Remote File-Server using Telegram API

> Team Members:
>
> 1- Asem Abdelhady (SD-01)
>
> 2- Mosab Mohamed (SD-01)
>
> 3- Hadi Saleh (CS-01)
>
> 4- Jaffar Totanji (SD-01)

Project Description:

A **Telegram bot** that accesses a remote server using `Telegram API`.

It allows the system admin to navigate, retrieve and manipulate any files that are present in the remote server. It also, allows for the execution of shell commands on the remote server through Telegram messages.

Additionally, the bot receives customized monitoring and logging messages from `Grafana`, and pushes these messages as notifications to the user via Telegram.

This is done by mounting your remote server's home directory on a running docker container for the Telegram bot server with which you can use the `Telegram API`.

Goals:

- Grant a system admin access to their remote server through Telegram in situations where they don't have access to sufficient tools to work with their server. Eliminating the need for certain systems/dependencies to access the server.
- Notify the admin about critical events which may require their immediate attention, and the ability to handle such situations promptly and remotely.
- **Use Cases**:
  - The admin is away from their system, and are required to make a quick immediate change in the server, they can make that change immediately via any device with Telegram being the only dependency.
  - The admin is away from the system, a critical situation arises e.g. Server CPU overheated. The sys admin can then simply access and shut down the server through their phone for example.

Tasks:

- Discuss the project idea and its features.
- Sketch the design and information flow for the infrastructure.
- Implement the basic app structure in python, using the pyrogram wrapper for `Telegram API`.
- Dockerize the application.
- Make docker-compose file to run `Grafana`, `Prometheus` and node export locally.

- Configure `Grafana` to add dashboards for `Prometheus` and `node-export`.
- Add alerts to send notifications to the Telegram bot from `Grafana`.
- Make all this run under `Ansible`.
- Add `Github Action` so that with every feature added to the Telegram bot server, it builds and deploys a docker image into `dockerhub`.

Methodology:

We adopted the `Waterfall` development methodology, due to the fact that the project was large in terms of requirements and research, and rather simple in terms of implementation.

We also opted to only work on the project whenever we were all available at the same time to avoid complexities, because of the nature of the project's idea and the fact that all project parts are dependent on one another.

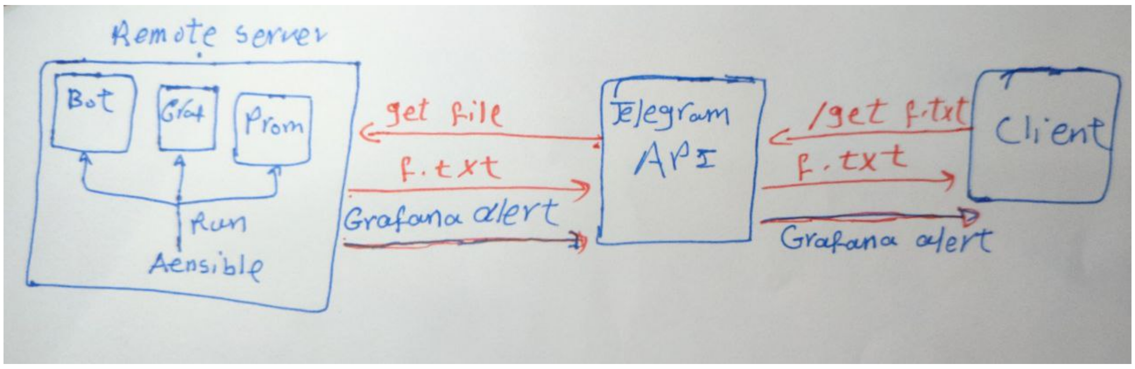
Hence, we organized meetings whenever everyone was free, and conducted it in the following way:

1. Free discussion of researched task.
2. Deliberations on implementation specifics.
3. Implementation.
4. Discussion of the next task.

During the time between meetings, members of the team were required to research the task that will then be further discussed and implemented in the next meeting.

Development:

- **Design Specification**



- **Implementation**
  1. Used `pyrogram` to implement three main features (`cd, ls, get`).
  2. Implemented `Dockerfile` that securely runs the Telegram bot server by mounting the home directory.
  3. Created `docker-compose` file to handle monitoring.
  4. `Prometheus`:
     - Created the `Prometheus` yaml file to fetch the system hardware performance metrics in order to monitor the system state.
     - Added the `Prometheus` configurations to the `docker-compose` file.
  5. `Node Export`:
     - Added `Node Export` configurations to the `docker-compose` file.
  6. `Grafana`:

        - Added `Grafana`'s configurations to the `docker-compose` file.
    7. Implemented The `run` feature, which executes `shell scripts` on the remote server.
    8. Created the `inventory.ini` to be specifically run on localhost.
    9. Created the `Ansible` configuration file (`.cfg`)
    10. `Ansible:`
        - Created `Ansible`'s `main.yaml` to run four tasks:
            1. Copy monitoring folder to `/tmp` in the development phase.
            2. Copy app folder to `/tmp` in the development phase.
            3. Run the `docker-compose` from `/tmp/monitoring`.
            4. Run the `docker-compose` from `/tmp/app`.
- **Testing**
    1. Clone the repository from another several other machines.
    2. Configured the `.env` file for each cloned repo.
    3. Ran the `ansible-playbook`.
    4. Tested the Telegram bots' functionality.
- **CI/CD**
    - Created a github CI/CD file.
    - Added one job to run two steps:
        1. Builds the `docker-image` on each commit.
        2. Deploys the `docker-image` to `dockerhub`.

Development Logs:

16/11/2022: Project Idea

- Discussing and Finalizing project idea.
- Agreeing on development methodology.
- Sketched the infrastructure design.
- git init

21/11/2022: Basic funcionality

- Deciding what features will be added.
- Adding basic features (`cd, ls, get`)

30/11/2022: Docker

- Made the app's `Dockerfile`.

05/12/2022: Ansible, Logging, Monitering

- Made the `docker-compose` file for monitoring.
- Added the `Prometheus`, `Node Export`, and `Grafana` configurations to the `docker-compose` file.
- Created the first draft of the `Ansible` file.

07/12/2022: Ansible, Run feature, Docker Compose

- Added the `run` feature.
- Made the `docker-compose` for the app.
- Finished `Ansible` to run both `docker-compose` files.

09/12/2022: Testing

- Tested the Telegram bot in different environments

11/12/2022: Github Actions

- Made the `Github Actions` CI/CD file.

12/12/2022: Readme, Report

- Wrote a comprehensive **README to explain how to setup the bot (Because it is extremely personalized)**.
- Reaching a conclusion and writing the report of the project.

17/12/2022: Demo, Finalization

- Made the demo showcasing the configurations and features.

- Final touches.

Difficulties:

Although we wanted to include unit tests in our project, we struggled to find proper features to test. Due to the fact that the Telegram bot is an always running process. Also due to the nature of this project and its focus on system administration there wasn't much focus on app features which led to little to no reasonable tests to add, so we opted to focus on other administrative areas instead, because it was more in line with the learning outcomes of this project.

We also faced some difficulties in how to make the `Docker` container secure in terms of limiting the user's access to root files and in the same time preventing other unwanted users with different credentials from the `.env` file to use the personalized bot of a specific user.

We solved this issue by making a function that checks the credentials of the user before responding to any command. Furthermore in the `Dockerfile` we mounted the home to `/app/storage` and changed the permissions so they are suitable for the container user.

Finally, we had some issues in making a `JSON` file suitable for the `Grafana` dashboard, so we used a pre-existing template.

Conclusion:

Throughout all the different stages of planning, development, testing, we learned about a handful of new technologies, their strengths and weaknesses. We were also able to improve our soft skills by working as a team and setting our own goals and expectations of each other. To name a few learning outcomes:

- We got to know more about `Prometheus` and `Node Export` which are monitoring tools that are extremely powerful in terms of system hardware and performance metrics and all the

different areas they could be utilized in.
- We learned about `Grafana` and how helpful it is in system performance logs and metrics visualization and how it could be used to produce customizable alerts.
- We gained more experience in using `Docker` and `docker-compose` which will be helpful in future projects, courses, and ultimately in the job market later on.
- The nature of this project made us more comfortable working in a team setting and our adoption of the waterfall development method made us recognize the value of predetermining the requirements and design to have a concrete base to build on instead of making it up along the way.

Finally, this project in specific -and the course as a whole- was heavily leaning towards the practical side, and infromed us about technologies and techniques that are widely-used, and required in practice (Monitoring tools, Logging and Log rotation, Bash Scripting, Ansible, CI/CD, Docker, and Docker-Compose). We believe that this approach is extremely beneficial and engaging for the students, and hope for it to be adopted by more courses in the future. So, we would like to thank everyone who was involved in the organization of this course. Big Thank you from our team!

Links

1- Project repository

2- Demo video

3- DockerHub

Notes

- It is extremely advisable to read the README to be able to setup the configurations.