MiniTwitter

Project Documentation

Authors:

Michał Szynkaruk
119273

Jakub Sienski
119269

Oliwia Witkowska
120867

29 January 2025

Contents

1 Overview

MiniTwitter is a distributed microblogging application that allows users to send and retrieve short messages (tweets) through a client-server architecture implemented with **gRPC** and **Protocol Buffers**. The project is designed as an educational tool to demonstrate software engineering best practices, including CI/CD, automated testing, and infrastructure deployment.

2 Getting Started

2.1 Prerequisites

- Python 3.9 or higher
- `pip` (Python package manager)
- gRPC tools (`grpcio` and `grpcio-tools`)

3 Installation

3.1 Clone the repository

```
git clone https://github.com/MlCHAL-S/MiniTwitter.git
cd MiniTwitter
```

3.2 Set up a virtual environment

```
python -m venv .venv
source .venv/bin/activate    # On Windows: .venv\Scripts\activate
```

3.3 Install dependencies

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

3.4 Generate gRPC code (if needed)

```
python -m grpc_tools.protoc -Iproto --python_out=. --
grpc_python_out=. proto/minitwitter.proto
```

4 Running the Application

4.1 Set PYTHONPATH

```
export PYTHONPATH=src:$PYTHONPATH
```

4.2 Starting the Server

To start the MiniTwitter server, run:

```
python -m src.server.server
```

The server listens on `localhost:50051` for gRPC connections.

4.3 Running the Client

To start the MiniTwitter client, run:

```
python -m src.client.client
```

The client provides a simple CLI for interacting with the server:

- `SEND <message>`: Send a tweet to the server.
- `GET <number>`: Retrieve the last `<number>` messages.
- `EXIT`: Exit the client application.

5 Architecture

5.1 Software Architecture

The MiniTwitter system uses a **client-server model**:

- **Client**:
  - Provides a Command-Line Interface (CLI) for users to send and retrieve messages.
  - Interacts with the server over a gRPC connection.
- **Server**:
  - Manages in-memory storage of messages.
  - Implements two gRPC methods:
    - `sendMessage`: Accepts and stores user messages.
    - `getMessages`: Retrieves a specified number of recent messages.

5.2 Deployment Architecture

The application is designed to run in a cloud environment using **Terraform** for Infrastructure as Code (IaC). Key components include:

- **Amazon Web Services (AWS):**
  - **VPC**: A dedicated Virtual Private Cloud to isolate network resources.
  - **EC2 Instance**: Hosts the MiniTwitter server.
  - **Security Groups**: Manage network access to the server.
- **Networking Setup:**
  - Subnets, Route Tables, and Internet Gateways ensure communication between clients and the server.

5.3 Logging System

A dedicated logs folder has been introduced to store logs generated by the MiniTwitter server and client. These logs help in debugging and monitoring system behavior over time. Logging system ensures that all major actions, including user commands, message transactions, and server activity, are recorded for analysis. A dedicated logs folder has been introduced to store logs generated by the MiniTwitter server and client. These logs help in debugging and monitoring system behavior over time.

6 Features and Functionality

6.1 Final Functionalities

- **Message Sending**:
  - Clients can send text messages (up to 80 characters) to the server.
  - Messages are stored in the server's memory.
- **Message Retrieval**:
  - Clients can retrieve the most recent messages, with a user-defined count.
  - The server responds with the requested messages or all available messages if the count exceeds stored messages.
- **Concurrency**:
  - Supports multiple clients simultaneously accessing the server.

6.2 Intermediary Steps

To ensure a structured development process, the project was built in increments:

1. **Initial gRPC Setup**:
   - Defined Protocol Buffers (`minitwitter.proto`) for `sendMessage` and `getMessages`.
   - Generated gRPC stubs and service classes.
2. **Server Implementation**:
   - Created `MiniTwitterServicer` to handle gRPC requests.
   - Added in-memory storage for messages.
3. **Client Implementation**:
   - Built a CLI to interact with the server.
   - Implemented error handling for invalid inputs and commands.
4. **Automated Testing**:
   - Developed unit tests for both client and server.
   - Configured `pytest` for test coverage.
5. **CI/CD Pipeline**:
   - Set up GitHub Actions for automated linting and testing on pull requests.
6. **Cloud Deployment**:
   - Automated infrastructure setup using Terraform.
   - Deployed the server to an AWS EC2 instance.

7 Implementation Details

7.1 gRPC and Protocol Buffers

- **Proto File (`minitwitter.proto`)**:
  - Defines the gRPC service with two methods:
    - `sendMessage(MessageRequest) -> MessageResponse`
    - `getMessages(MessageListRequest) -> MessageListResponse`
  - Messages:
    - `MessageRequest`: Contains the text of the message.
    - `MessageListRequest`: Specifies the number of messages to retrieve.

    - `MessageResponse` and `MessageListResponse`: Return status and message content.

- **Generated Files**:
  - `minitwitter_pb2.py`: Contains Python representations of the message types.
  - `minitwitter_pb2_grpc.py`: Provides the client and server classes for gRPC communication.

7.2 Client (`client.py`)

- **Features**:
  - CLI allows users to:
    - Send messages: `SEND <message>`
    - Retrieve messages: `GET <count>`
    - Exit the application: `EXIT`
  - Establishes a gRPC channel to communicate with the server.
- **Error Handling**:
  - Validates user input for commands and arguments.
  - Handles server responses gracefully.

7.3 Server (`server.py`)

- **Features**:
  - Implements the `MiniTwitterServicer` class with:
    - `sendMessage`: Appends a message to the in-memory list.
    - `getMessages`: Retrieves recent messages based on the requested count.
  - Runs on port `50051` using a thread-pool executor to handle concurrent clients.
- **In-Memory Storage**:
  - Messages are stored as a Python list shared across requests.
- **Logging**:
  - Prints server start-up information and request handling for debugging.

8 Testing

8.1 Unit Tests

- **Client Tests (`test_client.py`)**:
  - Mocked gRPC stubs to simulate server responses.
  - Validated commands (`SEND`, `GET`, `EXIT`) and error handling.
- **Server Tests (`test_server.py`)**:
  - Tested `MiniTwitterServicer` methods:
    - `sendMessage`: Ensured messages are stored correctly.
    - `getMessages`: Verified retrieval of the correct number of messages.
  - Smoke test for `serve` function to confirm the server starts properly.

8.2 Test Configuration

- **pytest**:
  - Test discovery and execution.
  - Configured in `pytest.ini` to include coverage analysis.
- **pylint**:
  - Enforced code quality and PEP8 compliance.

8.3 Code Coverage

Code coverage has been measured using `pytest`, and the results demonstrate a high level of test coverage:

```
Name                    Stmts   Miss  Cover   Missing
-------------------------------------------------------
src/client/client.py       39      5    87%   73-78
src/server/server.py       26      1    96%   77
-------------------------------------------------------
TOTAL                      65      6    91%
```

This confirms that over 90% of the code is covered by unit tests, ensuring strong reliability of both the client and server components. Additionally, the client and server components log their interactions, providing enhanced traceability and debugging capabilities.

8.4 Linting and Code Quality

`pylint` has been used to ensure adherence to coding standards.

- The `pylint` score for both `client.py` and `server.py` is **10/10**, indicating that the code follows best practices and is free from style and formatting issues.

9 Automation

9.1 Continuous Integration

- **GitHub Actions (`.github/workflows/ci.yaml`)**:
  - Pipeline runs on pull requests to `main`.
  - Steps include:
    * Code checkout.
    * Python environment setup (version `3.12`).
    * Dependency installation.
    * Linting with `pylint`.
    * Test execution with `pytest`.

9.2 Deployment Automation

To deploy the MiniTwitter server using Ansible, run the following command:
`ansible-playbook -i ansible/inventory.ini ansible/config.yaml`

- **Ansible**:
  - `ansible/config.yaml`: Defines deployment configuration.
  - `ansible/inventory.ini`: Specifies target hosts.
  - Automates deployment of the MiniTwitter server.

9.3 Infrastructure as Code

- **Terraform**:
  - `main.tf`: Provisions AWS resources including:
    * **VPC**: A virtual private cloud to isolate network resources.
    * **EC2 Instance**: Hosts the MiniTwitter server.
    * **Security Groups**: Define access control rules for incoming/outgoing traffic.
  - `vars.tf`: Contains configurable parameters, such as:
    * AWS region.

    - Instance type.
    - Key-pair for SSH access.
  - `output.tf`: Outputs important information post-deployment, such as:
    - Public IP address of the EC2 instance.
    - VPC ID for reference.

9.4 Workflow

- Initialize Terraform (`terraform init`).
- Validate the configuration (`terraform validate`).
- Plan infrastructure changes (`terraform plan`).
- Apply changes to create resources (`terraform apply`).

10 Team Contributions

10.1 Responsibilities

- **Proto Definitions**:
  - Developed by Michał Szynkaruk (`MlCHAL-S`).
  - Ensured compatibility with gRPC.
  - Created cloud infrastructure and automated deployment using Ansible and Terraform.
  - Implemented the CI pipeline using GitHub Actions.
- **Server Implementation**:
  - Led by Oliwia Witkowska (`olivblvck`).
  - Focused on gRPC servicer logic and message storage.
- **Client Implementation**:
  - Developed by Jakub Sienski (`KubaSienski`).
  - Created a user-friendly CLI and implemented input validation.
- **Infrastructure Setup**:
  - Configured by Michał Szynkaruk (`MlCHAL-S`).
  - Automated using Terraform and Ansible.
- **Documentation**:
  - Documentation written by Oliwia Witkowska (`olivblvck`) and reviewed by Michał Szynkaruk (`MlCHAL-S`).
  - Ensured comprehensive documentation.

10.2 Workflow

- **GitHub Projects**:
  - Tracked progress and assigned tasks as issues.
- **Pull Requests**:
  - Each feature was developed in a dedicated branch.
  - PRs were peer-reviewed before merging.

11 Results and Future Work

11.1 Results

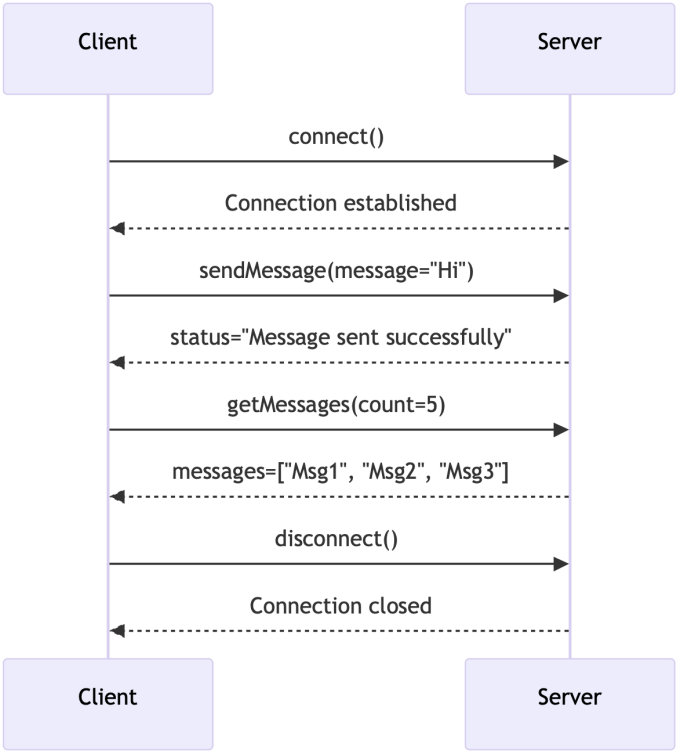
- Successfully implemented all planned functionalities.
- Deployed a working server to AWS, accessible by multiple clients.

11.2 Future Enhancements

- **Persistent Storage**:
  - Replace in-memory storage with a database for scalability.
- **Authentication**:
  - Add user authentication and message ownership.
- **Enhanced Error Handling**:
  - Improve validation for edge cases and add more descriptive error messages.

12 Appendix

12.1 Temporal Sequence Diagram



12.2 Cloud Architecture Diagram