

Mentorat technique

Plan du cours

01

Introduction

02

Architecture globale

03

Mise en pratique

04

Cycle complet

05

**Assurer la qualité
des livrables**

06

**Déploiement et
améliorations**

01

Introduction

Objectif du cours

Mettre en place un écosystème afin de pouvoir vérifier, corriger et améliorer vos prévisions

Passer d'une estimation théorique à une gestion de projet pilotée par la donnée



Prévisions

Combien de temps, de ressources, de complexité.



Mesures

Temps passé, qualité du code, réussite des builds.



Suivi

Relier les estimations à des données objectives.

02

Architecture globale

Quels outils et pour quoi faire?

GitHub



Webhooks riches (push, PR, releases...) vers vos systèmes CI/Chat/etc.
Alternatives: GitLab

TeamCity



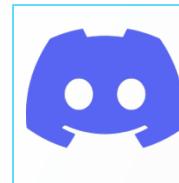
Déclenchement de builds par GitHub Checks Webhook Trigger ; renvoie l'état des builds dans GitHub
Alternatives: Jenkins

YouTrack



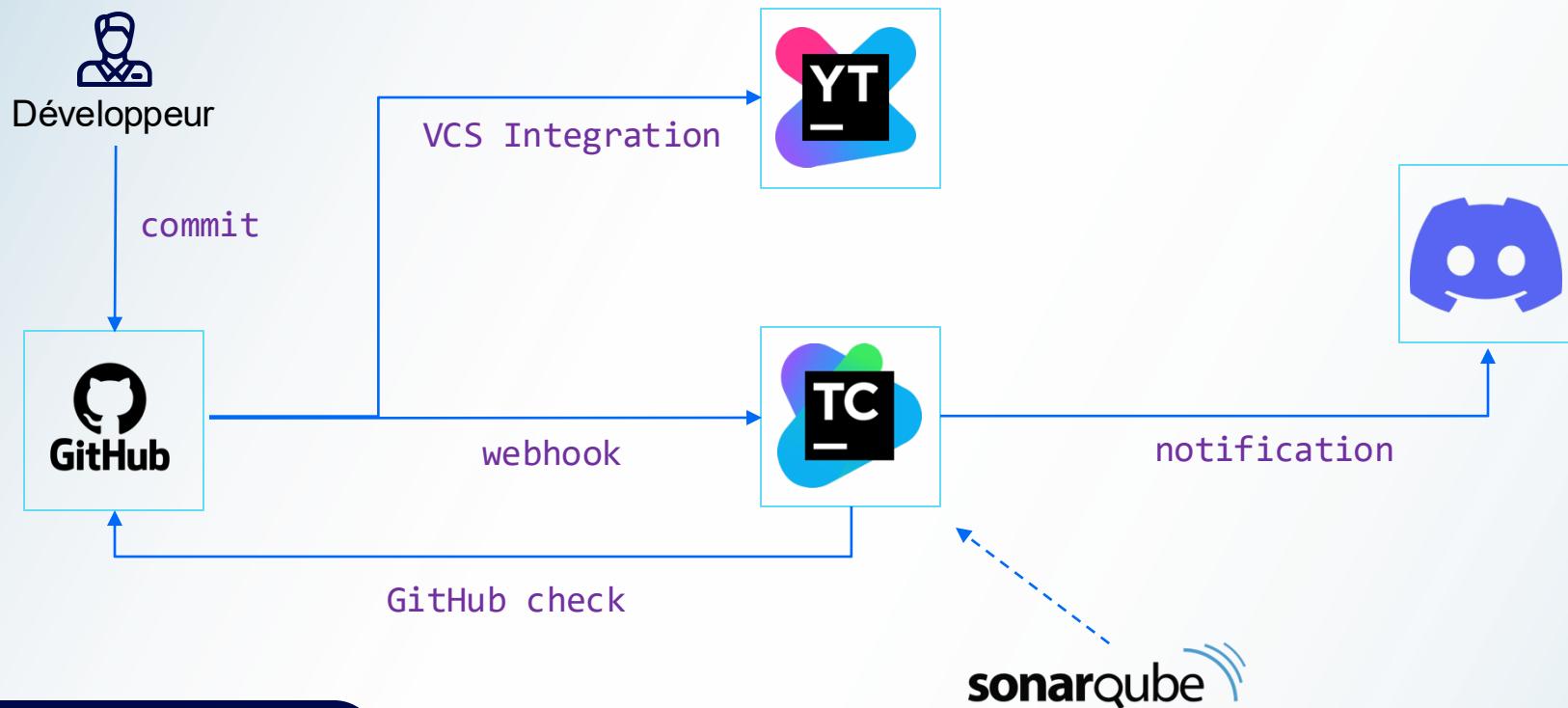
Gestion d'issues, board agile, intégration VCS.
Alternatives: Jira, Trello

Discord



Notifications d'événements (push, build, release, changements d'issues).
Alternatives: Slack

Architecture globale



03

Configuration et intégration

A. Création d'un repos GitHub

Objectif

Avoir un dépôt de code commun



Etape 1

Créer un dépôt GitHub vide avec une seule branche (main)



Etape 2

Ajouter l'ensemble des personnes de votre groupe au projet comme administrateurs



Etape 3

Faire un premier commit et laisser l'ensemble du groupe faire un clone

B. Configuration de TeamCity

Objectif Automatiser les builds (à chaque commit?)

- Créer une instance TeamCity sur le cloud JetBrains
- Créer un nouveau projet "MTECH"
- Ajouter des comptes utilisateurs pour les autres personnes du projet (optionnel)
- Ajouter un VCS Root connecté au dépôt GitHub.
- Activer un Build Trigger sur “VCS changes” (chaque push sur main déclenche un build).
- Exemple de build step (selon le projet) :
Commande : `npm install && npm test`
ou pour PHP : `composer install && composer test`
- Valider qu'un commit déclenche automatiquement un build.

Résultat attendu: à chaque git push, TeamCity récupère le code et exécute les tests.

C. Notifications Discord

Objectif Informer l'équipe en temps réel des builds et commits.

- Dans Discord, créer un serveur dédié (ex. Mtech).
- Créer un salon texte #builds.
- Aller dans les paramètres du salon → Intégrations → Webhook → Créer un webhook.
- Copier l'URL du webhook.
- Dans TeamCity, ajoute un Build Feature → Webhook et colle l'URL Discord.
- Choisir les événements à notifier : Build Started, Build Successful, Build Failed.

Résultat attendu : quand quelqu'un push, Discord affiche automatiquement :

- “Build #15 started by Alice”
- “ Build #15 succeeded (main branch)”

D. Intégration YouTrack - GitHub

Objectif

Suivre automatiquement les tickets via les commits

Dans YouTrack :

- Créer un projet MTECH
- Aller dans Project > VCS Integrations > GitHub > Add Repository.
- Connecter le dépôt GitHub (via token personnel si besoin).
- Appliquer les commandes depuis les messages de commit.
- Vérifier que les emails Git des utilisateurs correspondent à ceux de YouTrack.

Résultat attendu :

un commit comme "**MTECH-XXX State In Progress**" fait passer automatiquement le ticket METCH-XXX à "In Progress".

04

Flux complet de développement

A. Cycle de travail complet

Objectif

Appliquer les conventions et valider le bon fonctionnement du pipeline complet.

- Créer un ticket "MTECH-xxx – Créer l'écran d'accueil"
- Assigner le ticket à une personne
- Commencer le travail du ticket:

```
git commit -m "MTECH-XXX State In Progress"  
git push
```

- Continuer à travailler sur le ticket:

```
git commit -m "MTECH-XXX Spent 45m Comment 'Création du layout principal'"  
git push
```

- Finaliser le travail:

```
git commit -m "MTECH-XXX #Fixed"  
git push
```

Vérifier sur YouTrack que :

- Les commits apparaissent dans l'onglet VCS Changes.
- Le ticket a changé d'état automatiquement.
- L'historique du travail est complet.

B. Erreur et correction

Objectif Apprendre la gestion d'erreur et la réouverture de tickets

1. Simuler une erreur

- Simuler un bug dans le code: "throw new Exception [...] »
- Vérifier que TeamCity envoie une alerte Discord à propos du build échoué

2. Réagir à l'échec du build

```
git commit -m "MTECH-XXX State Reopened Comment 'Échec build, correction en cours'"  
git push
```

3. Corriger le code et valider

```
git commit -m "MTECH-XXX #Fixed Comment 'Tests corrigés, build réussi'"  
git push
```

Vérifier que: TeamCity repasse en build réussi, Discord reçoit la notification, le ticket est à nouveau fermé

Actions courantes en commentaire

Action	Commande	Exemple	Effet attendu
Fermer / marquer comme résolu	#Fixed	MTECH-123 #Fixed	Change l'état du ticket en "Fixed"
Changer l'état (State)	State <nouvel état>	MTECH-123 State In Progress	Passe le ticket à l'état "In Progress"
Rouvrir le ticket	State Reopened	MTECH-123 State Reopened Comment "Bug détecté"	Réouvre le ticket + ajoute un commentaire
Ajouter un commentaire	Comment "<texte>"	MTECH-123 Comment "Ajout des tests unitaires"	Ajoute le texte comme commentaire
Ajouter du temps / work item	Spent <durée>	MTECH-123 Spent 2h ou MTECH-123 Spent 30m	Ajoute un work item (temps passé)
Assigner à un utilisateur	Assignee <nom_utilisateur>	MTECH-123 Assignee alice	Attribue le ticket à l'utilisateur "alice"
Changer la priorité	Priority <niveau>	MTECH-123 Priority High	Définit la priorité du ticket à "High"
Ajouter un tag	Tag <nom_tag>	MTECH-123 Tag backend	Ajoute le tag "backend" au ticket
Retirer un tag	Untag <nom_tag>	MTECH-123 Untag backend	Supprime le tag "backend"

<https://www.jetbrains.com/help/youtrack/server/command-reference.html>

05

Assurer la qualité des livrables

B. Intégration de SonarQube

Objectif Scanner le code pour garantir un niveau de fiabilité et de qualité

1. Créer un compte SonarCloud

- Aller sur le site <https://www.sonarsource.com/products/sonarcloud/>
- Créer une organisation et ajouter le dépôt GitHub

2. Lier SonarCloud à TeamCity

- Générer un token SonarCloud et l'ajouter en tant que variable à TeamCity
`env.SONAR_TOKEN = <token>`
- Ajouter dans la configuration de build le SonarCloud Scanner

```
npx sonar-scanner \
-Dsonar.organization=<organisation_key> \
-Dsonar.projectKey=<project_key> \
-Dsonar.sources=. \
-Dsonar.host.url=https://sonarcloud.io \
-Dsonar.login=%env.SONAR_TOKEN% \
-Dsonar.qualitygate.wait=true
```

Vérifier le fonctionnement en lançant un nouveau build (Logs de TeamCity + Tableau de bord de SonarCloud)

Type d'analyse	Catégorie	Exemples d'outils
Analyse statique du code	SAST – Static Application Security Testing	SonarQube, Snyk Code, Checkmarx, Fortify, Semgrep
Analyse de dépendances / librairies tierces	SCA – Software Composition Analysis	Snyk, Dependabot, OWASP Dependency-Check
Analyse dynamique en exécution	DAST – Dynamic Application Security Testing	OWASP ZAP, Burp Suite
Analyse de qualité du code (non sécuritaire)	Code Quality / Code Review Automation	SonarQube, Codacy, Code Climate

06

Déploiement et améliorations

Déploiement

Ajouter une étape dans TeamCity qui upload votre application sur un FTP distant seulement si le commentaire de commit contient "#deploy" et que l'utilisateur est dans une liste autorisée à effectuer l'action.

Améliorations

En observant le fonctionnement actuel de votre pipeline (GitHub, TeamCity, YouTrack, Discord, SonarQube), identifiez des points d'améliorations possibles.

Formulez une hypothèse d'amélioration et mettez en place une solution technique pour la tester.

Pour inspiration:

- Si on ajoutait une étape d'analyse de vulnérabilités Snyk, on pourrait détecter les dépendances à risque avant le déploiement."
- Si on notifiait Discord uniquement sur les builds échoués, on réduirait le bruit et garderait l'attention sur les erreurs importantes."
- Si on utilisait une commande YouTrack automatique pour rouvrir un ticket quand un build échoue, on gagnerait en traçabilité."