

Rapport de stage

Stage du 3 Janvier au 11 Février 2022



Sommaire

Rapport de stage	1
Sommaire	2
Présentation de l'entreprise	4
Présentation du service	5
Présentation de l'environnement matériel et logiciel	6
Présentation de la méthodologie de travail	10
Présentation du contexte	13
La participation à des entretiens ou réunions	14
Les tâches réalisées	15
<u>Semaine 2</u>	15
Les demandes d'habilitation	15
Web Service d'orchestration	16
Pipeline :	16
Jenkinsfile* :	16
<u>Semaine 3</u>	20
Lister les dépendances présent sur les serveurs	20
Développement d'un fichier Jenkinsfile pour l'intégration continue	21
<u>Semaine 4</u>	23
Résolution du problème d'exécution du Jenkinsfile	23
Migration d'une application ICE vers PACIFIC	27
<u>Semaine 5</u>	28
Finalisation du WebService d'orchestration	28
Les recherches effectuées	34
Semaine 2	34
Semaine 3	35
Les problèmes rencontrés	36
Semaine 3	36
Les actions menées pour la résolution des problèmes	37
Semaine 3	37
Activité complémentaire	Error! Bookmark not defined.
Formation sur la retraite complémentaire	Error! Bookmark not defined.

Présentation de l'entreprise

AG2R LA MONDIALE est née en 2008 de la fusion des entreprises :

- La Mondiale créée en 1905, était une société d'assurance mutuelle sur la vie et de capitalisation
- AGRR créée en 1951, était une Association Générale de Retraite par Répartition, renommée AG2R en 1992

Officiellement, AG2R LA MONDIALE hérite de La Mondiale et de son historique.

Elle est donc recensée comme née en 1905.

Quelques chiffres de l'activité Retraite Complémentaire du Groupe; AG2R LA MONDIALE c'est :

- 414 000 entreprises gérées
- Soit 7,2 millions de salariés
- 3,7 millions de retraités
- 2617 collaborateurs (retraite complémentaire)



Présentation du service

J'ai été accueilli dans le service DSI Retraite Complémentaire et Action Sociale* avec mon tuteur de stage Mr Pasquet Olivier, responsable du service DSI RCAS*.

Je travaille avec plusieurs personnes en interne et externe sur le projet de migration de logiciels Java d'un serveur nommé ICE (Intégration Continue d'Entreprise) sur un nouveau serveur nommé PACIFIC (Plateforme Administrée Communautaire, d'Ingénierie logicielle, Fabrication et d'Intégration Continue).

- Mr LAANANI Abdallah, Chef de projet SI, interne
- Mr MARCHAU Charly, Intégrateur, interne
- Mme MANCEAU Jocelyne, Intégrateur, interne
- Mme KHELIFI Saoussen, concepteur développeur, externe
- Mr EL HENI Aymen, Chef de projet d'études, externe
- Mr ALLAMOU Othmane, concepteur développeur, externe

externe : Société **Atos** (Quartier des 2 lions, Tours)



Présentation de l'environnement matériel et logiciel

Environnement Matériel :

Chaque collaborateur a un pc portable pour pouvoir travailler sur site grâce à un dock et peut être en télétravail.



Durant ce stage, je travaille avec un PC portable qui m'est fourni :



Un Lenovo T14 :

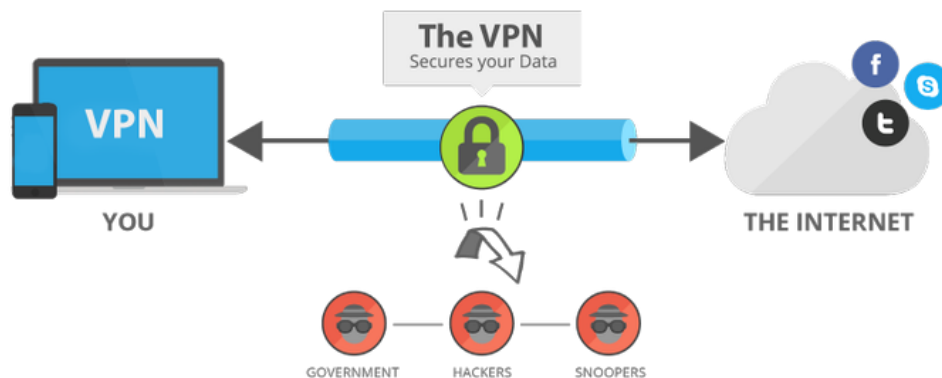
- Processeur : AMD Ryzen 5 pro 4650U (6 coeurs physique, 12 coeurs logique, 2.10Ghz)
- RAM : 16Go
- Stockage : SSD 256Go
- GPU : CPU graphique AMD Radeon Graphics (512 Mo de mémoire vive)
- OS : Windows 10 Entreprise

Environnement Logiciel :

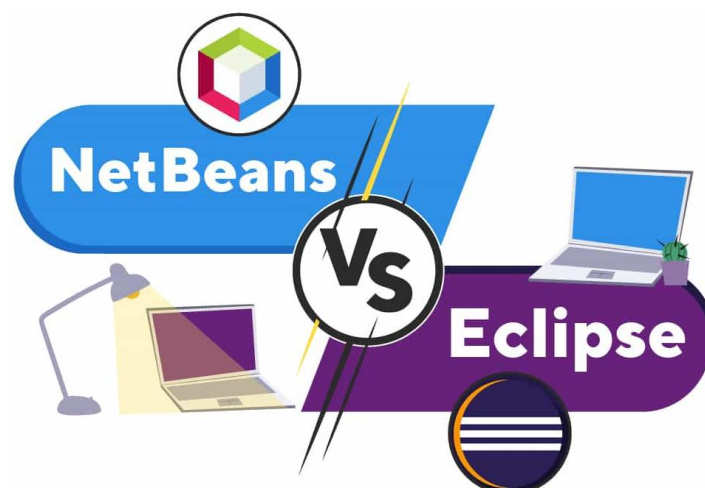
J'utilise la suite  Office 365 Entreprise, et plus particulièrement la messagerie Outlook et le logiciel de réunion Teams.



Suite aux annonces gouvernementales concernant le télétravail, je fais 3 jours de télétravail par semaine : Lundi, Mercredi et Jeudi. Cela oblige d'utiliser un VPN (pour sécuriser l'accès aux données de l'entreprise) fourni par l'entreprise : VALMA (VPN ALM Access)



Pour le développement des logiciels en Java, j'utilise l'IDE Eclipse (2020-03 4.15.0) qui est celui utilisé par les développeurs de l'entreprise. C'est le concurrent direct de l'IDE NetBeans.



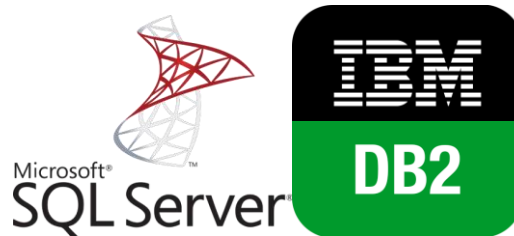
Pour le développement des applications Java, AG2R utilise **Maven**™ pour automatiser l'intégration continue.



Du côté serveur, **<APACHE ANT>** est utilisé pour faire office de serveur web et aussi de compilateur spécifiquement pour le Java

Côté SGBD :

- SQL Server est utilisé pour stocker les données des applications Java
- IBM DB2 est utilisé pour stocker les données des applications en Batch et Cobol



Pour vérifier la quality gate (qualité du code), les outils d'analyse utilisés sont :

- **sonarqube** est un logiciel libre permettant de mesurer la qualité du code source en continu.


Pour la partie tests unitaires :

- **sonarqube** est un logiciel libre permettant de faire des tests unitaires sur les langages : Java, PHP, C#, C++, Python et JavaScript.
- **JUnit 5** est un Framework de tests unitaires pour le langage Java.



CloudBees est un outil pour serveur qui sur la base d'un fichier jenkins file va récupérer et interagir avec différents outils comme GitLab pour déposer un tag; Maven pour build; SonarQube pour la qualimétrie, Hawaii* pour le déploiement. C'est un outil d'intégration continue.

*Hawaii est un outil de déploiement créée par AG2R

 **Nexus** est un serveur de stockage qui dans le cas d'AG2R est utilisé comme référentiel des dépendances Java.


Pour le versioning des applications, GIT est utilisé.


Pour la partie hébergement et de gestion de développement logiciel utilisant GIT, c'est GitLab qui est utilisé.







C'est un concurrent direct de bitbucket et de github.



Présentation de la méthodologie de travail

AG2R a mis en place la méthode agile kanban en s'appuyant sur le logiciel  qui est en relation directe avec GitLab.

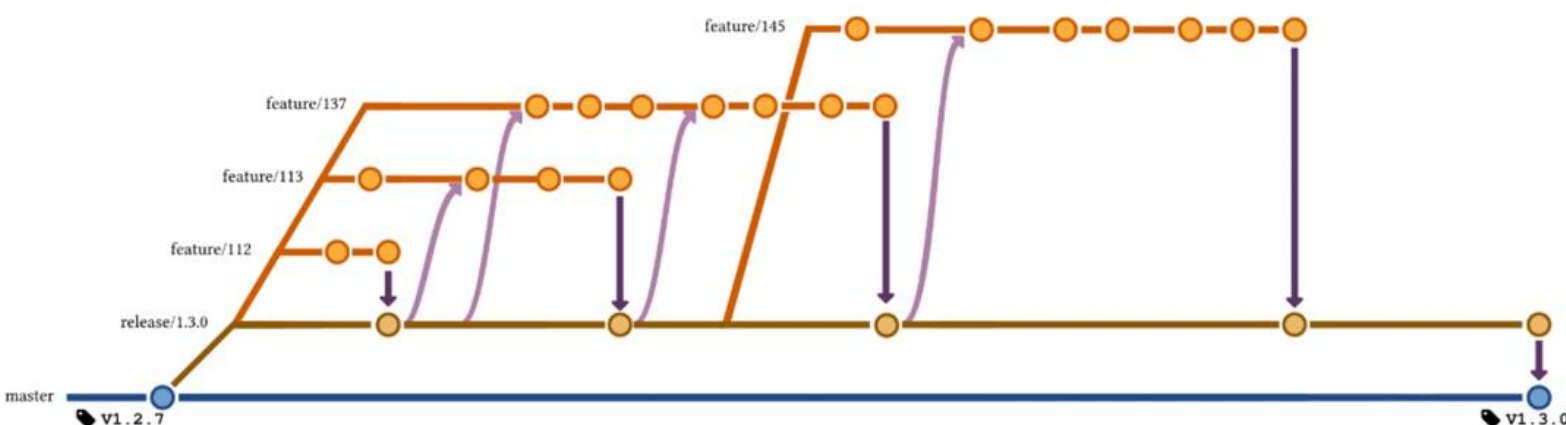
 est un logiciel ALM (Application Lifecycle Management) qui utilise la méthode agile pour du développement dit DevOps tout en favorisant la collaboration en faisant des équipes.
(Comme Bitbucket avec Jira)

À FAIRE	EN COURS	TERMINÉ
  	 	

Grâce au logiciel de versionning GIT, on va pouvoir créer des branches de manière méthodique pour que le projet avance avec une certaine cohérence :

- **feature** : Développement et gros correctifs
- **release** : Préparation d'une version
- **master** : Version de production

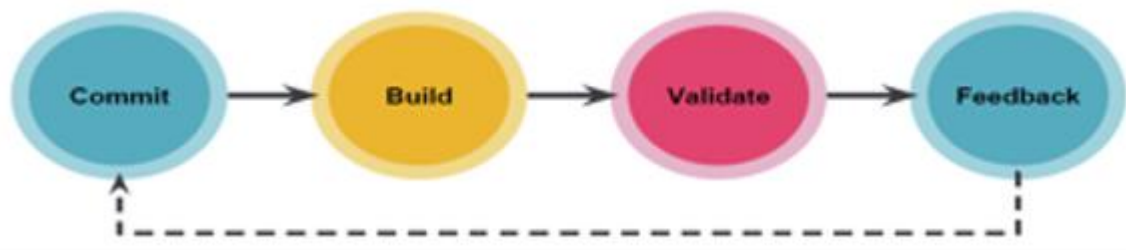
- 1) Une **release** par fonctionnalité
- 2) Des **features** sont créées depuis **release**
- 3) Les développements convergent sur **release**
- 4) Une version est créée par **release**
- 5) La version est publiée sur **master**



AG2R utilise le principe de l'intégration, livraison et déploiement continu :

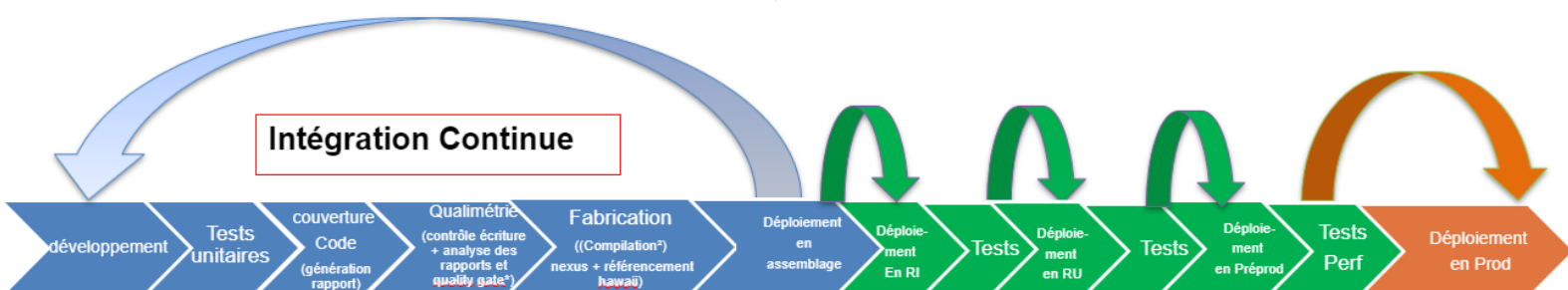
- **Intégration continue** : Ensemble de pratiques consistant à vérifier chaque modification de code source et que le résultat des modifications ne produit pas de régression dans l'application.
Le but principal est d'identifier les problèmes d'intégration pour améliorer la qualité du produit final et diminuer les coûts de correction.

L'intégration continue se fait en 4 étapes :

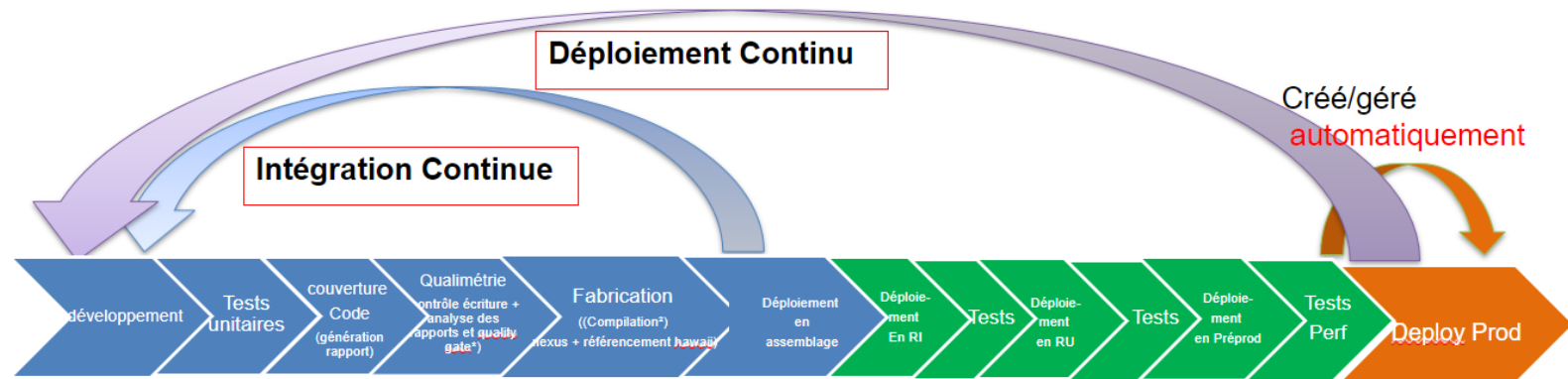


- **Commit** : Action de pousser dans le référentiel centralisé les modifications du code source
- **Build** : Action de compilation de l'application sur une version donnée du code source
- **Validate** : Validation selon les critères d'acceptance de la santé des applications
- **Feedback** : Notification à l'équipe concernée de l'application à l'instant T et la tendance dans le temps (amélioration / stabilité / régression)

- **Livraison continue** : Permet de mettre à disposition une version d'un logiciel sur les environnements hors production. On a donc **obligatoirement** un point d'arrêt avant l'installation en production (arrêt de la base de données, services, ...)



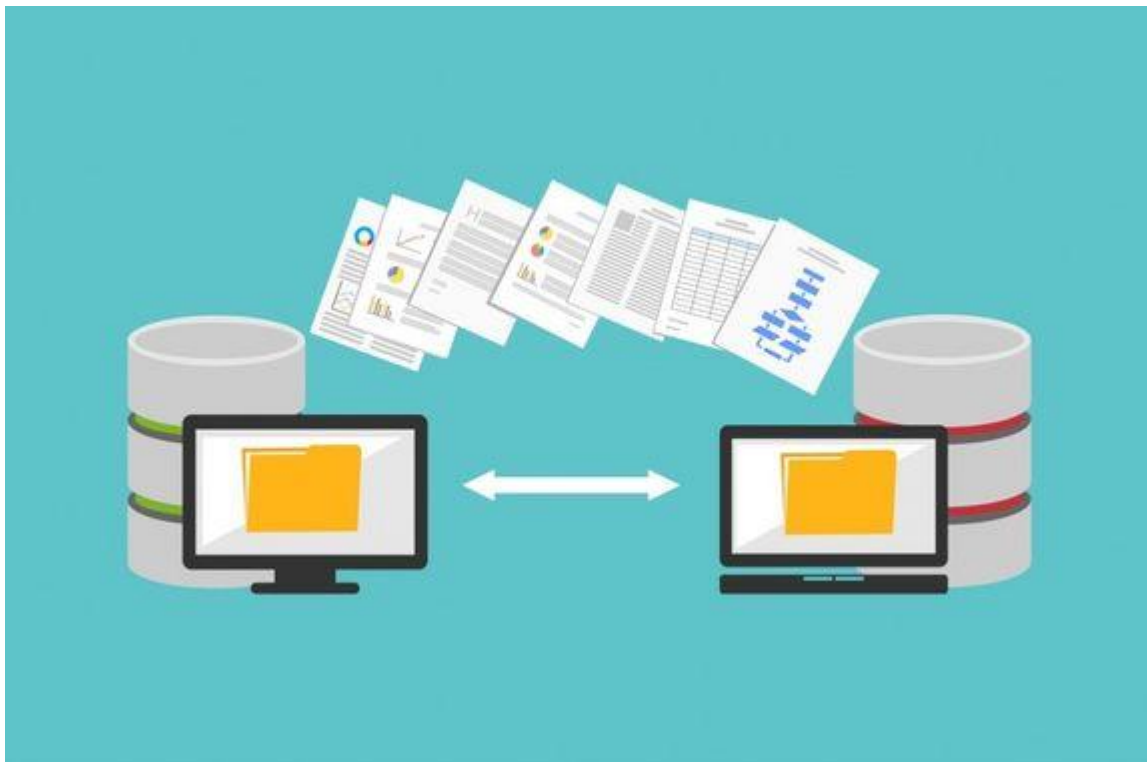
→ **Déploiement continu** : C'est une livraison continue sans point d'arrêt avant la mise en production. Dès la fin du développement, si tous les accords ont été donnés, ce dernier se retrouve en production quelques minutes après (en pleine journée, sans arrêt des bases de données et d'autres actions manuelles).



Malgré le principe d'intégration, livraison et déploiement continu, il y a des risques d'incident et problèmes. C'est pour cela qu'AG2R LA MONDIALE a créé son propre logiciel de gestion d'incident à ticket C@SSIUS qui est fortement comparable à GLPI.

Présentation du contexte

Le but du stage est de modifier des logiciels déjà existants (en Java avec Apache Maven) pour les mettre à jour et ainsi pouvoir les faire migrer sur un nouveau serveur.



La participation à des entretiens ou réunions

En raison du COVID19, je fais du télétravail 2 jours par semaine (mercredi et vendredi). Je participe à des entretiens avec mon tuteur de stage et les membres du projet pour voir l'avancement de celui-ci.

J'assiste aux réunions car je suis directement concerné par le projet qui m'est confié.



Les tâches réalisées

Semaine 2

Les demandes d'habilitation

Un des plus gros problèmes pour commencer un projet quand on a un profil de travail étudiant est d'avoir un profil restreint. Donc tout au long de la semaine, j'ai dû faire des demandes d'habilitation pour que je puisse accéder à tous les outils dont j'ai besoin pour commencer à travailler.

Web Service d'orchestration

Avant d'avoir l'entière responsabilité des habilitations afin de pouvoir travailler sur le projet migration de logiciels, ma première tâche en développement est de travailler avec l'équipe des développeurs pour terminer un livrable en cours qui est un web service d'orchestration :

1. Dupliquer une branche pour faire les tests du jenkinsfile* à posteriori.
2. Répertoire l'ensemble des dépendances du web service et les inclure sur [le serveur Nexus](#).
3. Créer un fichier jenkinsfile pour faire de l'intégration continue à chaque merge et push

Avant de définir ce qu'est un jenkinsfile, il faut comprendre ce qu'est un pipeline.

Pipeline :

Un pipeline DevOps est un ensemble de pratiques que les équipes de développement (Dev) et d'exploitation (Ops) mettent en œuvre pour créer, tester et déployer des logiciels plus rapidement et plus facilement. L'un des principaux objectifs d'un pipeline est de maintenir le processus de développement logiciel organisé et ciblé.

Avant de publier une application ou une nouvelle fonctionnalité pour les utilisateurs, on doit d'abord écrire le code. Ensuite, on s'assure que cela n'entraîne pas d'erreurs fatales qui pourraient faire planter l'application. Éviter un tel scénario implique d'exécuter divers tests pour détecter les bogues, les fautes de frappe ou les erreurs. Enfin, une fois que tout fonctionne comme prévu, on peut diffuser le code aux utilisateurs.

À partir de cette explication simplifiée, on peut conclure qu'un pipeline DevOps comprend les étapes de construction, de test et de déploiement.

Jenkinsfile* :

Un jenkinsfile est un pipeline reconnu par un serveur Jenkins / CloudBees.




```
@Library('hawaii')

// Declarative pipeline
pipeline {
    //definition de l'agent
    agent any
    tools {
        jdk "JDK sun 1.8"
        nodejs "NodeJS 10.16.3"
    }
    // declaration des options (connexion gitlab, configuration...)
    options {
        disableConcurrentBuilds()
        gitLabConnection('gitlab-jenkins')
        gitlabBuilds(builds: ['install', 'test', 'quality', 'build', 'buildZip'])
    }
    triggers {
        gitlab(
            triggerOnPush: true,
            triggerOnMergeRequest: true,
            skipWorkInProgressMergeRequest: true,
            triggerOpenMergeRequestOnPush: 'both',
            acceptMergeRequestOnSuccess: false,
            branchFilterType: "RegexBasedFilter",
            targetBranchRegex: ".*release.*|.feature.*|.hotfix.*"
        )
    }
    stages {
        stage('install') {
            steps {
                withNPM(npmrcConfig: 'npmrc_reponpm_ag2r1m_npmall') {
                    utilsExecuteCommand(cmd: "npm install")
                }
                updateGitlabCommitStatus name: 'install', state: 'success'
            }
        }
        stage('test') {
            steps {
                utilsExecuteCommand(cmd: "npm run test")
                updateGitlabCommitStatus name: 'test', state: 'success'
            }
        }
        stage('quality') {
            steps {
                utilsSonarQube(sonarScanner: "4.3")
                updateGitlabCommitStatus name: 'quality', state: 'success'
            }
        }
        stage('build') {
            steps {
                utilsExecuteCommand(cmd: "npm run build")
                updateGitlabCommitStatus name: 'build', state: 'success'
            }
        }
        stage('Build Zip') {
            steps {
                utilsExecuteCommand(cmd: 'npm run build-zip')
                updateGitlabCommitStatus name: 'buildZip', state: 'success'
            }
        }
        stage('upload'){
            when {
                expression { env.GIT_BRANCH?.startsWith('origin/release') }
            }
            steps{
                hawaiiUploadRest()
                gitlabSetTag()
                updateGitlabCommitStatus name: 'upload', state: 'success'
            }
        }
        stage('deploy'){
            when {
                expression { env.GIT_BRANCH?.startsWith('origin/release') }
            }
            steps{
                hawaiiDeployRest()
                updateGitlabCommitStatus name: 'deploy', state: 'success'
            }
        }
    }
}
```

Un exemple de résultat après l'exécution d'un jenkinsfile :



Chaque test est défini comme un carré, si dans un test, cela ne s'avère pas concluant (rouge), les tests s'arrêtent car ils ne correspondent pas aux normes qui sont imposées. Si tout est bon, le code est directement déployé en production.

Le web service d'orchestration est utilisé dans une application web sur le site d'AG2R LA MONDIALE.

Une fois connecté en tant que membre d'une entreprise affiliée à AG2R LA MONDIALE (n°Siren, n° tel, autre n°tel, mail, autre mail, civilité, nom, prénom, fonction, login), un formulaire de contact est disponible en cas de question. (message, pièces jointes 1,2 et 3)

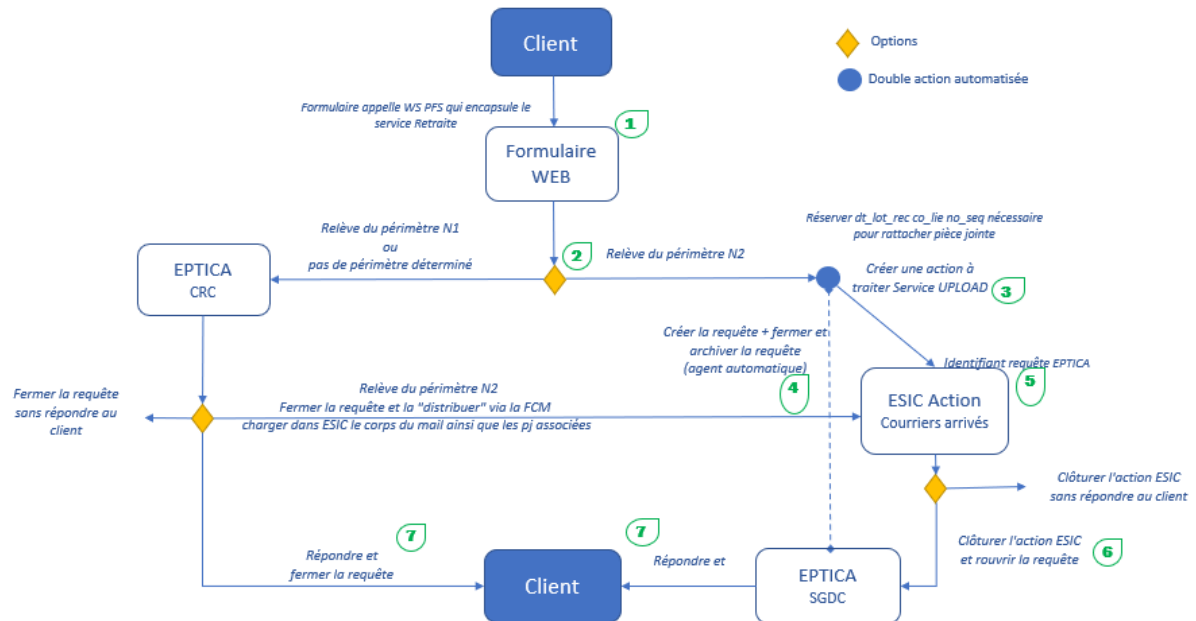
Il y a plusieurs case à cocher pour savoir dans quel service le futur mail va être envoyé. Une fois les cases cochées et le message écrit. Le WebService va prendre l'ensemble des données de connexion et vérifier si elles existent bien dans la base de données. Si tout est bon, selon le chemin qui a été choisi, le message peut partir dans deux sections différentes, CRC et Gestion.

CRC peut traiter les message uniquement de cette partie
Gestion peut traiter les messages CRC et les messages de leur partie.

Si le chemin est CRC, le message est envoyé sur les logiciels internes appelées :
Eptica qui Recense les formulaires / demandes à traiter
Esic qui est un suivi de client et d'entreprises.

Si le chemin est Gestion, cela va créer le message sous forme de fichier PDF et l'ajoute sur Esic.

Sur le logiciel Eptica, cela créer un formulaire spécial sous forme d'archive. Les archives sont disponibles uniquement pour les personnes aptes à traiter ces demandes.



- 1 Formulaire de EC devient le point d'entrée unique avec *formulaire.entreprise.retraite@*
Le client est le propriétaire de l'EC; les Tiers passeront par **leur propre EC** pour écrire sur leurs clients PM
Les flux par les BAL *entreprise.retraite@* et les *BIG-SGDC@* sont stoppés avec un message de redirection vers l'espace client (Messagerie)
ACTION: préciser quelles informations du formulaire remontent dans la fiche client EPTICA
- 2 Orientation N1N2 en fonction typologies du formulaire et de l'acteur (CRC ou CG) y attendant (mapping déjà réalisé)
- 3 Orientation vers CA (ESIC) en fonction du Portefeuille Responsable donné par le SMOG
- 4 Orientation vers EPTICA vers groupe de compte chapeau Retraite Service Entreprise : pas d'orientation vers groupes de compte/corbeilles
Fermeture des requêtes et archivage avec Agent Automatique
- 5 L'identifiant requête EPTICA ne sera pas le n° de requête et ne sera pas visible dans la Vue360 car pas de synchronisation des contacts avec le Hook pour les requêtes PM
Nx processus de synchronisation :
1- création d'un champs spécifique dans Fiche Client
2- Utilisation du nom de lot GED (requête entrante)
3- Récupération dans EPTICA via requêtes archivées avec champs requêtes ("nom de lot GED")
- 6 La réouverture de la requête archivée sera possible de n'importe quel groupe de compte
car la BAL *formulaire.entreprise.retraite@* sera paramétrée dans D2RC et DRC
Paramétrage pour que Agent Automatique ne clôture pas les requêtes rouvertes (requêtes avec historique)
- 7 Les réponses envoyées au client n'ont pas de processus d'historisation (Suppression du Hook)
ACTION: Mettre en place un tel processus

Semaine 3

Lister les dépendances présent sur les serveurs

Avant de commencer à exécuter une application, il faut savoir si les dépendances sont sur le serveur.

J'ai donc listé les dépendances présentes sur le serveur ICE et PACIFIC pour voir s'il n'en manquait pas.

Finalement, j'ai trouvé qu'il manquait le Framework Hibernate dans le serveur PACIFIC.



est un Framework gérant la persistance d'objets en base de données relationnelle (SQL) dans un environnement Java EE.

Fabien Bancharel, le technicien du serveur PACIFIC a pu faire le nécessaire pour ajouter le Framework dans le serveur.

Développement d'un fichier Jenkinsfile pour l'intégration continue

Sur le serveur ICE, les fichiers Jenkinsfile ne sont pas pris en charge. L'intégration continue se faisait via des programmes établis uniquement pour un projet à la fois.

L'avantage sur le serveur PACIFIC est qu'il prend en charge les Jenkinsfile. Cela permet de faciliter le développement de l'intégration continue via des commandes que le serveur va interpréter en exécutant des programmes pré-construits et ainsi faire gagner du temps lors du développement d'un Jenkinsfile et ainsi se concentrer sur le code source.

Pour tester les branches du WebService d'orchestration, on m'a demandé de développer un Jenkinsfile qui permet de vérifier la qualité du code, faire des tests unitaires et si tout est bon, build le projet.

Ci dessous, l'entièreté du code du fichier jenkinsfile que j'ai créé.

```
@Library('hawaii') _
// Déclaration de la pipeline
pipeline {
    //Définition de l'agent
    agent any
    //Indication des outils de développement
    tools {
        jdk "JDK sun 1.8"
        maven "Maven 3.2.5"
    }
    // Déclaration des options (connexion gitlab, configuration...)
    options {
        disableConcurrentBuilds() //Désactivation du build dans GitLab
        gitLabConnection('gitlab-jenkins') //Connexion à GitLab
        gitlabBuilds(builds: ['test&coverage', 'build']) //Étapes à suivre
    }
    //Condition d'exécution de la pipeline
    //Exécuter quand il y a un push ou un merge dans GitLab si cela vient des branches : release ou feature
    ou hotfix
    triggers {
        gitlab(
            triggerOnPush: true,
            triggerOnMergeRequest: true,
            skipWorkInProgressMergeRequest: true,
            triggerOpenMergeRequestOnPush: 'both',
            acceptMergeRequestOnSuccess: false,
            branchFilterType: "RegexBasedFilter",
            targetBranchRegex: ".*release/.*/.*feature/.*/.*hotfix/*"
        )
    }
    //Etapes
    stages {
        // Tests unitaires et couverture de code
        stage('test&coverage') {
            steps {
                utilsExecuteCommand(cmd: "mvn clean verify -Pcoverage")
            }
            //Maven "clean and build" + tests unitaires
            updateGitlabCommitStatus name: 'test&coverage', state: 'success'
            //Affichage du résultat "test&coverage" dans le tableau de résultat des étapes
        }
        // Build
        stage('build') {
            steps {
                utilsExecuteCommand(cmd: "mvn package -Dmaven.test.skip=true")
            }
            //Build Maven
            updateGitlabCommitStatus name: 'build', state: 'success'
            //Affichage du résultat "build" dans le tableau de résultat des étapes
        }
    }
    post {
        success {
            junit "target/surefire-reports/*.xml"
        }
    }
}
```

Semaine 4

Résolution du problème d'exécution du Jenkinsfile

Après avoir résolu une partie du problème concernant l'erreur du Jenkinsfile la semaine dernière, je me suis heurté à un autre problème. J'ai donc dû faire appel à Charly (Intégrateur), Saoussen (Développeuse) et Othmane (Développeur) pour voir ce qui n'allait pas étant donné que le build automatique ne se faisait plus et qu'une erreur incompréhensible est apparue.

Après une journée de recherche, on a trouvé que cela venait du pipeline qui s'était corrompu lors du commit and push dans le dépôt. En ré-essayant, le même problème s'est produit (on pense que cela vient de l'IDE Eclipse qui ne prend pas en charge l'extension .groovy).

Finalement, j'ai recréé le jenkinsfile de toute pièce directement sur le dépôt. Ça a fonctionné, mais une nouvelle erreur est apparue. Lorsque vient le moment de prendre le code source sur le dépôt gitlab, il y a une erreur :

```
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:  
WorkflowScript: 21: Undefined section "trigger" @ line 21, column 5.  
    trigger{  
    ^
```

Finalement, il s'est avéré que c'était une erreur de syntaxe au niveau du trigger car il peut y en avoir plusieurs, et donc ce n'était pas "trigger" mais "triggers". Après cela, le jenkinsfile à commencer à s'exécuter !

Stage View



Malheureusement, les tests unitaires ne passaient pas. En regardant dans la console, on a pu voir que toutes les dépendances n'étaient pas trouvées !


```
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - BUILD FAILURE
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Total time: 5.473 s
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Finished at: 2022-01-25T15:06:24+01:00
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Final Memory: 17M/309M
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - -----

[main] WARN org.apache.maven.DefaultMaven - The requested profile "coverage" could not be activated
because it does not exist.
[main] ERROR org.apache.maven.cli.MavenCli - Failed to execute goal on project wscontactmanagement:
Could not resolve dependencies for project fr.ag2rlamondiale.retraite:wscontactmanagement:war:0.0.1-SNAPSHOT:
The following artifacts could not be resolved: org.apache.tomcat:tomcat-dbc:jar:8.5.4,
org.springframework:springdoc-openapi-ui:jar:1.5.5, org.hibernate.validator:hibernate-validator:jar:7.0.1.Final,
com.systalians.spid.per.catalogue:CatalogueBomUploadAPI:jar:3.1.0, com.opencsv:opencsv:jar:5.5.2,
org.springframework.boot:spring-boot-starter-mail:jar:2.5.8, com.microsoft.sqlserver:sqljdbc4:jar:4.0,
com.systalians.qlf.fcl.catalogue:APIFCL:jar:r3_41_1,
com.systalians.spid.per.catalogue:CatalogueServicesBomPersonneMorale:jar:6.0.17,
fr.ag2rlamondiale.referentiel:model-pm:jar:1.3.6: Could not find artifact org.apache.tomcat:tomcat-
dbc:jar:8.5.4 in central.mirror (https://ledepotgitlabduprojet) -> [Help 1]
[main] ERROR org.apache.maven.cli.MavenCli - 
[main] ERROR org.apache.maven.cli.MavenCli - To see the full stack trace of the errors, re-run Maven with the
-e switch.
[main] ERROR org.apache.maven.cli.MavenCli - Re-run Maven using the -X switch to enable full debug logging.
[main] ERROR org.apache.maven.cli.MavenCli - 
[main] ERROR org.apache.maven.cli.MavenCli - For more information about the errors and possible solutions,
please read the following articles:
[main] ERROR org.apache.maven.cli.MavenCli - [Help 1]
http://cwiki.apache.org/confluence/display/MAVEN/DependencyResolutionException
```

Il s'avère que ce projet, initialement prévu pour aller sur PACIFIC a débuté avec ICE (et ses dépendances) pour prendre de l'avance. Hors dans PACIFIC, il n'a pas les mêmes dépendances que dans ICE. On se retrouve donc avec des dépendances non trouvés car les versions des dépendances ou les dépendances elles même ne sont pas intégrées dans PACIFIC. J'ai donc dû faire une demande pour intégrer ces dépendances au serveur.

Je suis toujours dans l'attente de la réponse des équipes s'occupant du serveur PACIFIC pour ajouter ces dépendances.

Voici un exemple de demande de DTR depuis le logiciel C@SSIUS :
Par soucis de confidentialité, les liens des dépendances ainsi que le dépôt GitLab ont été effacés.

Type de demande	Demande d'alimentation de repository nexus			
Numéro	DTR0156225	État	Ouvert	
Demandeur	LAANANI ABDALLAH	Priorité	4 - Bas	
Bénéficiaire	RODRIGUES VICTORIEN	* Groupe d'affectation	AEI SUPPORT METIS	
EC Impacté	ESPACE CLIENT SERVICES RETRAITE	Affecté à		
Service impacté	Service Web retraite	Type de budget	Code regroupement applicatif	
Date souhaitée	26/01/2022 00:00:00	Code regroupement applicatif	R_ESPACE CLIENT RETRAITE REUNICA	
Date planifiée	26/01/2022 00:00:00	SI Métier / Socle	RETRAITE COMPLEMENTAIRE & ACTION SOCIALE	
* Libellé	Demande d'ajout de dépendances dans NEXUS Groupe			
Description	<p>Bonjour, pouvez vous ajouter les dépendances ci-dessous dans NEXUS Groupe :</p> <p>Versions inexistantes :</p> <ul style="list-style-type: none"> - org.apache.tomcat:tomcat-dbc:jar:8.5.4 : - org.springdoc:springdoc-openapi-ui:jar:1.5.5 : - org.hibernate.validator:hibernate-validator:jar:7.0.1.Final : - org.springframework.boot:spring-boot-starter-mail:jar:2.5.8 : - com.microsoft.sqlserver:sqljdbc4:jar:4.0 : - com.opencsv:opencsv:jar:5.5.2 : <p>Dépendances inexistantes :</p> <ul style="list-style-type: none"> - com.systalians.spid.per.catalogue:CatalogueBomUploadAPI:jar:3.1.0 : - com.systalians.qllfcl.catalogue:APIFCL:jar:r3_41_1 : - com.systalians.spid.per.catalogue:CatalogueServicesBomPersonneMorale:jar:6.0.17 : - fr.ag2rlamondiale.referentiel.model-pm:jar:1.3.6 : <p>pom.xml du projet :</p> <p>Merci</p>			

En parallèle, Charly a vérifié mon jenkinsfile et l'a validé, donc une fois les dépendances ajoutées au serveur, le build devrait s'exécuter sans erreurs.

Migration d'une application ICE vers PACIFIC

Le Vendredi 28/01, j'ai commencé la migration d'une application ICE vers PACIFIC.

Le listage des dépendances fait auparavant a été très utile pour la savoir quel projet doit faire l'objet d'une demande de DTR pour intégrer de nouvelles dépendances ou de nouvelles versions de celles déjà existantes.

Grâce à la documentation interne concernant la migration des applications ICE vers PACIFIC, j'ai compris comment il faut que je m'y prenne pour que tout fonctionne :

Pour certains logiciels trop vieux, il faut créer un Jenkinsfile pour l'intégration continue. Pour les logiciels créés récemment, le Jenkinsfile sera automatiquement mise en place sur PACIFIC.

Il faut vérifier que pom.xml soit à jour au niveau des dépendances appelées.

Vérifier la documentation concernant les mises à jours des Framework pour potentiellement mettre à jour le code source de l'application.

Lancer un build sur le nouveau serveur pour confirmer le succès de sa migration.

Semaine 5

Contourner le problème des dépendances du WebService d'orchestration

Je me suis confronté à un problème pour l'ajout de certaines dépendances dans le serveur PACIFIC. En effet, les librairies "maison" d'AG2R LA MONDIALE ne passent pas dans le nouveau serveur.

Pour contourner le problème, nous avons donc pensé à faire un proxy. Un proxy permet de faire un lien / passerelle entre deux serveurs.

Le but de ce proxy est donc, dans un premier temps, que le programme aille chercher en premier lieux les dépendances dans Nexus Groupe et si il ne trouve pas, aller les chercher dans Nexus Réunion.

Avantages :

- Cela permet de toujours avoir une dépendance qui existe, donc un build assurer.
- Aucune demande d'ajout de dépendances dans Nexus Groupe, donc un gain de temps.
- Une migration des applications extrêmement rapide (pas de changement de code, uniquement créer un Jenkinsfile)

Inconvénient :

- Un temps de build plus important car il doit aller chercher les dépendances sur 2 serveurs au lieu d'un.

Pour que cela soit actif, dans le Jenkinsfile, il faut changer le type de l'agent :

Actuellement, c'est "agent any" qui est utilisé. Cela permet de prendre un agent lambda pour le build du projet.

Il faut donc changer "agent any" par un agent passerelle.

Agent utilisé actuellement :

```
@Library('hawaii') _  
// Déclaration de la pipeline  
pipeline {  
    //Définition de l'agent  
    agent any  
}
```

Nouvelle agent :

```
@Library('hawaii') _  
// Déclaration de la pipeline  
pipeline {  
    //Définition de l'agent  
    agent {  
        node {  
            label 'windows'  
        }  
    }  
}
```

Grâce à ça, j'ai pu exécuter la pipeline. En revanche, pour une raison que l'on ne connaît pas, le Jenkinsfile ne s'exécutait pas à part manuellement.

Après avoir fait vérifier ça par Charly(Intégrateur) et Aymen(Développeur); Charly a fini par créer un incident sur le dépôt du serveur PACIFIC qui a été résolu le Vendredi 4 Février

Migration de projets ICE vers PACIFIC

Grâce au Jenkinsfile créer pour le proxy entre les Nexus, cela à permit de créer un pipeline générique pour chaque projet avec le même niveau de sécurité.

Après s'être connecter sur le dépôt GitLab, on regarde quel est la dernière version d'une branche nommée release, feature ou hotfix.

Une fois la dernière version du projet trouvé, on upload le Jenkinsfile sur la racine.

On incrémente le numéro de version du projet de 1 et ajoute (s'il n'y est pas) "-SNAPSHOT" puis on commit.

Lors du commit, avec les changements apportés, cela va exécuter automatiquement le Jenkinsfile (grâce au code qu'il contient) pour faire les tests.

On va sur le dépôt du projet sur le serveur PACIFIC et on regarde le résultat.

Si tout est au vert, le projet est migré sinon il faut regarder l'erreur et la corriger.

Il y a 2 types d'erreurs :

- Soit on a oublié d'incrémenter le numéro de version du projet dans pom.xml
- Soit (selon le projet) une ligne du Jenkinsfile contenant l'incrémentation automatique de version de pom.xml à chaque commit n'est pas supporté.

Une fois ces erreurs corrigées, le build est relancé et le projet est migré.

La documentation complète via [ce lien](#)

Semaine 6

Finalisation du Jenkinsfile pour le WebService d'orchestration

Afin de terminer le projet projet qui doit arrivé sur PACIFIC, je me suis confronté à un dernier problème.

Le nouvel agent mise en place pour faire build les projet est bon pour les ancien projet, mais les nouveaux projet ne peuvent pas build avec une version de Maven si ancienne (obsolète).

Après la validation de mon tuteur, nous avons créé un ticket pour ajouter un agent Maven en 3.3.9 pour build les nouveaux projets.

Malheureusement, je ne pourrais pas voir ce projet ajouté à PACIFIC car l'agent sera inclus prochainement.

En revanche, j'ai préparé le Jenkinsfile pour que dès lors que l'agent sera ajouté, le Jenkinsfile se déclenche et ajoutera automatiquement le projet dans PACIFIC.

Les modifications sont minimales :

Agent pour les ancien projet:

```
tools {  
    jdk "JDK openjdk 1.8"  
    maven "Maven 3.0.4"  
}
```

Agent pour les nouveaux projets :

```
tools {  
    jdk "JDK openjdk 1.8"  
    maven "Maven 3.3.9"  
}
```

Finalisation de la migration des projets

Pour finir ce stage, j'ai terminé de migrer l'ensemble du lot de projets qui m'a été confié (30 projets).

J'ai pu ainsi corriger les différentes erreurs selon le projet à migré en question.

La gestion des erreurs a été ajoutée à [la documentation technique](#) créée la semaine précédente.

J'ai pu former un membre d'AG2R LA MONDIALE qui prend le relais pour la migration des prochains lots de projets à migrer.

Finalisation de la documentation technique

La semaine dernière, ayant commencer une documentation technique pour que le prochain membre d'AG2R LA MONDIALE puisse continuer et comprendre ce qu'il fait lors de la migration des projets ICE vers PACIFIC, j'ai finalisé cette documentation en ajoutant la gestion des erreurs afin de rapidement localiser le problème et le résoudre.

Les recherches effectuées

Semaine 2

Précédemment, dans les tâches à réaliser sur le WebService d'orchestration, j'ai parlé de créer un jenkinsfile. J'ai donc dû aller voir la [documentation](#) (in english please !) pour comprendre comment on crée un jenkinsfile :

1. Déclarer un agent:
 - "agent any" est pour l'exécution automatique du pipeline
2. Déclarer les outils:
 - "tools" doit indiquer quelles technologies on utilise et leur version (JDK, Maven, ...)
3. Options :
 - Dans notre cas, le but est de ne pas builder dans GitLab directement
4. Triggers:
 - Dans quelles conditions le pipeline s'exécute, dans notre cas lors d'un merge ou d'un push sur les branches
.master.|.release.*|.feature.*|.hotfix/*
5. Déclaration des étapes :
 - "Stages" englobe les tests à effectuer
6. Déclaration des tests :
 - "Stage("nameofstage")" permet de créer un bloc de tests
7. Déclaration des commandes de tests
 - "steps" englobe les commandes à exécuter (comme du code source)
8. when :
 - Équivalent au if
 - Est toujours suivi de "expression"
9. script :
 - Exécution d'un programme comme si c'est une méthode.
10. post :
 - Applique une condition sur chaque condition présente dans "stages"
 - On peut avoir plus de précision sur cette condition en ajoutant :
 - always => toujours
 - changed => modifié
 - fixed => fixe
 - regression => régression
 - aborted => avorté
 - faillure => échec
 - success => succès
 - unstable => instable
 - unsuccessful => échec
 - cleanup => nettoyage

Semaine 3

Lors de la création du fichier Jenkinsfile, j'ai consulté la [documentation officielle](#) ainsi que la documentation interne d'AG2R LA MONDIALE pour me permettre de créer ce fichier.



Les problèmes rencontrés

Semaine 3

Lors du merge de la pipeline dans GitLab, Aymen (Chef de projet) a testé le fichier en lançant un build. Malheureusement, dès la première ligne, il y avait une erreur :

```
rkflowScript: 4: unexpected token: pipeline @ line 4, column 1.
  pipeline {
    ^
1 error
   at org.codehaus.groovy.control.ErrorCollector.failIfErrors(ErrorCollector.java:310)
   at org.codehaus.groovy.control.ErrorCollector.addFatalError(ErrorCollector.java:150)
   at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:120)
   at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:132)
   at org.codehaus.groovy.control.SourceUnit.addError(SourceUnit.java:350)
   at
org.codehaus.groovy.antlr.AntlrParserPlugin.transformCSTIntoAST(AntlrParserPlugin.java:144)
   at org.codehaus.groovy.antlr.AntlrParserPlugin.parseCST(AntlrParserPlugin.java:110)
   at org.codehaus.groovy.control.SourceUnit.parse(SourceUnit.java:234)
   at org.codehaus.groovy.control.CompilationUnit$1.call(CompilationUnit.java:168)
   at
org.codehaus.groovy.control.CompilationUnit.applyToSourceUnits(CompilationUnit.java:943)
   at
org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:605)
   at
org.codehaus.groovy.control.CompilationUnit.processPhaseOperations(CompilationUnit.java:581)
   at org.codehaus.groovy.control.CompilationUnit.compile(CompilationUnit.java:558)
   at groovy.lang.GroovyClassLoader.doParseClass(GroovyClassLoader.java:298)
   at groovy.lang.GroovyClassLoader.parseClass(GroovyClassLoader.java:268)
   at groovy.lang.GroovyShell.parseClass(GroovyShell.java:688)
   at groovy.lang.GroovyShell.parse(GroovyShell.java:700)
   at org.jenkinsci.plugins.workflow.cps.CpsGroovyShell.doParse(CpsGroovyShell.java:142)
   at org.jenkinsci.plugins.workflow.cps.CpsGroovyShell.reparse(CpsGroovyShell.java:127)
   at
org.jenkinsci.plugins.workflow.cps.CpsFlowExecution.parseScript(CpsFlowExecution.java:571)
```

Les actions menées pour la résolution des problèmes

Semaine 3

Avec l'aide de Charly (Intégrateur), il a remarqué que j'avais oublié de mettre un underscore à la fin de ma déclaration de bibliothèque.

Mon code :

```
@Library('hawaii')
```

Code corrigé :

```
@Library('hawaii') _
```

Je n'avais pas remarqué cette erreur car dans la documentation de l'entreprise, il n'y avait pas cette indication car ils n'avaient pas mis à jour la documentation avec les mises à jour du langage.

Fact intéressant : un pipeline, un Jenkinsfile ont une extension .groovy.



est un langage de programmation orienté objet destiné à Java. Ce langage est inspiré de Python, Ruby et Smalltalk pour la partie développement pure ("stage" dans un jenkinsfile). Il reste très proche du Java par l'utilisation de ces accolades (en particulier pour un pipeline) et peut utiliser les mêmes bibliothèques.

Ce que m'a apporté ce stage

Ce stage m'a beaucoup appris dans la gestion de l'environnement de développement ainsi que le développement de pipeline.

D'une part, j'ai appris comment fonctionne l'intégration continue et en utilisant le langage Groovy.

D'autre part, j'ai appris comment l'environnement influence positivement ou négativement sur la performance au travail en particulier durant les périodes de télétravail.

Conclusion

Pour conclure, je dois dire que ce stage m'a bien plus en termes de connaissance acquise. Au niveau relationnel, j'ai appris à connaître des professionnels et à travailler avec eux en assimilant un maximum d'informations pour pouvoir correctement communiquer avec eux en ayant le vocabulaire qu'ils utilisent.

J'ai remarqué durant ce stage, que l'on ne parle pas assez de l'intégration continue et n'avons pas de situation concrète pour l'utiliser.

Grâce à ce stage, je veux continuer dans le développement logiciel car il y a besoin de réfléchir et ce n'est jamais la même chose que l'on produit.

Remerciement

Je tiens sincèrement à remercier Monsieur Pasquet Olivier mon tuteur de stage de m'avoir pris en stage et d'avoir eu l'occasion de travailler à ses côtés.

Je tiens à remercier Monsieur Laanani Abdallah, chef de projet, de m'avoir épaulé tout le long de ce stage en l'orientant sur les tâches à effectuées.

Je tiens à remercier Monsieur Marchau Charly, intégrateur, de m'avoir aider à réaliser les Jenkinsfile et de chercher les erreurs.

Je tiens à remercier l'équipe ATOS :

- Monsieur EL HENI Aymen, chef de projet
- Madame KHELIFI Saoussen, développeur
- Monsieur ALLAMOU Othmane, développeur

de m'avoir aider pour le projet du WebService d'orchestration ainsi que les autres projets auxquels ils sont liés.

Je tiens à remercier Madame FONTAINE Delphine pour la formation sur la retraite complémentaire ainsi que ses réponses à mes questions.

Je tiens à remercier Monsieur MIGAUD Xavier et Monsieur BELLENGUEZ Arnaud pour la réunion sur la sécurité des données personnelles de l'entreprise.

Je tiens à remercier Monsieur PIGNON Jean-Luc sans qui, je n'aurais pas pu obtenir ce stage.

Je tiens à remercier Monsieur DRAINNE Vincent qui reprend la suite du projet sur la migration des projets ICE vers PACIFIC.

Je tiens à remercier Monsieur RENONCOURT Didier et Monsieur DUBREUIL ROMAIN membres de la DOSI, qui m'ont soutenu dans ce projet

MERCI A TOUS POUR CES 6 SEMAINES DE STAGE !