AudioMuse-AI: Full Stack Local Deployment Guide with Docker Compose on Debian

This guide will walk you through deploying your complete AudioMuse-AI application stack on a Debian-based local machine using Docker Compose. This approach is ideal if you don't have a Kubernetes environment set up and want to run all services (Flask app, RQ worker, Redis, PostgreSQL) using pre-built Docker images, simplifying the setup process.

Prerequisites

Before you begin, ensure your Debian machine meets the following requirements:

- **Debian Operating System**: This guide is tailored for Debian 10 (Buster), Debian 11 (Bullseye), or Debian 12 (Bookworm).
- **Internet Connection**: Required to download Docker and container images.
- **Sudo Privileges**: You'll need `sudo` access to install software.

Step-by-Step Deployment

Step 1: Install Docker

First, you need to install Docker Engine on your Debian machine.

1. **Update your system's package index:**

```
sudo apt update
```

Listing 1: Update apt package index

2. **Install necessary packages to allow `apt` to use a repository over HTTPS:**

```
sudo apt install ca-certificates curl gnupg lsb-release -y
```

Listing 2: Install HTTPS transport packages

3. **Add Docker's official GPG key:**

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/
    keyrings/docker.gpg
```

Listing 3: Add Docker GPG key

4. **Set up the Docker repository:**

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://
      download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Listing 4: Configure Docker repository

5. **Install Docker Engine, containerd, and Docker Compose:**

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose
    -plugin -y
```

Listing 5: Install Docker components

6. **Verify Docker installation:**

```
sudo docker run hello-world
```

Listing 6: Verify Docker

You should see a message confirming Docker is working.

7. **Add your user to the `docker` group (optional, but recommended to run Docker without `sudo`):**

```
sudo usermod -aG docker $USER
newgrp docker # Apply group changes immediately, or log out and back in
```

Listing 7: Add user to docker group

Step 2: Create Project Directory and Docker Compose File

Instead of cloning the repository, you'll create a local directory and the `docker-compose.yaml` file within it.

1. **Create a directory for your project:**

```
mkdir ~/audiomuse-ai
cd ~/audiomuse-ai
```

Listing 8: Create project directory

2. **Create the Docker Compose file (`docker-compose.yaml`):** Create a file named `docker-compose.yaml` (no extension) in your `audiomuse-ai` project directory and paste the following content into it. This file defines all the services needed for your application, using the pre-built Docker image.

```
version: '3.8'

services:
  # Redis service for RQ (task queue)
  redis:
    image: redis:7-alpine
    container_name: audiomuse-redis
    ports:
      - "6379:6379" # Expose Redis port to the host
    volumes:
      - redis-data:/data # Persistent storage for Redis data
    restart: unless-stopped

  # PostgreSQL database service
  postgres:
    image: postgres:15-alpine
    container_name: audiomuse-postgres
    environment:
      POSTGRES_USER: "audiomuse"
      POSTGRES_PASSWORD: "audiomusepassword"
      POSTGRES_DB: "audiomusedb"
    ports:
      - "5432:5432" # Expose PostgreSQL port to the host
    volumes:
```

```
      - postgres-data:/var/lib/postgresql/data # Persistent storage for PostgreSQL data
    restart: unless-stopped

  # AudioMuse-AI Flask application service
  audiomuse-ai-flask:
    image: ghcr.io/neptunehub/audiomuse-ai:0.2.0-alpha # Your pre-built image
    container_name: audiomuse-ai-flask-app
    ports:
      - "8000:8000" # Map host port 8000 to container port 8000
    environment:
      SERVICE_TYPE: "flask" # Tells the container to run the Flask app
      JELLYFIN_USER_ID: "0e45c44b3e2e4da7a2be11a72a1c8575" # From jellyfin-credentials
          secret
      JELLYFIN_TOKEN: "e0b8c325bc1b426c81922b90c0aa2ff1" # From jellyfin-credentials secret
      JELLYFIN_URL: "http://jellyfin.192.168.3.131.nip.io:8087" # From audiomuse-ai-config
          ConfigMap
      DATABASE_URL: "postgresql://audiomuse:audiomusepassword@postgres:5432/audiomusedb" #
          Connects to the 'postgres' service
      REDIS_URL: "redis://redis:6379/0" # Connects to the 'redis' service
      TEMP_DIR: " temp_audio"
    volumes:
      - temp-audio-flask: temp_audio # Volume for temporary audio files
    depends_on:
      - redis
      - postgres
    restart: unless-stopped

  # AudioMuse-AI RQ Worker service
  audiomuse-ai-worker:
    image: ghcr.io/neptunehub/audiomuse-ai:0.2.0-alpha # Your pre-built image
    container_name: audiomuse-ai-worker-instance
    environment:
      SERVICE_TYPE: "worker" # Tells the container to run the RQ worker
      JELLYFIN_USER_ID: "0e45c44b3e2e4da7a2be11a72a1c8575" # From jellyfin-credentials
          secret
      JELLYFIN_TOKEN: "e0b8c325bc1b426c81922b90c0aa2ff1" # From jellyfin-credentials secret
      JELLYFIN_URL: "http://jellyfin.192.168.3.131.nip.io:8087" # From audiomuse-ai-config
          ConfigMap
      DATABASE_URL: "postgresql://audiomuse:audiomusepassword@postgres:5432/audiomusedb" #
          Connects to the 'postgres' service
      REDIS_URL: "redis://redis:6379/0" # Connects to the 'redis' service
      TEMP_DIR: " temp_audio"
    volumes:
      - temp-audio-worker: temp_audio # Volume for temporary audio files
    depends_on:
      - redis
      - postgres
    restart: unless-stopped

# Define volumes for persistent data and temporary files
volumes:
  redis-data:
  postgres-data:
  temp-audio-flask: # Volume for Flask app's temporary audio
  temp-audio-worker: # Volume for Worker's temporary audio
```

Listing 9: docker-compose.yaml

Important Notes for your `docker-compose.yaml`:

- **Image Source**: The `image` directive now points directly to `ghcr.io/neptunehub/audiomuse-ai:0.2.0-alpha`, meaning Docker will pull this pre-built image instead of building it locally.
- **Environment Variables**: I've directly included the values from your Kubernetes `Secret` and

`ConfigMap` into the `environment` section. For a production environment, you might prefer using a `.env` file for sensitive credentials.

- **Service Communication**: Services within a Docker Compose network can communicate using their service names (e.g., `postgres` for the PostgreSQL service, `redis` for the Redis service). This is why `DATABASE_URL` and `REDIS_URL` refer to `postgres:5432` and `redis:6379` respectively.
- `command` **Removed**: The `command` override for `audiomuse-ai-flask` and `audiomuse-ai-worker` has been removed. The `CMD` instruction within the pre-built Docker image (which uses the `SERVICE_TYPE` environment variable) will now correctly determine whether to run `app.py` or `rq_worker.py`.
- **Volumes**: Persistent volumes (`redis-data`, `postgres-data`) are defined to ensure your Redis data and PostgreSQL data are not lost when containers are stopped or removed. `emptyDir` equivalents (`temp-audio-flask`, `temp-audio-worker`) are used for temporary audio storage.
- `depends_on`: This ensures that `redis` and `postgres` services are started before the Flask app and RQ worker, as they depend on these services.

Step 3: Run the Docker Compose Stack

Once your `docker-compose.yaml` is in place, you can pull the images and run all services with a single command. Make sure you are in the directory where you created `docker-compose.yaml` (`~/audiomuse-ai`).

```
docker compose up -d
```

Listing 10: Run Docker Compose stack

- `docker compose up`: The command to start and run the services defined in `docker-compose.yaml`.
- `-d`: Runs the containers in "detached" mode (in the background).

This process will download the necessary Docker images (Redis, PostgreSQL, and your AudioMuse-AI image) and then start all the services. It might take a few minutes, depending on your internet speed.

Step 4: Access AudioMuse-AI

Your AudioMuse-AI application services should now be running in Docker containers.

(a) **Check container status:**

```
docker compose ps
```

Listing 11: Check container status

You should see `audiomuse-redis`, `audiomuse-postgres`, `audiomuse-ai-flask-app`, and `audiomuse-ai-worke` listed with status `running`.

(b) **View logs (optional, for debugging):** To see the combined logs of all services:

```
docker compose logs -f
```

Listing 12: View combined logs

To see logs for a specific service (e.g., the Flask app):

```
docker compose logs -f audiomuse-ai-flask
```

Listing 13: View specific service logs

(c) **Access the application:** Open your web browser and navigate to:

`http://localhost:8000`

Managing Your Docker Compose Stack

Here are some useful Docker Compose commands for managing your application stack:

- **Stop all services (without removing containers):**

```
docker compose stop
```

Listing 14: Stop all services

- **Start all services (if stopped):**

```
docker compose start
```

Listing 15: Start all services

- **Stop and remove all services, networks, and volumes (clean up):**

```
docker compose down -v
```

Listing 16: Stop and remove all services

  - `-v`: Removes volumes as well. Use this if you want a fresh start and don't care about persistent data (Redis, PostgreSQL).

- **Restart all services:**

```
docker compose restart
```

Listing 17: Restart all services

- **Pull updated images and restart services (if a new version of your image is released):**

```
docker compose pull && docker compose up -d
```

Listing 18: Pull and restart services

- **Scaling the RQ Worker (e.g., to run 2 workers):** To ensure proper functionality, your AudioMuse-AI application requires a minimum of **two** RQ worker instances. One worker is typically dedicated to handling the main tasks, while another handles any subtasks or child processes that might be spawned. Running with fewer than two workers might lead to tasks getting stuck or the application not functioning as expected.
  To run two (or more) instances of the `audiomuse-ai-worker` service, use the `--scale` flag with `docker compose up`. This will create additional containers for the specified service.
  First, ensure your existing services are stopped or down:

```
docker compose down
```

Listing 19: Stop existing services

  Then, start all services and scale the worker:

```
docker compose up -d --scale audiomuse-ai-worker=2
```

Listing 20: Scale worker service

  You can replace `2` with any higher number of worker instances if you need more processing power. To verify, run `docker compose ps` and you should see multiple `audiomuse-ai-worker-instance` containers.

This updated guide provides a comprehensive and robust way to deploy your full AudioMuse-AI application locally using Docker Compose, directly pulling the pre-built image and simplifying the setup process significantly.