

# Flyway 101

## Lunch and Learn

L'équipe du PaaP

Auteur : Said Mezghanni

# Plan

- ➊ Introduction
- ➋ Fonctionnalités
- ➌ Commandes
- ➍ Démonstration
- ➎ Discussion et Q&A

# Plan

- ① Introduction
- ② Fonctionnalités
- ③ Commandes
- ④ Démonstration
- ⑤ Discussion et Q&A

Flyway est un outil open-source de migration de base de données  
Le projet a commencé avec une première release en 2010.

## L'outil est disponible sous trois éditions :

- Community Edition
- Pro Edition
- Enterprise Edition

## Liste des outils offert par Flyway :

- Command-line : permet d'exécuter Flyway en ligne de commande utile pour les agents Docker.
- Maven-plugin : permet d'exécuter Flyway dans un build Maven.
- Gradle-plugin : permet d'exécuter Flyway dans des tasks Gradle.
- Java-API : permet d'exécuter Flyway à l'intérieur de code Java (fonctionnalité de migration au démarrage de l'application).
- Plugins communautaire : npm, Jenkins, SpringBoot, Quarkus,...

## Il est possible de configurer les outils Flyway par plusieurs manière :

- variables d'environnements
- fichier de config : flyway.conf
- fichier de config par migration : myscript.sql.conf
- placeholders : permet de faire du templating sur les scripts de migrations suivant des paramètres prédéfinis
- config SSL : il faut injecter des certificats dans le formats Java (keystore et truststore) a la JVM ou Flyway va s'exécuter puis configurer la connexion a la BD pour enforcer le ssl.

## Liste des bases de données supportés :

- Oracle (incl. Amazon RDS)
- MySQL (incl. Amazon RDS, Azure Database & Google Cloud SQL)
- MariaDB (incl. Amazon RDS)
- SQL Server (incl. Amazon RDS & Azure SQL Database)
- PostgreSQL (incl. Amazon RDS, Azure Database, Google Cloud SQL & Heroku)
- et plein d'autres base de données...

# Plan

① Introduction

② Fonctionnalités

③ Commandes

④ Démonstration

⑤ Discussion et Q&A



# Définitions

Avec Flyway, toutes les modifications apportées à la base de données sont appelées **migrations**.

Les migrations peuvent être :

- versionnées
- répétables

Les migrations versionnées se présentent sous 2 formes :

- régulière
- annulée

# Type de migrations

- **Les migrations versionnées** ont une version, une description et un checksum. La version doit être unique. Le checksum permet de détecter les changements accidentels. Les migrations versionnées sont appliquées dans l'ordre une seule fois.
- **Les migrations d'annulation** : en option, l'effet d'une migration régulière peut être annulé en fournissant une migration d'annulation avec la même version.
- **Les migrations répétables** ont une description et un checksum, mais pas de version. Au lieu d'être exécutés une seule fois, ils sont (ré)appliqués à chaque fois que leur checksum change.

# Langage de migration

Les migrations sont le plus souvent écrites en SQL. Ils sont généralement utilisés pour :

- Modifications DDL (CREATE/ALTER/DROP : TABLES, VIEWS, ...)
- Modifications simples des données (CRUD dans les tables)
- Modifications simples des données en masse (CRUD dans les tables)

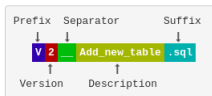
Les migrations basées sur Java ou sur des scripts Bash, Python, ... (en beta) conviennent parfaitement à toutes les modifications qui ne peuvent pas être facilement exprimées à l'aide de SQL :

- Modifications BLOB & CLOB
- Modifications avancées des données en masse (recalculs, changements de format avancés, ...)

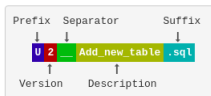
# Migrations SQL/Script

## Nommage :

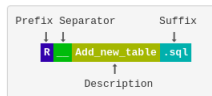
Versioned Migrations



Undo Migrations



Repeatable Migrations



## Emplacement des migrations :

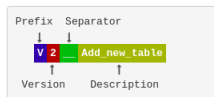
- classpath : utilisable dans les builds Java avec les plugins Maven et Gradle
- filesystem : utilisable avec le cli et les plugins
- s3/gcs : (en beta)



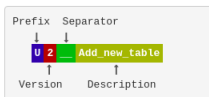
# Migrations Java

## Nommage :

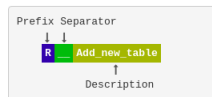
Versioned Migrations



Undo Migrations



Repeatable Migrations



## Emplacement des migrations :

- classpath : utilisable dans les builds Java avec les plugins Maven et Gradle. Il faut compiler les classes avant de lancer la migration.



# Table d'historique de schéma

Pour garder une trace des migrations qui ont déjà été appliquées, quand et par qui, Flyway ajoute une table d'historique de schéma. Il contient également les checksum, les temps et l'état d'exécutions des migrations.

flyway\_schema\_history

installed_rank	version	description	type	script	checksum	installed_by	installed_on	execution_time	success
1	1	Initial Setup	SQL	V1__Initial_Setup.sql	1996767037	axel	2016-02-04 22:23:00.0	546	true
2	2	First Changes	SQL	V2__First_Changes.sql	1279644856	axel	2016-02-06 09:18:00.0	127	true
3	2.1	Refactoring	JDBC	V2_1__Refactoring		axel	2016-02-10 17:45:05.4	251	true

## L'aspect transactionnel :

Par défaut, Flyway encapsule l'exécution de chaque migration dans une seule transaction.

Il est aussi possible de le configurer pour encapsuler l'exécution complète de toutes les migrations dans une seule transaction.

# Callbacks

Dans certaines situations, nous avons besoins d'exécuter des actions répétitives comme la recompilation des procédures, la mise-à-jour des vues matérialisées, les taches de maintenance...

Pour ces besoins, Flyway offre la possibilité de définir des scripts de Callbacks :

- **en SQL** : ils sont définits de la meme maniere que les scripts de migration. Le nom du script doit correspondre a l'événement écouté (beforeMigrate.sql, beforeEachMigrate.sql,...)
- **en Java** : il est aussi possible de programmer des Hook sur un ou plusieurs Callbacks en Java.

# Error Overrides

Lors de exécutions des migrations SQL :

- en cas d'erreur : Flyway marque la migration comme échoué et la rollback automatiquement si possible.
- en cas de warning : il est retourné.

Avec la fonctionnalité de redefinitions des erreurs(pro et enterprise), il est possible de :

- traitez une erreur comme un warning car la migration va la traiter correctement plus tard
- traitez un warning comme une erreur afin d'échouer rapidement et de pouvoir résoudre le problème plus tôt
- effectuer des actions supplémentaires lorsqu'une erreur/warning spécifique est émis



# Dry Runs

La fonctionnalité de dry-run (Pro et Enterprise) est disponible pour les migrations Java et SQL et permet de :

- prévisualiser les modifications que Flyway apportera à la DB
- soumettre les instructions SQL pour examen à un DBA avant de les appliquer
- utiliser Flyway pour générer le script de mise à jour, mais utiliser un outil différent pour appliquer les modifications

# Plan

- ① Introduction
- ② Fonctionnalités
- ③ Commandes
- ④ Démonstration
- ⑤ Discussion et Q&A

# Migrate/Undo

- **Migrate** : migre le schéma vers la dernière version. Flyway créera automatiquement la table d'historique de schéma si elle n'existe pas.
- **Undo** : annule la migration versionnée la plus récemment appliquée (pro et Enterprise)

## Options des commandes migrate/undo (beta) :

- **Cherry pick** : spécifiez les migrations à appliquer avec l'option `-cherryPick`. Exemple : `flyway migrate -cherryPick="5,create_view"`
- **Mark as applied** : avec l'option `skipExecutingMigrations=true`, l'exécution du script est ignoré, en mettant à jour la table d'historique de schéma.

# Info/Validate

- **Validate** : valide les migrations appliquées par rapport à celles disponibles.
- **Info** : imprime les détails et les informations d'état de toutes les migrations. Les états possible d'une migration :
  - **success** : la migration est réussit
  - **pending** : la migration est détectée et n'est pas appliquée
  - **failed** : lorsque la migration échoue (BD ne supporte pas les transactions)
  - **undone** : la migration versionnée est annulée par la commande undo.
  - **outdated** : la migration répétable dont la checksum a changé
  - **futur** : la migration versionnée appliquée a une version supérieure à la version la plus élevée connue.

# Clean/Baseline/Repair

- **Clean** : supprime tous les objets dans les schémas configurés.
- **Baseline** : la commande baseline sert à introduire Flyway dans les bases de données existantes en les basant sur une version spécifique.
- **Repair** : répare la table d'historique de schéma en :
  - supprimant les entrées de migration ayant échoué
  - réalignant les checksum, les descriptions et les types des migrations appliquées avec les migrations disponibles
  - supprimant les migrations répétées supprimé (beta)

# Plan

- ① Introduction
- ② Fonctionnalités
- ③ Commandes
- ④ **Démonstration**
- ⑤ Discussion et Q&A

## Scenario 1 : type de migration et migration au démarrage

Cette demo est réalisé avec une version d'essai enterprise :

- 1 migration et Undo en SQL
- 2 migration repetables avec chargement de checksum
- 3 migration Bash (bulk insert)
- 4 migration Java (anonymize)
- 5 migration Python
- 6 migration au démarrage de l'application (Java API)

## Scenario 2 : conflit de version de migration

Cette demo est réalisé avec une version d'essai enterprise :

- 1 créer un migration avec V1\_\_Create\_person\_table.sql
- 2 lancer la migration
- 3 créer un migration avec V1\_\_Create\_email\_table.sql
- 4 lancer la migration et visualiser l'erreur
- 5 résoudre le conflit
- 6 lancer la migration



# Plan

- ① Introduction
- ② Fonctionnalités
- ③ Commandes
- ④ Démonstration
- ⑤ Discussion et Q&A

## Pros :

- Riche en fonctionnalités
- Supporte plusieurs langage/platforme/BD
- Facilite les tâches de maintenance pour des usecases complexes

## Cons :

- L'édition communautaire est limitée
- Modèle de facturation par schéma pour les versions payantes

## Alternatives :

- Liquibase
- Alembic
- Orcas DB
- Dbmate



- Documentation : <https://flywaydb.org/documentation/>
- Blog : <https://flywaydb.org/blog/flyway-7.0.0-beta>
- FAQ : <https://flywaydb.org/documentation/faq>

Merci pour votre attention !