**Mini-Projet DevOps**

**Team Members: ​**

Abidellaoui Mariem​

Baccar Hamdi​

Khadraoui Feryel​

**​Année universitaire : 2024-2025**

**Table des matières**

[Introduction 3](#_Toc184874378)

[1. Objectifs principaux 3](#_Toc184874379)

[Dataset et Modèles de Machine Learning 4](#_Toc184874380)

[1. Description de la Dataset : GTZAN 4](#_Toc184874381)

[2. Présentation des Modèles 4](#_Toc184874382)

[Prétraitement des Données et Codage Base64 5](#_Toc184874383)

[Architecture du Projet 7](#_Toc184874384)

[1. Description de l’Architecture Globale 7](#_Toc184874385)

[2. Conteneurs Docker 7](#_Toc184874386)

[3. Rôle de Chaque Composant dans le Projet 8](#_Toc184874387)

[4. Interactions dans le Projet 9](#_Toc184874388)

[Implémentation Technique 10](#_Toc184874389)

[Conteneurs Docker 10](#_Toc184874390)

[Docker-Compose 10](#_Toc184874391)

[Intégration Continue avec Jenkins 11](#_Toc184874392)

[Conclusion 12](#_Toc184874393)

**Table des figures**

[Figure 1 Jenkins Pipeline 12](#_Toc184874420)

# Introduction

La classification de genres musicaux est un domaine fascinant qui combine les avancées en Machine Learning et le traitement du signal audio. Dans le cadre de ce projet, nous avons développé un environnement Machine Learning pour classifier les genres musicaux à partir d'un fichier audio au format WAV. Ce projet repose sur des modèles d'apprentissage automatique (SVM et VGG19) et est encapsulé dans des conteneurs Docker pour garantir une isolation et une reproductibilité optimales.

## Objectifs principaux

Ce projet a pour objectifs :

1. Intégrer deux modèles de classification de genres musicaux dans des services Flask distincts :
   * **SVM\_service** : un service utilisant un modèle de Machine Learning basé sur le SVM.
   * **VGG19\_service** : un service exploitant un réseau de neurones convolutifs pré-entraîné, le VGG19.
2. Orchestrer ces services à l'aide de Docker pour simplifier leur exécution et leur interaction.
3. Créer un front-end permettant de tester ces services via une interface utilisateur intuitive.

# Dataset et Modèles de Machine Learning

## Description de la Dataset : GTZAN

Le dataset **GTZAN** est un ensemble de données bien connu pour les tâches de classification des genres musicaux. Il contient :

* **Nombre de fichiers** : 1000 fichiers audio au format WAV.
* **Genres** : 10 genres musicaux différents (Blues, Classical, Country, Disco, Hip-Hop, Jazz, Metal, Pop, Reggae, Rock), chacun représenté par 100 fichiers.
* **Durée des fichiers** : Chaque fichier dure 30 secondes.
* **Taille totale** : Environ 1,2 Go.

Ce dataset est utilisé comme point de départ pour entraîner et évaluer des modèles de classification musicale grâce à sa richesse et son équilibrage entre les classes.

## Présentation des Modèles

Dans ce projet, deux modèles de classification, **SVM** et **VGG19**, sont implémentés dans des microservices séparés. L'utilisateur peut choisir dynamiquement l'un des deux modèles pour prédire le genre musical d'un fichier audio.

**1. Service basé sur Support Vector Machine (SVM)**

Le SVM est un algorithme de Machine Learning supervisé, efficace pour des problèmes de classification.

* **Fonctionnement** :
  + Les fichiers audio sont prétraités pour extraire des caractéristiques acoustiques (par exemple, les coefficients MFCC).
  + Le modèle SVM classe ensuite les données en traçant une frontière optimale entre les genres musicaux dans un espace multidimensionnel.
* **Avantages** :
  + **Performance rapide** : Convient pour des fichiers audio avec des caractéristiques extraites.
  + **Classification robuste** : Surtout pour des données avec des caractéristiques bien définies comme celles du dataset GTZAN.
* **Utilisation dans le microservice** :
  + Le fichier audio est encodé en **Base64** avant d'être transmis au microservice via une API Flask.
  + Le modèle traite les données encodées et retourne le genre musical prédit.

**2. Service basé sur le réseau de neurones convolutif VGG19**

Le modèle VGG19 est un réseau de neurones convolutifs (CNN) pré-entraîné conçu pour la classification. Ici, il est adapté pour traiter des fichiers audio transformés en spectrogrammes.

* **Fonctionnement** :
  + Les fichiers audio sont convertis en spectrogrammes, qui représentent visuellement les fréquences sonores dans le temps.
  + Ces spectrogrammes sont fournis en entrée au modèle VGG19, qui prédit le genre musical.
* **Avantages** :
  + **Apprentissage profond** : Identifie des motifs complexes dans les données, ce qui améliore la précision des prédictions.
  + **Transfert d’apprentissage** : Utilise un modèle pré-entraîné pour accélérer le développement et améliorer les performances.
* **Utilisation dans le microservice** :
  + Le fichier audio est également encodé en **Base64**, puis transmis au service Flask.
  + Le spectrogramme est généré à partir des données encodées et traité par le modèle pour produire la prédiction.

## Prétraitement des Données et Codage Base64

1. **Prétraitement** :
   * Les fichiers audio sont d'abord analysés pour extraire leurs caractéristiques acoustiques (par exemple, MFCC pour SVM, spectrogrammes pour VGG19).
   * Ces étapes garantissent que les données sont compatibles avec les exigences des deux modèles.
2. **Codage Base64** :
   * Avant la transmission au microservice, chaque fichier audio est **encodé en Base64**.
   * Ce format d'encodage permet de transformer le contenu binaire d'un fichier WAV en une chaîne de caractères ASCII. Cela facilite l'envoi du fichier audio via des requêtes HTTP.
   * Une fois reçu, le service décode les données Base64 pour les traiter avec le modèle approprié.
3. **Transmission des Données** :
   * Le fichier encodé est transmis en tant que paramètre via une API RESTful.
   * Ce processus standardisé assure une communication fluide entre le front-end et les microservices Flask.

Grâce à cette approche, le système permet une transmission fiable des fichiers audio et une prédiction flexible via les deux microservices indépendants. L'encodage Base64 joue un rôle clé en assurant la portabilité et la compatibilité des fichiers audio dans un environnement distribué.

# Architecture du Projet

## Description de l’Architecture Globale

Le projet repose sur une architecture conteneurisée orchestrée à l'aide de Docker et Docker-Compose. Chaque fonctionnalité est encapsulée dans un conteneur dédié pour assurer modularité et maintenabilité. Voici une vue d'ensemble des principaux conteneurs et leurs rôles.

## Conteneurs Docker

1. **SVM\_Service**
   * **Rôle** : Fournit un service web Flask qui utilise un modèle SVM pour prédire le genre musical d'un fichier audio.
   * **Fonctionnement** :
     + Reçoit un fichier WAV encodé en Base64.
     + Décode le fichier, extrait les caractéristiques acoustiques (MFCC) et utilise le modèle SVM pour prédire le genre.
     + Retourne le genre musical prédit au client sous forme de réponse JSON.
2. **VGG19\_Service**
   * **Rôle** : Implémente un service Flask utilisant le modèle VGG19 pour prédire le genre musical à partir d’un fichier audio.
   * **Fonctionnement** :
     + Accepte un fichier audio encodé en Base64.
     + Transforme le fichier en spectrogramme, qui est ensuite traité par le modèle VGG19.
     + Retourne la prédiction du genre musical au client sous format JSON.
3. **Front-End**
   * **Rôle** : Fournit une interface utilisateur pour interagir avec les services de classification.
   * **Fonctionnalités** :
     + Permet à l'utilisateur de choisir entre SVM ou VGG19 comme méthode de classification.
     + Charge un fichier audio, l’encode en Base64 et l’envoie au service sélectionné.
     + Affiche le genre musical prédit à l’utilisateur.
   * **Technologies** : Réalisé avec Angular.
4. **Service d'Orchestration (Docker-Compose)**
   * **Rôle** : Coordonne le démarrage, l’interconnexion et la gestion des conteneurs.
   * **Fonctionnalités** :
     + Définit les dépendances réseau entre les conteneurs.
     + Gère les volumes partagés si nécessaire.

## Rôle de Chaque Composant dans le Projet

|  |  |  |
| --- | --- | --- |
| **Composant** | **Rôle** | **Technologies Utilisées** |
| SVM\_Service | Classification des genres musicaux à l’aide du modèle SVM. | Flask |
| VGG19\_Service | Classification des genres musicaux à l’aide du modèle VGG19. | Flask |
| Front-End | Interface utilisateur pour choisir la méthode et visualiser les résultats. | Angular |
| Docker-Compose | Orchestration des conteneurs, gestion des dépendances réseau. | Docker |

## Interactions dans le Projet

1. L’utilisateur interagit avec le **front-end** pour charger un fichier audio et choisir le modèle (SVM ou VGG19).
2. Le fichier est encodé en Base64 et envoyé via une requête HTTP au microservice choisi.
3. Le microservice (SVM ou VGG19) traite le fichier et renvoie la prédiction au **front-end**.
4. **Docker-Compose** garantit la communication entre les conteneurs.

Cette architecture modulaire assure une séparation claire des responsabilités, facilitant la maintenance et l’évolution future du projet.

# Implémentation Technique

## Conteneurs Docker

Les services du projet sont encapsulés dans des conteneurs Docker pour garantir une isolation et une exécution indépendante. Voici les descriptions des Dockerfiles pour chaque service :

1. **Dockerfile pour SVM\_Service** Ce fichier configure le conteneur dédié au service de classification utilisant le modèle SVM. Il installe les dépendances nécessaires, copie les fichiers du projet et expose un port pour l'API Flask. Le service est conçu pour recevoir des fichiers audio encodés en Base64 et retourner le genre musical prédit.
2. **Dockerfile pour VGG19\_Service** Similaire au SVM\_Service, ce fichier configure un conteneur pour le modèle VGG19. Ce conteneur permet de traiter des fichiers audio encodés en Base64, en exploitant la puissance des réseaux de neurones convolutifs pour prédire le genre musical.
3. **Dockerfile pour le Front-End** Ce fichier est divisé en deux étapes :
   * Une étape de build pour compiler le projet Angular.
   * Une étape de déploiement qui utilise Nginx pour servir les fichiers statiques générés. Cela garantit une performance optimale et une distribution efficace des ressources.

## Docker-Compose

Le fichier docker-compose.yml permet d’orchestrer tous les services du projet en définissant leurs dépendances et les configurations.

* **Frontend** : Le conteneur du front-end se connecte aux services Flask pour interagir avec les modèles SVM et VGG19.
* **SVM\_Service** : Conteneur dédié au service utilisant le modèle SVM. Il est exposé sur le port 5001 et configuré pour traiter des requêtes POST avec des fichiers encodés en Base64.
* **VGG19\_Service** : Conteneur dédié au service utilisant le modèle VGG19, exposé sur le port 5002. Il offre une classification avancée grâce au modèle de réseau de neurones convolutifs.

## Intégration Continue avec Jenkins

Pour automatiser le processus de construction et de déploiement, un pipeline Jenkins a été mis en place. Voici les étapes principales :

1. **Récupération du Code Source** :
   * Jenkins utilise le plugin Git pour cloner le dépôt contenant les fichiers du projet.
   * Le code est récupéré depuis la branche principale du dépôt GitHub.
2. **Construction des Images Docker** :
   * Les images Docker pour les services SVM, VGG19 et Front-End sont construites automatiquement via le fichier docker-compose.yml.
3. **Déploiement des Services** :
   * Les conteneurs Docker sont lancés en arrière-plan à l'aide de la commande docker-compose up -d.

**Avantages de Jenkins** :

* Automatisation des tâches répétitives, comme la construction et le déploiement.
* Réduction des erreurs humaines grâce à un pipeline structuré.
* Gain de temps et amélioration de l'efficacité du développement.

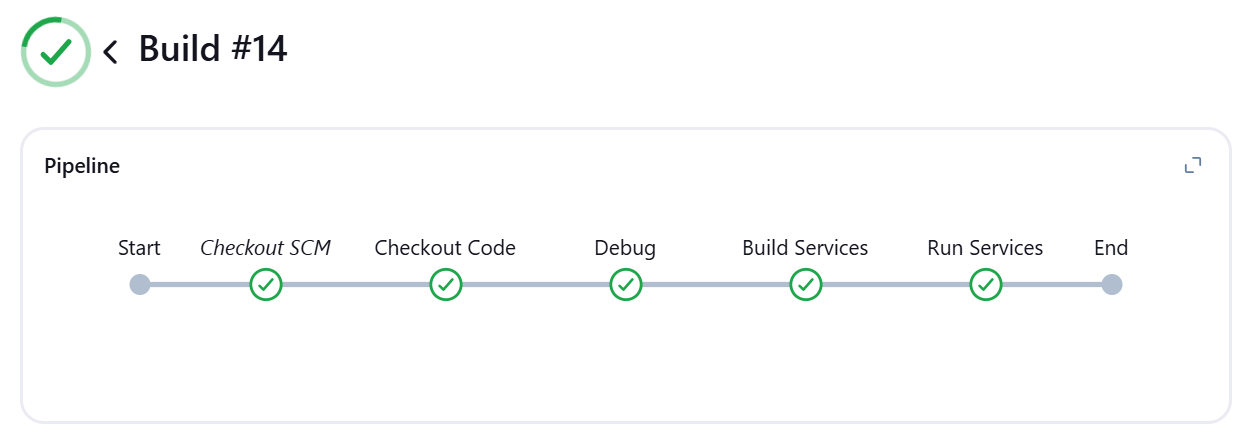


Figure 1 Jenkins Pipeline

# Conclusion

Ce projet a mis en œuvre une architecture moderne et modulaire, basée sur des conteneurs Docker et orchestrée via Docker-Compose, pour développer une solution de classification musicale. L’utilisation de microservices dédiés (SVM et VGG19) permet une séparation claire des responsabilités, offrant flexibilité et évolutivité. Le front-end, développé avec Angular et déployé via Nginx, assure une interface utilisateur réactive et accessible.

L’intégration continue avec Jenkins a permis d’automatiser les principales étapes du cycle de développement, notamment la construction, le déploiement et le lancement des services. Cette approche garantit une exécution fluide des processus tout en minimisant les erreurs humaines.

En exploitant des technologies robustes comme Flask, Angular, et Docker, le projet démontre comment des outils modernes peuvent être intégrés pour créer une solution performante et efficace. Ce système modulaire peut être facilement étendu à d’autres applications d’apprentissage automatique ou adapté à différents types de données, ce qui en fait une base solide pour des projets similaires à l’avenir.