Accessibility
   - **What to Do**: Open a web browser and type the URL of the Campus Companion application. The homepage should load without errors.
   - **What You're Looking For**: Ensure that the website is accessible, and the homepage appears as expected without any error messages.
2. Test the Login Function
   - **What to Do:** Attempt to log in using a known user account.
   - **What You're Looking For**: After entering your credentials, you should be redirected to the main user interface of the application. Ensure you can log in and log out without any errors.
3. Navigate Through Key Pages
   - **What to Do**: Click through the main sections of the application available in the navigation menu.
   - **What You're Looking For:** All pages should load correctly, and all links should take you to the correct sections without errors or delays.
4. Perform Basic User Interactions
   - **What to Do:** Try adding, editing, and deleting information where the application permits (like profile updates, posting updates, etc.).
   - **What You're Looking For:** Each action should complete successfully without errors, and changes should be visible immediately.

5. Check Connectivity to the Database
    - **What to Do:** Look for any functionalities in the app that require database interaction, such as retrieving historical data or statistics.
    - **What You're Looking For:** Data should load correctly, and there should be no errors indicating a failure to retrieve data.
6. Verify Security Features
    - **What to Do:** Ensure the login page is served over HTTPS and log out to see if the session ends properly, preventing re-access without credentials.
    - **What You're Looking For**: Check for the HTTPS in your browser's address bar on login and other sensitive pages, ensuring data is encrypted.

Technical Verification Steps

1. API Server Container Verification
   Verification Methods:
    - API Response Checks: Send HTTP requests to the API endpoints and verify the responses are as expected. Use tools like Postman or curl for these requests.
        - Example: Run the below curl command this the cloud instance where the application is deployed to test the main chat funtinoality

```
curl -X POST http://localhost:8000/api/chat \
-H "Content-Type: application/json" \
-d '{"messages": [{"role": "USER", "content": "Hello"}]}'
```

    - Log Review: Check the logs for any startup errors or warnings that could indicate problems. You can find the logs in the cloud instance where they are deployed in the directory **"var/www/2024SpringTeam30-Entrepreneurs/campuscompanion/logs/"**.
      Here you will find:
        - access.log
        - error.log

      You can vrows this files for any warnings or eros the application might encounter further enhancing your verification process

2. Database Population Verification
   Verification Methods:

- Manually Investigate MongoDB: Log into your MongoDB database and browse through the database collection. You should be able to see that is populated. By manually checking the objects of data that has been populated you can verify that in the deployment process the data population step is done properly. Below is an example: