COMP0239 Coursework

David Roman Frischer

April 23, 2024

Contents

1 Introduction

A (fake) customer service company has asked me to analyse calls between customer service agents and customers in order to improve the quality of their service. To achieve this goal, the company wants to identify calls that have resulted in poor conversations and then analyse why the customer is dissatisfied. This report demonstrates a working workflow to meet these requirements. The pipeline of this project has been tested for at more than 24 hours on over 621 audio files, each approximately 30 minutes in length for an average of 9 minutes and 8 seconds to process it, with pipeline and hardware being monitored during runtime.

2 Methodology

The pipeline in Figure 1 illustrates the Directed Acyclic Graph (DAG) for analysing a conversation stored in an audio file. As shown in the figure, this file is initially stored in the `new_audios` folder. At the start of the pipeline, the file is moved to the `pipelines_audios` folder to begin processing. The first step is to convert the audio file to WAV format if it is an MP3 file. Then the diarisation task is run to transcribe and separate the words of the customer and the customer support representative. Finally, three successive tasks are performed to evaluate and summarise the conversation, to understand how it went and to determine whether further steps need to be taken by the company. All results are saved in a database for access through a web platform designed specifically for this project.

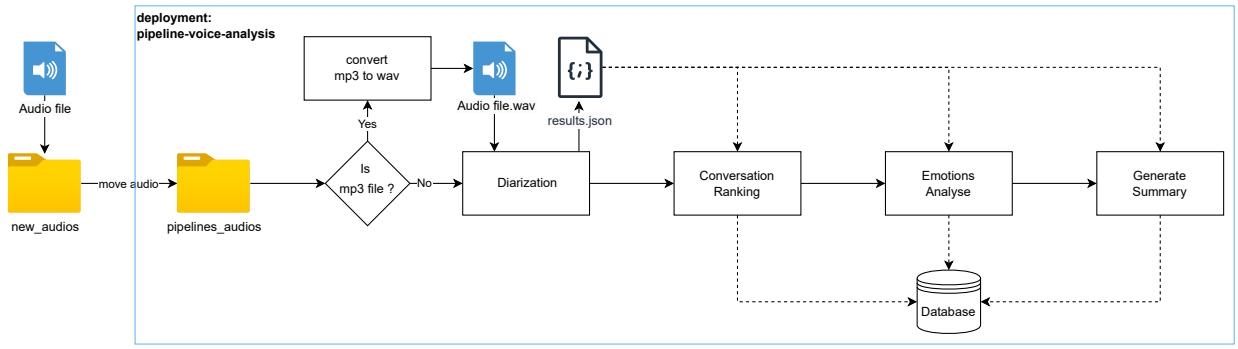

Figure 1: Pipeline Voice Analysis

3 Requirements

During the software development process, it is obvious to consider idempotency, robustness and easy connectivity as key elements to ensure successful delivery of the product. Additional specifications must also be considered based on the requirements provided by the company.

3.1 Distributed data analysis

The distributed system must be resilient and scalable. In anticipation of the need to analyse more data in the future, the system should be horizontally scalable and easy to set up. This will ensure that it can efficiently handle increased data volumes while maintaining performance and reliability.

3.2 Pipeline and hardware monitoring

It should be possible to monitor the pipeline throughout the process until completion or failure is detected. In addition, users should be able to track the number of audio files remaining to be processed, those in progress, those completed and those that have failed. This comprehensive monitoring capability ensures transparency and facilitates efficient management of pipeline operations.

3.3 Add new data without code

The platform should allow new audio files to be added to the processing queue, even after pipelines have started. This process should be straightforward for any user, allowing seamless integration of new data without the need for coding knowledge.

3.4 Viewing and Retrieving Pipeline Output

Once the pipeline has completed its execution, users should be able to access the results in a conventional file format. They should also be able to download the data for further analysis if required. This ensures that users can easily review the output and perform additional analysis if required.

4 Choice of the Distributed System Engine

Choosing the right orchestrator for our distributed system is critical to ensuring efficient operation and scalability. Airflow, Spark or Celery could all be suitable options for the project. However, Prefect[1] was chosen because of its user-friendly and modern interface (also suitable for screen phones), the seamless integration of Prefect with Python using decorators, and the requirement for a single Docker image to start a Prefect server or worker in the cluster.

5 Data Management

The data storage strategy should distinguish between storing files and storing records. Storing audio in a relational database, as well as storing the results of a flow run in a file for long term rather than a database, would generally be considered bad practice. Although technologies such as Hive and Trino provide options for managing data within a database environment, they are typically used for structured data stored in distributed file systems such as Hadoop's HDFS or cloud storage systems. For the platform, it was decided to use NFS (Network File System) for file sharing between VMs and a Postgres database within the Swarm network for data storage and management.

5.1 Network File System (NFS)

The Network File System is used to store new audio files for sharing across the cluster, as well as any files generated during a flow run. The NFS is mounted on the client VM, providing a total storage capacity of approximately 217GB across the machines.

5.2 Relational Database Postgres

A Postgres database will store all the results of a pipeline. Figure 2 illustrates the schema design of the relation between the models.

Audio Results: This table contains the metadata results of a pipeline. It includes the `flow_run_id` generated by the Prefect API, the audio file path given by the user, the worker ID (the local IP address), the full transcript, and a summary of the conversation.

Speakers: The Speakers table records the individual's positive contributions to the conversation, such as a transcript of just their part of the conversation.

Emotion Scores: Each speaker is associated with six emotions, which are scored from 0 to 1, representing the absence (0) or presence (1) of the emotion. The emotions analysed are "sadness", "joy", "love", "anger", "fear" and "surprise". These emotions are linked to both the `flow_run_id` and the `speaker_id` to establish a relationship with the Speakers table.

[1] `https://www.prefect.io/`

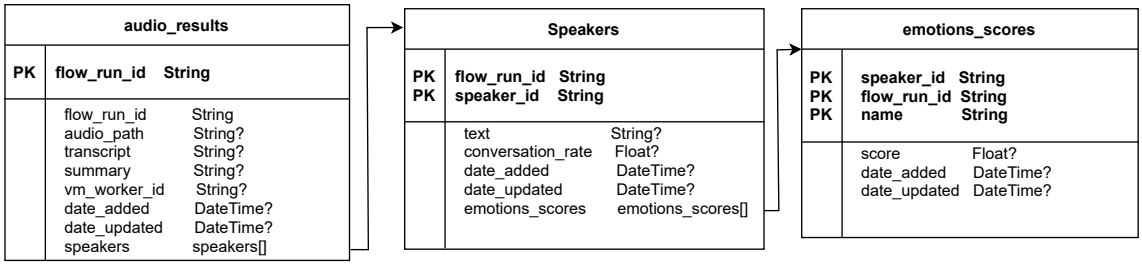

Figure 2: Database Models for the analyse

6 System Architecture

Figure 3 illustrates all the services within the system.

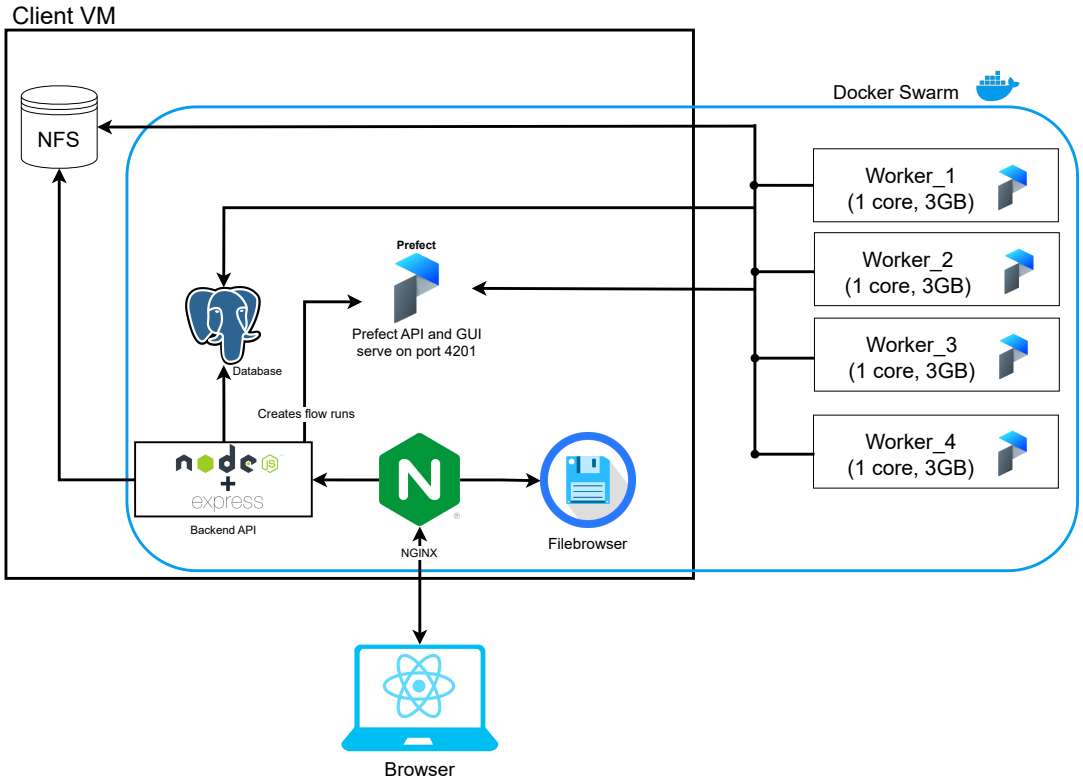

Figure 3: System Architecture

Docker Swarm

The cluster is built using the Docker Swarm network. Docker Swarm orchestrates the cluster to ensure the proper functioning of all containers and updates them as new releases are published to Docker Hub. It also facilitates connectivity between all VMs within the same network. Docker Swarm assigns roles to VMs by labeling them, designating the initiator as the manager and others as workers.

Client VM (Manager)

The Client VM acts as the central component of the software, hosting all services except the Prefect workers in the distributed system:

- **Prefect**: The Prefect GUI, accessible from the Prefect server container, provides the API for orchestrating workflows. The GUI is accessible at `<hostname>:4201` and the API is accessible at `<hostname>:4201/api`.

- **Filebrowser**: This service allows users to drag and drop audio files into the `new_audios` folders.
- **Backend API**: The backend API checks for new audio files to process and then creates new Prefect flow runs in the Prefect queue.
- **NGINX**: Acting as the entry point to the platform for the user, NGINX provides a simple frontend to display results and acts as a reverse proxy for the file browser and backend Express.js API.
- **Postgres database**: This database stores the transcript and results of a pipeline, along with additional metadata such as creation and update date of a record.

Cluster VMs (Workers)

Each VM in the cluster has a speech analyzer deployment ready to run. When a new task enters the queue, the first Worker to retrieve it starts a new flow run and processes it accordingly. Access to the required files is facilitated by a mounted directory on NFS hosted on the client VM.

7 Cloud Infrastructure Setup

Ansible scripts are used to install the necessary dependencies to run the platform. Using Docker containers offers the advantage of reduced installation requirements, as most dependencies are already included in the Docker images.

However, managing Docker images and storing files requires a significant amount of data storage. Therefore, a network file system is mounted on the client VM for file sharing, and each VM in the cluster has additional storage for downloading Docker images.

The installation steps are outlined in Figure 4, where the flow can be understood by following the ascending numbers on the directional arrows. The 'Host VM' refers to a low resource machine from which the installation is initiated, while the 'Cluster machines' (Client + Cluster VMs) are the machines on which the installation is performed.

Common Role: The common role will mount the extra storage, install the Docker service, download the GitHub repo, and finally pull the Docker images. This last step may take some time as the Prefect image is quite large (10GB).

NFS Role: As demonstrated in the COMP2039 lectures, installing NFS is fairly straightforward. All VMs will have a directory mounted to the client VM at `/mnt/data/shared`.

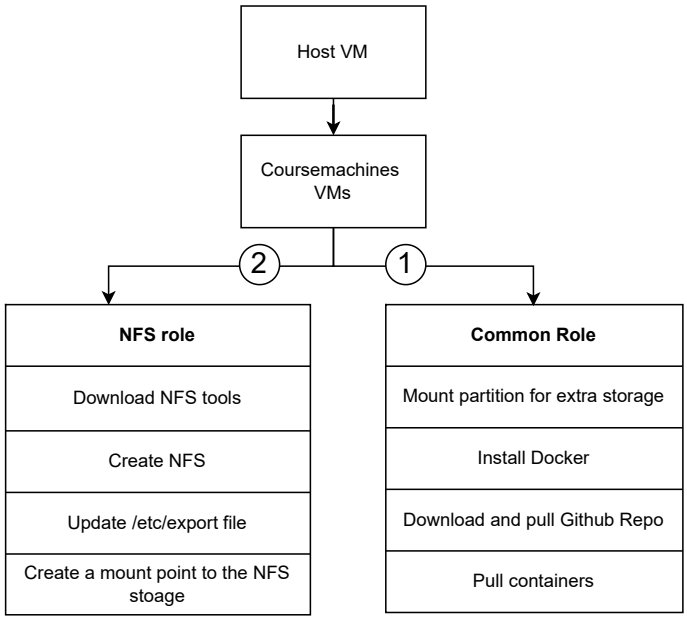

Figure 4: Ansible roles and tasks

8 Task Distribution

8.1 Publisher - Subscriber model

The distributed system works on a publisher/subscriber model. Prefect Workers listen to the Prefect server API for new flow runs in the queue. Task distribution is handled by the first available worker. While each worker can handle multiple pipelines simultaneously, it's preferable to limit them to one task at a time due to memory constraints on the cluster VMs.

8.2 Adding Tasks and Collating Results

8.2.1 Adding Files for Analysis

To add a new file for analysis, users can add the audio file to the `"/mnt/data/shared/new_audios"` folder. They can either copy the file directly into the folder using a terminal, or use the filebrowser GUI, accessible at `www.<my_domain>.com/filebrowser` (username: admin, password: admin) to drag and drop the file. The React front-end application for the platform has a redirect button that can take users directly to the correct folder on the filebrowser GUI.

8.2.2 Starting pipelines

If there are audio files in the `new_audios` folder, users can initiate processing of these files by clicking the green "Start processing files" button on the frontend React app. Behind the scenes, this action triggers a POST API call to the backend at the URL `POST /v1/pipelines/start-processing`. This tells the backend to create flow runs for Prefect and add them to its queue.

8.2.3 Saving and Viewing Results

All temporary and final results are stored in the `pipelines_audios` folder, following the naming convention `"<timestamp>_<audio_file_name>"`. A timestamp prefix is added to avoid overwriting folders. Users can view a brief summary of the results in the front-end React app and download a JSON file containing the data by clicking on the displayed download button.

9 Monitoring and Logging

The platform has monitoring capabilities for administrators and users to track VM metrics and the current status of pipelines.

9.1 Hardware logging

Figure 5 illustrates the system architecture required to monitor all the hardware components of the platform. Each VM runs CADvisor and node_exporter containers to expose resource usage data to the Prometheus database. The Prometheus database and the Grafana platform are hosted on the client VM. Prometheus is configured to collect VM information, while Grafana uses a template dashboard from the official website to visualise this data. The Grafana platform is accessible through port 4200 on the client VM (username: admin, password: admin).

9.2 Flow Run Logging

All pipeline logs are accessible through the Prefect UI. The Prefect UI landing page provides a quick overview of the overall pipeline status. Figure 6 shows in a glance an example of all flow runs status. By clicking on a specific flow run in the flow runs page, users can view all associated logs and a graphical representation of the pipeline stages using a Directed Acyclic Graph (DAG).

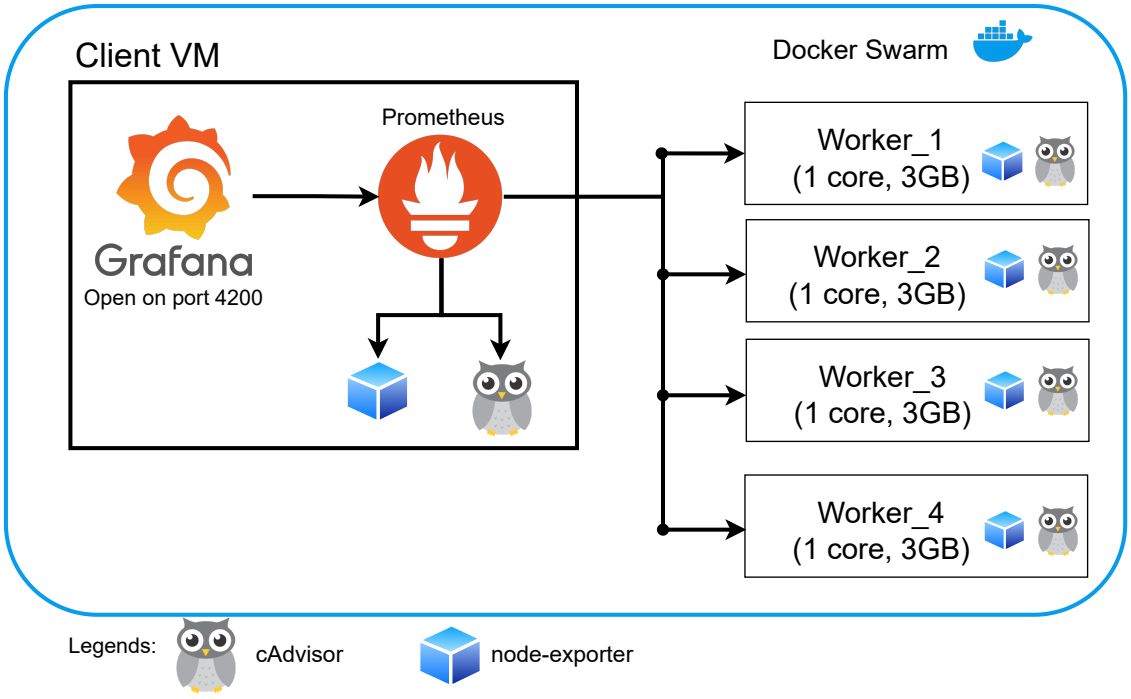

Figure 5: System Architecture for monitoring VMs

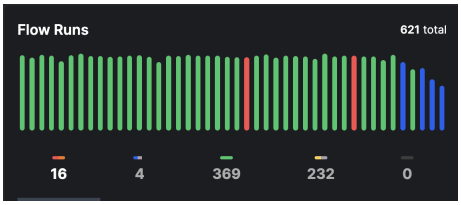

Figure 6: Prefect Flow runs overview

10 Platform Deployment

10.1 Push Docker Images with GitHub Workflows

To automate the platform further, a GitHub workflow has been developed to build and push Docker images to the david712 Docker Hub repository (https://hub.docker.com/u/david712). Developers with the appropriate permissions can insert their credentials into GitHub secrets, enabling them to build and push the latest versions.

10.2 Deploy the application

Setup and startup guides are available in the project's GitHub repository in the docs folder. They provide "copy/paste" instructions for running the platform. These guides assume that the developer already has access to the VMs and that the host VM has already connected to the other VMs using an SSH key.

11 Testing

You can test the pipeline by analysing a sample audio file provided directly in the GitHub repository. Testing is crucial to ensure that the pipeline works from start to finish without crashing. However, it does not guarantee the quality of the results.

The pipeline test can be run locally, but it has also been integrated as a GitHub workflow. This integration ensures that every time you push to the main branch, the pipeline is tested to confirm the continuous integration of the project by displaying a small green arrow next to the commit message on Github, as shown in Figure 7.

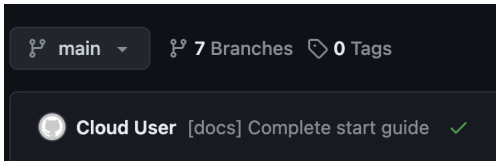

Figure 7: Github Actions on main branch

12 Challenges and Trade-offs

12.1 Hardware

The provided hardware for the project is considered sufficient for running the pipeline. Table 1 outlines the specifications of the available VMs.

	Quantity	Cores	RAM	Swap Memory	Disc Storage
Host VM	1	1	7.4G	0G	9.4G
Client VM	1	4	7.4G	0G	220.7G
Cluster VM	4	2	3.5G	0G	108G

Table 1: Hardware Specifications

12.2 No reverse-proxy for Prefect and Grafana

For simplicity's sake, Prefect and Grafana have not been placed behind NGINX as a reverse proxy. Furthermore, attempts to install Prefect behind a reverse proxy have proven unsuccessful, with the Prefect maintainers is currently adressing the issue on GitHub[2].

12.3 Filebrowser and frontend improvement

Allowing direct folder access via a browser raises significant security concerns. While this approach makes it easier to add files to folders, it exposes sensitive data to potential hacking. Alternatively, implementing a React front-end component to pass audio files to the ExpressJS backend for storage in the `new_audios` folder could improve security.

12.4 NFS and Beegfs

Initially, Beegfs was plan to provide shared data storage across VMs, with the host VM acting as the manager and other VMs acting as storage nodes to provide a storage capacity up to 500 GB. However, problems arose, particularly during beegfs management service restarts, which made the shared data storage unstable and occasionally inaccessible. As a result, a simpler (if slightly less secure) approach was adopted using NFS storage. The client VM was use to be mounted. for the accessibility across the cluster. Revised Ansible files for Beegfs installation for the project based on the lectures class can be found from this commit[3].

13 Results of the deployment run test

A free audio files databank was used to test the platform. It contains 207 audio files of conversation between two persons. The average conversation length is 26.18 minutes and the average time of a single audio file run is 9 minutes and 8 seconds. To ensure that the distributed system will work for at least 24 hours, the audio files were submitted to the Prefect API in batches of the total database data 3 times. Hence, there are 624 files to be processed.

It's important to note that the artificial nature of the generated conversations may affect the accuracy of the scoring and that the conversations does not represent real interactions between customer service and customers.

It's also important to remember that the primary goal of this project is to develop a stable distributed system with machine learning tools that can run continuously for long periods of time, even in the event of flow failures during

[2] `https://github.com/PrefectHQ/prefect/issues/11472`

[3] `https://github.com/davidfrisch/cw_data_eng_2/tree/e6a6c321bba400486723151b634e064c5f906c82/ansible/roles`

processing. Therefore, while the accuracy of the scoring may be affected by the nature of the conversations, the focus remains on the stability and resilience of the system.

14 Future Improvements

Several improvements can be made to improve the user experience and stability of the platform.

One possible improvement is to replace NFS with Beegfs, which offers increased storage capacity and a more secure approach to connecting to the storage network, requiring a secret file for authentication.

Another improvement could be to remove the filebrowser service and use only the React front-end application for adding new audio files. This consolidation simplifies the user interface and reduces potential security vulnerabilities associated with direct folder access.

Finally, while Docker Swarm provides a simple method for creating a network cluster between VMs, a more professional approach would involve moving to a Kubernetes cluster for improved orchestration and monitoring capabilities.

15 GitHub repository

The source code for the project is accessible on GitHub at the following link: `https://github.com/davidfrisch/cw_data_eng_2`. The repository includes all the source code, installation instructions, system startup guidelines, and examples for using the system from a python client

16 Conclusion

In summary, this project has covered several critical aspects of data engineering, including infrastructure construction using programming tools, development of fully distributed pipelines, shared data storage, deployment, testing and monitoring. Although the quality of the results is not adequate, the system has demonstrated resilience and potential for the development of more realistic pipelines.