Publisher-Subscriber System - README

Overview

This project implements a **Publisher-Subscriber system** using a central **indexing server** that facilitates communication between multiple **peer nodes**. The **indexing server** manages topics, subscriptions, and message publishing for peers, allowing them to create, subscribe to, and retrieve messages from topics.

Components

1. **Indexing Server**: The central server that manages peer nodes and topics.
2. **Peer Nodes**: Clients that can register, create topics, publish messages, and subscribe to topics.
3. **Deployment Script**: A deploy.sh script that automates the startup of the indexing server and peer nodes.

Prerequisites

- **Python 3.6+** installed on your machine.
- **Virtual Environment**: It's recommended to use a virtual environment to isolate dependencies.
- **Pip**: Python's package manager to install dependencies.

File Structure

```
PA2
├── Code
│   ├── deploy.sh              # Deployment script to start the indexing server and peers
│   ├── indexing_server.py     # Central indexing server script
│   ├── peer_node.py           # Peer node script
│   ├── api_benchmark.py       # Script for benchmarking API performance
│   ├── response_time_test.py  # Script for measuring response times under load
│   ├── test_p2p.py            # Script for testing API functionalities
│   ├── requirements.txt       # List of required dependencies
│   ├── Makefile               # Makefile for setting up environment and installing dependencies
├── Files
│   ├── Report                 # Report
│   ├── Design Doc             # Design documentation
│   ├── ReadMe                 # README file
├── Out
│   ├── Logs                   # Directory for storing log files
│   ├── Images                 # Directory for storing generated images and graphs
```

Setup Instructions

Follow these steps to set up the project and run the indexing server and peer nodes.

1. Set Up a Virtual Environment

You can either use the provided **Makefile** to set up the virtual environment or do it manually.

Option 1: Using Makefile (Recommended)

The Makefile provides automated commands for setting up the virtual environment and installing dependencies:

1. **Set up the virtual environment and install dependencies**:

   ```
   make install
   ```

2. **Freeze dependencies to** requirements.txt (if you add or update packages):

   ```
   make freeze
   ```

Option 2: Manually

1. **Create a virtual environment**:

   ```
   python3 -m venv venv
   ```

2. **Activate the virtual environment**:

   - **Linux/macOS**:

     ```
     source venv/bin/activate
     ```

   - **Windows**:

     ```
     venv\Scripts\activate
     ```

3. **Install dependencies**:

   ```
   pip install -r requirements.txt
   ```

2. Running the Program

You can either run the system manually or use the provided **deployment script** to automate the process.

Option 1: Manual Run

1. **Start the Indexing Server**:

   Run the indexing server on port **5000**:

   ```
   python indexing_server.py
   ```

2. **Start Peer Nodes**:

   Open another terminal window and run the peer nodes on different port:

   ```
   # Peer1
   python peer_node.py 5001 peer1
   ```

Option 2: Automated Deployment

Use the provided deploy.sh script to automatically start the **indexing server** and 3 peer nodes.

1. **Make the script executable**:

   ```
   chmod +x deploy.sh
   ```

2. **Run the deployment script**:

   ```
   deploy.sh
   ```

   This will start the indexing server on port **5000** and the a peer node on ports **5001.**

3. **Stop the services**: Press **Ctrl+C** to stop the indexing server and peer nodes when you're done.

3. API Endpoints and cURLs

Below are the key API endpoints available for interacting with the **indexing server** and **peer nodes**.

Register a Peer Node

cURL: `curl -X POST http://localhost:5000/register -H "Content-Type: application/json" -d '{"peer_id": "peer3"}'`

Method: POST
Endpoint: /register_peer_node
Body:

```
{
  "peer_id": "peer1"
}
```

Unregister a Peer Node

cURL: `curl -X POST -H 'Content-Type: application/json' -d '{"peer_id": "peer1"}' http://localhost:5000/unregister_peer_node`

Method: POST
Endpoint: /unregister_peer_node
Body:

```
{
  "peer_id": "peer1"
}
```

Create a Topic

cURL: `curl -X POST http://localhost:5001/create_topic -H "Content-Type: application/json" -d '{"topic": "sports"}'`

Method: POST
Endpoint: /create_topic
Body:

```
{
  "topic": "news"
}
```

Publish a Message to a Topic

cURL: `curl -X POST http://localhost:5001/publish -H "Content-Type: application/json" -d '{"topic": "sports", "message": "Team A won the match!"}'`

Method: POST
Endpoint: /publish
Body:

```
{
  "peer_id": "peer1",
  "topic": "news",
  "message": "Breaking news: AI is transforming the world!"
}
```

Subscribe to a Topic

cURL: `curl -X POST http://localhost:5002/subscribe -H "Content-Type: application/json" -d '{"topic": "sports"}'`

Method: POST
Endpoint: /subscribe
Body:

```
{
  "peer_id": "peer1",
  "topic": "news"
}
```

Pull Messages from a Topic

cURL: `curl -X POST http://localhost:5002/pull_messages -H "Content-Type: application/json" -d '{"topic": "sports"}'`

Method: POST
Endpoint: /pull_messages
Body:

```
{
  "peer_id": "peer1",
  "topic": "news"
}
```

Query Peers Subscribed to a Topic

cURL: `curl -X GET "http://localhost:5000/query_peers?topic=technology"`

Method: GET
Endpoint: /query_peers
Parameters: ?topic=news

Delete a Topic

cURL: `curl -X DELETE http://localhost:5001/delete_topic -H "Content-Type: application/json" -d '{"topic": "sports"}'`

Method: DELETE
Endpoint: /delete_topic
Body:

```
{
  "topic": "news"
}
```

4. Logging

Logs are generated for both the **indexing server** and each **peer node**:

- **Indexing server logs**: Stored in a file named indexing_server_<timestamp>.log.
- **Peer node logs**: Each peer node has its own log file named {peer_id}_{timestamp}.log, which records all actions performed by that peer.

5. Stopping the System

- **If using the deployment script**: Press **Ctrl+C** to stop the indexing server and peer nodes.

- **If running manually**: Simply stop the server and peer processes in each terminal.

6. Troubleshooting

- **Port Already in Use**: Ensure that ports 5000, 5001 are free before starting the system.
- **Virtual Environment Issues**: Ensure the virtual environment is activated before running any Python commands.
- **Missing Dependencies**: If any package is missing, ensure that requirements.txt is up-to-date by running make freeze or manually adding the package.

7. Additional Commands

Clean the Virtual Environment

If you want to remove the virtual environment and start fresh:

```
make clean
```

Conclusion

This **Publisher-Subscriber system** allows multiple peer nodes to interact through a central indexing server. The system can be easily set up, deployed, and managed using the provided scripts. If you need further assistance or have questions, feel free to reach out.