RAPPORT DE STAGE DE M1 INFORMATIQUE

du 7 avril au 13 juin 2025

Mise en place d'une infrastructure CI/CD

Établissement

Université d'Angers

Faculté des sciences

Responsable pédagogique

M. Benoît DA MOTA

Entreprise

Mors Smitt France

Sablé sur Sarthe

Maître de stage

M. Adrien HAPPI

Antonin CHERRÉ

Table des matières

Introduction

Dans le cadre de ma première année de **Master Informatique** à l'**Université d'Angers – Faculté des Sciences**, il m'a été demandé de réaliser un stage de deux mois en entreprise, afin de compléter ma formation par une expérience pratique. Ce stage a pour objectif principal d'assurer à l'étudiant une **formation concrète en lien avec les enseignements suivis à l'université**, et d'initier une première insertion dans le monde professionnel.

Intéressé par les domaines du **développement logiciel** et des **processus industriels associés**, j'ai saisi l'opportunité d'intégrer l'entreprise **Mors Smitt**, filiale du groupe Wabtec, spécialisée dans les systèmes embarqués pour applications ferroviaires.

Le sujet de mon stage porte sur l'**intégration de pratiques DevSecOps** dans le processus de développement et de livraison de solutions logicielles pour des systèmes embarqués. Cette mission m'a permis de me confronter à un **environnement technique exigeant**, nécessitant la mise en œuvre de certaines technologies que je n'avais encore jamais abordées en cours, telles que **GitLab CI/CD**, la configuration d'environnements de développement avec des fichiers `.devcontainer` dans Visual Studio Code, ou encore l'usage d'outils **Docker** dans un cadre industriel.

Encadré tout au long de mon stage par **Adrien Happi**, ingénieur logiciel chez Mors Smitt, j'ai pu progresser en autonomie grâce à l'appui de nombreuses ressources documentaires internes, ainsi qu'en échangeant régulièrement avec l'équipe en place. Le stage s'inscrit dans un projet global d'une durée de **six mois**. **Ce rapport se concentre donc uniquement sur les deux premiers mois de travail**, correspondant à la période d'**avril à juin 2025**.

1 Présentation de l'entreprise

1.1 Présentation générale

Mors Smitt France, implantée à **Sablé-sur-Sarthe**, est une entreprise spécialisée dans la fabrication de composants électriques de haute fiabilité, principalement destinés au secteur ferroviaire. Elle est particulièrement reconnue pour sa maîtrise de la technologie des relais électromécaniques, utilisés dans des environnements critiques où la sécurité et la robustesse sont essentielles.

Depuis 2012, l'entreprise fait partie du **groupe Wabtec** (*Westinghouse Air Brake Technologies Corporation*), un acteur mondial majeur dans la conception et la fabrication de systèmes pour le transport ferroviaire, tant pour le fret que pour les passagers. Basé aux États-Unis, Wabtec emploie environ 27 000 collaborateurs à travers le monde et développe des solutions innovantes autour de la performance, de la sécurité et de la durabilité des systèmes ferroviaires.

Le site de **Sablé-sur-Sarthe** joue un rôle stratégique au sein du groupe Wabtec en assurant la conception, la production et le contrôle qualité de composants critiques. Les équipes locales, composées d'ingénieurs, de techniciens et d'opérateurs qualifiés, collaborent étroitement avec d'autres entités du groupe à l'échelle internationale. Cette implantation permet à Mors Smitt France de combiner savoir-faire industriel historique et intégration dans une dynamique d'innovation à l'échelle mondiale.

1.2 Le service d'accueil

Au sein de Mors Smitt France, j'ai été accueilli dans le **bureau d'étude**, un service transversal impliqué dans de nombreuses activités techniques de l'entreprise : informatique industrielle, systèmes embarqués, électronique, tests produits, innovation et conception mécanique (plans 3D, etc.).

Encadré par **Adrien Happi**, ingénieur logiciel expérimenté, j'ai été intégré à un projet centré sur l'introduction de pratiques DevSecOps dans le cycle de développement. Cette intégration m'a permis de découvrir les outils et méthodologies employés dans un environnement industriel structuré, tout en bénéficiant d'un accompagnement attentif et de ressources documentaires et techniques internes solides.

2 Contexte et missions du stage

2.1 Origine du projet

Le projet dans lequel s'inscrit ce stage est né d'une **volonté interne de moderniser les processus de développement logiciel** au sein de Mors Smitt France. Jusqu'à présent, les nouveaux arrivants passaient plusieurs jours, voire plusieurs semaines, à mettre en place un environnement de développement complet, avec les bons outils, configurations et dépendances. Cette lenteur représentait un frein à la productivité, en particulier dans un contexte industriel où la fiabilité et la sécurité sont primordiales.

Dans ce contexte, Mors Smitt a souhaité amorcer un projet permettant la mise en place d'une **infrastructure CI/CD moderne**, avec pour objectif de faciliter l'installation d'environnements standardisés, reproductibles, et compatibles avec les contraintes de sécurité et de qualité imposées par le domaine ferroviaire. Bien que certaines automatisations existaient déjà dans l'entreprise (pipelines GitLab, runners, etc.), le projet visé consiste à définir et construire une nouvelle infrastructure, plus intégrée et adaptée aux outils de développement modernes comme VS Code et Docker.

2.2 Objectifs et enjeux

Le stage a pour objectif principal de **concevoir et déployer une infrastructure CI/CD** cohérente avec les besoins des équipes de développement embarqué de Mors Smitt. Plus précisément, il s'agit de :

- Définir l'architecture technique et les composants de l'infrastructure.
- Mettre en place un projet GitLab complet, avec protection des branches, workflows et gestion des accès.
- Configurer des environnements de développement sous Docker compatibles avec VS Code et Eclipse.
- Rédiger une documentation exhaustive pour chaque étape (Wiki sur GitLab).
- Réaliser une démonstration de compilation modulaire orientée BDD pour valider les choix techniques et préparer l'intégration future des pipelines CI.

Les enjeux sont multiples : **réduire drastiquement le temps de prise en main des environnements**, **standardiser les processus** et **renforcer la qualité et la sécurité** tout au long du cycle de développement. Le projet s'inscrit aussi dans une volonté à moyen terme d'adoption à l'échelle de l'entreprise.

2.3 Planning du stage

Le stage s'est déroulé sur une durée de deux mois, du **7 avril au 13 juin 2025**. Il constitue la **première phase d'un projet plus vaste initialement pensé en trois étapes**, visant à moderniser l'infrastructure de développement logiciel chez Mors Smitt France.

- **Phase 1 : Fondation** — Installation et configuration des outils de base (GitLab, Docker), création d'une image Docker centralisée, standardisation des environnements de développement (VS Code), rédaction de la documentation associée, et implémentation d'une démonstration de compilation modulaire orientée BDD.
- **Phase 2 : Intégration** — Mise en place des pipelines GitLab CI, intégration des runners, automatisation des tests et outils d'analyse.
- **Phase 3 : Optimisation** — Améliorations des performances, renforcement de la sécurité, supervision de l'infrastructure et documentation finale.

Ce rapport couvre exclusivement la **Phase 1**, réalisée dans le cadre du stage de Master. À l'issue de celui-ci, la suite du projet pourra être reprise ultérieurement en interne ou par un autre intervenant selon les priorités de l'entreprise.

Un diagramme de Gantt synthétique est présenté ci-dessous pour illustrer les principales tâches effectuées durant cette première phase.

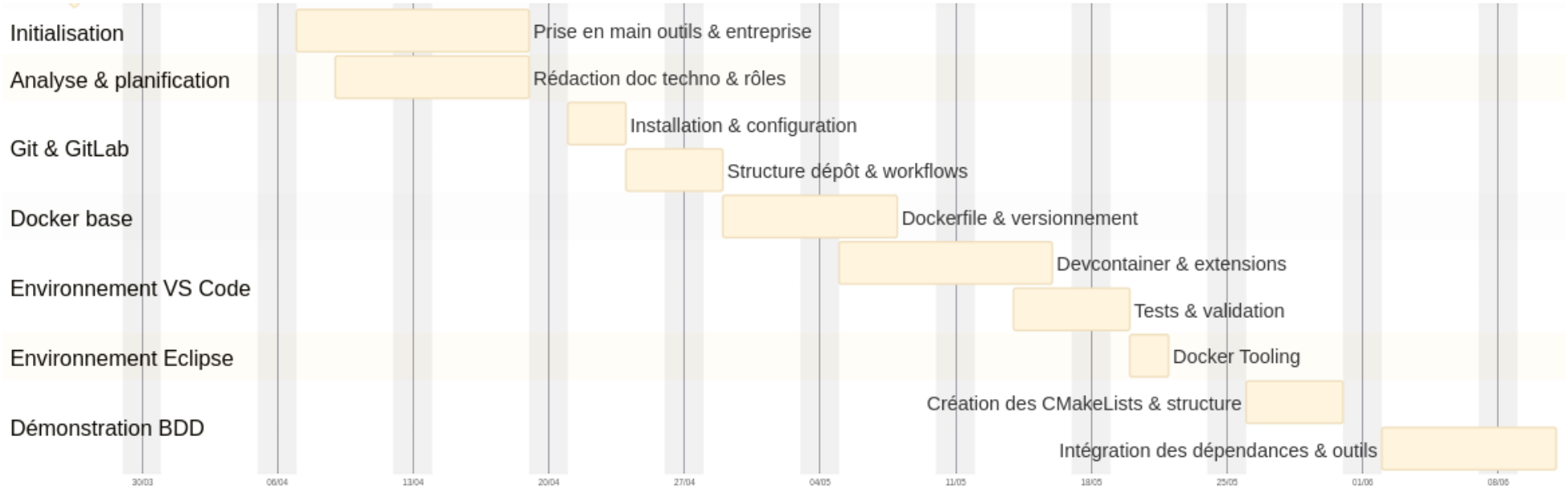

FIGURE 2.1 – Planning du stage

3 Développement et mise en œuvre

3.1 Semaine 1–2 : Analyse des besoins et planification

Choix des outils et architecture

La mise en place d'une infrastructure CI/CD nécessite des choix techniques structurants, en tenant compte des outils existants, des usages en place, et des objectifs de reproductibilité et de sécurité. Une cartographie des composants a été réalisée dès la première semaine afin de visualiser les relations fonctionnelles entre les différents éléments de l'écosystème de développement.

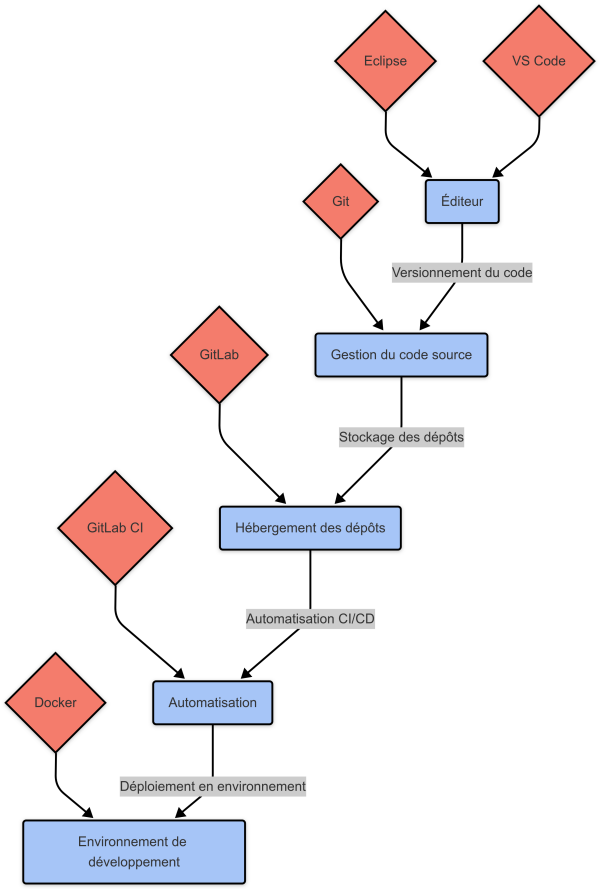

FIGURE 3.1 – Cartographie des outils utilisés dans l'infrastructure

Les principaux outils retenus sont :

— **Visual Studio Code (VS Code)** et **Eclipse**, éditeurs de code extensibles adaptés aux projets embarqués.
— **Git**, pour la gestion des versions et le travail collaboratif.
— **GitLab**, pour l'hébergement de dépôts en interne, accompagné de **GitLab CI** pour l'automatisation.
— **Docker**, pour créer des environnements de développement reproductibles.

Des extensions et plugins spécifiques ont été configurés afin d'optimiser l'expérience de développement :

— Pour VS Code : *Dev Containers*, *GitLab Workflow*, *Remote Extension Pack*, *Live Share*.
— Pour Eclipse : *Docker Tooling*, *eGit*, *GitLab for Eclipse*.

Les langages utilisés dans les projets embarqués en cours sont principalement le `C++` et le `Python`.

Définition des rôles, gouvernance et compétences requises

Dès le début du projet, une analyse des rôles impliqués a permis de clarifier la gouvernance de l'infrastructure, d'identifier les interlocuteurs à chaque étape, et d'anticiper les compétences à mobiliser.

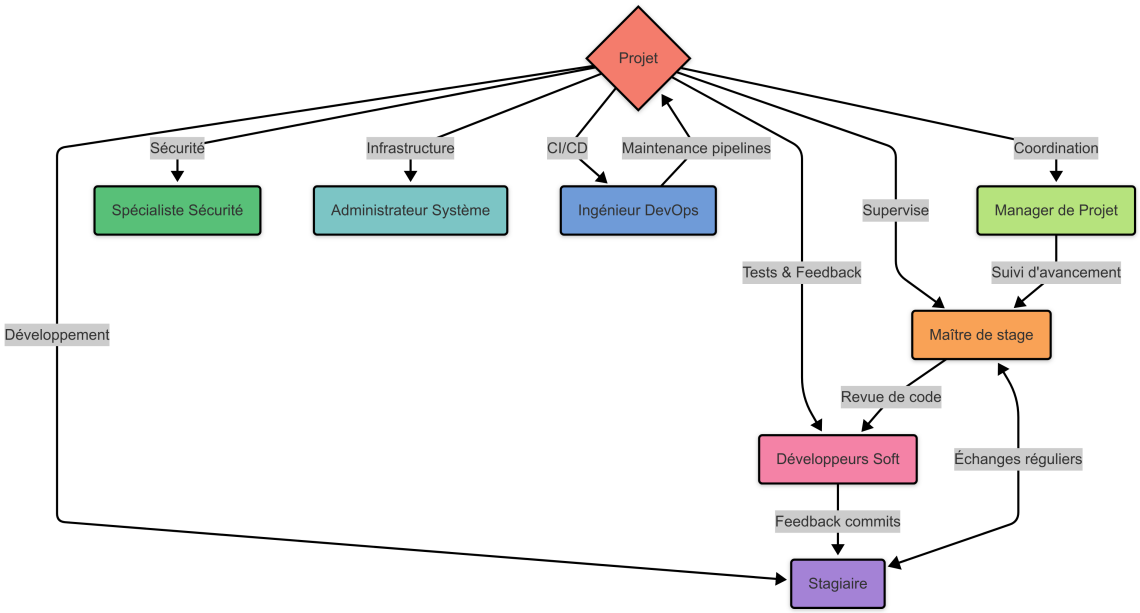

FIGURE 3.2 – Organisation des rôles dans le projet CI/CD

Le schéma ci-dessus illustre les rôles clés :

- Le **stagiaire** (moi-même), en charge de l'étude, de l'implémentation et de la documentation des outils mis en place.
- Le **maître de stage**, qui assure le suivi du projet, oriente les choix techniques majeurs et valide les étapes clés.
- D'autres membres de l'équipe (**développeurs, ingénieur DevOps, administrateur système**) ont pu ponctuellement répondre à mes questions ou me fournir des retours sur certains aspects (outils, intégration, contraintes internes, etc.).

Chaque modification majeure devait être validée, et une communication régulière était assurée via des réunions ou des échanges informels.

Les compétences ciblées à ce stade incluent :

- la configuration de projets GitLab avec pipelines CI,
- l'utilisation de Docker en environnement de développement,
- le suivi de bonnes pratiques de versionnement et de sécurité.

3.2 Semaine 3 : Mise en place de GitLab et Git

Installation et configuration de GitLab

La troisième semaine du stage a été consacrée à l'installation et à la configuration du dépôt GitLab du projet, ainsi qu'à la mise en place d'une stratégie Git structurée. L'objectif principal était de disposer d'un espace de travail collaboratif et sécurisé, prêt à accueillir les futures fonctionnalités de l'infrastructure CI/CD.

La création du projet s'est faite sur l'instance GitLab interne de Wabtec, accessible à l'adresse `gitlab.corp.wabtec.com`. Après avoir obtenu les droits d'accès nécessaires, plusieurs étapes ont été menées :

- Paramétrage du compte utilisateur : génération d'un *Access Token* pour les accès extérieurs et ajout d'une *clé SSH sécurisée* pour le push via terminal.
- Création d'un projet vierge nommé `auto-installation-logiciels-ci-cd` dans le répertoire `incubator` de l'équipe `energy-measurement-system`.
- Initialisation du dépôt avec un `README.md`, un fichier `.gitignore` adapté aux projets Docker, ainsi qu'un dossier `.devcontainer` servant à configurer l'environnement de développement.

Un effort particulier a été porté à la configuration des permissions et à la protection des branches. La branche `main` a été verrouillée pour n'autoriser que les *merge requests* validées par au moins deux développeurs logiciels. Cette politique vise à garantir la qualité du code intégré, tout en forçant la revue de chaque contribution par un pair.

La visibilité du projet a été maintenue en mode *privé* et les droits d'accès sont hérités du répertoire parent, conformément aux règles de gouvernance internes à l'entreprise.

Structuration des dépôts et stratégie Git

La mise en place d'une stratégie Git rigoureuse est apparue essentielle pour structurer le développement sur les 3 phases du projet. Le dépôt a donc été organisé selon une architecture modulaire, avec une branche dédiée à chaque objectif mensuel.

Chaque branche suit une progression jalonnée : un *milestone* par semaine (environ) est défini, correspondant à une tâche ou une fonctionnalité à livrer. Cette structuration permet de suivre les avancements de manière fine et de valider régulièrement les livrables intermédiaires avant fusion dans la branche `main`.

Pour accompagner ce découpage, l'extension `Git Graph` (ou équivalent) dans Visual Studio Code permet une visualisation claire de l'historique Git, facilitant la compréhension des dépendances entre branches.

Enfin, des règles spécifiques ont été définies pour les *merge requests* :

- Exécution obligatoire des pipelines GitLab CI avant validation ;
- Relecture par au moins deux membres de l'équipe Développement ;
- Activation par défaut de l'option *Squash commits*, permettant de regrouper les commits d'une branche en un seul lors des merges, afin de conserver un historique Git propre et lisible.

Cette organisation a permis de construire une base solide pour les futures intégrations, tout en garantissant un suivi clair et sécurisé des modifications tout au long du projet.

3.3 Semaine 4 : Création d'un environnement Docker de base

Objectifs et stratégie

L'objectif de cette phase du projet était de créer une image Docker standardisée pour fournir un **environnement de développement complet, reproductible et à jour**, adapté aux besoins spécifiques de Mors Smitt France. Cette image doit pouvoir être utilisée aussi bien localement (par les développeurs) que dans un contexte d'intégration continue (runners GitLab).

Elle intègre à la fois les **outils de compilation** (GCC, Clang), de **build** (Make, Ninja), de **gestion de projets** (CMake) et des **analyseurs de code et générateurs de documentation** (Cppcheck, Doxygen, Uncrustify). Le choix de ces outils découle des besoins internes identifiés lors de l'analyse initiale.

Implémentation technique

Le développement de cette image s'est articulé autour de plusieurs composants complémentaires, visant à assurer la pérennité, la portabilité et la maintenabilité de l'environnement :

- **Un Dockerfile unique**, codé depuis zéro, basé sur l'image officielle `ubuntu:24.04` récupérée depuis Docker Hub. Il intègre plus de 60 paquets système, une vingtaine d'outils Python, ainsi qu'une dizaine de dépendances compilées manuellement depuis leurs sources officielles (GitHub, Kernel.org, etc.) afin de garantir des versions à jour et durables ;
- **Un fichier `docker-compose.yml`**, facilitant la construction, le lancement, le montage des volumes et le versionnage de l'image grâce à l'utilisation de variables ;
- **Un fichier `.env`**, centralisant l'ensemble des versions logicielles (Git, GCC, Clang, CMake, etc.), permettant une mise à jour simplifiée et une traçabilité des évolutions ;
- **Un script `post-build.ps1`** automatisant le processus de tagging (`version` et `latest`) et le push vers le registre GitLab interne à l'entreprise ;
- **Un fichier de spécification technique**, listant les outils installés, leur fonction, leur version, et leur provenance, afin de documenter précisément l'image et d'en faciliter la maintenance future.

Contrairement à une approche basée uniquement sur les dépôts `apt`, l'installation manuelle depuis les dépôts Git des éditeurs permet de garantir une compatibilité avec les dernières normes de compilation, tout en évitant les problèmes liés à l'obsolescence rapide des versions packagées. Cela rend l'image particulièrement adaptée à une utilisation de long terme dans un contexte industriel.

Chaque phase du build est minutée et suivie d'une vérification explicite des versions installées, afin de valider la cohérence et l'intégrité de l'environnement. L'image est ainsi prête à être utilisée comme socle commun par l'ensemble des développeurs, que ce soit dans un terminal local, via VS Code ou par les GitLab Runners à venir.

Résultats obtenus

Une première version de l'image a été générée avec succès dès la fin de la quatrième semaine du stage. Elle est taguée `1.0.0`.

L'image est poussée automatiquement dans le registre de conteneurs GitLab interne de l'entreprise, au sein du projet `auto-installation-logiciels-ci-cd`, grâce au script post-build.ps1.

Deux tags sont générés à chaque build : un tag de version (ex : `1.0.0`) et un tag générique `latest`, facilitant l'intégration dans les pipelines.

L'image ainsi construite est conçue pour être utilisée comme base de développement dans les environnements Docker de Visual Studio Code et Eclipse, mais aussi comme socle de référence pour les runners GitLab CI utilisés dans les étapes d'automatisation du projet.

Perspectives et évolutivité

L'architecture retenue, via la séparation des fichiers de configuration (`Dockerfile`, `.env`, `docker-compose.yml`), rend l'image facilement modifiable. Chaque mise à jour de composant ne nécessite qu'une modification dans `.env` suivie de l'exécution du script `post-build.ps1`.

Le processus mis en place permettra à l'avenir :

- d'ajouter de nouveaux outils selon les besoins des projets ;
- d'adapter des variantes de l'image selon les contextes (production, test, validation) ;
- de renforcer l'usage de cette image dans l'ensemble des projets GitLab de l'entreprise.

3.4 Semaine 5–6 : Environnement de développement VS Code

Fichier `.devcontainer` et extensions

Dans la continuité de l'intégration de l'image Docker dédiée à l'environnement CI/CD, une configuration spécifique a été mise en place pour Visual Studio Code afin de permettre son utilisation directe comme environnement de développement, sans configuration manuelle.

Deux fichiers ont été ajoutés au projet :

- Un fichier `devcontainer.json`, placé dans le dossier `.devcontainer`, permet de lancer un conteneur à partir de l'image Docker précédemment construite et poussée dans le registre GitLab de l'entreprise.
- Un fichier `extensions.json`, situé dans le dossier `.vscode`, contient la liste des extensions recommandées pour VS Code.

Le fichier `devcontainer.json` spécifie notamment :

- le nom de l'image Docker à utiliser (`msf-env:stable`) depuis le registre GitLab ;
- le répertoire de travail dans le conteneur (`/home/developer/workspace`) ;
- un montage automatique avec un dossier local `workspace` pour synchroniser les fichiers entre l'hôte et le conteneur ;
- des personnalisations de l'environnement utilisateur, comme les réglages de l'éditeur et la liste des extensions installées automatiquement.

La configuration permet à tout utilisateur du projet de cloner le dépôt, d'ouvrir VS Code, et de lancer un conteneur prêt à l'emploi en quelques clics, sans avoir à reconstruire manuellement l'image Docker ni configurer les outils à la main.

Le fichier `extensions.json`, de son côté, recommande plusieurs extensions utiles pour le développement dans un environnement conteneurisé, notamment :

- `ms-vscode-remote.remote-containers`, pour l'intégration des conteneurs Docker dans VS Code ;
- `GitLab.gitlab-workflow`, pour une meilleure interaction avec les projets GitLab ;
- des extensions facilitant le développement en C++, Python, Markdown, etc.

Une documentation complète à destination des futurs utilisateurs du projet a également été rédigée en Markdown. Elle présente les prérequis (Docker Desktop, VS Code), l'organisation des dossiers nécessaires pour assurer le bon fonctionnement du montage de volumes, et les différentes étapes pour lancer le projet dans un conteneur via VS Code. Cette documentation vise à faciliter la prise en main de l'environnement par de nouveaux collaborateurs, y compris ceux ne disposant pas des droits d'administration sur leur machine Windows.

Tests et validation du workflow

La validation du workflow s'est appuyée sur plusieurs essais réalisés en local dans un environnement simulant les conditions réelles d'utilisation. Les tests ont porté sur les points suivants :

- lancement du conteneur à partir de l'image stockée dans le registre GitLab de l'entreprise ;
- bon montage du répertoire `workspace` local dans le conteneur ;
- installation automatique des extensions recommandées ;
- bon fonctionnement des outils de développement dans le conteneur (terminal intégré, outils CI/CD préinstallés, support des langages C++, Python, etc.) ;
- compatibilité avec des systèmes hôtes sous Windows.

L'ensemble du processus s'est avéré fonctionnel et stable, après plusieurs ajustements itératifs sur l'image Docker, dont la version stable finale utilisée est la `1.0.9`. Les développeurs peuvent ainsi démarrer rapidement un environnement de développement complet, identique à celui des autres membres de l'équipe, et bénéficier des outils préconfigurés sans avoir à gérer manuellement les dépendances ou installations.

Des vérifications supplémentaires ont été réalisées afin de s'assurer de la disponibilité et du bon fonctionnement des outils principaux. Les versions des compilateurs et interpréteurs installés (tels que `gcc`, `cmake`, `python`) ont été contrôlées, et de petits programmes de test en C++ et en Python ont été compilés et exécutés avec succès dans le conteneur.

3.5 Semaine 7 : Exploration de l'environnement Eclipse (non finalisée)

Dans le prolongement du travail réalisé sur Visual Studio Code, une phase d'exploration a été entamée pour proposer un environnement de développement similaire via **Eclipse**, à l'aide de l'extension *Eclipse Docker Tooling*.

L'objectif initial était de permettre aux développeurs utilisant Eclipse de lancer des conteneurs Docker configurés de manière identique à ceux utilisés dans VS Code, facilitant ainsi la cohérence des environnements CI/CD.

Une documentation technique a été rédigée à cette occasion, détaillant :

- l'installation d'Eclipse et de l'extension Docker Tooling ;
- la configuration d'une connexion TCP avec Docker Desktop ;
- les étapes de lancement d'un conteneur depuis Eclipse avec montage dynamique du workspace ;
- la résolution des problèmes courants liés à l'intégration Docker.

Toutefois, en concertation avec le maître de stage, cette piste a été mise en pause, afin de prioriser la mise en œuvre de compilation modulaire orientée BDD. L'exploration menée reste néanmoins documentée et pourra servir de base pour une reprise ultérieure.

3.6 Semaine 8-9 : Démonstration BDD - construction modulaire et packaging

Une démonstration de type *BDD* (Behavior-Driven Development) a été réalisée afin de tester une infrastructure de compilation modulaire, portable et généralisable à l'ensemble des projets embarqués de l'entreprise. Elle illustre la capacité à structurer un projet complexe en plusieurs modules indépendants, tout en intégrant des dépendances externes via des téléchargements depuis leurs dépôts officiels.

Architecture du projet : le projet est organisé selon trois modules principaux :

- **Main** : binaire principal simulant une application embarquée typique, avec journalisation (`Log4CXX`) et configuration ;
- **Test** : tests unitaires, d'intégration et tests BDD, réalisés avec `GoogleTest`, `Benchmark` et `Cucumber-cpp` ;
- **IT** : module prévu pour des tests techniques ou de bout en bout (non encore finalisé).

Chaque module est accompagné d'un `CMakeLists.txt` dédié, et les dépendances sont téléchargées depuis leurs sources officielles (`GitHub`, `Apache`, etc.) dans leur version la plus récente (généralement depuis la branche `main` ou `master`). Celles-ci sont ensuite compilées manuellement au sein du conteneur Docker. Cette approche permet de contourner les limitations des gestionnaires de paquets classiques (`apt`), souvent en retard sur les dernières versions, et garantit un environnement homogène, stable et à jour sur le long terme.

Dépendances intégrées (extraits) :

- **Main** : `OpenSSL`, `Boost`, `fmt`, `Expat`, `APR`, `APR-Util`, `Log4CXX` ;
- **Test** : `GoogleTest`, `Google Benchmark`, `Cucumber-cpp`, `CURL`, `Xerces-C`.

Fonctionnalités implémentées :

- Compilation modulaire avec gestion des dépendances par `ExternalProject_Add` ;
- Formatage de code avec `Uncrustify`, `Clang-Format` ;
- Analyse de code avec `Cppcheck`, `Clang-Tidy` ;
- Génération de documentation via `Doxygen` ;
- Packaging avec `CPack` : création d'archives distinctes pour chaque module (`Main`, `Test`, `IT`), contenant les exécutables et leurs dépendances compilées.

Objectifs visés :

- Fournir un socle de compilation reproductible et maintenable ;
- Garantir l'homogénéité des environnements de test en interne ;
- Préparer l'intégration future de ce projet dans des pipelines CI/CD automatisés.

L'arborescence du projet a été pensée pour s'adapter aux besoins récurrents des projets embarqués développés chez Mors Smitt France. Bien qu'il ne s'agisse pas d'un projet fonctionnel destiné à la production, cette démonstration constitue une **preuve de concept réutilisable et généralisable**, qui servira de base à l'automatisation du processus de compilation et de test dans les futurs projets.

3.7 Compétences acquises

Ce stage a représenté ma première immersion dans un environnement informatique professionnel, au sein d'une entreprise industrielle. J'y ai découvert une réalité de terrain très différente du cadre académique : à l'université, les documents pédagogiques fournis permettent généralement de bien cerner les objectifs, les contraintes et les solutions envisageables. En entreprise, ces repères sont absents : il faut construire ses propres références à partir d'une veille technique, de recherches et de tests, souvent à partir d'informations éparses.

Sur le plan technique, j'ai développé de nombreuses compétences autour des environnements virtualisés, notamment grâce à Docker et à la génération d'images personnalisées adaptées à un usage professionnel. J'ai également appris à utiliser des outils d'automatisation et de configuration modernes comme `CMake`, `Doxygen`, `Google Test`, et les `DevContainers` dans VS Code.

L'ensemble des concepts liés au CI/CD (intégration et déploiement continus), à GitLab et à la structuration de projets partagés en entreprise étaient entièrement nouveaux pour moi. Leur apprentissage m'a donné une vision beaucoup plus concrète de ce que signifie industrialiser le développement logiciel.

Enfin, j'ai découvert l'importance centrale de la **documentation** en entreprise : que ce soit pour le code, les décisions techniques ou l'accueil de nouveaux collaborateurs, elle fait partie intégrante du développement d'un projet durable et maintenable.

3.8 Retour sur le projet

Le projet, bien que dense, a été très formateur. La difficulté principale a résidé dans la courbe d'apprentissage : il a fallu assimiler rapidement de nouveaux outils, adopter des pratiques industrielles, et faire preuve d'autonomie face à des sujets encore peu abordés dans ma formation.

Mais cette même difficulté a constitué l'aspect que j'ai le plus apprécié : la satisfaction de surmonter les blocages, de comprendre les mécanismes en profondeur, et de produire des environnements de développement fiables et réutilisables. Le projet m'a permis de toucher à l'ensemble de la chaîne de développement : préparation, environnement, compilation, tests, documentation, distribution. Chaque étape m'a apporté une compréhension plus fine du processus d'industrialisation logicielle dans un contexte embarqué.

Deux mois peuvent sembler courts pour appréhender pleinement les processus internes d'une entreprise. Le temps nécessaire à comprendre les outils, les pratiques en place, et les attentes spécifiques réduit d'autant la durée réellement productive. Cette contrainte rend d'autant plus important l'effort de documentation et de transmission du travail réalisé.

Conclusion

Ce stage chez Mors Smitt France a représenté une étape importante dans mon parcours. Il m'a permis de découvrir l'informatique industrielle appliquée aux systèmes embarqués, dans un cadre structuré, exigeant, mais aussi très stimulant.

Je mesure mieux aujourd'hui les exigences du monde professionnel : rigueur, documentation, standardisation, gestion des dépendances, reproductibilité des environnements. Ce sont des préoccupations parfois absentes dans les projets académiques, mais essentielles dans une entreprise qui conçoit des produits embarqués à haute criticité, à destination de grands acteurs comme la SNCF.

Les résultats obtenus (création d'une image Docker stable et maintenable, intégration dans VS Code, démonstration BDD avec dépendances externes) sont une base solide pour les prochaines étapes du projet. Je suis satisfait du travail accompli, et encore plus de tout ce que j'ai pu apprendre en si peu de temps.

Enfin, ce stage m'a donné envie de continuer à explorer les problématiques liées à l'industrialisation du développement logiciel, et à approfondir mes compétences dans les environnements CI/CD modernes.

Bibliographie et webographie

Sites et documentations consultés

— **Wabtec Corporation**. Site officiel — https://www.wabtec.com
— **Mors Smitt France**. Site officiel — https://www.morssmitt.fr
— **GitLab Documentation**. GitLab Inc. — https://docs.gitlab.com/
— **GitLab CI/CD Pipelines**. GitLab Inc. — https://docs.gitlab.com/ee/ci/
— **Docker Documentation**. Docker Inc. — https://docs.docker.com/
— **Docker Hub**. Docker Inc. — https://hub.docker.com/
— **Visual Studio Code Docs**. Microsoft — https://code.visualstudio.com/docs
— **Dev Containers Specification**. Microsoft — https://containers.dev
— **CMake Documentation**. Kitware — https://cmake.org/documentation/
— **CPack Documentation**. Kitware — https://cmake.org/cmake/help/latest/module/CPack.html
— **Doxygen Documentation**. Dimitri van Heesch — https://www.doxygen.nl/manual/index.html
— **Cppcheck Documentation**. Cppcheck — http://cppcheck.sourceforge.net/

Technologies et outils mentionnés

— `Git`, `GitLab`, `GitLab CI`, `runners`, `pipelines`
— `Docker`, `Dockerfile`, `docker-compose`, `.devcontainer`
— `Visual Studio Code` et extensions : Dev Containers, GitLab Workflow, Remote Extension Pack, Live Share
— `Eclipse` et extensions : Docker Tooling, eGit, GitLab for Eclipse
— `GCC`, `Clang`, `Make`, `CMake`, `Ninja`
— `Cppcheck`, `Doxygen`, `Uncrustify`
— `OpenSSL`, `Boost`, `fmt`, `Expat`, `APR`, `APR-Util`, `Log4CXX`
— `GoogleTest`, `Google Benchmark`, `Cucumber-cpp`, `CURL`, `Xerces-C`

Remerciements

Je remercie l'équipe de Mors Smitt pour l'opportunité de ce stage et pour leur accueil. Ce fut une première expérience très enrichissante dans le monde du développement logiciel en entreprise, tant sur le plan technique que professionnel.

Merci également à mon maître de stage Adrien HAPPI pour son encadrement et sa disponibilité.