

28/01/2025

Wroblewksi Martin

**CER**

Rôles

* Tuteur : Aurélien
* Animateur : Erwan
* Secrétaire : Martin
* Scribe : Thomas
* Gestionnaire : Louis

Analyse du besoin

**Analyse** **du** **besoin** :

Nous cherchons à connaitre les bases du .NET et C#.

**Généralisation** :

Je résous le problème grâce aux environnements de développements.

Sommaire

**Sommaire**

[I. Mise en situation 2](#_Toc184106791)

[A. Définition des mots clés. 2](#_Toc184106792)

[B. Contexte. 3](#_Toc184106793)

[C. Contraintes. 3](#_Toc184106794)

[D. Livrable. 3](#_Toc184106795)

[E. Problématique 3](#_Toc184106796)

[II. Résolution. 4](#_Toc184106797)

[A. Plan d’action. 4](#_Toc184106798)

[B. Résolution. 5](#_Toc184106799)

[III. Cours 7](#_Toc184106800)

[A. IDPS 7](#_Toc184106801)

[B. Pare-feu 9](#_Toc184106802)

[C. UTM 11](#_Toc184106803)

[D. DMZ 13](#_Toc184106804)

[E. Proxy 14](#_Toc184106805)

CER

# Mise en situation

## Définition des mots clés.

Mots clés :

* POO (Programmation Orientée Objet) : Un paradigme de programmation qui organise le code en objets et classes. Les objets représentent des entités avec des attributs et des méthodes, et la programmation se concentre sur l'interaction entre ces objets.
* Build : Le processus de compilation et de création d'une version exécutable d'un programme à partir de son code source. Cela inclut la génération des fichiers binaires, la gestion des dépendances, et parfois l'optimisation du code.
* Maintenabilité : La facilité avec laquelle un logiciel peut être modifié pour corriger des défauts, améliorer ses performances ou ajouter de nouvelles fonctionnalités. Une bonne maintenabilité implique un code bien structuré et documenté.
* Déploiement : L'action de rendre un logiciel disponible pour les utilisateurs, en le transférant dans un environnement de production. Cela inclut souvent la configuration, l'installation et la mise en service des applications.
* Modularité : Un principe de conception qui consiste à diviser un logiciel en composants indépendants appelés modules. Cela permet de mieux organiser, tester et maintenir le code.
* Tests unitaires : Des tests automatisés qui vérifient le bon fonctionnement de petites unités de code, comme une fonction ou une méthode, de manière isolée. Ils aident à détecter rapidement les erreurs dans le code.
* Classe : En POO, une classe est un modèle ou un gabarit pour créer des objets. Elle définit les attributs (données) et les méthodes (comportements) associés à ces objets.
* Anomalie : Un comportement ou un résultat inattendu dans un logiciel qui dévie de la normale. Une anomalie peut être une erreur, un bogue ou un dysfonctionnement.
* Windows : Un système d'exploitation graphique développé par Microsoft, utilisé principalement sur des ordinateurs personnels. Il est souvent utilisé pour développer des applications qui nécessitent une interface utilisateur.
* Développement multiplateforme : Un processus qui consiste à créer des applications pouvant fonctionner sur plusieurs systèmes d'exploitation (Windows, macOS, Linux, etc.) ou appareils (ordinateurs, smartphones, etc.) sans modifications majeures du code.
* Méthodes de travail : Les approches, processus ou techniques utilisés pour organiser et exécuter les tâches de développement logiciel. Cela peut inclure des méthodologies comme Agile, Scrum ou Waterfall.
* Portabilité : La capacité d'un logiciel à être exécuté sur différentes plateformes avec peu ou pas de modifications. Un logiciel portable peut être utilisé sur plusieurs systèmes d'exploitation sans réécriture majeure.
* Objets : En POO, un objet est une instance d'une classe. Il représente une entité réelle ou abstraite et possède des attributs et des méthodes. Les objets interagissent entre eux pour accomplir des tâches.
* Version : Une version d'un logiciel représente un état particulier du développement, généralement marqué par un numéro de version. Cela aide à identifier les différentes étapes du cycle de vie du logiciel et à différencier les versions avec de nouvelles fonctionnalités ou corrections.
* Régression : Un type d'erreur où une fonctionnalité qui fonctionnait correctement dans une version précédente cesse de fonctionner après une mise à jour ou une modification du code.
* Environnement : L'ensemble des conditions dans lesquelles une application est exécutée, y compris le système d'exploitation, les bibliothèques, les outils, les configurations et autres éléments externes nécessaires au bon fonctionnement du logiciel.

## Contexte.

2F Roaming fait face à des problèmes du à des méthodes de travail qui doivent évoluer.

## Contraintes.

Les contraintes sont :

* Collaboration entre développeurs
* Windows
* Temps

## Livrable.

Un CER avec des cours.

## Problématique

Comment améliorer le cycle de vie d’une application grâce à de nouveaux outils de développement ?

# Résolution.

## Plan d’action.

Le plan d’action se déroule en plusieurs étapes :

1. Études des documents (corbeilles + workshop)
2. Notion fondamentale d’un environnement
3. Comprendre .NET Core
4. Comprendre l’intégration continue

CER

## Résolution.

**Etudes des documents (corbeille + workshop)**

Voir corbeille et workshop.

**Notion fondamentale d’un environnement**

Un environnement de développement désigne l’ensemble des ressources logicielles et matérielles dans lequel un logiciel est créé, testé et exécuté. Il est crucial de comprendre ces environnements pour s’assurer que le logiciel soit cohérent, reproductible, et fonctionne correctement dans divers contextes. Voici les différents types d'environnements utilisés dans le cycle de développement :

Types d'environnements :

* Environnement de développement : C'est l'environnement où les développeurs écrivent et testent leur code. Il comprend généralement un IDE (Integrated Development Environment) comme Visual Studio ou VS Code, ainsi que des outils de gestion de version comme Git.
* Environnement de test : Utilisé pour tester les applications de manière isolée. Ce type d'environnement est configuré pour imiter aussi fidèlement que possible l'environnement de production sans risquer de perturber les utilisateurs finaux.
* Environnement de pré-production : Avant de déployer l’application en production, cet environnement est utilisé pour valider que le logiciel se comporte comme attendu dans un environnement similaire à la production.
* Environnement de production : L’environnement où le logiciel est effectivement utilisé par les utilisateurs finaux. Ce doit être un environnement stable et sécurisé.

Gestion des environnements :

* La portabilité des environnements est essentielle. Utiliser des outils comme Docker permet de créer des conteneurs qui encapsulent tout le nécessaire pour exécuter l’application, garantissant que le logiciel fonctionne de manière identique quel que soit le système.
* La reproductibilité des environnements garantit que l’application peut être facilement déployée dans n'importe quel environnement sans nécessiter une configuration complexe ou des ajustements manuels**.**

**Comprendre .NET Core**

.NET Core est un framework open-source développé par Microsoft pour la création d’applications multiplateformes. C’est une version allégée et plus performante de l’ancien .NET Framework, permettant de développer des applications pour Windows, macOS, et Linux.

Principes clés de .NET Core :

* Multiplateforme : .NET Core peut être exécuté sur Windows, macOS et Linux. Cela permet aux applications de s’exécuter sur plusieurs systèmes sans modifications majeures du code.
* Performance : .NET Core est conçu pour être léger et performant. Il est adapté à des applications exigeantes en termes de ressources, comme des applications web, des API, ou des microservices.
* Modularité : .NET Core est modulaire, c’est-à-dire qu’il est possible d’inclure uniquement les composants nécessaires pour une application donnée. Cela permet de réduire la taille de l’application et d’améliorer les performances.
* Support pour les microservices : .NET Core est particulièrement adapté pour le développement de microservices, où chaque service est indépendant, évolutif, et utilise des technologies comme des API REST ou gRPC.

Points clés à maîtriser :

* Installation et configuration de .NET Core (installation de l'IDE Visual Studio et de .NET SDK).
* Création d’un projet dans .NET Core, en utilisant la CLI (command-line interface) ou Visual Studio.
* Gestion des dépendances avec NuGet (gestionnaire de paquets).
* Développement d’applications web et API avec ASP.NET Core, qui est un framework intégré à .NET Core pour développer des applications web légères et performantes.

Pour bien comprendre .NET Core, il est conseillé de :

* Lire la documentation officielle de Microsoft : <https://dotnet.microsoft.com/>
* Expérimenter avec des exemples de code simples, comme créer une application console, puis une API REST avec ASP.NET Core.

**Comprendre l’intégration continue (CI)**

L'intégration continue (CI) est une pratique qui consiste à intégrer fréquemment le code des développeurs dans une branche principale (par exemple la branche main ou develop d’un dépôt Git), où il est ensuite automatiquement testé et déployé.

Les principes de l'intégration continue :

* Automatisation des builds : Chaque fois qu'un changement est effectué, le code est compilé et testé automatiquement. Cela permet de s'assurer que les nouvelles modifications n'ont pas introduit de régressions.
* Tests automatisés : Les tests unitaires et d'intégration sont exécutés à chaque commit pour garantir que le code est fonctionnel et ne casse pas de fonctionnalités existantes.
* Déploiement automatisé : Une fois que le code est testé avec succès, il peut être automatiquement déployé dans un environnement de pré-production ou même de production.

Outils populaires de CI :

* Jenkins : Un serveur d'automatisation open-source qui permet de gérer les pipelines d'intégration continue et de déploiement continu (CD).
* GitLab CI/CD : Une plateforme complète d'intégration et de déploiement continu intégrée à GitLab.
* GitHub Actions : Un outil de CI/CD intégré à GitHub qui permet d’automatiser les workflows de développement.
* CircleCI : Une autre plateforme populaire de CI/CD qui permet d’automatiser les tests et déploiements.

Mise en place d'une pipeline CI :

* Configurer un dépôt Git : Commence par utiliser un service comme GitHub ou GitLab pour héberger ton code source.
* Configurer une pipeline CI : Crée un fichier de configuration (ex. .gitlab-ci.yml pour GitLab ou .github/workflows pour GitHub) qui définit les étapes de build, de tests et de déploiement.
* Tests unitaires et intégration : Ajoute des tests automatisés à ton code, et configure la pipeline CI pour qu'ils s'exécutent automatiquement à chaque commit.
* Automatisation du déploiement : Déploie automatiquement l’application sur un serveur de pré-production à chaque merge dans la branche principale.

# Cours

## .NET Core et .NET Framework

**.NET Core**

.NET Core est un framework open-source et multiplateforme conçu pour créer des applications modernes et légères. Il prend en charge les applications web, les API, les microservices et les applications console. Sa compatibilité multiplateforme lui permet de s'exécuter sur Windows, macOS et Linux. Il est destiné à remplacer l'ancienne version .NET Framework pour des scénarios où la portabilité et la performance sont des priorités.

* **Caractéristiques principales** :
  + Multiplateforme : Fonctionne sur Windows, macOS, et Linux.
  + Open source : Le code est disponible gratuitement, et la communauté peut y contribuer.
  + Léger et modulaire : Inclut uniquement les composants nécessaires pour une application, réduisant ainsi la taille du déploiement.
  + Performanceoptimisée : Conçu pour des applications à haute performance.

**.NET Framework**

Le .NET Framework est un framework développé par Microsoft pour la création d'applications Windows (principalement pour desktop et serveurs). Bien que riche en fonctionnalités, il est limité à Windows et ne supporte pas nativement d'autres systèmes d'exploitation.

* **Caractéristiques principales** :
  + Windows**-**only : Il est conçu uniquement pour les systèmes Windows.
  + Richesse fonctionnelle : Fournit un grand nombre de bibliothèques pour le développement d'applications.
  + Support de longue date : Il est stable et très utilisé dans des applications d’entreprise sur Windows.

**Rôle dans .NET**

* **.**NET Core est conçu pour des applications modernes et performantes sur des environnements multiplateformes, tandis que **.**NETFramework est destiné aux applications Windows legacy qui nécessitent des fonctionnalités de bureau complètes.

## Evolution entre .NET Core et .NET Framework

**Portabilité** :

* .NET Core : Multiplateforme (Windows, macOS, Linux).
* .NET Framework : Limité à Windows.

**Performance** :

* .NET Core a été conçu pour de meilleures performances et une exécution plus rapide, particulièrement pour les applications web et les microservices.

**Modularité** :

* .NET Core est plus modulaire, permettant d'inclure uniquement les parties nécessaires de l'application.
* .NET Framework est plus "monolithique", incluant plus de bibliothèques par défaut.

**Écosystème et développement** :

* .NET Core est optimisé pour le cloud, les microservices, et les environnements modernes.
* .NET Framework continue à être utilisé pour les applications de bureau et les systèmes existants sur Windows.

**Interopérabilité** :

* .NET Core est conçu pour être plus flexible et permettre une meilleure interopérabilité avec d'autres langages et environnements.

## CoreCLR et CLR

**CLR (Common Language Runtime) :**

Le CLR est la machine virtuelle de .NET. Il est responsable de l'exécution du code .NET et de la gestion de la mémoire, de la sécurité et des exceptions.

* **Composants principaux** :
  + JIT (Just-In-Time Compiler) : Compiles le code IL (Intermediate Language) en code machine pour le processeur cible.
  + Garbage Collection : Le CLR gère la mémoire en allouant de l’espace pour les objets et en libérant automatiquement la mémoire lorsqu’elle n’est plus utilisée (garbage collection).
  + Sécurité : Le CLR assure que le code .NET respecte des règles de sécurité et de gestion des accès.

**CoreCLR (Core Common Language Runtime) :**

CoreCLR est l’implémentation de CLR pour .NET Core. Il est conçu pour être plus léger et optimisé pour les environnements multiplateformes.

* **Différences avec CLR classique** :
  + Multiplateforme : CoreCLR fonctionne sur Windows, macOS, et Linux.
  + Optimisation : Il est plus léger et plus rapide pour les applications cloud et microservices.

## Assemblage .NET

Un assemblage (.DLL ou .EXE) est un fichier qui contient du code compilé dans un format exécutable. Il peut inclure des fichiers de métadonnées et des ressources. Dans .NET, un assemblage est la plus petite unité de déploiement et de versioning.

**GAC (Global Assembly Cache) :**

Le GAC est un endroit où les assemblies partagés sont stockés pour être utilisés par plusieurs applications sur un même système.

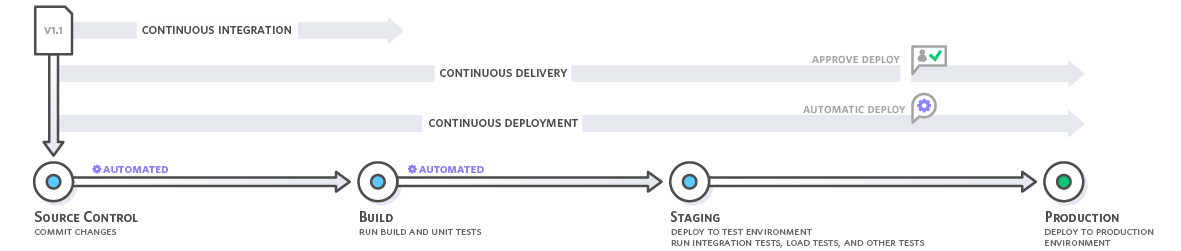
* Dans .NET Framework : Les assemblies peuvent être installés dans le GAC pour une utilisation partagée.
* Dans .NET Core : Le GAC n'existe pas. Les applications .NET Core dépendent des versions locales des assemblages, ce qui permet un déploiement plus isolé et flexible.

## Chaîne d’intégration continue

L’intégration continue est une méthode de développement de logiciel DevOps par laquelle les développeurs intègrent régulièrement leurs modifications de code à un référentiel centralisé, suite à quoi des opérations de création et de test sont automatiquement exécutées. L'intégration continue désigne généralement l'étape de création ou d'intégration du processus de publication de logiciels, et implique un aspect automatisé (par exemple un service d'intégration continue ou de création) et un aspect culturel (par exemple apprendre à intégrer fréquemment). Les principaux objectifs de l'intégration continue sont de trouver et de corriger plus rapidement les bogues, d'améliorer la qualité des logiciels et de réduire le temps nécessaire pour valider et publier de nouvelles mises à jour de logiciels.

Les outils CI (Continuous Integration) permettent d'automatiser les tests, la construction et le déploiement des applications. Voici les principaux outils :

* Jenkins : Open-source, il permet d'automatiser les builds et tests.
* GitLabCI : Intégré à GitLab, offre des pipelines d’intégration et de déploiement.
* GitHub Actions : Outil intégré à GitHub pour automatiser les workflows CI/CD.
* CircleCI : Un autre outil populaire pour CI/CD qui s'intègre bien avec GitHub et Bitbucket.



## Principes du versioning

**Principes de gestion de versions :**

* Versionner chaque changement pour suivre l’évolution du code.
* Branches : Utiliser des branches pour séparer les nouvelles fonctionnalités du code principal.
* Commits : Enregistrer des modifications locales.
* Pull Requests (PR) : Proposer des changements pour fusionner dans la branche principale.

Commandes GIT:

* Initialiser un dépôt :

git init

* Ajouter des fichiers :

git add .

* Faire un commit :

git commit -m "Message du commit"

* Pousser un dépôt distant :

git push origin main