Automatisation du développement

Présentation du module

L'objectif de ce module est de comprendre les enjeux de l'automatisation du développement. Nous aborderons, à travers différents points, la manière d'articuler son projet pour le rendre plus facilement automatisable, les méthodes de travail pour faciliter cette automatisation et les outils permettant de la mettre en place.

Avant de commencer

Automatiser son travail permet de se concentrer sur les tâches à plus forte valeur ajoutée. On peut ainsi se concentrer sur la conception et la réalisation de son projet.
L'automatisation des processus de travail permet également de mettre en place des outils de qualité et de sécurité appliqués de manière systématique.
Enfin, en obligeant à configurer un projet de manière reproductible, l'automatisation facilite le travail en équipe ainsi que la maintenance du projet.

Toutes ces notions peuvent sembler abstraites ou trop complexes/lourdes pour de petits projets. Pourtant, les appliquer dès le départ, même sur des projets modestes, permet de se familiariser avec ces processus, et de se développer un kit d'outils que l'on peut réutiliser sur des projets plus conséquents par la suite.

En résumé :

- Gain de temps sur les tâches répétitives
- Meilleure qualité du projet
- Meilleure sécurité du projet
- Meilleure maintenabilité du projet
- Meilleure collaboration sur le projet

L'élément indispensable pour démarrer est de disposer d'un environnement facilement reproductible. Il convient donc de figer les versions de chaque langage et framework utilisés.
Ensuite, il faut définir une méthode simple et précise pour reproduire cet environnement. Nous évoquerons plus tard les outils de qualité et les tests qui seront déclenchés automatiquement sur le projet.
Une fois tout cela en place, l'automatisation du déploiement en production viendra compléter le processus.
Il est facile de prendre en main ces concepts en les amenant progressivement dans son projet.

Versions de langage et framework

Note : Pour tout le module, nous travaillerons avec des projets en PHP, et un front en JavaScript.

Il est indispensable d'utiliser un gestionnaire de dépendances pour figer les versions de langage et de framework, ainsi que pour les maintenir à jour. Voici donc un rappel des principaux outils utilisés.

Composer

On utilisera donc Composer pour PHP.
Composer permet de charger les dépendances d'un projet et de les référencer avec leur version dans un

fichier `composer.json`.
Il est également recommandé d'y inscrire la version de PHP et les extensions nécessaires au bon fonctionnement du projet (par exemple : ext-mysql, ext-bcmath, etc.).
Le fichier `composer.lock`, généré automatiquement, liste les versions exactes des dépendances et permet de reproduire exactement l'environnement de travail. Il est indispensable de garder ce fichier.
Composer permet aussi de lancer des scripts utiles pour automatiser diverses tâches.

Quelques commandes à bien connaître :

- `composer install` : Installe les dépendances du projet. Il crée également le fichier `composer.lock` s'il n'existe pas. S'il existe, il installe les dépendances listées dans ce fichier.
- `composer update` : Met à jour les dépendances et le fichier `composer.lock` avec les nouvelles versions.
- `composer require` : Ajoute une dépendance au projet (met à jour le fichier `composer.json` et le fichier `composer.lock` avec la nouvelle dépendance). Utiliser l'option `--dev` pour une dépendance de développement.

Npm

Pour le front, on utilisera npm pour gérer les dépendances. Il est possible d'utiliser d'autre gestionnaire de dépendance, comme yarn ou pnpm. Pour des raisons de simplicité nous utiliserons npm dans les exemples, mais le principe sera le même avec les autres gestionnaires.

Comme Composer, npm permet de lister les dépendances dans un fichier `package.json`. Il possède également un fichier `package-lock.json` qui liste les versions exactes des dépendances. Il également possible (et conseillé) de configurer des scripts avec npm (compiler le front, lancer les tests, lancer le serveur de dev, etc.).
Configurer ses scripts dans`package.json` (et dans `composer.json`) permet d'être sûr que toutes les personnes travaillant sur le projet utiliseront les mêmes outils, et les mêmes versions. Il permet également d'appeler ces commandes de manière plus facile via d'autre script bash, nous verront l'utilité de cela plus tard.

Quelque commande à bien connaitre :

- `npm install` : Installe les dépendances du projet. Il créé ou met à jour le fichier `package-lock.json` .
- `npm ci` : Installe les dépendances via `package-lock.json`. Très utile pour les reproductions d'environnement.
- `npm update` : Met à jour le `package.json` avec les nouvelles versions des dépendances.
- `npm install --save <ma-dependance>` / `npm install --save-dev <ma-dependance>` : Ajoute une dépendance au projet. Il met à jour le fichier `package.json` et le fichier `package-lock.json` avec la nouvelle dépendance.

Git

Vous devriez déjà connaître git, et l'utiliser pour gérer votre projet. Vous l'utilisez peut-être même déjà de manière avancée.
Nous allons voir ici des bonnes pratiques pour travailler en équipe, et pour faciliter l'automatisation de votre projet ainsi que la gestion des développements.

Gitflow

Gitflow est une méthodologie permettant de structurer le développement d'un projet en plusieurs branches. Elle est particulièrement adaptée aux projets web, car elle permet de gérer le développement de nouvelles fonctionnalités, les corrections de bugs, et les publications de nouvelles versions. Cette approche s'intègre parfaitement aux bonnes pratiques DevOps, comme la livraison continue.

Branches

Principales branches :

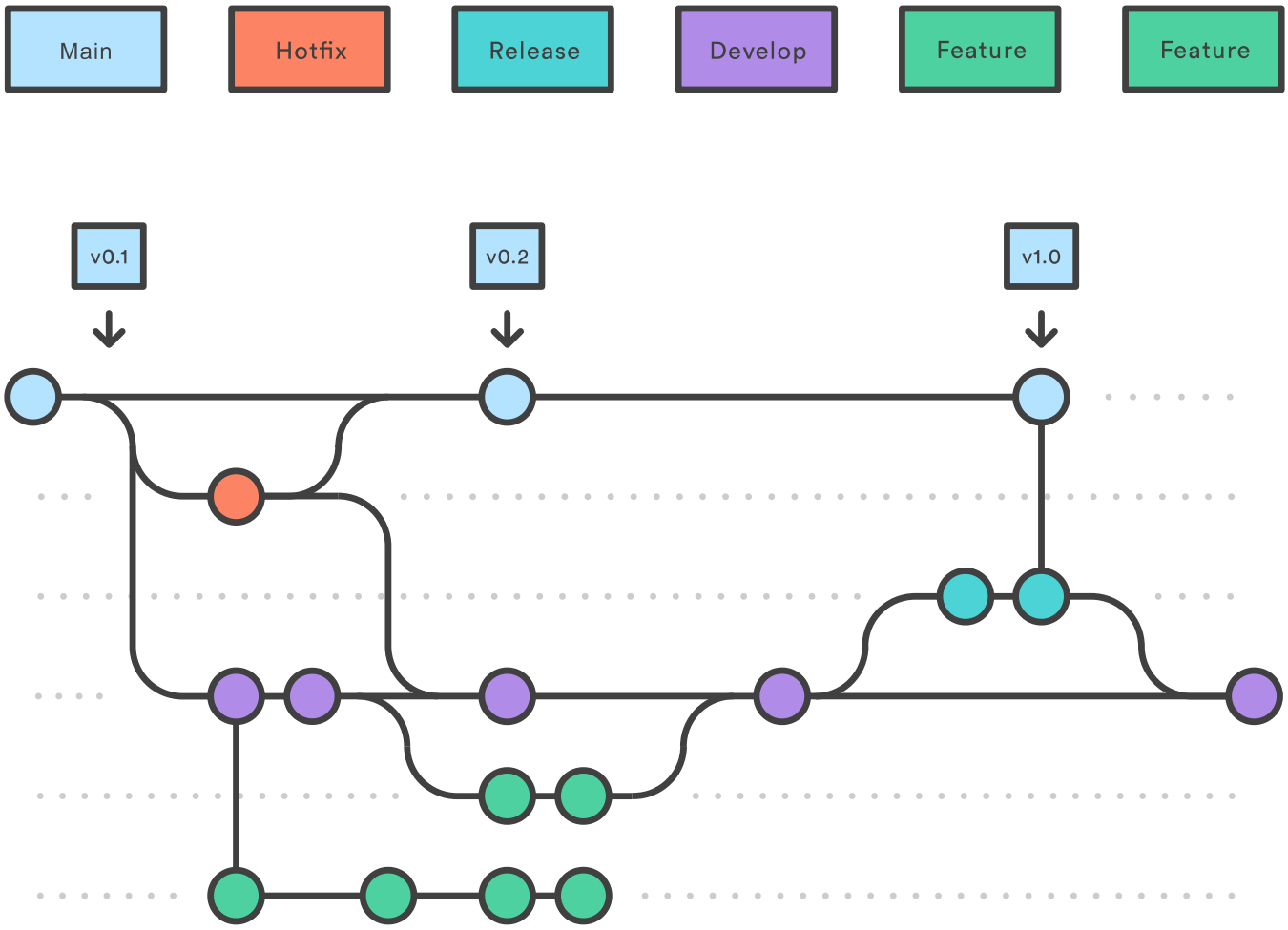
- **main** (ou anciennement **master**) : branche liée à la mise en production
- **develop** : branche de développement collaborative entre tous les devs

Branches secondaires : Mais si tout le monde commence à travailler sur la même branche, l'historique peut rapidement devenir fouilli et les modifications risquent de se court-circuiter. Ainsi on a second niveau de branche :

- `feature/XXX` : pour le développement de nouvelles fonctionnalités, créées à partir de develop puis mergées dessus une fois terminées
- `release/XXX` : faire la liaison entre la branche develop et la branche main. En général cette branche contiendra des commits pour préparer une nouvelle release (on va changer le numéro de version dans le fichier readme par exemple).
- `hotfix/XXX` : pour corriger rapidement un bug depuis main, puis fusion sur main et develop

Exemple de workflow

Représentation graphique du workflow:



Je souhaite développer une nouvelles fonctionnalité à mon projet.
Par exemple sur une application de gestion de tâches, je souhaite ajouter la possibilité de mettre des tags sur les tâches.

- Je crée une nouvelle branche `feature/add-tags` à partir de la branche `develop`
- Je fais mes commits sur cette branche
- Une fois que j'ai terminé, je fais une pull request pour fusionner ma branche `feature/add-tags` sur la branche `develop`
- Une fois la pull request validée (idéalement par un autre membre de l'équipe), je peux merger et supprimer ma branche `feature/add-tags`.
- Je peux créer une nouvelle branche `release/1.0.1` à partir de la branche `develop`
- Si des modifications sont à apporter pour la release, je peux les faire sur cette branche.
- Une fois la release prête, je peux fusionner la branche `release/1.0.1` sur la branche `main` et `develop`
- Je peux flagger la release via git et lui donner un nom en respectant la semantique des numéros de version.

Ce processus peut sembler lourd, mais il garantit une branche de développement fonctionnelle et permet de travailler en parallelle sur plusieurs fonctionnalités ou de faire des corrections rapides sur la production, tout en facilitant les revues de code (pull requests) et l'automatisation.

Je souhaite corriger un bug en production sur mon projet.

- Je crée une nouvelle branche `hotfix/fix-login-button` à partir de la branche `main`
- Je fais mes commits sur cette branche

- Une fois que j'ai terminé, je fais une pull request pour fusionner ma branche `hotfix/fix-login-button` sur la branche `main`
- Une fois la pull request validée, je peux merger et supprimer ma branche `hotfix/fix-login-button`.
- Je reporte les modifications sur la branche `develop`.

On voit ici que Gitflow permet d'intervenir facilement et rapidement sur une version en production, tout en gardant une branche de developpement fonctionnelle. C'est une intervention plus rapide, et plus sereine.

- voir plus en détail : Documentation d'Atlassian sur gitflow

Docker

Votre projet est maintenant versionné, et vous savez la version de chaque langage et framework utilisé. Il est maintenant temps de mettre en place un environnement de travail reproductible pour toute les personnes travaillant sur le projet.

Vous avez probablement déjà utilisé Docker. Nous allons voir ici comment utiliser Docker et Docker Compose pour gérer plusieurs conteneurs simultanés, exécuter des commandes dans un conteneur, et manipuler un fichier `.env` pour les variables d'environnement..

Quelques commandes `docker` utiles pour le développement :

- `docker logs <CONTAINER ID>` : Affiche les logs du container
- `docker stop <CONTAINER ID>` : Stoppe le container

Dockerfile

Le Dockerfile est un fichier qui permet de décrire comment construire une image docker : dépendances, commandes à lancer, extensions nécessaires, etc.
Privilégiez les images officielles (par exemple `mariadb` pour la base de données) et limitez la création d'images personnalisées à ce qui est indispensable (souvent pour PHP).
Le seul cas où nous aurons besoin de créer une image sera pour l'image PHP. Nous utiliserons une image officielle de php, et nous ajouterons les extensions nécessaires au projet.

Vous pourriez être tenté de créer une image avec tous les outils nécessaires au projet, mais cela n'est pas une bonne pratique et peut complexifier la maintenance de l'image, et la rendre plus lourde. C'est rendra également plus difficile à maintenir certaines dépendances précises (changement de version de Nginx mais pas de php ? Je dois reconstruire l'image entière).

Docker-compose

Docker-compose est un outil qui permet de lancer plusieurs conteneurs en même temps. Il simplifie grandement la gestion des conteneurs.

Pour un projet php, l'idéal est d'avoir au moins ces 3 conteneurs :

- Un conteneur pour le serveur HTTP (nginx ou apache)
- Un conteneur pour la base de données (mariadb, mysql, postgresql…)
- Un conteneur pour l'application web (PHP)

- Éventuellement, un conteneur front (Node.js, npm...) ou mailer de dev (ex. mailpit) Si votre projets envoi des mails, vous pourriez également avoir un conteneur pour le serveur de mail de dev (comme mailpit par exemple).

Exemple de fichier docker-compose.yml , pour un projet php avec nginx, mariadb et php:

```
version: "3.8"

services:
  server:
    user: "${UID}:${GID}" # Set user to match host user > can be hardcoded
in .env if needed, use "id -u" and "id -g" to find values. USE ONLY IF
NECESSARY
    image: nginx:alpine
    ports:
      - "${SERVER_PORT:-8080}:80"
    volumes:
      - ./:/var/www/html
      - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf
  php:
    build:
      context: ./docker/php
    volumes:
      - ./:/var/www/html

  database:
    user: "${UID}:${GID}"
    image: mariadb:10
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWD:-password}
      MARIADB_DATABASE: ${DB_DATABASE:-database}
      MARIADB_USER: ${DB_USER_NAME:-user}
      MARIADB_PASSWORD: ${DB_USER_PASSWORD:-password}
    ports:
      - "${DB_PORT:-3306}:3306"
    volumes:
      - db-data:/var/lib/mysql

  mailer:
    image: axllent/mailpit
    ports:
      - "${MAILER_PORT_SMTP:-1025}:1025"
      - "${MAILER_PORT_HTTP:-1080}:1080"

  vite:
    image: node:18-alpine
    user: "${UID:-1000}:${GID:-1000}"
    volumes:
      - .:/app
    ports:
      - "${DEV_SERVER_PORT:-3000}:3000"
```

```
    working_dir: /app
    command: npm run dev
```

Pour chaque variable d'environnement, une valeur par défaut est définie : `${MA_VARIABLE:-valeur-par-defaut}`. Cela permet de pouvoir lancer le projet sans avoir à créer de fichier `.env`, mais de pouvoir tout de même l'adapter localement si besoin.

Commandes utiles

Quelques commandes `docker compose` utiles pour le développement :

- `docker compose up` : Lance les conteneurs (ajouter `-d` pour lancer le tout en arrière-plan)
- `docker compose stop` : Arrête les conteneurs
- `docker compose down` : Arrête et supprime les conteneurs
- `docker compose exec <nom-du-conteneur> <commande>` : Execute une commande dans un conteneur actif. (ex : `docker compose exec php composer install`)
- `docker compose run --rm <nom-du-conteneur> <commande>` : Demarre un conteneur, execute une commande. Par exemple `docker compose run --rm php composer install` pour installer les dépendances du projet. L'option `--rm` permet de supprimer le conteneur une fois la commande terminée, ce qui permet de ne pas avoir de conteneur inutile qui traine.

Fichier .env

Le fichier `.env` sert à définir des variables d'environnement utilisées par le projet (PHP, Docker, etc.). Idéalement, il n'est pas versionné mais un fichier `.env.example` versionné permet de donner un exemple de fichier `.env` à utiliser, et de lister les variables d'environnement utilisées dans le projet.

Fichier README

Non technique mais indispensable, ce fichier documente le projet : présentation, instructions d'installation, commandes utiles, et explications de certains choix contextuels.

Liste minimale conseillée :

- Instructions d'installation
- Commandes essentielles pour le développement
- Brève description du projet

Imaginez devoir reprendre un projet inconnu : ce fichier doit permettre de le lancer et comprendre la stack en quelques minutes.

Arborescence recommandée pour un projet classique

```
mon-projet/
├── docker-compose.yml
├── .env                # variables d'environnement
├── .env.example        # pour le versionning
├── package.json
```

```
├── package-lock.json
├── composer.json
├── composer.lock
├── docker/
│   ├── php/
│   │   └── Dockerfile
│   └── ...
└── src/
    └── ...
├── public/
    └── ...
```