BRAUDE
College of Engineering, Karmiel

AudioWave
Maintenance guide

Submitted in partial fulfillment for the degree of

Bachelor of Science

in

Software Engineering

for

The Software Engineering Department

at

Braude - College of Engineering, Karmiel

By

Ward Zidani

Ahmad Bsese

Under guidance of

Zeev Frenkel

Project code

24-1-D-44

Summer 2024

1. Maintenance Guide: AudioWave Platform

The Maintenance Guide is designed to ensure that the AudioWave platform remains operational, adaptable, and updatable after the initial development phase. This guide outlines the required system environment, custom software installation steps, and procedures for making future updates and improvements to the platform.

1.1. System Environment

Software Infrastructure:

- **API Gateway**: Nginx gateway is used to manage and route API requests.
- **Microservices**: Developed using .NET Core and deployed within Docker containers.
- **Message Broker**: RabbitMQ is used for communication between services.
- **Databases**:
  - **SQL Server**: Manages user data and audio metadata.
  - **MongoDB**: Manages playlist data and associated content.

Hardware Requirements:

- **Server Environment**: Virtual Private Cloud (VPC) with at least 4 vCPUs and 8GB of RAM for each instance.
- **Storage**: Storage requirements are minimal due to the use of Amazon S3 for audio file storage, but the server should have at least 100GB for system logs and temporary data.
- **Network**: High-speed network connection for handling large traffic volumes and real-time streaming.

1.2. Installation Instructions

1.2.1. Pre-requisites

Before installing the custom software components of AudioWave, ensure that the following system dependencies are installed:

- **Docker**: Docker is required to run the containerized microservices.
- **Docker Compose**: For orchestrating the deployment of multiple containers.
- **RabbitMQ**: Installed and configured for message queuing between services.
- **nginx Gateway**: Installed for API routing and reverse proxy.

1.2.2. Installation Steps for AudioWave

Clone the Repository:
Clone the AudioWave repository from the official GitHub repository
git clone from https://github.com/WardZid/AudioWave.git

1.2.3. Set Up Docker:

- Ensure Docker is installed and running on your system. Verify the installation by running:

```
docker --version
```

1.2.4. **Database Configuration:**

Set up the SQL Server and MongoDB instances by editing the `.env` file in the project directory to point to your database credentials and URLs:

```
SQL_SERVER_CONNECTION_STRING=<your-sql-connection>
MONGO_DB_CONNECTION_STRING=<your-mongo-connection>
```

1.2.5. **Run Docker Compose:** Navigate to the project directory and run the following command to start all services:
docker-compose up -d

1.2.6. **Set Up API Gateway:** The nginx Gateway configure Nginx to act as a reverse proxy for routing requests to the respective microservices by editing the `/etc/nginx/nginx.conf` file.

1.2.7. **Credential Encryption Key Generator Application**:

- Set up the external application responsible for generating encryption keys. Ensure MongoDB is configured to securely store these keys.

sudo apt-get install nginx

1.2.8. **Run the Application:** Once all services are running, the platform can be accessed through the web and mobile clients by pointing them to the API Gateway's public IP or domain.

1.3. Applying Updates and Improvements

1.3.1. **System Updates**: To ensure the system remains up-to-date with bug fixes and new features, follow these steps for applying updates:
Pull Latest Changes:

- Periodically check the repository for updates and pull the latest changes:
  git pull origin main
-
-

1.3.2. **Rebuild Docker Containers:** After pulling new changes, it is necessary to rebuild and restart the Docker containers

```
docker-compose down
docker-compose build
docker-compose up -d
```

1.3.3. **Database Migrations**: If the new updates involve database schema changes, run the provided migration scripts to update the databases:

1.4. Adding New Features

The AudioWave platform was built with flexibility in mind, allowing for the easy addition of new features. Here is the general process for adding a new microservice or functionality:

- **Create a New Microservice:**
  - Develop the new feature as an independent microservice, following the same .NET Core and Docker setup as the existing services. Ensure that it fits into the existing architecture (user management, playlists, audio metadata, etc.).
- **Update API Gateway:**
  - Configure Kong Gateway to route traffic to the new microservice by editing the `kong.conf` file and adding the necessary routes:
    kong route create --service <service_name> --hosts <hostname>

    Test the New Feature:

- Thoroughly test the new feature using unit tests, integration tests, and functional tests to ensure it interacts with the other microservices correctly.

    Deploy the Service:

- Use Docker to containerize the new service and add it to the `docker-compose.yml` file. This will ensure the service is deployed alongside the other services in the platform.

1.5. Maintenance Tips

- **Monitoring:**
  Set up a monitoring solution like Prometheus or Grafana to keep track of system performance, database health, and message broker activity.
- **Log Management:**
  Enable centralized logging using tools like ELK Stack (Elasticsearch, Logstash, and Kibana) or Splunk to collect logs from all services in one place for easier debugging and analysis.
- **Regular Backups:**
  Schedule regular backups for both SQL Server and MongoDB databases to prevent data loss in case of unexpected failures. Utilize Amazon S3 for backing up audio files and metadata.

Reference:

1. Docker Documentation:

Docker, Inc. (n.d.). Docker Documentation. Retrieved from
https://docs.docker.com

2. RabbitMQ Documentation:

Pivotal Software. (n.d.). RabbitMQ Documentation. Retrieved from
https://www.rabbitmq.com/documentation.html

3. Nginx Documentation:

Nginx, Inc. (n.d.). Nginx Documentation. Retrieved from https://nginx.org/en/docs/