IATIC 4 – 2018/2019

Rapport de stage

DevOps sur PISTE

Auteur :
M. Clément Lefevre

Encadrants :
Mme Sondes Kallel
M. Matthieu Jahan

Version 1.4 du
7 juillet 2019

Résumé

Résumé : Né d'une volonté stratégique de faire évoluer la culture du développement, et de répondre aux difficultés rencontrées par les entreprises sur ce plan, le DevOps se présente aujourd'hui comme un enjeu de transformation vital pour assurer leur compétitivité.

C'est dans ce contexte que l'entreprise Sopra Steria, leader européen de la transformation numérique, m'a accueilli pour mon stage, du 23 Avril au 16 Août 2019, au cours duquel j'ai intégré un projet relativement récent et en plein développement, mobilisant près d'une dizaine de personnes.

Le projet PISTE (Plateforme d'Intermédiation des Services pour la Transformation de l'Etat), piloté par l'AIFE (Agence pour l'Informatique Financière de l'Etat), s'inscrit dans le cadre du programme France Connect Plateforme, qui vise à bâtir le socle numérique de l'Etat plateforme. Il consiste en la mise en œuvre d'une plateforme mutualisée de gestion d'API de l'Etat et de la sphère publique.

J'ai donc été chargé du développement et de la mise en place de pratiques DevOps au sein de ce projet, manipulant ainsi les technologies suivantes :
- Java 2 Enterprise Edition (J2EE),
- Maven,
- Jenkins,
- Ansible,
- GitLab,
- Artifactory.

Dans ce cadre, j'ai pu appréhender les notions globales d'un projet, et participer activement à son développement, notamment en prenant des responsabilités sur les diverses tâches qui m'ont été confiées.

Mots clés : DevOps – Industrialisation – Automatisation – Agile – API

Abstract

Résumé : Born from a strategic desire to change the culture of development, and to respond to the difficulties faced by companies in this area, DevOps is today a vital transformation challenge to ensure their competitiveness.

It is in that context that Sopra Steria, European leader in digital transformation, welcomed me for my internship, from April 23 to August 16, 2019, during which I integrated a recent and growing project, mobilizing nearly a dozen people.

The PISTE (Platform for Intermediation of Services for State Transformation) project, led by the AIFE (Agency for Financial Computing of the State), is part of the program France Connect Platform, which aims to build the digital base of the State-platform concept. This project consists in the implementation of a shared management platform of APIs of the State and the public sphere.

I was responsible for the development and implementation of DevOps practices, handling the following technologies :

— Java 2 Enterprise Edition (J2EE),
— Maven,
— Jenkins,
— Ansible,
— GitLab,
— Artifactory.

In this context, I was able to apprehend the global notions of a project, actively participate in its development, and take responsibility for the various tasks entrusted to me.

Keywords : DevOps – Industrialization – Agile – Automation – API

Remerciements

Avant d'ouvrir ce rapport, je tiens à remercier l'ensemble des personnes ayant contribué à sa réalisation, ainsi qu'à celle du stage dont il fait l'objet.

En premier lieu, je tiens à remercier l'ISTY (Institut des Sciences et Techniques des Yvelines) pour m'avoir permis d'effectuer ce stage très enrichissant, et ainsi de mettre en pratique dans un contexte professionnel les notions étudiées durant ma formation.

Je tenais également à remercier le groupe Sopra Steria, pour leur confiance et leur accueil au sein de leur entreprise pour ce stage. L'environnement et les conditions dans lesquelles j'ai pu évoluer ont renforcé ma motivation et mon investissement. Ainsi, je remercie particulièrement Matthieu JAHAN (Chef de projet et tuteur), Bruno TRINTA (Architecte), David BERTRAND (Référent technique), pour leur confiance, et leur accompagnement sur les diverses tâches que j'ai été amené à réaliser. Leur disponibilité, leur écoute et leurs précieux conseils ont grandement facilité mon intégration dans l'entreprise.

De manière plus générale, je remercie les équipes du Lot 4 et de PISTE, au sein desquelles j'ai pu évoluer, pour leur bienveillance et leur bonne humeur quotidienne.

Enfin, je souhaite remercier mon tuteur pédagogique, Mme Sondes KALLEL, pour sa disponibilité, son efficacité et sa réactivité face aux sollicitations que j'ai pu lui émettre.

Table des matières

Introduction

Objectif du document

Dans le cadre de ma deuxième année de cycle ingénieur en informatique au sein de l'ISTY (Institut des Sciences et Techniques des Yvelines), j'ai pu réaliser un stage d'une durée de 17 semaines, du 23 avril au 16 août 2019.

Ayant eu plusieurs opportunités pour ce stage, je me suis finalement tourné vers celle qui m'a été offerte par l'entreprise Sopra Steria, leader européen de la transformation digitale, du fait notamment du caractère Recherche et Développement (R&D) de la mission, qui m'intéressait particulièrement.

L'objectif de ce rapport est donc de rendre compte de mes pratiques professionnelles, en mettant en évidence ma capacité à identifier et analyser les problèmes que j'ai pu rencontrer lors de la réalisation des missions qui m'ont été confiées.

A l'heure de la rédaction de ce rapport, mon stage n'est pas encore terminé. Néanmoins, j'ai essayé de prendre un maximum de recul, afin d'avoir une vue d'ensemble sur les diverses tâches que j'ai pu réaliser. Ce rapport constitue donc une synthèse des connaissances et compétences acquises durant ce stage.

Destinataires

Ce rapport est destiné à toute personne de l'ISTY participant à mon évaluation, ainsi qu'à toutes les personnes m'ayant encadré dans le cadre de mon stage à Sopra Steria.

Confidentialité

Bien que ce rapport ne soit soumis à aucune forme de confidentialité, le projet dont il fait l'objet relevant du domaine public, j'ai tout de même pris soin de masquer des éléments que l'on pourrait qualifier de sensibles (telles que les adresses IP des machines, par exemple).

Chapitre 1

Contexte

1.1 Contexte du stage

1.1.1 Sopra Steria Group

Historique

Historiquement, Sopra Steria Group est l'une des Entreprises de Services du Numérique (ESN) parmi les plus anciennes d'Europe.
Elle est née le 31 décembre 2014, de l'opération de fusion-absorption de l'entreprise Groupe Steria (ESN fondée en 1969) par l'entreprise Sopra Group (ESN fondée en 1968).

FIGURE 1.1 – Logo de Sopra

FIGURE 1.2 – Logo de Steria

Parmi les nombreux projets, qui ont contribué au développement et à la renommée de l'entreprise, on pourrait citer :

- La participation de Steria au développement du Minitel, en 1981 ;
- Le projet d'automatisation du RER A, soumis par Steria, en 1987 ;
- La première solution de reporting bancaire pour le BAFI (Base des Agents FInanciers), par Sopra, en 1990
- L'élaboration de l'aéroport de Jakarta (capitale de l'Indonésie), par Steria, en 1993.

Aujourd'hui

Aujourd'hui encore, Sopra Steria Group s'affirme comme un des leaders européen de la transformation numérique, avec un chiffre d'affaires de 4,1 milliards d'euros en 2018, pour un résultat net de 125,1 millions d'euros (soit environ 3% du chiffre d'affaires).
Le groupe rassemble également plus de 44 000 collaborateurs, présents dans plus de 25 pays, dont environ de 19 000 en France.

Forte d'un portefeuille d'offres très complet, divisé en deux grandes catégories que sont la production de services numériques (représentant 85% du chiffre d'affaires), et l'édition de

solutions (représentant 15% du chiffre d'affaires), l'entreprise accompagne ses clients dans leur transformation digitale, en leur apportant des réponses adaptées et innovantes aux enjeux de développement et de compétitivité rencontrés.

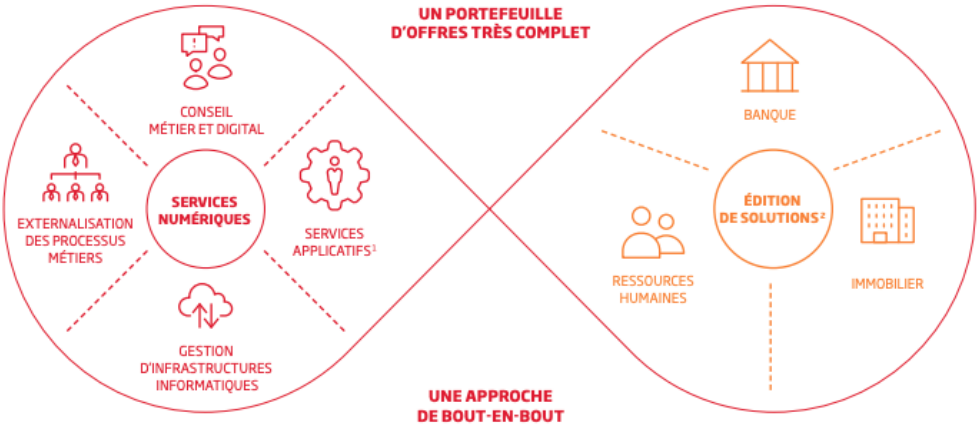

FIGURE 1.3 – Portefeuille d'offres de Sopra Steria

Secteurs d'activités

Sopra Steria intervient sur plusieurs secteurs d'activités, matérialisés en Business Unit (BU) :

- Aéronautique et spatial,
- Assurance et protection sociale,
- Banque,
- Défense,
- Distribution,
- Energie,
- Secteur Public,
- Securité intérieure,
- Télécoms, médias et jeux,
- Transport.

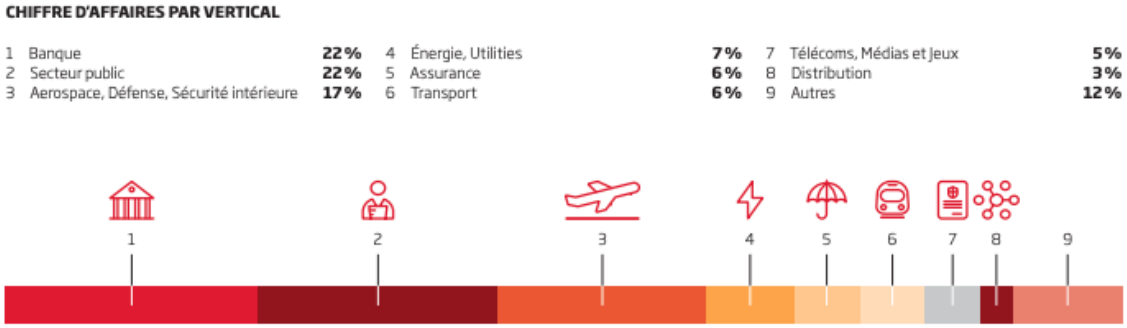

FIGURE 1.4 – Répartition du Chiffre d'Affaires

Comme nous le montre la figure 1.4, trois principaux secteurs d'activités se démarquent au sein de Sopra Steria, en terme de chiffre d'affaires notamment :

— Le secteur bancaire (22% du chiffre d'affaires)
— Le secteur public (22% du chiffre d'affaires)
— Le secteur aérospatial, défense et sécurité intérieure (17% du chiffre d'affaires).

Pour la suite de ce rapport, nous nous intéresserons particulièrement au secteur plublic, au sein duquel j'ai pu réaliser mon stage.

1.1.2 Le pôle Secteur Public

Confronté à l'obligations de suivre les évolutions réglementaires, et porté par une vague continue de réformes, le secteur public poursuit la transformation de ses métiers, de son organisation et de son offre de services aux usagers.
Pour répondre à ces besoins, Sopra Steria propose des offres de modernisation des systèmes d'informations métier, à travers des programmes de transformation digitale, et des solutions de mutualisation des principaux acteurs de la sphère publique.

Le pôle Secteur Public est subdivisé en quatre agences, regroupées par secteur d'activités et basées au sein de grandes villes françaises et étrangères.
Ces agences représentent l'identité du secteur public, et possèdent leurs propres clients (voir figure 1.5).

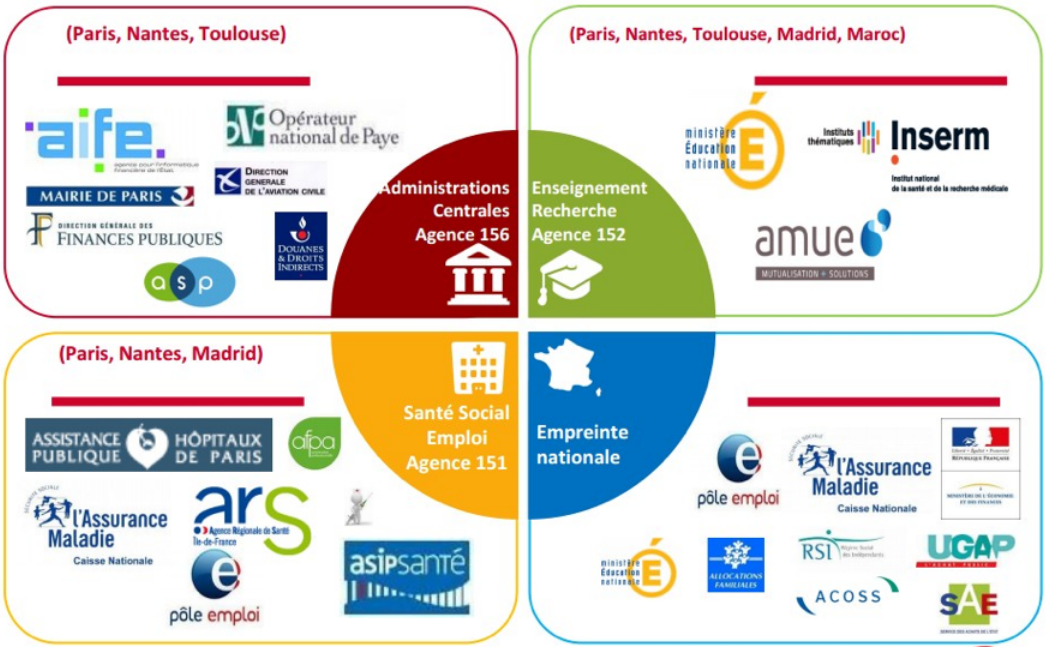

FIGURE 1.5 – Présentation du pôle Secteur Public

J'ai pu réaliser mon stage au sein de l'agence 156, dans les locaux du client, l'AIFE, basée à Noisy-le-Grand (93).

1.1.3 Le client : L'AIFE

L'AIFE (Agence pour l'Informatique Financière de l'État) est un service à compétence nationale (SCN) et à gouvernance interministérielle. Il a été créé par le décret du 11 Février 2005, puis amendé par celui du 7 Mai 2014.

FIGURE 1.6 – Logo de l'AIFE

Ses principales missions sont :
- Le pilotage de l'urbanisation du Systèmes d'Informations (SI) Financières de l'État,
- La maintenance du Systèmes d'Informations (SI) Chorus de gestion financière de l'État et de dépôt des factures électroniques de la sphère publique,
- La maintenance de PLACE, la plateforme des achats de l'État,
- Le pilotage de nouveaux projets interministériels ou ministériels et leur intégration dans le SI Chorus,
- L'accompagnement du changement au sein des ministères et auprès des utilisateurs.

1.1.4 Organisation du projet

Chorus est le nouveau Systèmes d'Informations (SI) de gestion des finances de l'Etat, au service de sa modernisation.
Mis en œuvre suite à la Loi Organique relative aux Lois de Finances (LOLF) du 1er Août 2002, ce programme phare du Ministère des Finances s'inscrit dans une démarche d'amélioration de la performance et de transparence de la gestion publique.

La maintenance du projet Chorus est gérée par un groupement entre les entreprises Capgemini et Sopra Steria, dont la responsabilité est répartie sous forme de lots (voir figure 1.7).

Chacun de ces lots est chargé de la maintenance de différentes parties du Systèmes d'Informations Chorus, ainsi :
- le lot 1 est chargé de Coeur Chorus, module SAP de la plateforme,
- le lot 2 est lui, responsable de l'Infocentre,
- le lot 3 gère la solution Chorus Pro ainsi que d'autres projets annexes,
- le lot 4 est responsable des systèmes d'échanges Chorus et Chorus Pro

Dans le cadre de ce stage j'ai évolué en tant que développeur sur le projet PISTE. Néanmoins, pour des raisons de place insuffisante dans leur bureau, j'ai été positionné au sein de l'équipe Lot 4 (les deux bureaux n'étant qu'à quelques mètres l'un de l'autre).

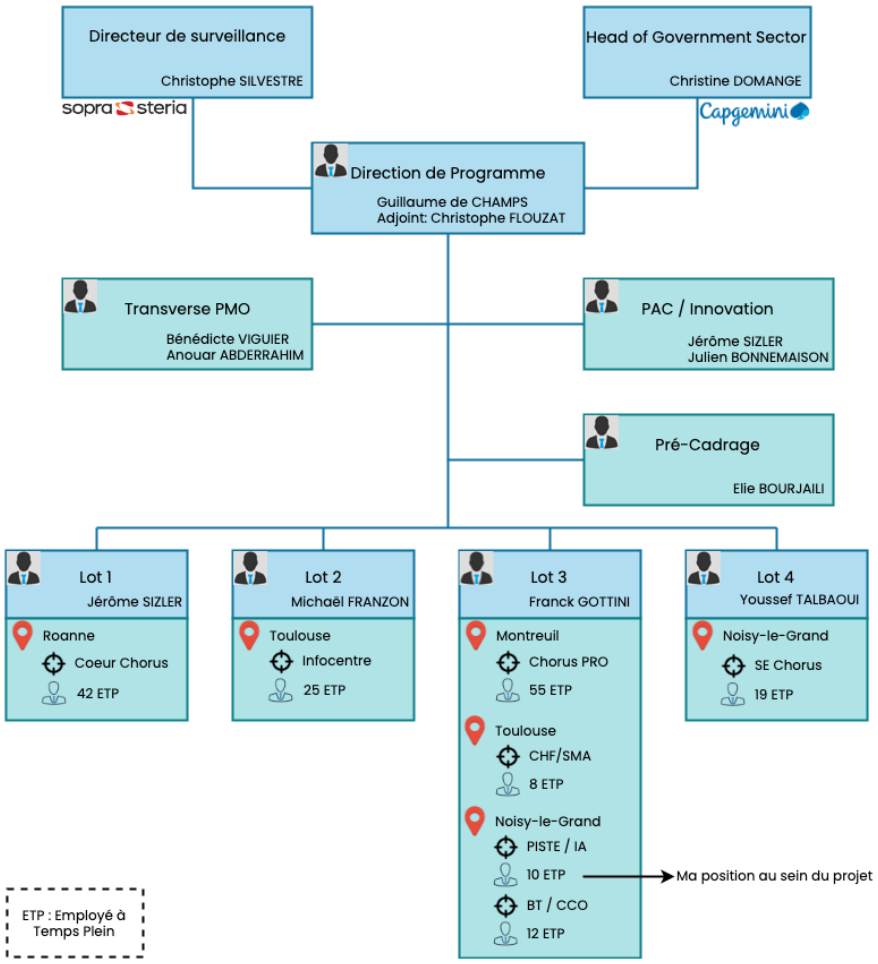

FIGURE 1.7 – Organisation du projet

1.1.5 Quotidien au sein de l'équipe

Accueil

Avec deux autres stagiaires, Lina ABOU SLEIMAN et Nicolas BLOT, nous avons rejoint Sopra Steria le 23 Avril 2019.
Nous avons été reçus dans un premier temps à la tour Manhattan (située dans le quartier de la Défense), où sont centralisées les activités de Ressources Humaines de Sopra Steria.
Durant la matinée, nous avons pu rencontrer nos tuteurs de stages respectifs, nous avons été formé sur différentes formalités administratives (notamment la saisie du compte-rendu d'activités), et nous avons également récupéré notre équipement de travail (sac à dos, ordinateur, etc.).
L'après midi, nous avons pu intégrer les équipes au sein desquelles nous allions évoluer durant la suite de notre stage, dans les locaux de l'AIFE à Noisy-le-Grand (93).

Environnement de travail

Le bureau au sein duquel j'ai été affecté est aménagé en open-space, scindé en deux parties communiquantes :

- Une première partie occupée par des Business Analysts (analystes d'affaires), chargés, entre-autres, de réaliser des analyses du besoin, de rédiger des spécifications formelles pour les développeurs, et de tester les développements ;
- Une seconde partie, occupée par des Solutions Builder (développeurs), au sein de laquelle j'ai été placé.

Réunions

Afin de prendre connaissance de l'avancée de chacun des membres de l'équipes, et de partager les éventuels points de blocages entre collaborateurs, des réunions quotidiennes à heure fixe, sous la forme de stand-up meetings (DSTUMs) ont lieu chaque matin.
Ainsi, avec les développeurs, les business analysts et le responsable d'équipe, nous nous réunissons, et prenons la parole à tour de rôle, partageant ainsi nos difficultés rencontrées, nos tâches accomplies récemment et celles à venir.
Ces réunions s'intègrent dans la mise en œuvre d'une approche Agile, plus souple et adaptée que les méthodes de gestion de projet traditionnelles, plaçant les besoins du client au centre des priorités du projet.

Chaque semaine, des réunions avec d'autres acteurs (notamment le bureau technique), appelées V1, ont également lieu. Elles ont pour but de donner une vue d'ensemble sur l'avancement du projet, et sur les prochaines évolutions annoncées.

Enfin, PISTE ayant été choisi comme projet pilote pour la mise en place de pratiques DevOps au sein de l'AIFE, j'ai eu l'occasion de participer à des réunions bi-mensuelles avec certains de leurs représentants, traitant de l'avancement de cette partie du projet.

1.2 Contexte du projet - PISTE

1.2.1 Historique - Besoin adressé

Dans le cadre du programme Chorus Pro (solution de gestion des factures), l'AIFE a mis en œuvre une plateforme de gestion d'API, opérationnelle depuis fin 2016. Partagée par l'ensemble des administrations et les entreprises, cette plateforme propose aujourd'hui plus d'une centaine d'APIs.

Lancé dans le cadre du Plan de Transformation Numérique des Ministères Economiques et Financiers, et s'inscrivant dans le cadre du programme France Connect Plateforme [1], visant à bâtir le socle numérique de l'Etat plateforme, le projet PISTE consiste en la mutualisation de cette plateforme de gestion des APIs.

1.2.2 Présentation générale

PISTE est donc une plateforme mutualisée des services API de l'État, et plus généralement des services publics, mise en place par l'AIFE (Agence pour l'Informatique Financière de l'État).

Figure 1.8 – Logo du projet PISTE

Fonctionnalités de la solution PISTE

La plateforme PISTE permet aux développeurs œuvrant pour les services publics ou non, après inscription, d'intégrer les API qui y sont exposées par des fournisseurs de l'Etat ou de la sphère publique.
La plateforme se veut simple, sûre et disponible.

Aussi, la plateforme possède les fonctionnalités suivantes :

— La **consultation de catalogues d'APIs** : le catalogue exposé dépend du compte utilisé. De la même manière, si l'utilisateur est déconnecté, seules les APIs publiques lui sont exposées.

1. Projet entendant décloisonner les données de l'administration pour offrir de meilleurs services publics numériques aux citoyens

— La **création d'applications** : Les applications clientes (utilisant une ou plusieurs APIs exposées donc), consommeront l'API au travers de l'application créée dans le portail de service PISTE.
— Le **test des APIs** : via une fonctionnalité de try-it permettant de réaliser un appel à l'API, et d'évaluer le retour, directement sur la plateforme
— La **consommation d'APIs** : au travers de la plateforme
— L'**auto-approvisionnement d'informations d'identifications** : au travers d'un système de credentials (informations d'identifications) et de clé API (API Key).
— La **consultation de l'usage des APIs**
— **L'administration des APIs**

Acteurs

Différentes catégories d'acteurs sont amenés à utiliser la plateforme :
— Les **administrateurs** : équipes de développement, de maintenance de la solution PISTE,
— Les **consommateurs** : applications qui interagissent avec la plateforme, via des appels APIs,
— Les **développeurs d'applications** : utilisateurs qui développent les applications consommatrices, et qui utilisent le portail de services de la plateforme PISTE,
— Les **fournisseurs d'APIs** : partenaires de l'AIFE qui exposent leurs APIs via la plateforme.

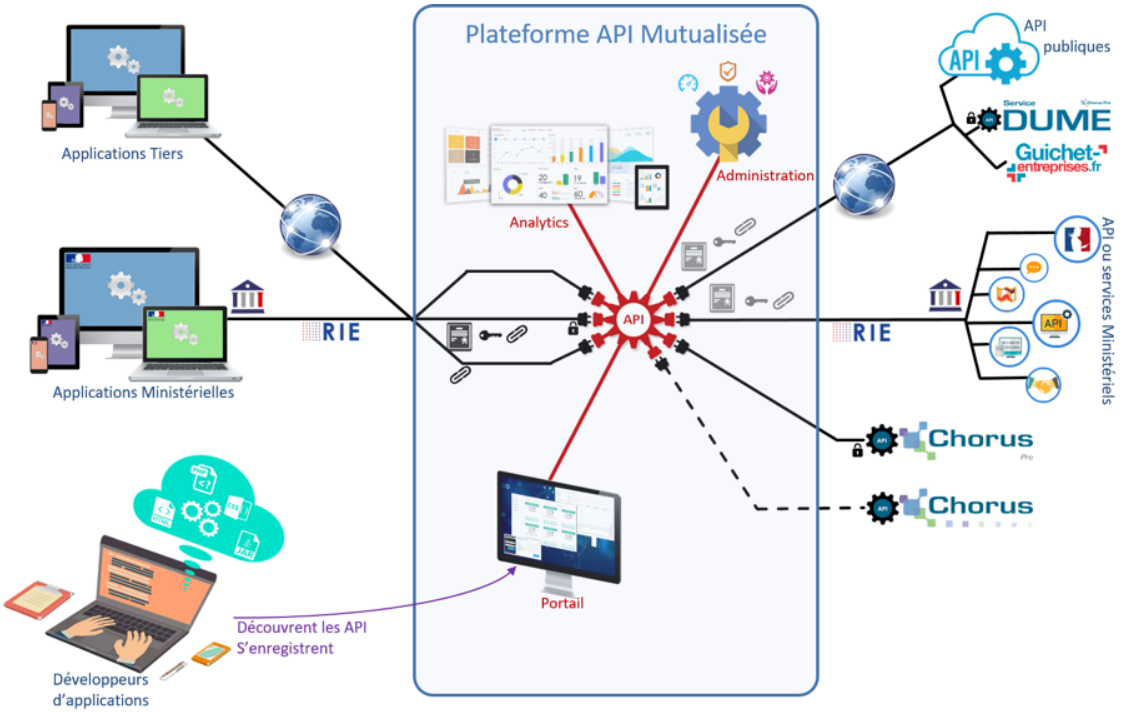

FIGURE 1.9 – Interactions sur la plateforme PISTE

1.2.3 Exigences

La plateforme PISTE, est, comme tout autre projet, soumis à un certain nombre d'exigences, qui se doivent d'être respectées. De plus, celle-ci est considérée comme un composant critique du Systèmes d'Informations (SI) de l'AIFE.

Plage de service

La plateforme doit être accessible 24 heures sur 24, 7 jours sur 7.
Même en cas d'opérations de mise en production, ou de maintenance, son taux de disponibilité ne doit pas être inférieur à 99,7%.

Volumétrie et temps de réponse

La plateforme doit être en capacité de traiter 1200 requêtes par secondes, et le temps de réponse d'une requête doit être de moins de 100ms.

Interruption et perte de données

En cas de bascule sur le site de secours, la durée maximale admissible d'interruption (DMIA) de la plateforme est de 24 heures.
La plage de perte de données maximale admissible (PDMA) est, elle, de 12 heures.

1.2.4 Architecture

L'API Management

L'API Management est un système de gestion des APIs, définissant un ensemble de moyens et de procédures, permettant :

- l'**exposition**, la **publication**, et la **protection** des API,
- la **gestion de leur cycle de vie**,
- leur **surveillance** et **audit**,
- leur éventuelle **monétisation**,
- la **collecte de données relatives à leur utilisation**, à des fins statistiques notamment

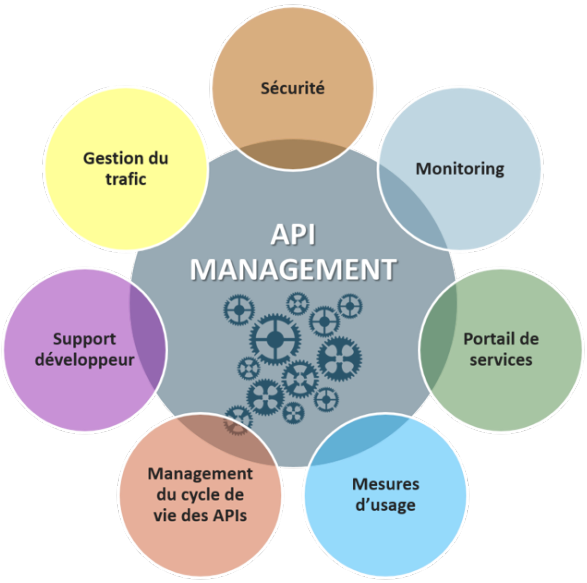

FIGURE 1.10 – Fonctionnalités d'une plateforme d'API Management

Composants

Sur PISTE, les fonctionnalités présentées sur la figure 1.10 sont rendues possible grâce aux composants suivants :

Composant	Produit	Fonctionnalité d'API Management
Portail de services	API Portal (Axway)	Portail de services, Support développeur
API Gateway	API Gateway (Axway)	Exposition, Sécurité, Mesures d'usages
Portail d'administration des Gateway	API Gateway (Axway)	Monitoring, Gestion du trafic
Portail d'administrations des APIs	API Gateway (Axway)	Management du cycle de vie des APIs
Portail de supervision	Embedded Analytics (Axway)	Monitoring, Mesures d'usages
Unité de calcul	Embedded Analytics (Axway)	
Transmission de données	Filebeat (Elastic)	

TABLE 1.1 – Liste des composants de la plateforme PISTE

Comme nous le montre le tableau ci-dessus, les solutions logicielles utilisées sont principalement produites par l'éditeur Axway.
Ainsi, les deux principaux composants de la plateforme PISTE sont :

- L'API Portail : Un portail web, basé sur le Système de Gestion de Contenu (SGC) *Joomla !*, et implémenté en PHP. A disposition des développeurs, ce portail permet les fonctionnalités suivantes :
  - la création de compte et la gestion de profil,
  - la consultation des catalogues d'APIs,
  - la gestion d'applications (via des credentials)
- L'API Gateway : Une solution logicielle assurant l'exposition des APIs et la sécurisation de la plateforme.
  Les API Gateways fonctionnent en haute disponibilité, sur plusieurs serveurs, et exécutent des policies (politiques d'utilisations) définies par les développeurs.

Dans le cadre de mon stage, mes travaux ont essentiellement porté sur ce dernier composant.

Interaction avec les composants

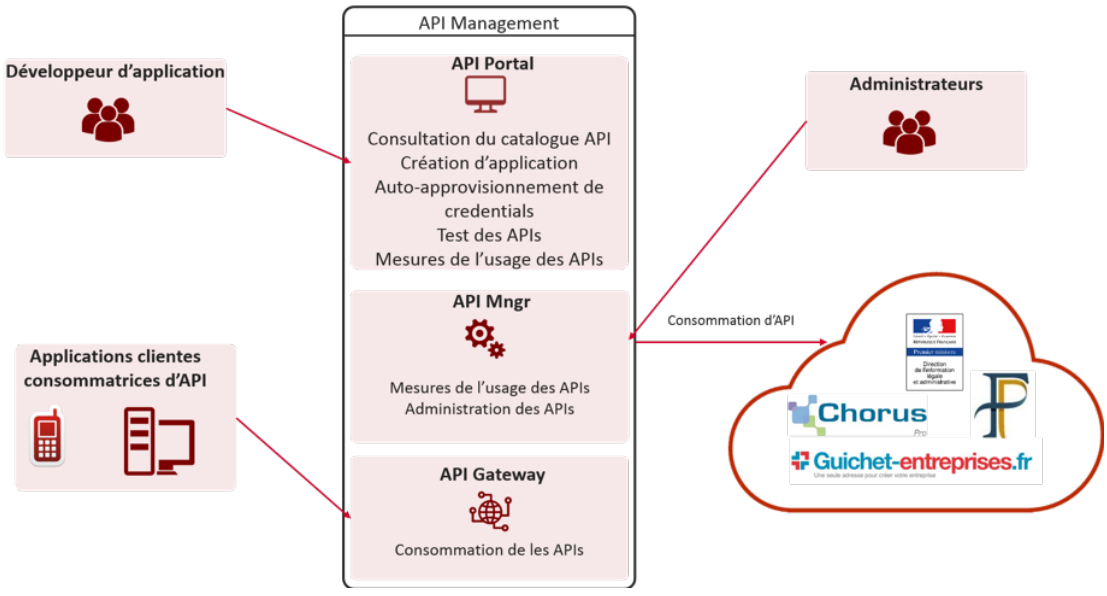

FIGURE 1.11 – Schéma représentant les interactions utilisateur - composant sur la plateforme

Chapitre 2

Missions

Ce chapitre traite de ma principale mission dans le cadre du stage : la mise en place d'une chaîne d'outils DevOps sur PISTE.
En premier lieu, il convient de définir au mieux la notion de DevOps, avant de considérer la mise en oeuvre de pratiques DevOps au sein de PISTE. Enfin, je clorai ce chapitre en détaillant les différents travaux que j'ai pu effectuer dans le cadre de ma mission.

2.1 Le DevOps

2.1.1 Définition

Le terme DevOps est issu de la contraction des deux mots anglais Development (développement) et Operations (exploitation).
Il est apparu en 2009, lors d'une conférence intitulée *DevOpsDays*, menée par un consultant en informatique belge, Patrick Debois, et visant à améliorer la collaboration jusqu'alors étroite entre les équipes de développements (Devs) et les équipes d'exploitations (Ops).
En effet, cette relation est souvent symbolisée par un mur de la confusion (figure 2.1, séparant les Devs, constamment à la recherche de changements (nouvelles fonctionnalités, nouvelles technologies), des Ops, à la recherche, eux, de stabilité.

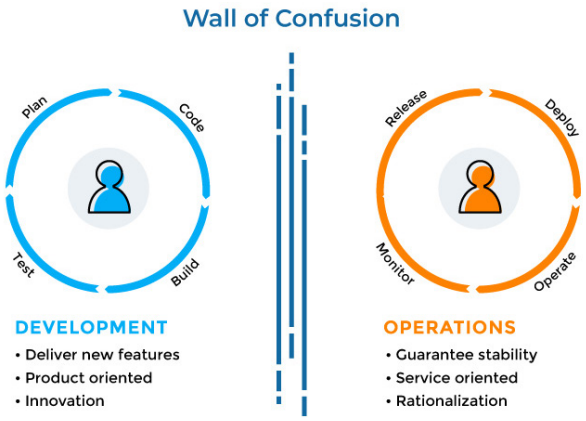

FIGURE 2.1 – Le mur de la confusion séparant les Devs des Ops

Le DevOps vise donc à créer une culture et un environnement au sein desquels la conception, les tests et la diffusion de logiciels peuvent être réalisés rapidement, fréquemment et

efficacement. Il ne s'agît donc pas d'une simple méthodologie, mais d'une réelle philosophie de travail.

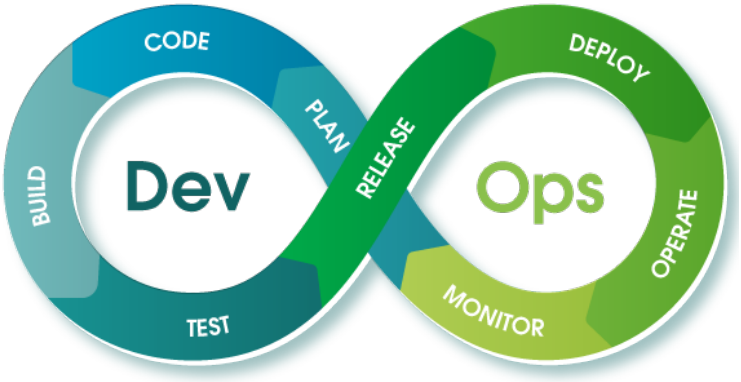

FIGURE 2.2 – Le DevOps

2.1.2 Principes piliers

Il n'existe pas encore réellement de référentiel de bonnes pratiques sur lesquelles s'appuyer pour implémenter le DevOps. Néanmoins, cinq pilliers sont communément acceptés : les C.A.L.M.S (Culture Automation Lean Measurements Sharing).

Culture

Le DevOps requiert un changement de culture organisationnelle, particulièrement difficile du fait du mur de la confusion mentionné précédemment.
L'idée est donc de favoriser la collaboration transverse.

Automation (automatisation)

Les tâches manuelles répétitives sont souvent peu rentables et peuvent occasionner des erreurs.
Pour pallier à cela, il faut automatiser le plus possible les tâches, en exploitant des processus reproductibles qui créent des systèmes fiables.

Lean (rationalisation)

L'élimination des activités à faible valeur ajoutée fait également partie intégrante de la culture DevOps.
Il faut donc identifier, en permanence, des opportunités d'amélioration continue.

Measurements (mesures)

Afin d'évaluer les possibles améliorations, il faut être en capacité de les mesurer. Il existe pour cela de nombreux outils et technologies.
Ainsi, il peut être intéressant de mesurer l'utilisation d'un produit, la fréquence d'apparition de bugs ou défaillances, etc.

Sharing (partage)

Toujours dans l'optique d'outrepasser le mur de la confusion, le partage exprime la nécessité d'une communication continue entre les équipes de développements et celles d'exploitations. On parle ici de partage des résultats, des données et des analyses, mais également de partage des responsabilités, aussi bien en situation d'échec que de succès.

2.2 Le DevOps sur PISTE

2.2.1 Contexte

Actuellement, sur PISTE, et sur la quasi-totalité des projets de l'AIFE, la mise en place de pratiques DevOps est presque inexistante.

Du fait de sa récence, et de sa petite échelle, l'AIFE a donc choisi PISTE, comme projet pilote pour la mise en place d'une chaîne d'outils DevOps, afin d'en estimer la faisabilité, le temps, le coût, et les risques avant d'étendre cela à d'autres projets sous leur responsabilité.

C'est donc dans ce contexte que sont effectuées des réunions bi-mensuelles avec des responsables de l'AIFE, afin de leur offrir une visibilité sur les avancements effectués, les éventuels points de blocages rencontrés, et d'étudier ensemble les solutions à y apporter.

2.2.2 Outils

GitLab

GitLab est un outil de gestion de dépôt Git [1] offrant de nombreuses fonctionnalités supplémentaires, parmi lesquelles un système de développement et d'intégration continue, un système de suivi des bugs, et un wiki. [2]

Sur PISTE, GitLab est utilisé pour :
— versionner le code source de l'applicatif et les scripts de déploiement,
— la documentation, via le wiki intégré (voir figure 2.3).

1. Logiciel de gestion de versions décentralisé
2. Application web permettant la création, la modification et l'illustration collaborative de pages web

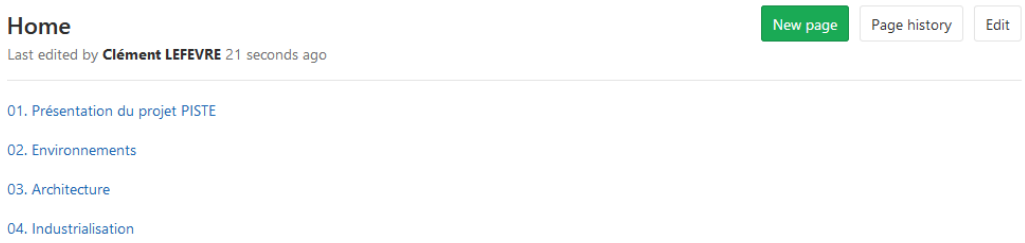

FIGURE 2.3 – Organisation du wiki de PISTE

L'arborescence est organisée de manière à respecter la règle d'un sous-dépôt par brique applicative. On a donc un sous-dépôt pour l'API Gateway et un second pour l'API Portal.
Dans le cadre de ma mission, j'ai principalement été amené à travailler sur le dépôt de l'API Gateway.

Artifactory

JFrog Artifactory est un gestionnaire de dépôt d'objets binaires (binary repository manager en anglais).
Il permet donc de gérer de manière pérenne, distribuée et sécurisée des artefacts (livrables), issus de n'importe quelle technologie, et surtout, de les versionner.

Sur PISTE, deux dépôts sont utilisés :
- `piste_generic_development`, contenant les binaires et livrables propres à la plateforme.
- `external_maven_release`, commun à tous les projets de l'AIFE, et contenant les dépendances Maven [3] utilisées par les différents programmes Java.

Ansible

Ansible est un logiciel open source permettant la réalisation de déploiements, l'exécution de tâches et la gestion de configurations sur une ou plusieurs machines simultanément.
Il est sans agents (ne nécessite pas de serveur), et accède aux machines à distance via le protocole SSH.

Ansible permet donc de créer des playbooks [4], décrivant les tâches à accomplir sur les différentes machines.
Au sein des playbooks sont définis les hôtes (machines), les éventuelles variables, et les tâches à exécuter. Chaque tâche possède un nom, et utilise un module Ansible (par exemple le module `copy`, permettant de copier des fichiers sur une ou plusieurs machines distantes).

Sur PISTE, Ansible est utilisé pour l'installation des composants et le déploiement de livrables. Une machine est entièrement dédiée à son utilisation.

3. Principalement des librairies Java
4. Scripts utilisant la syntaxe YAML, simple et aisément compréhensible

Jenkins

Jenkins est un outil open source d'intégration continue. Relativement simple à utiliser, et fort d'une communauté très nombreuse et active, il possède de nombreuses extensions (plus de 1000 plugins), qui le rendent particulièrement polyvalent.

De manière générale, le fonctionnement d'un pipeline Jenkins est le suivant :

1. Un ou plusieurs développeur(s) modifie(nt) le code source de l'application, et pousse(nt) la modification sur le gestionnaire de code source (ici GitLab).
2. Jenkins détecte le changement de code source, et prépare un nouveau build.
3. Si le build rencontre une erreur, le(s) développeur(s) sont notifiés, sinon, des tests automatisés sont effectués.
4. Une fois les tests effectués, Jenkins génère un message de retour, et notifie le ou les développeur(s).

Sur PISTE, Jenkins permet d'orchestrer les différentes chaînes d'outils automatisées.

2.2.3 Enjeux

Les enjeux de la mise en place d'une chaîne d'outils DevOps sur PISTE font écho aux principales douleurs rencontrées à l'heure actuelle sur le projet, à savoir :

— des installations et des configurations manuelles des produits, engendrant des écarts entre les différents environnements,
— des livraisons fréquentes (toutes les deux semaines) et des déploiements manuels fastidieux, complexes et chronophages, engendrant un risque d'erreurs important,
— un code source applicatif non-versionné, qui rend compliqué les retours en arrière en cas d'erreurs.

Les réponses qui doivent être apportées par la chaîne d'outils DevOps sont donc :

— des installations automatisées, rapides et uniformes, afin d'avoir des environnements alignés en terme de configuration,
— des déploiements automatisés des livrables, offrant à la fois un gain de temps, et de sécurité,
— une historisation du code source (sur GitLab) et des binaires (sur Artifactory).

2.3 Travaux effectués

Au sein de cette partie sont décrits les différents travaux que j'ai pu réaliser dans le cadre de mon stage.

Comme mentionné précédemment au sein de ce rapport, les missions qui m'ont été confiées gravitent principalement autour du composant API Gateway d'Axway.
En effet, ma mission était de réaliser une chaîne d'industrialisation, allant de la génération de livrables à partir de sources, jusqu'à leur déploiement sur un environnement spécifique, en passant par leur environnementalisation.

Cette partie vise donc à décrire chacun des maillons de cette chaîne d'industrialisation que j'ai pu mettre en place. Pour chacun d'eux, j'évoquerais le besoin énoncé, sa conception, et sa réalisation, ainsi que les éventuelles problématiques rencontrées, et les solutions mises en places pour les surmonter.

2.3.1 Création du livrable à partir des sources

(a) Besoin

Comme mentionné lors de la présentation des composants de PISTE, les API Gateway sont des composants exécutant des policies (politiques d'utilisations), définies par les développeurs du projet.
Ces policies peuvent être générées par le biais de l'IDE (Environnement de développement) Policy Studio, également produit par Axway.
Au stade de développement, ces policies se présentent sous forme de projet, correspondant concrètement à un dossier, composé de plusieurs fichiers au format XML.
Afin d'être déployé sur les API Gateway, ces policies doivent être transformées sous forme d'archives :

- Une archive au format `.pol`, contenant les données relatives au policies (politiques d'utilisations).
- Une archive au format `.env`, contenant les données relatives à l'environnement (paramétrages particuliers liés à l'environnement).

Auparavant, la transition de l'état de projet à celui d'archive était effectuée par un script Jython [5] intitulé `projpack.py`, qui prenait en paramètres le dossier contenant les sources XML, ainsi que diverses options (nom de l'archive générée, passphrase, etc.).
Cependant, ce script n'était présent que sur certaines machines virtuelles du projet, et contenait de nombreuses dépendances, pour divers cas d'utilisations inexploités dans le cadre du projet.
De plus, la chaîne était peu pertinente, puisqu'il était nécessaire d'exécuter ce script depuis une machine de l'environnement, puis de déposer les livrables générés sur Artifactory afin de les redéployer sur l'ensemble des machines de l'environnement.

5. Interprète Java écrit en Python, offrant de nombreuses fonctionnalités, dont celle d'utiliser des objets Java dans un code Python.

La première mission qui m'a été confiée dans le cadre de mon stage a donc consisté à ré-implémenter une version simplifiée et allégée de ce script Jython en un exécutable Java (JAR), dont les sources seront stockées sur un dépôt GitLab et ainsi versionnées.

(b) Conception

Pour cette première mission, j'ai évolué en autonomie, avec l'appui de mon référent technique, M. David BERTRAND.
N'étant arrivé que très récemment au sein de l'entreprise, j'ai naturellement eu besoin d'un temps d'adaptation, durant lequel je me suis documenté sur le composant API Gateway et l'IDE Policy Studio.

J'ai ensuite dû adopter une démarche de rétro-ingénierie[6], afin de comprendre le fonctionnement du script `projpack.py`. En effet, il aurait été improductif de simplement recopier le script, en modulant uniquement la syntaxe du Python vers celle du Java, d'autant que la version à produire se devait d'être simplifiée et allégée.
Il était donc nécessaire de comprendre la quasi-totalité de la logique derrière le script original.

Concernant les technologies utilisées, la seule contrainte m'ayant été fixée était l'utilisation du langage Java. J'ai donc pu utiliser les librairies de mon choix, sous réserve qu'elles soient open-source et fassent l'objet d'un suivi relativement régulier.
Aucun IDE ne m'ayant été imposé pour réaliser mes développements, je me suis tourné vers IntelliJ de JetBrains, que j'avais déjà eu l'occasion d'utiliser lors de mon stage de IATIC3 l'an dernier.
Enfin, j'ai également utilisé l'outil open-source Maven, permettant de faciliter la gestion et la construction d'un projet Java.

(c) Réalisation

Parsing

Après avoir relativement étudié le script en lui-même, j'ai commencé par ré-implémenter la partie parsing[7], qui se trouve également être la première partie abordée dans le script original.

Initialement, le script pouvait prendre de nombreuses options en paramètres, dont la plupart n'étaient jamais utilisées dans le cadre du projet.
J'ai donc réduit les options à celles utilisées, en consultant mon référent technique. Ainsi, sur la version réimplémentée en Java, les options sont désormais :

- `--name` : le nom des archives `.pol` et `.env` générées – requise
- `--dir` : le chemin vers un dossier projet (contenant des sources) – requise
- `--version` : la version du livrable généré – requise

Pour implémenter cette partie, j'ai choisi d'utiliser la librairie `commons-CLI` d'Apache, qui offre un système de parsing relativement simple à mettre en œuvre, semblable à celui que l'on

6. Etude d'un objet afin d'en déterminer le fonctionnement interne
7. Analyse de la ligne de commandes

peut trouver sur les commandes des systèmes Unix [8].
J'ai également fait en sorte de renvoyer une erreur en cas de mauvais usage des options, tout en affichant un message d'aide contenant l'usage préconisé.

Génération des livrables

La génération des livrables `.pol` et `.env`, qui est la principale partie du script, était gérée par des objets Java, dépendances auxquelles je n'avais alors pas accès, puisque seul le script Jython m'avait été fourni. J'ai donc rencontré ici un premier point de blocage, que j'ai pu surmonter rapidement en allant récupérer les fichiers JAR de dépendances sur une des machines virtuelles.

Une fois les fichiers récupérés, je pensais alors pouvoir récupérer les sources (celles-ci peuvent être inclues au sein du JAR), mais ce n'était pas le cas. Pour pallier à cela, j'ai donc dû trouver une autre alternative : la décompilation. Cette fonctionnalité, offerte notamment par l'IDE IntelliJ, permet de reconstituer le code source ayant permis la génération d'un exécutable (ici au format JAR).
Toutefois, cette pratique n'est pas toujours autorisée, puisqu'elle dépend de la licence d'utilisation du logiciel. J'ai donc dû consulter mon référent technique, qui, après s'être renseigné sur cette licence, m'a autorisé à décompiler.

J'ai donc dû décompiler et ré-implémenter un grand nombre de classes Java (environ 70), organisées en packages, tout en évaluant pour chacune d'elles sa nécessité, ainsi que celle de ses attributs et fonctions.

Gestion des dépendances `.so`

Après avoir grandement avancé sur la partie précédente, j'ai rencontré un autre point de blocage, concernant l'utilisation, de librairies `.so` (shared object, ou librairie partagée).
Une librairie partagée est destinée à être associée aux programmes au moment où ils sont exécutés. Par exemple, dans le code Java initial, le chargement de la bibliothèque était effectué via l'appel `System.loadLibrary(libname)`, puis, les fonctions contenues dans cette librairie pouvaient être directement appelées depuis le code Java.

Néanmoins, ces librairies sont relativement coûteuses à charger et à configurer (notamment à cause d'un système de liens symboliques), et les conserver impliquerait, dans l'absolu, de devoir les re-configurer à chaque exécution du programme. [9]

Après analyse, il est apparu que ces librairies étaient chargées pour un encodage de données en base64 [10], qui n'était pas utilisé dans le cadre du projet. Il m'a donc été conseillé de supprimer l'utilisation de ces librairies au sein des classes Java ré-implémentées. Néanmoins, si les besoins du projet étaient amenés à évoluer vers une utilisation de cet encodage, il serait tout à fait possible de ré-implémenter cette partie directement en Java.

8. Forme : `-o value` ou `--longOption=value`
9. Le workspace Jenkins, sur lequel est exécuté le programme, est effacé après chaque job
10. Codage de l'information utilisant 64 caractères

Amélioration de l'expérience utilisateur - logs

Enfin, pour améliorer l'expérience utilisateur/testeur/développeur, il m'a été demandé de mettre en place des messages de logs de différents niveaux sur l'ensemble du programme.
Pour cela, j'ai choisi d'utiliser la librairie `log4j` d'Apache, qui propose de nombreuses options, dont celle de stocker les logs au sein de fichiers, tout en personnalisant l'affichage (choix des champs et du format).

Pour ma part, j'ai choisi de personnaliser les logs de manière à ce qu'ils s'affichent à la fois dans la console, lors de l'exécution, mais également qu'ils soient stockés dans un fichier HTML (figure ci-dessous), ce qui permet d'assurer une certaine traçabilité.
J'ai choisi de conserver les champs par défaut, ceux-ci étant amplement suffisants dans le cadre de ce programme.

Time	Thread	Level	Logger	File:Line	Message
1000	main	INFO	MergeAPIProjects	MergeAPIProjects.java:755	Merging command line package properties(pol and env) with properties from api projects.
1000	main	INFO	MergeAPIProjects	MergeAPIProjects.java:756	**********
1001	main	INFO	MergeAPIProjects	MergeAPIProjects.java:784	Finished merging command line package properties(pol and env) with properties from api projects.
1001	main	INFO	MergeAPIProjects	MergeAPIProjects.java:785	**********
1002	main	DEBUG	ProjectPacker	ProjectPacker.java:178	[Builder] OK
1002	main	INFO	ProjectPacker	ProjectPacker.java:180	Successfully setted up the projectloader & the builder
1003	main	INFO	ProjectPacker	ProjectPacker.java:76	Building project...
1373	main	DEBUG	MergeAPIProjects	MergeAPIProjects.java:546	Temp dir created at path : C:\Users\CLELEF~1\AppData\Local\Temp\7763879426020711753
1374	main	DEBUG	MergeAPIProjects	MergeAPIProjects.java:547	Exists ? true
1375	main	DEBUG	MergeAPIProjects	MergeAPIProjects.java:548	Is directory ? true
1375	main	DEBUG	MergeAPIProjects	MergeAPIProjects.java:549	Able to write/read ? true / true
1376	main	DEBUG	MergeAPIProjects	MergeAPIProjects.java:550	**********

FIGURE 2.4 – Extrait des logs

Une des principales difficultés de cette partie a résidé dans le choix des niveaux de logs, qui se devait d'être le plus pertinent possible, afin de ne pas induire en erreur le développeur/testeur/utilisateur, notamment sur le niveau de criticité. Ce niveau peut d'ailleurs être configuré, pour les différents cas d'utilisations.
Ainsi, j'ai utilisé les niveaux de logs suivants, par ordre croissant de criticité :

- **DEBUG** - Message de débogage du programme, utile pour le développeur et le testeur notamment
- **INFO** - Message d'information sur le déroulement du programme
- **WARN** - Message rapportant un avertissement (non-bloquant)
- **ERROR** - Message rapportant une erreur (non-bloquante)
- **FATAL** - Message rapportant une erreur fatale (bloquante)

(d) Conclusion

Après plusieurs tests réalisés par moi-même, sur des projets sources m'ayant été fourni par mon référent technique, le programme s'est révélé fonctionnel. Il m'a fallu consacrer environ deux semaines de temps à cette mission. L'approche choisie par mes responsables s'est donc révélée pertinente, dans la mesure où développer un outil entier aurait pris plusieurs mois.

Du fait de certaines contraintes à respecter (notamment celle de la logique de l'éditeur Axway à conserver), le programme Java résultant demeure relativement conséquent (plus de

70 classes Java).

Après plusieurs tests effectués en environnement réel, il apparaît que l'outil développé et simplifié est en moyenne 1,6 fois plus rapide que le script original.

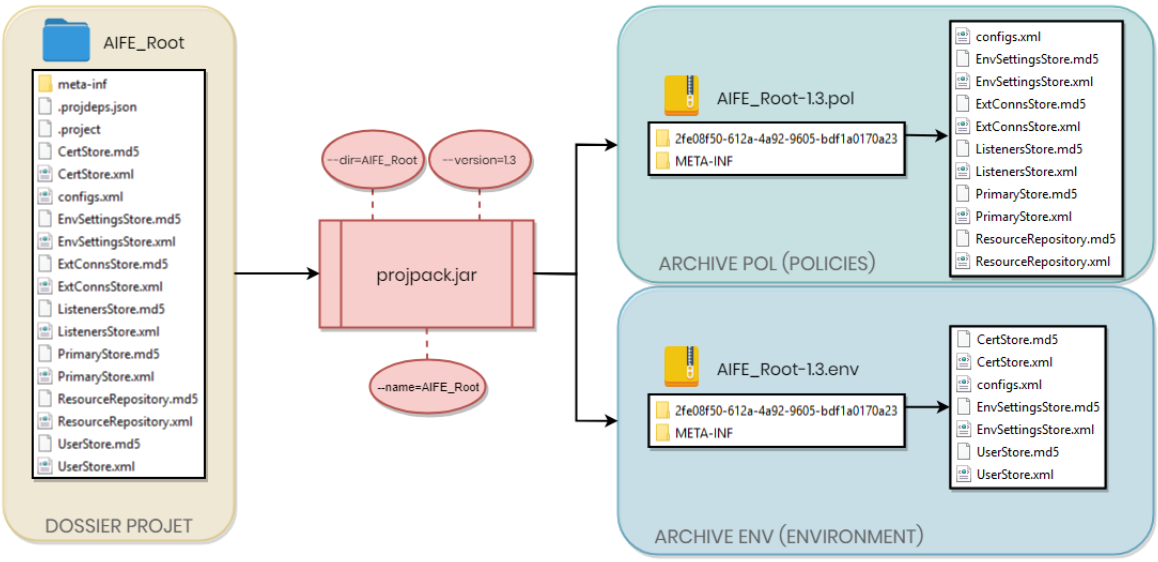

FIGURE 2.5 – Schéma de synthèse du programme projpack

2.3.2 Environnementalisation

(a) Besoin

Une fois l'outil de création de livrable développé, une seconde mission, succédant de manière logique à la première, m'a été confiée.
En effet, à la fin de la première étape de la chaîne d'industrialisation, deux archives sont générées :

— L'archive `.pol`, contenant les politiques d'utilisations
— L'archive `.env`, contenant les données relatives à l'environnement de l'API Gateway.

Or, le contenu de l'archive `.env` diffère selon l'environnement sur lequel le déploiement est effectué. De fait, afin de rendre le déploiement le plus générique possible, il apparaissait nécessaire de spécialiser l'archive `.env` en fonction de l'environnement désiré.

L'éditeur Axway avait préalablement fourni un script Python (intitulé `generate_env.py`), chargé de cela, mais, toujours dans une volonté d'uniformiser les technologies utilisées au sein du projet, il m'a été demandé de ré-implémenter ce script sous forme d'exécutable Java (JAR).

(b) Conception

Comme pour la première mission, il m'a fallu adopter dans un premier temps une démarche de rétro-ingénérie, afin d'étudier le fonctionnement du script Python.

Toutefois, ce dernier était moins complexe que le précédent, puisqu'il n'utilisait pas d'objets Java.

L'idée était donc de développer un programme Java, prenant en paramètres un fichier `.env` générique, ainsi qu'un fichier `.properties` (contenant l'ensemble des variables propres à l'environnement), et générant un fichier `.env` spécialisé.

J'ai évolué en autonomie sur cette tâche, avec l'appui de mon référent technique, M. David BERTRAND, et de M. Bruno TRINTA, architecte sur le projet.

(c) Réalisation

Après étude, le script original effectuait les actions suivantes :

1. Décompression de l'archive `.env` passée en paramètre
2. Récupération du fichier `EnvSettingsStore.xml`, présent dans l'archive contenant les différents paramètres liés à l'environnement
3. Remplacement des clés dans le fichier `EnvSettingsStore.xml` par les valeurs du fichier `.properties` passé en paramètre
4. Re-compression de l'archive `.env` (et éventuellement renommage).

Le seul fichier de l'archive .env à modifier était donc `EnvSettingsStore.xml`, contenant uniquement des balises `<entity>`, tel que l'extrait ci-dessous :

```
<entity xmlns="http://www.vordel.com/2005/06/24/entityStore"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" type="EnvironmentalizedFieldString"
    entityPK="-8860844603398104038" parentPK="-4861299783025092970">
   <fval name="displayName"><value>Read consistency level:</value></fval>
   <fval name="entityFieldName"><value>throttlingReadConsistencyLevel</value></fval>
   <fval name="index"><value>0</value></fval>
   <fval name="multiline"><value>0</value></fval>
   <fval name="value"><value>ONE</value></fval>
</entity>
```

Listing 2.1 – Extrait du fichier EnvSettingsStore.xml original

Toutefois, il était impossible d'identifier efficacement à quelle clé chaque balise `<entity>` correspondait. Après concertation avec des membres d'Axway, il est apparu nécessaire de remplacer les valeurs manuellement dans `EnvSettingsStore.xml`, selon la nomenclature suivante : `key.type.env.store.name`, où `type` correspond à `text`, `number` ou `password`. Ainsi, la balise précédente devait être transformée en :

```
<entity xmlns="http://www.vordel.com/2005/06/24/entityStore"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" type="EnvironmentalizedFieldString"
    entityPK="-8860844603398104038" parentPK="-4861299783025092970">
   <fval name="displayName"><value>Read consistency level:</value></fval>
   <fval name="entityFieldName"><value>throttlingReadConsistencyLevel</value></fval>
   <fval name="index"><value>0</value></fval>
   <fval name="multiline"><value>0</value></fval>
   <fval name="value"><value>key.text.env.store.throttlingReadConsistency</value></fval>
</entity>
```

Listing 2.2 – Extrait du fichier EnvSettingsStore.xml générique

La solution mise en oeuvre a donc été de décompresser l'archive `.env`, de modifier toutes les valeurs du fichier `EnvSettingsStore.xml` (comme ci-dessus), puis de la recompresser, obtenant ainsi une archive `.env` générique. Néanmoins, cette solution n'est pas pérenne, puisqu'en cas de modification de données liées à l'environnement au sein des projets sources, il sera nécessaire de réitérer cette manipulation. Une solution alternative est donc en cours de conception par les consultants Axway rattachés à PISTE.

Malgré cela, j'ai pu poursuivre le développement du programme Java chargé de l'environnementalisation. Celui-ci possède quatre classes :

- `Main`, point d'entrée du programme
- `Parser`, chargée d'analyser la ligne de commande, de contrôler les options passées au programme, et de vérifier l'existence des fichiers
- `Environmentalizer`, chargée d'effectuer, entre autres, le remplacement des clés par les valeurs du fichier `.properties`.
- `ZipFileUtil`, permettant de décompresser et de re-compresser l'archive `.env`

Le programme effectue donc les actions présentées sur la figure 2.6, en remplaçant les clés du fichier `EnvSettingsStore.xml` générique par leurs valeurs dans le fichier `.properties`.

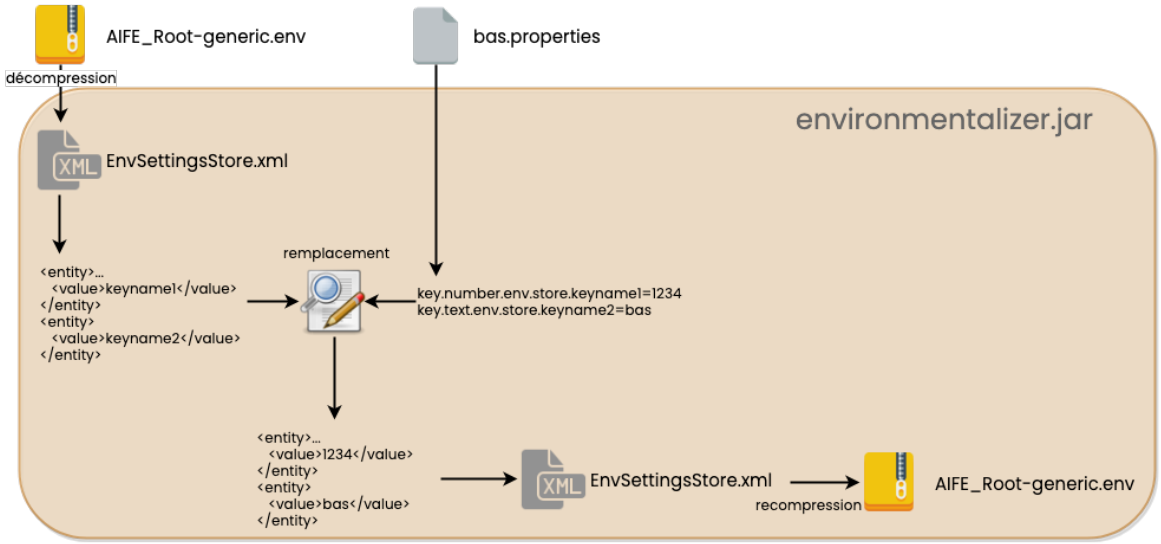

FIGURE 2.6 – Spécialisation de l'archive .env

(d) Conclusion

Bien que la solution de générisation de l'archive `.env` ne soit que temporaire, la réalisation de cette tâche s'est relativement bien déroulée.
Le projet Java développé était bien plus réduit que lors de la tâche précédente, j'y ai naturellement consacré moins de temps (environ deux journées). Néanmoins, cette partie m'a permis de monter en compétences sur les technologies Java et Maven.

2.3.3 Déploiement

(a) Problématique

Une fois les livrables `.pol` et `.env` générés et le `.env` spécialisé, le dernier maillon de la chaîne d'industrialisation consistait en le déploiement de ces livrables sur les API Gateways de l'environnement spécifié.
Le déploiement peut s'effectuer sur différents environnements :

- **BAS** (Bac à sable)
- **DEV** (Développement)
- **TST** (Test)
- **REC** (Recette)
- **PROD** (Production)

A chacun de ces environnements correspondent des architectures et des configurations de machines différentes.
Le besoin énoncé était donc de déployer les livrables `.pol` et `.env` (spécialisé) d'une version donnée, sur un environnement spécifié, via Ansible.

(b) Conception

Dans un premier temps, j'ai du monter en compétences sur ce langage, en apparence relativement simple, mais qui requiert le respect d'un certain nombre de bonnes pratiques, notamment en ce qui concerne la gestion des variables et de leur portée.

L'idée était ici de réaliser un playbook [11], chargé d'effectuer un déploiement exhaustif sur les API Gateways d'un environnement spécifié. On parle ici de déploiement exhaustif dans la mesure où l'on ne déploie pas uniquement les livrables `.pol` et `.env`, générés précédemment, mais également d'autres éléments, liés au fonctionnement de l'API Gateway, tels que :

- un fichier `envSettings.props`, contenant des variables liées à l'environnement (et indépendantes de celles du `.env`),
- un fichier `app.config`, contenant des variables de configuration,
- des templates d'e-mails, au format HTML.

Afin de rendre le développement le plus générique possible, il a été convenu de définir un rôle indépendant pour chacun de ces éléments (soit au total quatre rôles), appelés successivement par le playbook.

Après répartition des tâches, la réalisation du playbook ainsi que celle du rôle chargé de déployer les livrables `.pol` et `.env` sur les machines de l'environnement, m'ont été confiées.

Suite à une réflexion et une concertation avec mon référent technique, et du fait d'une contrainte liée à Ansible, j'ai dû distinguer le développement du rôle de déploiement des livrables en deux sous-rôles :

- `checkout.apigw`, s'exécutant sur la machine Ansible (en local donc), chargé de récupérer les archives depuis Artifactory, et de spécialiser l'archive `.env` en fonction de l'environnement, grâce au programme précédemment développé.

11. Script Ansible permettant d'automatiser des tâches séquentielles

— `deployment.apigw`, s'exécutant sur les machines de l'environnement spécifié, effectuant le déploiement en appelant un script prévu à cet effet (`projdeploy.sh`).

(c) Réalisation

[checkout.apigw] Récupération des archives

La récupération des deux archives (`.pol` et `.env` générique), et du programme d'environnementalisation s'effectue simplement au moyen du module `get_url` d'Ansible.

```
# tasks file for checkout.apigw
- name: Recuperation du POL
  get_url:
     url: "{{ artifactory_url }}/{{ basename }}-{{ version }}.pol"
     dest: "{{ dest_dir }}"
     headers:
        X-JFrog-Art-Api: UneCleAPI
```

Listing 2.3 – Téléchargement de l'archive .pol depuis Artifactory

La seule particularité notable est ici l'utilisation d'une clé API, pour l'authentification sur Artifactory, qui est une alternative pertinente et sécurisée à la saisie de mots de passe en clair. La procédure est la même pour les trois artefacts à télécharger. Afin d'assurer plus de modularité, les noms de variables sont définis dans un dossier `defaults` :

```
# defaults file for checkout.apigw
artifactory_url: https://artifactory.aop.chorus.aife/artifactory/piste_generic_developpement
basename: AIFE_Root
dest_dir: /data/ansible/template_project/roles/deployment.apigw/files
```

Listing 2.4 – Téléchargement de l'archive .pol depuis Artifactory

Suite à ce téléchargement, l'archive `.pol` est directement placée dans les fichiers du rôle de déploiement, mais l'archive `.env` doit elle être d'abord spécialisée.

[checkout.apigw] Génération du fichier .properties et environnementalisation

Afin d'environnementaliser l'archive `.env` à l'aide de l'outil préalablement développé, il est nécessaire de générer le fichier `.properties` correspondant à l'environnement.
Ce fichier, dont le contenu se présente sous la forme `key=value`, est généré à l'aide du module `template` d'Ansible.

Ceci requiert la création d'un fichier template `.properties.j2` (format requis pour le templating), dont le contenu se présente, lui, sous la forme `key={{ var }}`.
Les variables et leurs valeurs sont alors résolues par le moteur de templating d'Ansible, qui outrepasse (override) les valeurs des variables selon leur ordre de priorité.
Ainsi, dans notre cas, les variables du fichier template sont principalement stockées au sein de deux fichiers :

1. un fichier YAML propre à l'environnement (par exemple `bas_properties.yml` pour l'environnement BAS),

2. le fichier contenant les variables par défaut du rôle (`defaults/main.yml`)

La bonne pratique consiste donc à définir, pour toutes les variables utilisées dans le fichier `.properties.j2`, une valeur par défaut, puis de déclarer uniquement celles dépendant de l'environnement dans le fichier YAML qui lui est propre (voir figure 2.7).

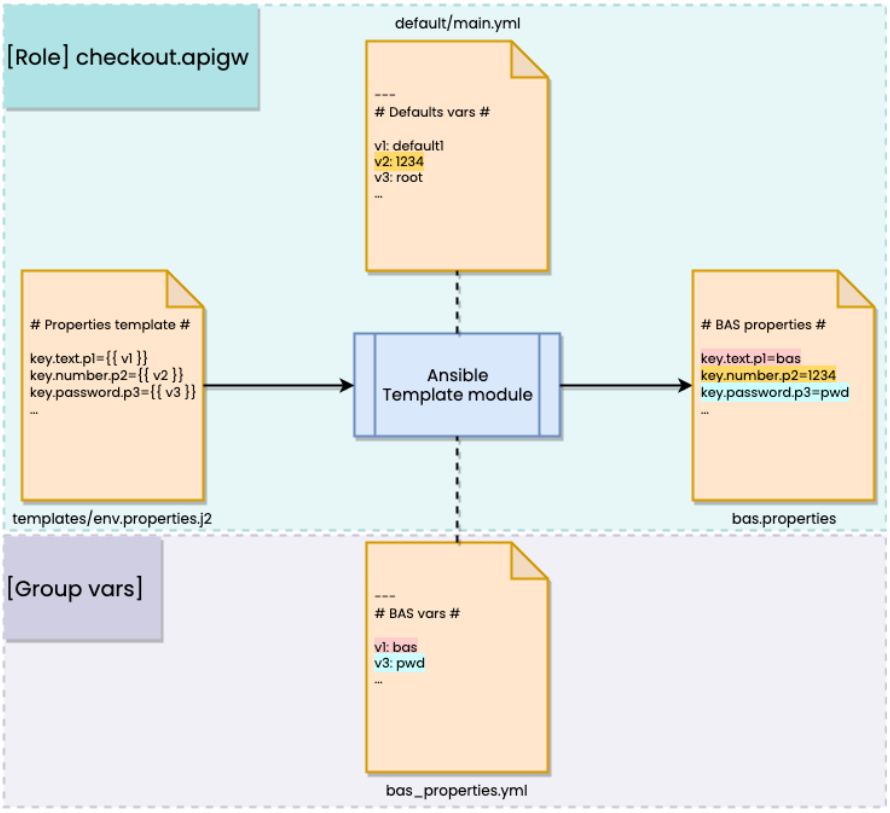

FIGURE 2.7 – Génération du fichier .properties propre à l'environnement

La suite du rôle consiste donc simplement en l'appel du module `template` et l'exécution de l'outil d'environnementalisation, via le module `shell`, générant ainsi l'archive `.env` spécialisée.

```
- name: Templating properties file
  template:
    src: templates/envfile.properties.j2
    dest: "{{ conf_file }}"
- name: Execution de l'environnementaliseur
  shell: "java -jar {{ role_path }}/files/{{ environmentalizer }} --conf={{ conf_file }}
      --env={{ generic_env }} --targetEnv={{ target_env }}"
```

Listing 2.5 – Utilisation du module de templating

[deployment.apigw] Copie des archives et déploiement

Ce rôle permet de déployer les archives `.pol` et `.env` (spécialisée) sur les machines de l'environnement spécifié.

Le déploiement sur l'environnement est effectué depuis une machine directrice, l'Admin Node Manager, via le script `projdepoy`. L'Admin Node Manager pilote ensuite le déploiement, sur les différentes machines du groupe spécifié, via un autre composant intermédiaire présent sur chaque machine, le Node Manager (voir figure 2.8).

Sur chaque environnement, les machines sont distinguées en deux groupes, AIFE et AIFE_TRYIT. Pour le moment, les déploiements sont effectués uniquement sur le groupe AIFE, mais une solution permettant de gérer le groupe sous forme de variable du rôle Ansible est actuellement en cours de conception.

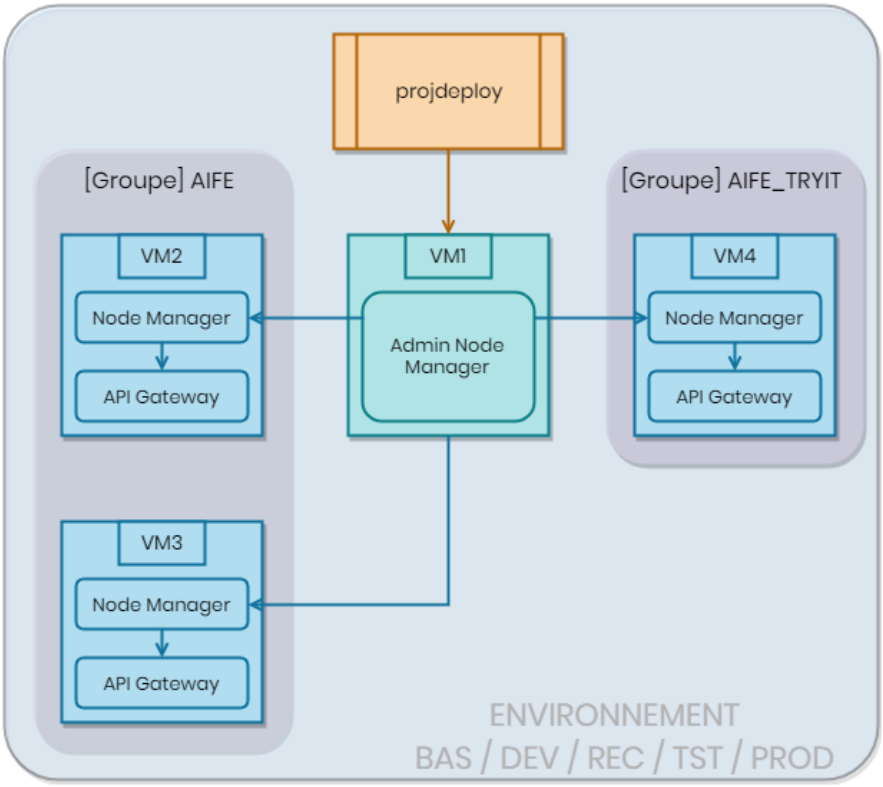

FIGURE 2.8 – Déploiement du livrable via le script projdeploy.sh

Une fois l'appel au script `projdeploy` effectué, il est possible de consulter les logs, stockés au sein d'un fichier, toujours dans une démarche de traçabilité.

```
Loading policy archive file '/tmp/archives/AIFE_Root-0.1.pol' to Node Manager...
Loaded policy archive '9acc9ca2-93a1-407c-98da-d070753e2f5c' to Node Manager.
Loading environment archive file '/tmp/archives/AIFE_Root-bas.env' to Node Manager...
Loaded environment archive and created deployment archive '6fcd99c5-0fbf-4228-a7a0-9b5f158ed958' to hosts running group.
Deploying to API Gateway 'INST_AIFE_ALL_f4slsea010'...
Deployed to API Gateway 'INST_AIFE_ALL_f4slsea010' successfully.
Completed successfully.
Gateway instance [INST_AIFE_ALL_f4slsea010] deployed with [0] error(s).
```

FIGURE 2.9 – Message informant du succès du déploiement

(d) Conclusion

La réalisation de cette partie m'a permis de découvrir Ansible, qui se révèle être un outil d'automatisation particulièrement puissant en ce qui concerne la gestion d'un parc informatique.

Contrairement aux autres missions précédemment mentionnées, j'ai ici eu la chance de pouvoir évoluer dans un environnement préalablement configuré. J'ai donc pu me consacrer pleinement au développement en lui-même et à la mise en œuvre des bonnes pratiques.

Enfin, Bruno TRINTA, architecte sur le projet, a été mon principal interlocuteur sur ce sujet. Il a su me former et m'accompagner, en apportant des réponses toujours pertinentes aux questions que j'ai pu lui soumettre.

2.3.4 Orchestration

(a) Problématique

Afin de lier tous les maillons de la chaîne d'industrialisation, et toujours selon l'approche CI/CD [12], il était nécessaire d'automatiser les étapes précédentes et leurs enchaînements.
En effet, il paraît relativement inutile et improductif que le développeur intervienne à chacune des étapes de la chaîne d'industrialisation.
Ainsi, il était donc nécessaire d'orchestrer les travaux précédents, afin que la totalité de la chaîne soit autonome.

(b) Conception

Ce travail n'a donc pas été réellement réalisé à la suite des précédents, mais plutôt parallèlement à ceux-ci, afin de mener des tests durant la phase de développement.
Pour cette partie, il m'a donc été demandé d'utiliser l'outil Jenkins, qui permet de créer une pipeline d'intégration continue et de développement continu (CI/CD).

Initialement, il était prévu de ne réaliser qu'un seul et unique job Jenkins, qui se chargerait d'exécuter l'ensemble de la chaîne d'industrialisation, depuis la création des livrables à partir des sources jusqu'à leur déploiement.
Seulement, au fil de l'avancement du projet et de notre réflexion, il est apparu judicieux de le scinder en deux jobs :

- **piste-pack-apigw**, chargé du paquetage des sources et du dépôt du `.pol` généré sur Artifactory
- **piste-deploy-apigw**, chargé de la récupération des archives `.pol` et `.env` générique depuis Artifactory, de la spécialisation de l'archive `.env` et du déploiement du livrable sur l'environnement spécifié.

12. Développement Continu / Intégration Continue

(c) Réalisation

J'ai donc créé les deux jobs Jenkins, dans un premier temps à l'aide de l'interface graphique de l'outil. Il existe différents formats pour les jobs Jenkins, et il m'a été recommandé d'utiliser le format "Declarative Pipeline". Ce format permet de définir les différentes étapes du job sous forme de code, au sein d'un fichier intitulé `Jenkinsfile`, utilisant la syntaxe JSON.

Une des possibilités offertes par l'outil Jenkins est de récupérer ce fichier depuis un dépôt Git (configuré) avant chaque exécution du job. De cette manière, ce fichier peut être versionné et traçabilisé.

Le job se présente sous forme d'étapes (stages), elles-mêmes contenant des sous-étapes (steps), au sein desquelles il est possible d'appeler des plugins.

```
pipeline {
    // Agent sur lequel sera execute le pipeline
    agent any

    // Eventuels parametres
    parameters {
    }

    // Stages
    stages {
        stage ('Premier stage') {
            steps {
                echo "Test de la commande echo"
                sh '''
                    # script Shell
                '''
            }
        }
    }
}
```

Listing 2.6 – Syntaxe d'un pipeline Jenkins

piste-pack-apigw

La première action effectuée par ce pipeline est la récupération des dernières sources du code applicatif depuis le dépôt GitLab. L'authentification est réalisée au moyen de credentials (certificats de connexion), configurés dans Jenkins, afin de ne pas avoir les mots de passe affichés en clair.

```
stage('Checkout des sources') {
    steps {
        git url: "https://url_du_depot_gitlab.com", credentialsId: "credentials"
    }
}
```

Listing 2.7 – Récupération des sources depuis le dépôt GitLab

Suite à cela, l'outil Java `projpack` est construit, puis exécuté avec les sources des projets Policy Studio présents sur le GitLab.
Les archives `.pol` et `.env` sont ainsi créées, et la version passée en paramètre du job Jenkins leur est associée. L'archive `.pol` est ensuite déposée sur Artifactory, et le dépôt GitLab est tagué avec cette version. Le tag permet de marquer le point de publication (v1.0, v1.2), et donc de lier le livrable (`.pol`) à son code source (projet Policy Studio), et au code source de l'outil `projpack`.

La figure 2.10 présente une exécution de ce job. Les deux premières étapes, générées par défaut, permettent à Jenkins de récupérer le Jenkinsfile depuis GitLab, puis d'initialiser les outils [13] pour la suite du job. On retrouve ensuite la succession d'étapes mentionnées précédemment. Jenkins offre également plusieurs indicateurs :

— La couleur, qui représente l'état de l'exécution de chaque stage. Elle est verte pour un succès (code de retour 0), et rouge pour un échec (code de retour différent de 0). A noter que si l'exécution d'un stage rencontre une erreur, les suivants ne sont pas exécutés.
— Le temps d'exécution de chacun des stages, ainsi que leur temps d'exécution moyen. [14]

Ces indicateurs sont liés au pilier Mesure (Measurement) des C.A.L.M.S, permettant d'avoir une vue d'ensemble sur le déroulement de l'exécution, et ainsi de localiser rapidement les erreurs.

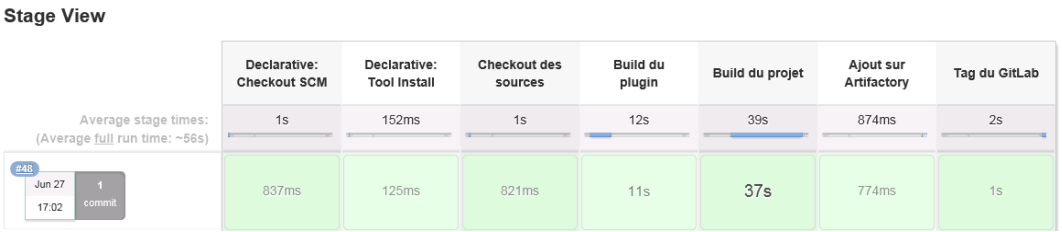

FIGURE 2.10 – Compte-rendu du job pack

piste-deploy-apigw

Plus concis que le précédent, ce job se distingue en deux étapes :

— La récupération des sources sur le dépôt GitLab, depuis le tag correspondant à la version passée en paramètre du job.
— La copie du code de déploiement Ansible sur la machine Ansible, chargée d'effectuer le déploiement.

13. Ici Java et Maven, utilisés pour le projpack
14. Les valeurs moyennes ne sont ici pas représentatives, la plupart des exécutions ayant été effectuées pour des tests durant le développement du pipeline.

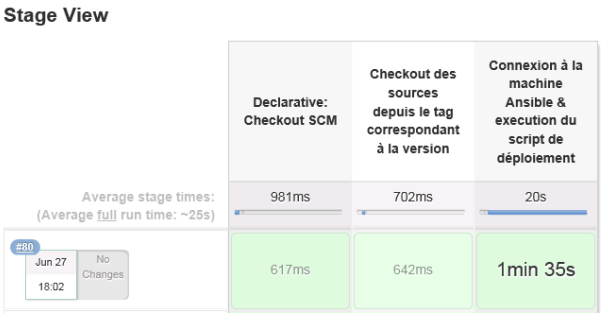

FIGURE 2.11 – Compte-rendu du job deploy

L'étape particulièrement intéressante est ici l'interaction avec la machine Ansible, chargée d'effectuer le déploiement. L'accès à cette machine s'effectue via le protocole de connexion sécurisé Secure Shell (SSH), basé sur un échange de clé.

Le développement de cette partie s'est révélé relativement complexe à mettre en œuvre, puisqu'aucun échange de clés SSH n'avait été configuré préalablement entre les machines Jenkins et Ansible. De plus, cette solution n'a pas été retenue car elle aurait demandé, dans l'absolu, une mise en oeuvre trop compliquée, puisqu'il aurait fallu réaliser un échange de clé pour chaque projet de l'AIFE.

Si aucun échange de clé n'est configuré, la commande `ssh` propose à l'utilisateur de saisir son mot de passe de manière interactive, via le prompt. Néanmoins, dans le cadre d'un job Jenkins autonome, il n'y a pas d'interaction entre l'utilisateur et la machine, il n'a donc pas le contrôle sur le prompt. Pour pallier à cela, nous avons demandé l'installation d'un utilitaire permettant la saisie du mot de passe directement dans la ligne de commande : `sshpass`.
Afin que le mot de passe ne soit pas affiché en clair, nous avons trouvé judicieux d'utiliser à nouveau les credentials Jenkins.

Une autre difficulté rencontrée au sein de cette partie a été la gestion des droits en écriture et en exécution, particulièrement stricte sur cette machine. Bien que totalement justifiées, du fait de sa puissance de contrôle et des données qui y sont stockées, ces mesures m'ont contraint à changer d'approche pour le déploiement. Ainsi, pour pouvoir accéder au répertoire `\data\ansible`, environnement au sein duquel doivent être effectués les appels aux playbooks Ansible, il était nécessaire de passer en super-utilisateur `ansible_piste_dev`.

L'idée a donc été de procéder en trois temps :

1. La copie du code Ansible (playbooks et rôles) [15] lié au déploiement de l'API Gateway, ainsi que d'un script shell `deploy.sh`, dans un dossier temporaire `/tmp/ansible_apigw_deploy`, auquel l'accès était autorisé en exécution et en écriture. Cette partie a pu être réalisée grâce à la commande `scp`, permettant un transfert de fichiers via le protocole SSH.
2. L'exécution à distance, via la commande `ssh`, du script `deploy.sh`, effectuant les actions suivantes :

15. Récupérés préalablement depuis GitLab par Jenkins

(a) le passage en super-utilisateur `ansible_piste_dev`, grâce à la commande `sudo su - ansible_piste_dev`,

(b) la copie du code de déploiement Ansible dans le dossier approprié,

(c) l'exécution du playbook chargé du déploiement de l'API Gateway `deployment.apigw.yml`

3. La suppression du dossier temporaire, afin de libérer l'espace disque.

```
stage('Connexion a la machine Ansible et execution du script de deploiement') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'ansiblecredentials_dev',
            passwordVariable: 'SSH_PASSWORD', usernameVariable: 'SSH_USERNAME')]) {
            sh '''
                ## Copie de tous les scripts Ansible
                sshpass -p $SSH_PASSWORD scp -vpr deployment/ansible/*
                    $SSH_USERNAME@vm_ansible:/tmp/ansible_apigw_deploy

                ## Execution du script de deploiement
                sshpass -p $SSH_PASSWORD ssh $SSH_USERNAME@vm_ansible
                    "/tmp/ansible_apigw_deploy/deploy.sh ${VERSION} ${ENVIRONMENT}"

                ## Suppression du dossier temporaire
                sshpass -p $SSH_PASSWORD ssh $SSH_USERNAME@vm_ansible "rm -rf
                    /tmp/ansible_apigw_deploy/*"
            '''
            }
        }
    }
```

Listing 2.8 – Interaction avec la machine Ansible

(d) Conclusion

Cette partie m'a permis d'acquérir énormément de compétences sur un outil particulièrement utilisé dans le monde professionnel, permettant la mise en place d'une démarche de développement et d'intégration continus.

Les principales difficultés que j'ai pu rencontrer au cours de ces développements concernaient les interactions entre les machines et leurs configurations. Pour la plupart des points de blocages, je n'avais aucun pouvoir de décision et d'action, j'ai donc dû interagir avec les personnes compétentes en la matière, souvent par échange de courriels.
Afin de ne pas rester passif dans l'attente d'un retour de leur part, j'ai également effectué des recherches, qui m'ont amené à leur effectuer quelques propositions (notamment l'utilisation de `sshpass`).

Aujourd'hui, les deux jobs Jenkins sont utilisés, sur les différents environnements, à l'exception de celui de production (PROD), pour lequel il sera nécessaire d'apporter quelques modifications. En effet, cet environnement étant particulièrement critique, il est important de tracer toutes les actions qui l'impactent. C'est pourquoi il nous a été demandé d'implé-

menter un système de connexion, et non d'utiliser un compte générique, comme c'est le cas actuellement.

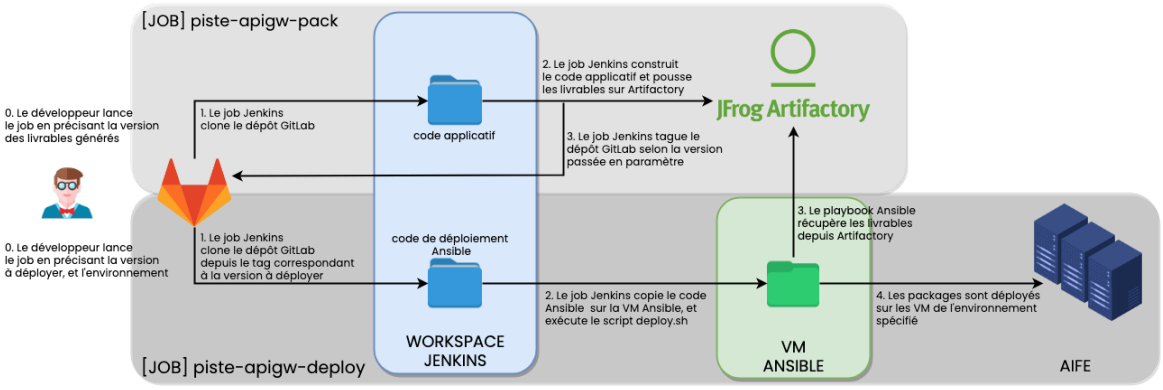

FIGURE 2.12 – Schéma de synthèse des deux jobs Jenkins mis en place

Conclusion et perspectives

Je suis très satisfait d'avoir pu effectuer mon stage de IATIC4 au sein de l'entreprise Sopra Steria. Cela m'a permis de vivre une forte expérience de travail en entreprise, puisque j'ai eu l'opportunité d'évoluer au sein d'un projet relativement récent et particulièrement ambitieux. J'ai d'ailleurs eu la chance de participer concrètement à ses enjeux, au travers de missions variées, tel que celle de la réalisation d'une chaîne d'outils DevOps de bout en bout, que j'ai particulièrement apprécié.

Bien qu'il ne soit pas encore terminé, ce stage m'a déjà énormément apporté, tant sur le plan technique que sur le plan humain.

Ainsi, sur le plan humain, j'ai pu découvrir le travail en open-space au sein d'une équipe relativement nombreuse, qui s'est révélé être à la fois un réel atout et une contrainte forte. En effet, du fait de la diversité des profils, cette configuration favorise l'entraide et la richesse des échanges entre les collaborateurs, mais oblige également à une certaine rigueur, dans la mesure où le travail d'une personne impacte celui des autres. D'un naturel réservé, la bienveillance de mes collaborateurs, et plus particulièrement de mes référents techniques, toujours disponibles à mon égard, m'a permis de m'épanouir et ainsi de stimuler mon intérêt et mon implication au sein du projet.

Ce stage m'a également permis de découvrir la culture DevOps, enjeu majeur pour la transformation numérique des entreprises, et sa mise en œuvre au sein d'un environnement professionnel. Sur ce point, le projet PISTE fait face à une période charnière, et je suis très fier d'avoir pu contribuer à cette restructuration. Néanmoins, bien que les premiers résultats soient très satisfaisants, ce processus exige une adaptation permanente face à des besoins et des technologies en perpétuelle évolution.

Sur le plan technique, j'ai pu progresser sur des technologies nouvelles, particulièrement utilisées en milieu professionnel, telles que Jenkins et Ansible.
J'ai également pu appliquer des notions, qu'elles soient techniques, ou conceptuelles, acquises au cours de ma formation, en y modulant le contexte à celui du projet.

Enfin, ce stage m'a également permis d'en apprendre plus sur moi-même. Ainsi, au cours des différentes missions que j'ai pu réaliser, j'ai découvert que l'aspect Recherche et Développement (R&D) était celui qui m'intéressait le plus. Conjuguant à la fois mon intérêt pour la veille technologique et mon besoin de créativité, cet aspect est également essentiel pour assurer l'innovation et la compétitivité des entreprises. Je pense donc m'orienter à l'avenir vers des postes de ce type.

Sigles et acronymes

AIFE Agence pour l'Informatique Financière de l'Etat. 2, 10–16, 44

BU Business Unit - Unité Commerciale. 9

C.A.L.M.S Culture Automation Lean Measurements Sharing – Culture Automatisation Rationalisation Mesure Partage. 20, 37

CI/CD Continuous Integration/Continuous Development – Intégration Continue/Développement Continu. 35

DSTUM Daily Stand-Up Meeting. 13

ESN Entreprise de Service du Numérique. 8

IATIC Ingénierie des Architectures Technologiques de l'Information et de la Communication. 25, 41

IDE Integrated Development Environment – Environnement de Développement Intégré. 24–26

ISTY Institut des Sciences et Techniques des Yvelines. 4, 7

J2EE Java 2 Enterprise Edition. 2, 3

JAR Java Archive. 25, 26, 28

JSON JavaScript Object Notation. 36

LOLF Loi Organique relative aux Lois de Finances. 11

PISTE Plateforme d'Intermédiation des Services pour la Transformation de l'Etat. 2, 4, 11, 13–19, 21–24, 30, 41

SCN Service à Compétence Nationale. 11

SGC Système de Gestion de Contenu. 18

SI Systèmes d'Informations. 11, 16

SSH Secure Shell. 22, 38

Glossaire

Agile (Méthode) Approche itérative et collaborative, capable de prendre en compte les besoins initiaux du client et ceux liés aux évolutions . 2, 13

API (acr. de Application Programming Interface - Interface Applicative de Programmation) Ensemble de fonctions permettant à un développeur d'utiliser plus simplement une application dans son programme. 2, 14–16, 18, 32

automatisation Substitution d'une ou plusieurs machines à l'homme, pour réaliser de manière automatique un programme déterminé d'opérations. 2, 8, 35

credentials (Informatique) Identifiants de connexion à un espace à accès restreint. 15, 18, 36, 38

DevOps Mouvement en ingénierie informatique visant à l'unification du développement logiciel (Dev) et l'exploitation des infrastructures informatiques (Ops) . 2, 13, 19–21, 23, 41, 46

Etat plateforme Conception de l'Etat comme une plateforme mettant des ressources à disposition de la société, en lui laissant la liberté de développer des biens et des services par l'utilisation de ces ressources. 2, 14

industrialisation Actions entreprises pour rationaliser, standardiser, optimiser les services rendus par les infrastructure informatiques. 2, 24, 28, 31, 35, 46

log Type de fichier dont la mission principale consiste à stocker un historique des évènements, afin de permettre l'analyse de l'activité interne d'un processus. 27, 34

Table des figures

Table des codes sources

Annexes

Avancement des tâches

Les figures ci-après représentent le tableau kanban [16] mis en place durant le stage, spécialement pour la partie DevOps/industrialisation.

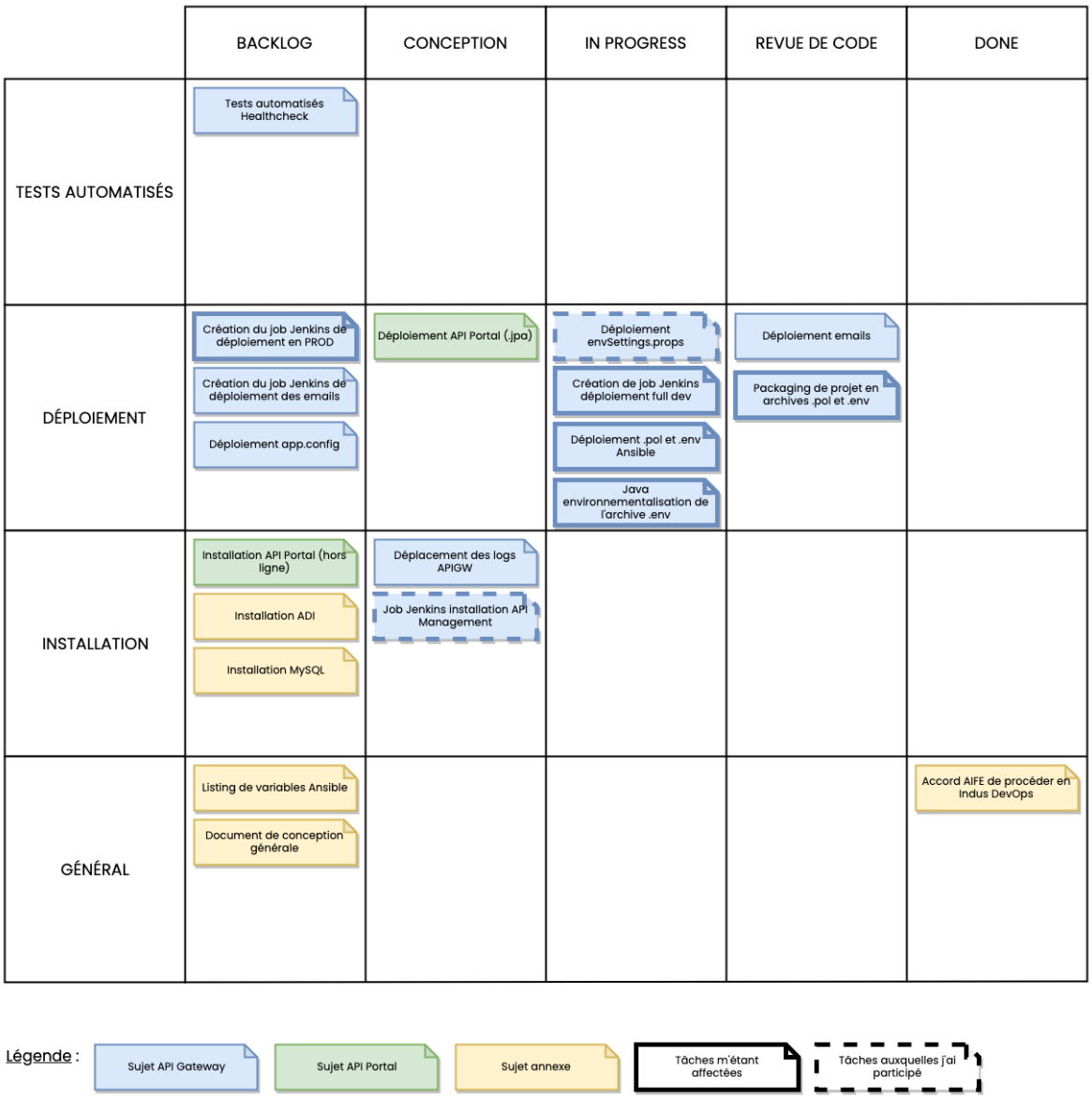

FIGURE 13 – Avancement de l'industrialisation sur PISTE au 15/05/2019

16. Outil de travail permettant l'application de la méthode agile Kanban à un flux de travail

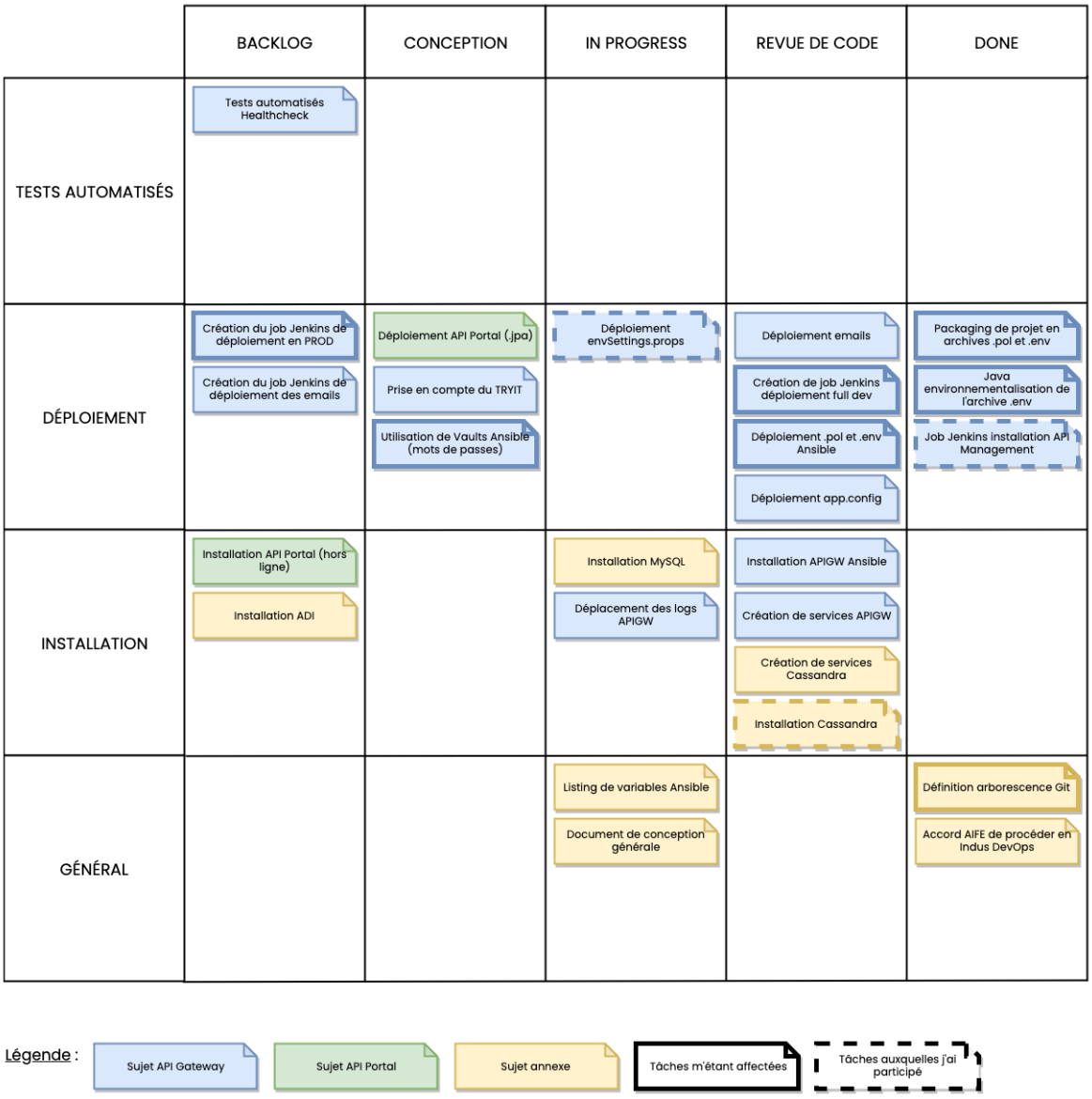

FIGURE 14 – Avancement de l'industrialisation sur PISTE au 21/06/2019

Utiliser ce type de représentation permet de mieux visualiser le flux de travail, et offre une visibilité sur les tâches en cours à tous les niveaux, favorisant ainsi la surveillance, l'adaptation et l'amélioration.

Bibliographie

[1] Comprendre le DevOps
. https://www.redhat.com/fr/topics/devops.

[2] Documentation d'Ansible
. https://docs.ansible.com/.

[3] Documentation d'Artifactory
. https://www.jfrog.com/confluence/.

[4] Documentation d'Axway
. https://docs.axway.com/.

[5] Documentation de GitLab
. https://docs.gitlab.com/.

[6] Documentation de Jenkins
. https://jenkins.io/doc/.

[7] Documentation de Maven
. https://maven.apache.org/guides/index.html.

[8] Le Secteur Public de Sopra Steria
. https://www.soprasteria.com/fr/secteur-activite/secteur-public.

[9] A propos de Sopra Steria
. https://www.soprasteria.com/fr/a-propos.

[10] Présentation de l'AIFE
. http://www.economie.gouv.fr/aife/presentation-aife.

[11] Site de la plateforme PISTE
. https://developer.aife.economie.gouv.fr/.

[12] Wikipedia
. https://wikipedia.com/.