



*Ecole Nationale
Supérieure d'Electronique,
Informatique,
Télécommunications,
Mathématique et
Mécanique de Bordeaux*

RAPPORT DE STAGE

NOM : Juillard
PRENOM : Sandrine
FILLIERE : Télécommunication

DATE DU STAGE : du 29/06 au 31/08 , soit 9 semaines

DENOMINATION DE L'ORGANISME D'ACCUEIL : BLU AGETM

ADRESSE : 32 av Léonard de Vinci, 33600 PESSAC

PAYS : FRANCE

EXPERIENCE INTERNATIONALE : NON

CONVENTION DE STAGE : OUI

Responsable stage 2A : Astien Eric / eric.astien@bordeaux-inp.fr

Abstract

Ce rapport fait office de compte-rendu de mon stage de 2eme année à Blu Age, dans le cadre de l'obtention de mon diplôme à l'ENSEIRB MATMECA. Ce stage de deux mois c'est dérouler d'une manière assez particulière, étant donnée les circonstances de la crise du COVID19 de 2020. Il c'est déroulé des manière hybride : en présentiel lorsque c'était nécessaire, et en télétravail la plus part du temps. Malgré le contexte exceptionnelle, ce stage à été, pour ma part, une très bonne expérience.

Tables de Matière

Introduction	p.4
I. Blu Age	p.5
II. Contexte Technique	p.7
1) Serverless COBOL for AWS solution	p.7
2) L'environnement Amazon Web Services	p.7
III. Réalisation de chaine de déploiement	p.9
1) Etape de recherche	p.9
2) Solution retenu	p.10
1. Aperçu global	p.10
2. Détails de la solution	p.12
IV. Travaux complémentaires	p.15
1) BASCAC (<METTRE SE QUE 9A VEUX DIRE>)	p.15
2) Export de la vue problème d'Analyzer	p.16
3) Plugin Blu Age Cobol — Tâche 1 : Phase de vérification	p.17
4) Plugin Blu Age Cobol — Tâche 2: Ajout d'une commande	p.18
Conclusion	p.20
Références	p.21
Annexes	p.22

Listes des Abréviations

AWS	Amazone Web Services	AWS	Amazone Web Services
Cobol	Common Business Oriented Language	Cobol	Common Business Oriented Language
API	Application Programming Interface	API	Application Programming Interface
DevOps	Software <u>Development</u> (<i>Dev</i>) IT <u>operations</u> (<i>Ops</i>)	DevOps	Software <u>Development</u> (<i>Dev</i>) IT <u>operations</u> (<i>Ops</i>)
VM	Virtual Machin	VM	Virtual Machin
IDE	Integrated development environment	IDE	Integrated development environment
SPA	Sigle-page application	SPA	Sigle-page application
VSC	Visual Studio Code	VSC	Visual Studio Code

Introduction

Fin 2018, à l'occasion de la conférence AWS Re:invent à Las Vegas, BluAge présente officiellement "[serverless COBOL for AWS solution](#)", Composé de deux produits : une extension VSC servant de compilateur COBOL vers Java et un Framework « Vélocity », les deux produits combinés permettent de déployer de manière « serverless » une application initialement codée en Cobol. C'est-à-dire, à déployer leur application en utilisant une offre Cloud émergeante d'amazon ou la gestion des ressources est manager par le fournisseur [1] [2].

Pourquoi la nécessiter de compiler en Java un code écrit en COBOL ? Car le COBOL est en langage que l'on pourrais qualifié d'obsolète, pourtant, il est encore très présent dans les entreprises. BluAge à pour but d'accompagner ces entreprises dans leurs transitions digital. La plus partie de leurs solutions sont des outils pour assister les développeurs dans la traduction de leurs applications du COBOL vers le Java ou le .Net. Néanmoins, il est parfois difficile pour des développeur COBOL de se reconvertis vers le Java. Le compilateur proposé par BluAge, permet d'éviter les licenciements, sans restreindre l'entreprise dans leurs conversion digital en permettant au développeur de continuer à écrire en COBOL, tout en générant des applications dans un langage plus moderne, le Java. Ce compilateur est le 1^e produit BluAge à traduire du code à 100%, sans nécessité aucune intervention humaine.

A l'heure actuelle, [le framework Vélocity de BluAge est mise à disposition du client de manière manuelle](#). A partir de la console graphique de Amazon, l'API est déployer manuellement dans les régions souhaiter. L'opération peut prendre une demi-journée, voir une journée complète pour chaque nouvelle version publiée. Il semble judicieux de réfléchir à l'automatiser du déploiement de ces produits.

Objectif : Crée une chaîne de déploiement pour l'API vélocity de Blu Age.

Vélocity est une API développée en Java. Il existe déjà une chaîne d'intégration en charge de compiler et tester le produit. A l'issue de ce traitement, les fichiers de l'API sont archivés sous forme de zip. La chaîne de déploiement sera en charge de mettre à disposition du client ce fichier sur une des plateformes de stockage d'Amazon, les buckets S3. En résumé, le travail à effectuer est de concevoir, réaliser et finaliser une pipeline qui effectuera de manière 100% automatique les tâches présentées plus tôt.

Blu Age

Netfective Technology, l'organisation mère

La révolution numérique à bouleversé notre société et en particulier dans le monde des entreprises. Depuis l'essor des nouveaux outils et possibilités qu'offre le digital, de nouveau besoin, émergeant des entreprise, aussi bien privé que public, ont vu la jour. En particulier, les évolutions très rapides de ces technologie demandent de se renouveler constamment. C'est pour répondre à cette demande qu'a été créée Netfective Technology, la corporation mère de Blu Age. Crée en 2000 et diriger par Christian Champagne, la firme organise son activité autour de l'accompagnement des grandes entreprises et organismes publics vers le digital. Malgré seulement vingt ans d'ancienneté, la firme compte déjà plus de 160 collaborateurs, dont 80% d'ingénieur. Implanté dans 3 pays : La France, le Maroc et les États-Unis; On retrouve parmi ces clients et partenaires de grande entreprise tel que Amazones, BNP PARIBAS, Orange, Spora Steria, Accenture.. Mais aussi des administrations gouvernemental tel que L' Administration de la sécurité sociale des États-Unis, Le département du Travail et des Retraites britannique, ou encore La direction des finances publique française. Ces activité sont séparé dans ces deux filial : Blu Age et OptTeam.

Blu age d'une part, charger de développer des solutions logiciels et OpTeam pour le consulting.

Année de création	Dirigeant
2000	Christian Champagne
Age moyen	Nombre de Collaborateurs
32 ans	160
Pays implanté	Turnover
France (Pessac, Surenne)	5%
Maroc (Casablanca, Rabat)	Chiffre d'affaire
États-Unis (Dallas)	17M€

Figure 1 : Tableau informatif sur Blu Age

Les activités de Blu Age : La réécriture d'applications

Blu Age base ces techniques de transcription sur une approche MDA : Comme son nom l'indique, cette technique se base sur la récupération d'un modèle. La suite logiciel « Blu Age Classic », va automatiquement extraire la logique de l'application, et facilitera le travail du programmeur chargé de la réécrire [2]. Cette suite fournira en plus des outils de générations automatique de code des outils de visualisation du code et des outils d'analyse fig.[2].

Ce logiciel, et tous les autres logiciels de la suite Blu Age sont régulièrement mise à jour et déployer sur les serveur AWS.

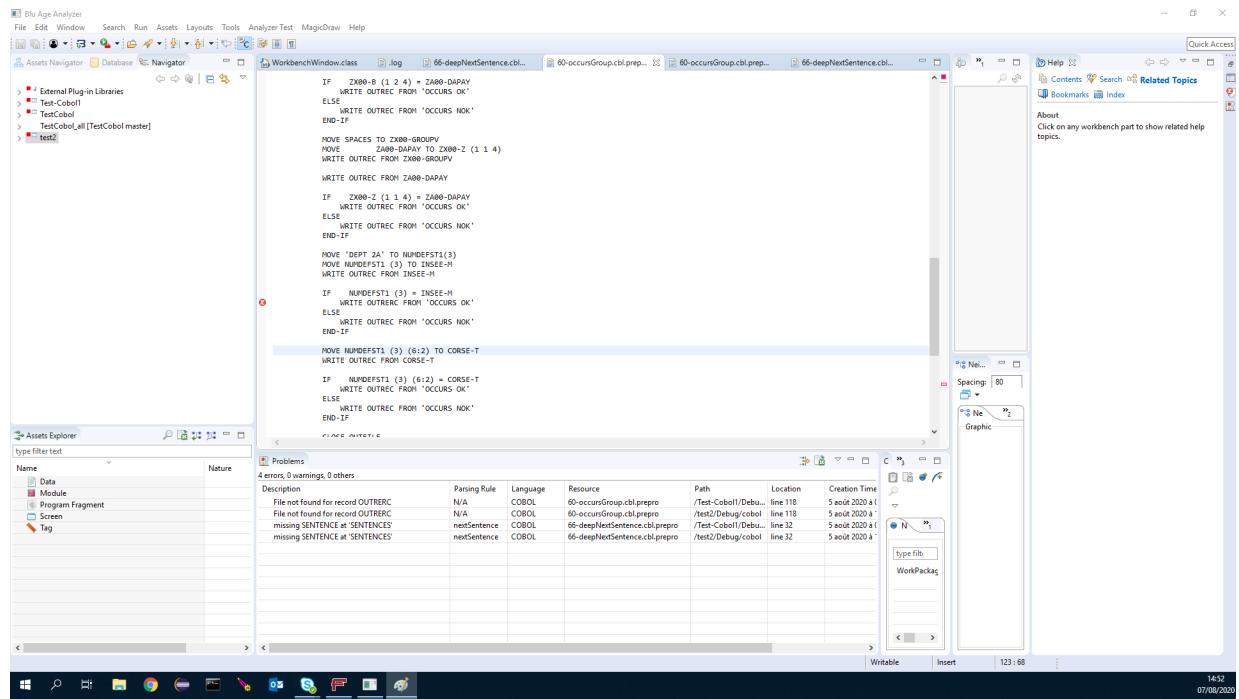


Figure 2 : Capture d'écran du Logiciel Analyzer

Capture dd'écran Analyser + petite explication

Le centre Blu Age Pessac

Le centre BluAge de pessac est un des 5 point d'implenation de l'entreprise. Elle compte dans son service une soixantaine d'employé séparé dans 3 domaine d'activité distincts :

Le partie R&D : Service dans lequelle s'integre mon stage. Le pôle R&D dirigé par Alexis Henry est charger du developpement des nouveaux produit BluAge.

La partie BluSight : Les outils de traduction que propose BluAge ne traduise pas une application à 100%. L'intervention d'un developpeur est nécessaire. Bien que les outils sont consus pour faciliter le travail, il peux arriver que des entreprises ne souhaitent pas donner cette tâche à un developpeur en interne. Le pôle BluSight, dirigé par Youssef Iraoui, est donc en charge d'utilisé les outils BluAge, afin de livré directement à l'entreprise qui en à fait la demande, un application traduite à 100%.

Le partie e-commerce : Les activité du pôle e-commerce sont excentré des deux autre pôle. Ce services dirigé par Sébastien PRADET n'a qu'un seul client, un seul projet : Intermarché. Son but est de maintenir et amélioré le site de e-commerce de la grande enseigne.

Contexte technique

La solution serverless de Velocity a pour but de procurer au client une solution pour déployer, sur une Lambda Amazone, une application java initialement codées en COBOL. Elle

Il est constituée de deux produits : d'une part un compilateur COBOL vers java, déployer sous forme d'une extension Visual Studio code. Cette extension permet au client de générer un fichier jar à partir de son projet Cobol. Ce fichier contient le code minimum de l'application : Il ne contient pas les librairies nécessaires à son fonctionnement. Ces dépendances sont fournies par le second produit, le Framework Velocity de Blu Age. Ce Framework est déployé sous Amazon Web Services en tant que « Layer ».

Ainsi, pour déployer son application, le client devra créer une Lambda, à laquelle il ajoutera le Framework Velocity en tant que Layer, puis il pourra y exécuter le fichier jar fourni par l'extension VSC.

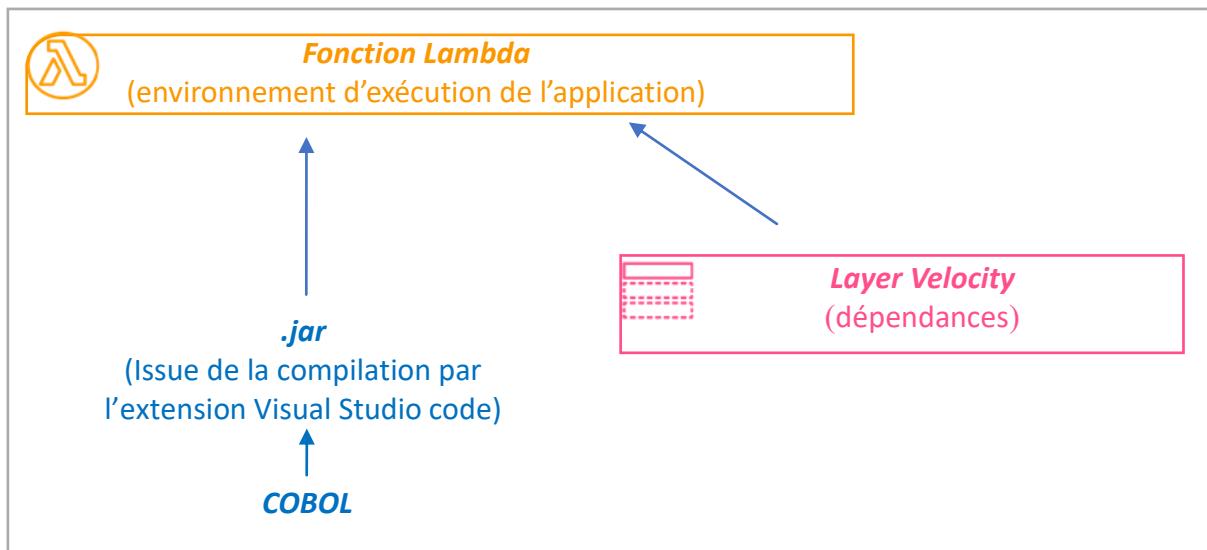


Figure 3 : Schéma fonctionnelle de la solution serverless COBOL for aws solution

L'environnement Amazon Web Services

Amazon Web Services est une plateforme qui propose des services informatiques à destination des entreprises comme des particuliers. Elle propose un grand nombre de services, pour stocker, manager, et déployer des données et des applications. En particulier, cette plateforme est spécialisée dans le Cloud Computing.

Dans les parties précédentes, nous avons mentionné certains des services qu'il propose :

Tous d'abord, **les Lambda AWS**. C'est l'environnement d'exécution sur laquelle le client pourra déployer son application. Une Lambda est un service qui fournit un environnement d'exécution de code qui ne nécessite ni mise en service, ni gestion de serveur. C'est le fournisseur d'accès (c-a-d Amazon) qui se charge de l'administration des ressources, entre autres, la maintenance des serveurs, le dimensionnement et la mise à l'échelle de capacité, la surveillance et la journalisation des exécutions. De plus, quand la Lambda est appelée plusieurs fois, des fichiers temporaires sont conservés permettant ainsi d'économiser sur le temps de préparation de l'environnement d'exécution.

L'environnement Amazon Web Services

Pour l'utilisateur, ce service permet de déployer simplement n'importe quel type d'application en étant facturer uniquement pour le temps de calcul utilisé. Lorsque la fonction Lambda ne s'exécute pas, l'utilisateur débourser rien. Ce service d'Amazon et également très adapté à la gestion des autres ressources et services que Amazon propose.

Ensuite **les Layers**. C'est une couche supplémentaire que l'on peut ajouter à une Lambda. : Une couche est une archive ZIP qui contient des bibliothèques, un environnement d'exécution personnalisé ou d'autres dépendances. Elle permet de créer un runtime personnalisé. Ainsi, il n'est plus nécessaire d'inclure les bibliothèques utilisées par la fonction Lambda dans le package de déploiement.

Pour déployer un Framework sur une Layer, une pratique possible (et c'est celle que nous allons utiliser pour la chaîne de déploiement), est de stocker les fichiers de la Layer sur une unité de stockage d'Amazon (un bucket) qui se trouve dans la même région que la Layer, puis importer le code de la Layer depuis ce bucket.

Enfin, ces **unités de stockage appelé bucket** que nous allons utiliser : Un bucket est un compartiment de stockage fourni par le service Amazon Simple Storage Services (Amazon S3). Un compartiment permet de stocker des Object ; un objet est la somme d'un fichier et de tout le méta data qui le décrit.

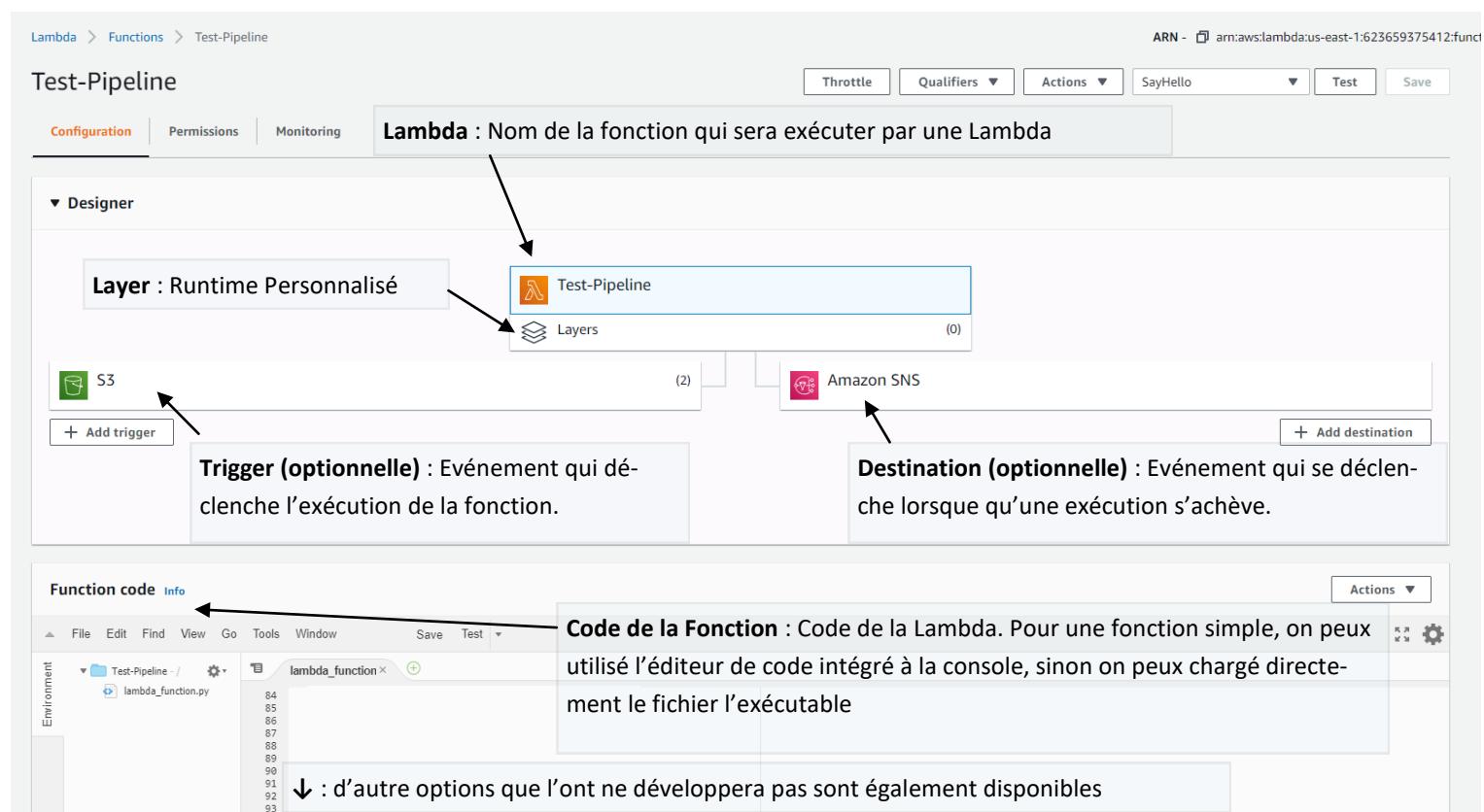


Figure 4 : Capture d'écran de la console AWS pour le paramétrage d'une Lambda

III. Réalisation de chaîne de déploiement

1) Etape de recherche

Livrables

La chaîne de déploiement sera en charge de mettre à disposition du client ce fichier sur une des plate-forme de stockage d'amazone, les buckets s3.

Elle devra également :

- S'assurer que l'accès à ces espaces de stockage est configuré correctement et que seuls les clients ont accès au produit.
- Tester l'API en tant que Layer, (soit en l'ajoutant en tant que Layer à une Lambda pour programmer pour faire des tests).
- Faire l'ensemble de ces étapes, dans toutes les régions du Cloud où l'on souhaite que l'API soit disponible.

Le prototype sera également être de préférence facilement paramétrables, sécurisé et peu coûteuse.

Solutions non-retenu

Jenkins : Usuellement, un pipeline est développé avec un outil tel que Jenkins. C'est d'ailleurs cet outil qui est utilisé pour l'intégration (compilation, test...) du Framework Velocity. Une première solution envisagée était d'utiliser le même outil pour la chaîne de déploiement. Cela est possible en utilisant un conteneur sur lequel serait téléchargé l'interface en ligne de commande que propose Amazon. L'inconvénient de cette méthode est qu'elle nécessite de faire toutes les commandes en dehors du cloud Amazon, et

Step Function et CodeDeploy : Amazon propose plusieurs services pour la réalisation de pipeline et de work flow. En ce qui concerne Amazon CodeDeploy, le choix de ne pas le retenir vient d'une part de son coût, et d'autre part du fait qu'il soit surtout conçu pour être utilisé avec les autres services d'Amazon (Services d'Amazon qui permettent d'héberger toutes les étapes du développement, git, intégration, déploiement) tandis qu'on ne veut l'utiliser uniquement pour le déploiement. Amazon Step Fonction quant à lui, utilise les Lambda comme environnement d'exécution, ce service rajoute du code supplémentaire pour lier les Lambda entre elles (en un langage propre à Amazon, proche du JSON). Ce service est très bien pour obtenir un joli aspect visuel du work flow, mais il m'a semblé plus judicieux d'utiliser les fonctions Lambda seule. Certes, on n'obtient pas l'aspect visuel du pipeline, mais l'utilisation est identique. On évite ainsi un coût supplémentaire pour l'utilisation du service et on évite d'ajouter du code supplémentaire à l'architecture.

N.B : Plus que de m'être renseigné sur ces outils, je les ai testés et j'ai réalisé des petites ébauches de pipeline. En plus des arguments cités une petite partie de ma décision est aussi liée à mes préférences personnelles.

2) Prototype retenu

Pourquoi choisir cette solution ?

Le choix de cette solution est motivé d'un part, par sa flexibilité, mais également pour son coût très faible comparé au autre solution proposée. Enfin, l'avantage de cette méthode comparativement à l'utilisation de Jenkins par exemple est que l'on utilise quand même des outils Amazon, et les opérations sont effectuées de manière interne au compte Amazon de l'entreprise. En d'autre terme, on élimine la nécessité de devoir se logger et donc de transmettre des identifiants ni en clair. Cette solution propose le meilleurs compromis entre sécurité, flexibilité et coût.

Aperçu global du prototype

Le déploiement démarre lorsque la chaîne d'intégration se termine. À l'issue de ce traitement, la nouvelle version du Framework, sous forme de jar va être chargé dans le Bucket Source (cf fig). Ce fichier, c'est celui indiqué sur le schéma sous le nom de "Layer File". Cette action va alors déclencher la chaîne de déploiement. Une fonction que l'on nomme "Main Pipeline", exécuté sur une Lambda va alors récupérer le fichier, publier la Layer, la tester, et ajouter les permissions aux comptes AWS qui doivent avoir accès au Framework. Toutes ces opérations sont effectuées dans la région par défaut (canoniquement, en Virginie du Nord us-east-1). Une fois le déploiement de la Layer effectué dans la région par défaut, et si l'opération c'est déroulé correctement, alors le déploiement sur toute la région pourra commencer.

La fonction "Main Pipeline" avant de se terminer, va déclencher de manière asynchrone (C'est à dire simultanément, comme des threads) N fonction lambda nommée "Déployer". Chacune avec comme argument une région différente. Leurs rôles, effectuer le déploiement de la layer dans la région qui leur a été attribué. Dans la région qui leur a été assigné, les fonctions vont créer une copie du contenu du Bucket Source dans un bucket temporaire, pour pouvoir ensuite reproduire les étapes de déploiement (publier, tester, ajouter les permissions).

Les étapes de déploiement, étant commune à la fois de la fonction Main Pipeline et Déployer seront effectuées par une autre Lambda lancé de manière synchrone par leurs mères ; ainsi on évite de dupliquer du code.

En résumé, le flow d'exécution du pipeline est codé par 3 fonctions Lambdas codées en Python. Grâce à la bibliothèque d'Amazon Boto 3, ces fonctions peuvent interagir avec les autres ressources et services que propose Amazon.

La première fonction principale déclenchée par le déploiement d'une nouvelle version de la Layer sur un bucket auxquelles celle-ci est relié, déclenchera à son tour les autres fonctions.

De plus, une fonction Lambda codée en Java devra être chargée de tester la Layer (fonction que je n'ai pas codée, elle est matérialisée dans le prototype par une fonction qui ne fait rien). Et enfin, une dernière fonction qui sera en charge de gérer les journaux d'exécution et l'envoi des notifications du statut du déploiement. Cette dernière permet de centraliser les informations des différents maillons de la chaîne qui s'exécute indépendamment et en parallèle.

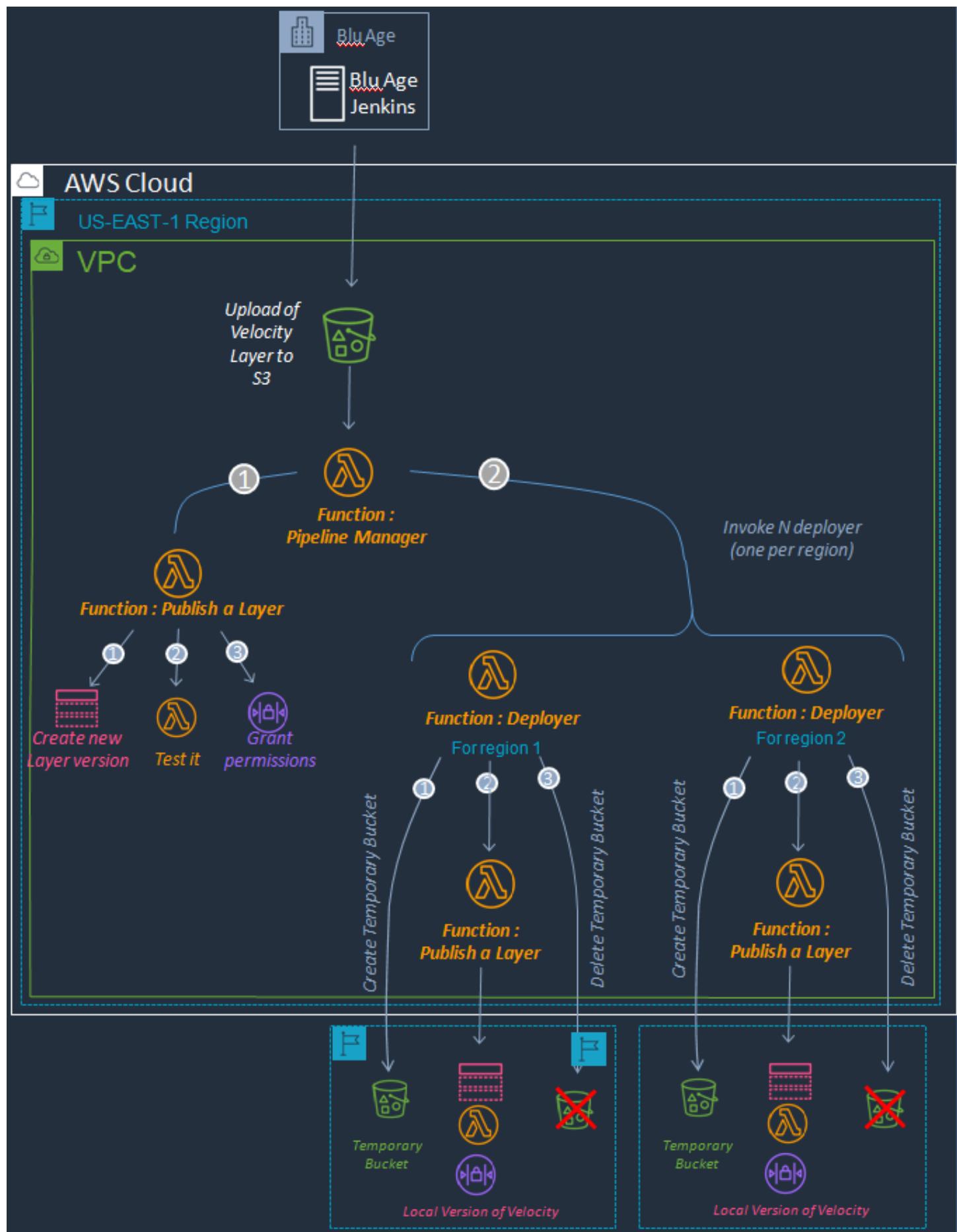


Figure 5 : Schéma de l'architecture de la chaîne de déploiement

Détails de la chaîne de déploiement

Paramétrage des fonctions

Tous les paramètres susceptibles de devoir être modifié sont entré sous forme de variable d'environnement de la fonction Lambda principale. Voici la liste des paramètres modifiables :

- SOURCE_BUCKET : Nom unique du bucket source.
- LAYER_NAME : Nom de la Layer que l'on souhaite donner à la future Layer
- FCT_NAME : Nom de la fonction en charge de tester la Layer
- ACCOUNT_ID : ID du compte sur laquelle d'exécute la Pipeline
- FILE_NAME_REQUIREMENT : Règle sur le nom du fichier qui permettra de filtré les fichiers qui déclencherons effectivement la pipeline.
- FILE_EXTENSION : Règle sur l'extension du fichier (soit zip soit jar pour qu'elle soit reconnu en tant que Layer).
- DEPLOYER_NAME : Nom de la fonction de Déploiement
- PUBLISHER_NAME : Nom de la fonction Publisher
- region_list : Liste des régions sur lesquelles on souhaite déployer la Layer.
- TOPIC_SNS : Nom du topic 'SNS' sur lequel envoyer les notifications.

Préquis pour le fonctionnement du pipeline : La pipeline requière l'existence de toute les ressources mise en paramètre (bucket source, topic SNS, fonction Lambda de teste) ainsi que la présence de la fonction lambda de teste dans toutes les régions où l'on souhaite déployer la Lambda. Le bucket source doit également contenir une fichier ACL.json (pour la gestion des permissions). Ce fichier d'ACL (Access Control) devra suivre le Template fig[??]. Il servira à l'administration des permissions de la layer.

Journaux d'exécution

Par défaut, Amazon stock les journaux d'exécution à travers un service nommée Cloud Watch. Le principal inconvénient de ces journaux pour notre pipeline multifonctions est que les logs de chaque fonction se retrouve séparé les uns des autres. Pour faciliter la lecture, un système de journaux qui centralise les logs de toutes les fonctions de la chaîne a été mis en place. Dans le bucket source va se trouver un dossier Log (ou crée s'il n'existe pas déjà). A chaque exécution, un nouveau fichier texte contenant les logs est créé. Pour se différentier des autres exécutions, le nom de ce fichier est l'heure et la date, à la milliseconde près, du début de l'exécution de la chaîne.

Cette information d'horodatage est générée par la 1^{er} fonction Lambda (au démarrage) et est transmise en argument à toute les autres fonctions invoquées.

Une fonction Lambda que l'on nommera Notification-Manager sera chargé de gérer l'écriture des Logs. Lorsqu'une des fonctions voudra écrire dans les logs, il invoquera cette fonction avec en paramètre l'information d'horodatage mentionné plus tôt ainsi que le message et sa nature (Erreur / warning/info).

```

log17-07-2020_09.47.24.txt - Bloc-notes
Fichier Edition Format Affichage Aide
LOGS 17-07-2020_09.47.24 :
Version 150 deploy sucessfuly in us-east-1
Version 121 deploy sucessfuly in us-east-2
<!WARNING!> in eu-west-1 : test-pipeline--layer-eu-west-1-temporary BucketA]
Version 23 deploy sucessfuly in eu-west-1
<!ERROR!> in eu-west-2 :ACL exceptions , An error occurred (ResourceConflictE

```

Figure 6 : Résultat du journal d'exécution du pipeline

Système de Notification

Le service SNS (Simple Notification Service) d'Amazon fournit un système de messagerie pour la communication A2P (Application à personne), ou system à système. Le service est basé sur un système de publication/Abonnement : un « topic » est créé, la/les applications/ micro-services lié au topic vont push les informations sur le topic. Les utilisateurs peuvent s'abonner à un topic et récupérer ces informations. La fonction Notification-Manager (celle mentionnée dans la partie précédente pour la gestion de journaux d'exécution) est aussi chargé de la gestion des messages de notifications SNS.

Un message SNS est publié pour chaque layer déployer avec succès ou lorsqu'une erreur se produit. Le schéma ci-dessous montre comment est managé la gestion de l'envoie des messages pour que même dans le cas où une erreur non-intercepté se produise, un message SNS soit tous de même envoyé, ou pour éviter la répétition des messages.

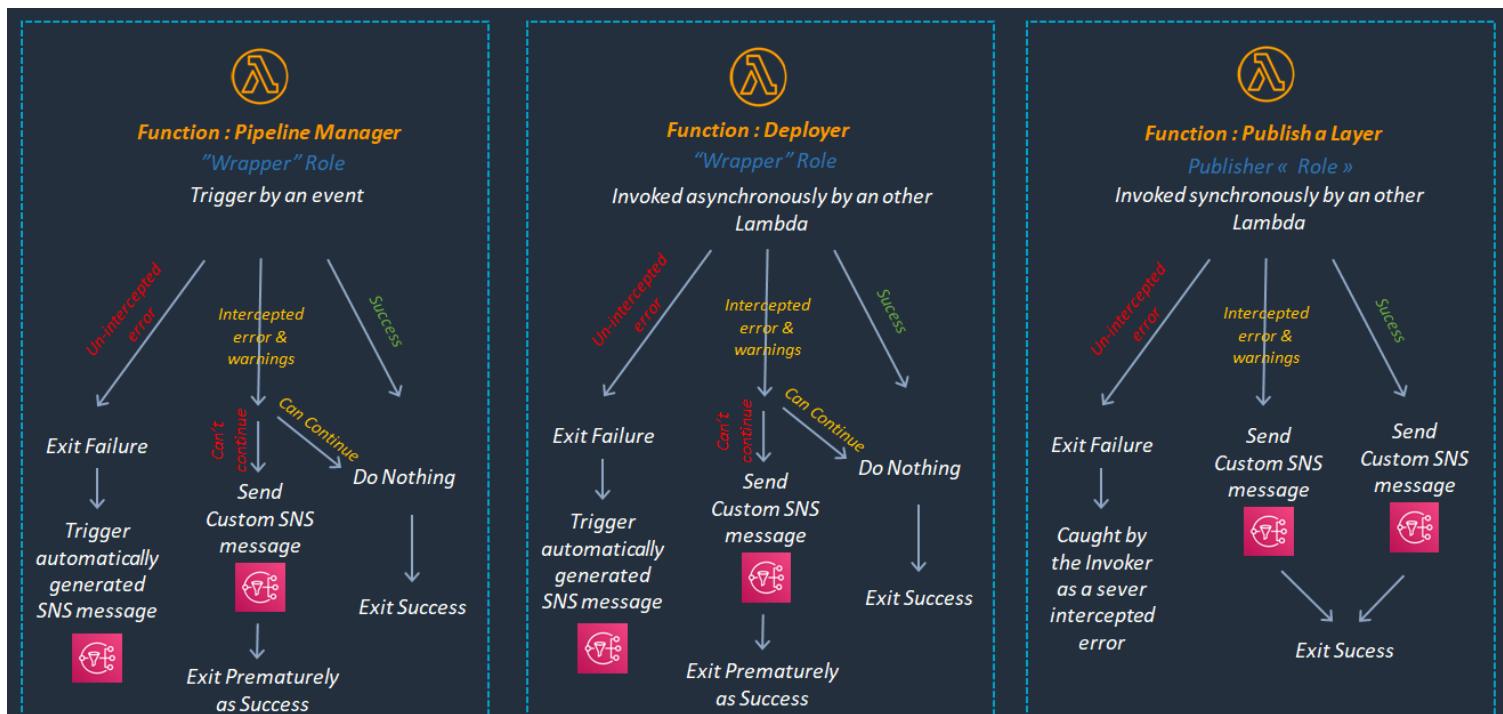
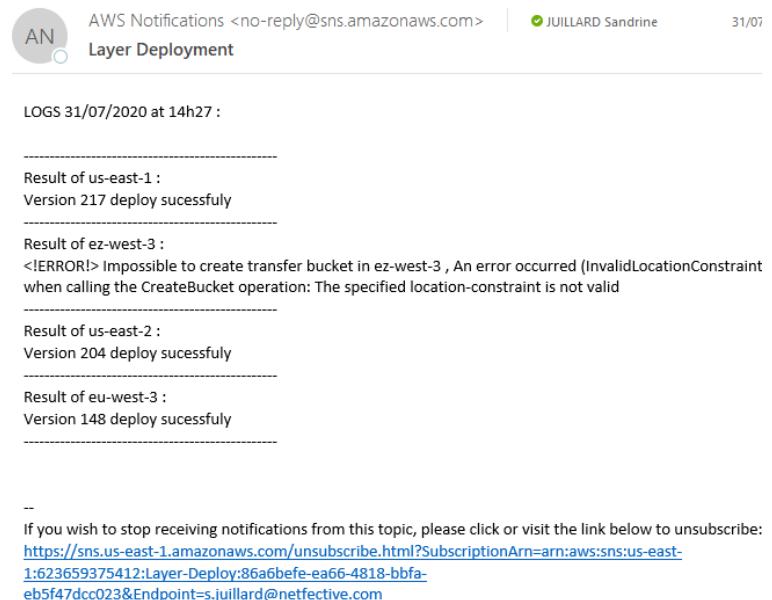


Figure 7 : Schéma de la gestion des Notifications SNS selon la fonction

Dans le cas où l'erreur n'a pas été intercepté par la Lambda elle-même, et qu'elle n'est pas en capacité d'invoqué. C'est grâce au mécanismes mise en place par Amazon que l'on envoie un message SNS généré automatiquement. Ce mécanisme est identique à celui qui lie le bucket source avec la 1^{er} fonction du pipeline et est configurable dans la console d'aws



Gestion des ACL

Pour éviter de devoir éditer à la main le fichier ACL.json, une fonction lambda annexe a été créée, afin de gérer automatiquement la mise à jour ce fichier. Blu Age stock les informations de ces clients dans une ressource Amazon sous forme de base de données : DynamoDB. La fonction ACL Conversion a pour but de convertir la base de données en un fichier ACL.json. Cette fonction est déclenchée par la modification de la base de données. Elle effectue une requête pour récupérer la liste des clients ainsi que leurs identifiant, puis édite le fichier ACL à partir de ces informations. Enfin, elle charge ce fichier sur le bucket source donnée en paramètre (en tant que variable d'environnement de la fonction).

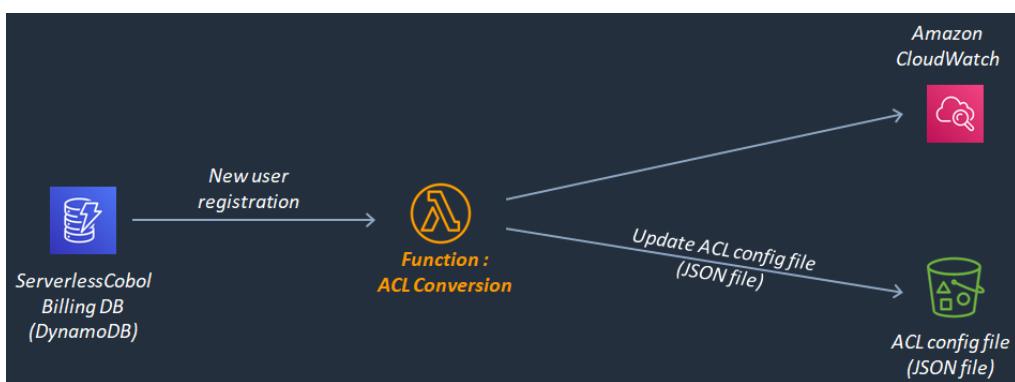


Figure 8 : Schéma fonctionnelle de la mise à jour automatiquement le fichier ACL

Génération de bucket temporaire

Un bucket doit avoir un nom unique. Si un bucket porte le même nom il sera alors impossible de le créer. Le nom d'un bucket temporaire. Le nom d'un bucket temporaire est généré de la manière suivante : [SOURCE_BUCKET] -[REGION]-temporary. (Ex : in us-east-2 à Test-Pipeline--Layer-us-east-2-temporary). Si ce nom est déjà pris, une fonction implémentée de manière récursive sera chargée de trouver un nom disponible en générant une chaîne de caractère aléatoire et de taille variable qui sera ajouter au nom par défaut.

IV - Travaux complémentaires

BASCAC

BASCAC est une petite application web qui permet de simplifier la gestion de la facturation de Velocity au client. Celle-ci s'appuie sur la base de données client (identique à la base de données qu'utilise le pipeline pour récupérer les ACL). L'application permet entre autres, d'ajouter/retirer la permission à un client d'utiliser Velocity, ou encore de gérer le pourcentage de réduction accordé au client. En conséquence, son accès doit être limité car elle gère des données de facturation.

L'objectif de cette tâche est de mettre en ligne BASCAC de manière sécurisée, en limitant son accès par un mot de passe. J'ai utilisé l'IDE Visual Studio Code pour y ajouter mes modifications. L'outil qui m'a été recommandé est AWS Cognito, un service d'Amazon pour les applications web, qui propose de gérer les mots de passe.

L'application a été réalisé avec Angular, soit codé en TypeScript, html et css. L'application a été naturellement une "single-page application" ou SPA. J'ai dû rajouter à cette application un mécanisme de routage afin d'y ajouter une page d'authentification. Une fois cela terminé, j'ai essayé d'intégrer Cognito à ma page d'authentification. C'est à cette étape que j'ai pris connaissance de Amplify. Un autre service d'Amazon qui permet de mettre en ligne des applications. Pour la mise en place de Cognito, son utilisation a été nécessaire. En me renseignant plus sur cet outil, je me suis aperçu que je pouvais déployer BASCAC, après l'avoir compilé, de manière non-programmatique sur la console graphique d'Amazon. Puis une fois cela fait, ajouter la gestion des mots de passe, qui apparaît comme un pop-up en haut de la page. Au final, je suis revenu sur mon travail d'ajout de page d'authentification et d'utilisation de Cognito et j'ai opté pour un déploiement et une gestion des mots de passe non-programmatique, à l'aide d'Amazon.

N.B : Sur demande de son créateur, j'ai également modifié légèrement le style de la page en ajoutant du contenu au feuilles de style de l'application.

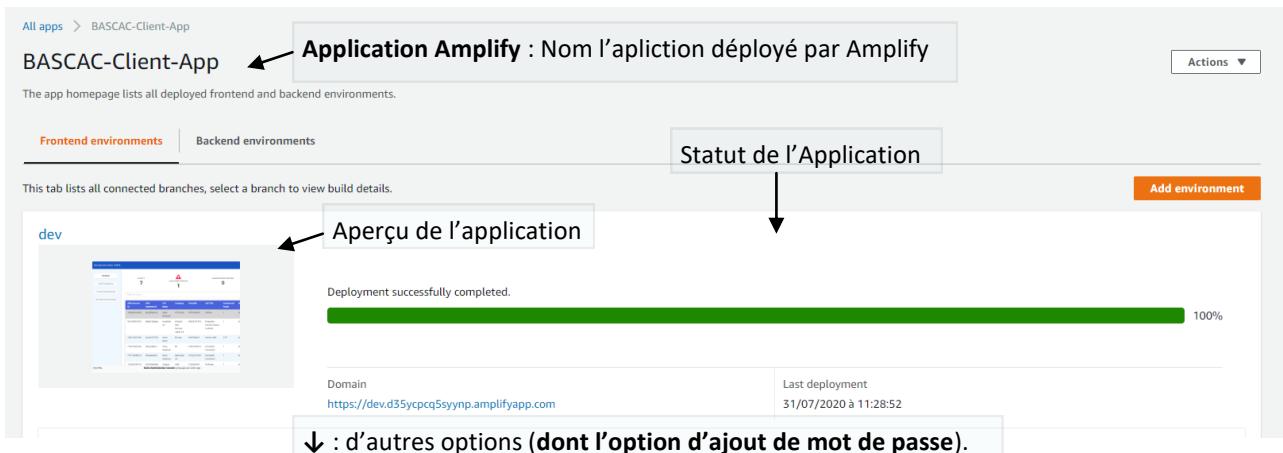


Figure 9 : Capture d'écran de la console Amazon Amplify

Exportation de la Vue Problème de Blu Age Analyzer.

Analyzer est un produit de Blu Age de la V7 <ALORS LA J'AI BESOIN D'EXPLIQUATION POUR SON USAGE>. Lorsqu'un client rencontre des problèmes avec son code, il peut contacter Blu Age afin de recevoir de l'aide. On lui demander alors le contenu de l'onglet « problème », l'onglet dans lesquels est affiché les erreurs et les warnings issue de l'a compilations du/des projets en cours.

L'objectif de cette tâche est d'ajouter, dans l'onglet problème une option pour exporter le contenu de la vue au format .csv (tableurs).

Blu age Analyser est une IDE développé sur la base d'Eclipse. Elle possède à la fois des onglets, vues et fonctionnalités développées par Blu Age et d'autre issue de Eclipse. En xml, on peut ajouter des éléments graphiques à Analyzer, et les liés avec des handlers, commandé en java. La difficulté de cette tâche réside dans l'adaptation de code interne à Eclipse. L'onglet « problèmes » d'Analyzer, est une vue dont le code est interne et il est donc difficile d'en récupérer les éléments. Pour cette raison, il n'est pas possible de récupérer le contenu de la vue problème à proprement parlé. Pour récupérer ce contenu, il a fallu aller à la source de ce contenu : On va récupérer les informations « à la source », soit dans les markers d'un projet. Un marker est un ensemble de donnée générée dans notre cas à la racine du projet dans lesquels les informations relatives à celui-ci sont stocker. C'est en récupérant ces informations que j'ai pu reconstituer la vue problème, puis l'exporter sous le format csv. Après son implémentation, j'ai pris connaissance d'une option similaire, qui avait déjà été implémenter, mais qui avait un usage différent. Cette option proposait d'exporter, à la racine du projet, dans plusieurs formats possibles, les problèmes relatifs à un projet en faisant une clique droite sur celui-ci dans la vue navigation. J'ai donc dans un deuxième temps, adapter ma solution pour quelle s'intègre au travail déjà effectuer. Ainsi, sans retirer l'option déjà implémenté, j'ai ajouté à la vue problème une option qui propose d'exporter la vue problème (problèmes de tous les projets ouvert), dans plusieurs formats différents, dont le csv, et de l'enregistrer par un menu contextuel en choisissant le nom et l'emplacement du rapport d'erreurs. Une fois la tâche terminer, j'ai pris connaissance des protocoles de Blu Age afin de manager le suivit des commit. Avec l'aide de Oliver <Nom de famille> j'ai revu l'écriture de mon code, j'ai rempli mon message de commit avec un identifiant qui correspondais à ma tâche, puis on a rempli ensemble cette fiche pour présente l'option implémenté.

N.B : J'ai également dû réfléchir à la méthode programmatique que j'utilisais pour écrire dans écrire dans un fichier. Le premier jet que j'ai rendu était trop lent.

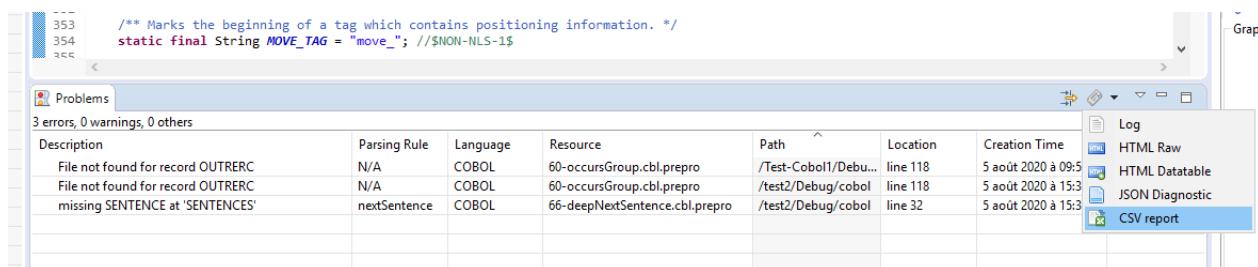


Figure 10 : Vue problème et son menu déroulant permettant l'export des problèmes

Plugin Blu Age Cobol – Tâche 1 : Phase de vérification

Afin de permettre à un développeur COBOL de compiler leur application dans un autre langage, Blu Age met disposition de leurs client un Plugin VSC. Ce plugin ajoute à l'IDE plusieurs commandes pour compiler leurs codes, en faisant appel à un server par l'intermédiaire d'une requête http (basé sur une architecture REST [10]). Plusieurs types d'opérations sont proposées par l'extension : D'une part, des commande « backend » pour la compilation du code Cobol vers java, produisant un fichier jar. Et d'autre part, une commande « frontend » pour la génération d'une application web à partir de ressource CICS (dont fichier bms[11]), produisant un fichier war.

Lorsque le client appelle une de ces commandes sur Visual, une phase de vérification, va au préalable empêcher la compilation si le fichier sélectionné n'est pas du cobol, et ce, peut-importe la nature de la commande invoquée. Or, pour la compilation d'un fichier BMS, la présence d'un fichier Cobol n'est pas nécessaire.

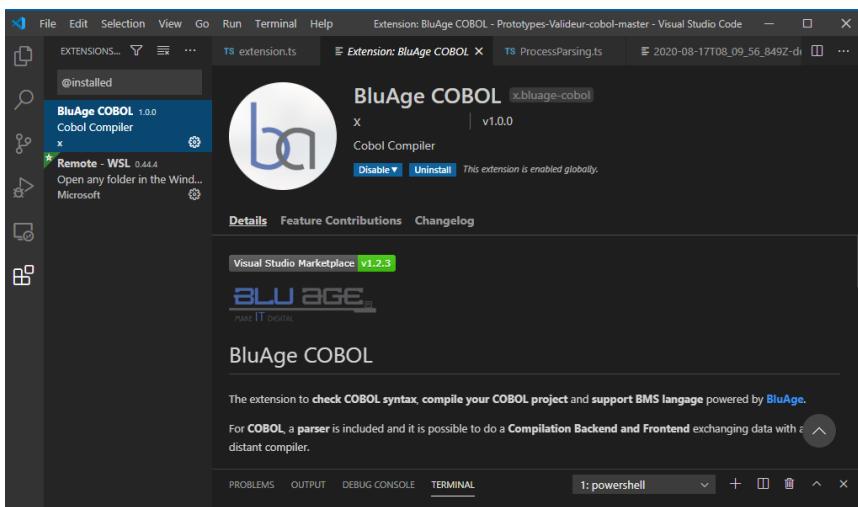


Figure 11 : Capture d'écran de l'extension BluAge COBOL dans l'IDE VSC

L'objectif de cette première tâche est de séparer les phases de vérification pour qu'elle soit adapter au type de compilation invoqué. La modification un fois effectué, j'ai généré à partir du code un fichier 'vsix' ce fichier pourra permettre à un utilisateur d'ajouté l'extension.

```

PROBLEMS 71 OUTPUT DEBUG CONSOLE TERMINAL
BluAge Console

*** Frontend compilation command called ***
[17-08-2020 14:47:38] ERROR - File to parse is not BMS : extension-output-#4.
*** Frontend compilation command ended ***

*** Frontend compilation command called ***
[17-08-2020 14:47:49] Cobol code validation.
*** Frontend compilation command ended ***

*** Frontend compilation command called ***
[17-08-2020 15:07:07] ERROR - File to parse is not BMS : MKCT00.cbl.
*** Frontend compilation command ended ***

*** Compilation command called ***
[17-08-2020 15:07:27] ERROR - File to parse is not cobol : MKCTSET.bms.
*** Compilation command ended ***

```

Figure 12 : Résultat dans la console de la phase de vérification, on voit que lorsque le type de fichier sélectionné n'est pas bon, la phase de validation échoue

NB : Le fichier jar produit, issue de la compilation backend par l'extension, pourra ensuite être exécuter sur une fonction Lambda d'Amazon ayant une Layer contenant le Framework Velocity (c'est la layer dont nous avons construit la chaîne de déploiement). (cf. contexte technique).

Plugin Blu Age Cobol – Tâche 2: Ajout d'une commande

Les applications CICS (Customer Information Control System) sont des systèmes qui permettent d'effectuer des opérations transactionnelles (en général consultation ou mise à jour de bases de données ou de fichiers). Généralement codé en COBOL, c'est couramment ce type d'application qui doit être transcrit par l'extension de Blu Age. En plus des fichiers COBOL, des fichiers bms, pour l'apparence de l'application et un fichier textuel de description de ressource sont généralement présents. Lors de la compilation, le texte de description de ressource est parsé pour en faire une suite de requêtes SQL qui permet de former et alimenter une base de données liée à l'application.

L'objectif de cette tâche est d'ajouter une commande à l'extension Blu Age Cobol afin de récupérer un fichier SQL généré à partir d'un fichier textuel de description de ressources.

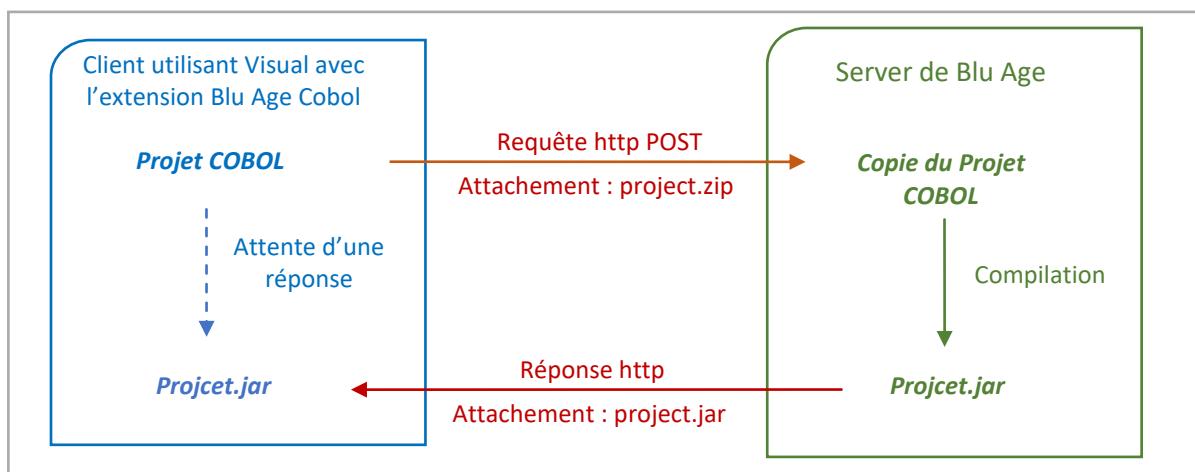


Figure 12 : Schéma fonctionnelle de la compilation par l'extension Blu Age Cobol (architecture REST [10])

Le développement des mécanismes présentés a évidemment été inspiré des commandes de compilations Blu Age déjà présente. Mon travail a été d'ajouter à une architecture déjà présente, une fonctionnalité supplémentaire. Tous d'abord, côté frontend, une commande nommée « Blu Age parsing CSD to SQL » a été créé en type script avec l'IDE Visual Studio Code. Lorsque celle-ci est invoquée, le fichier CSD sélectionné va être transmis au serveur COBOL par le biais d'une requête http. Puis côté server, la requête est traitée : A l'aide des outils déjà présents, le fichier CSD réceptionné est traduit en SQL, zip-pé, puis renvoyé au client. (Développement en java, avec Eclipse). Ensuite, à nouveau côté client, la réception des réponses du serveur a été adaptée pour que le fichier soit récupéré correctement.

Le code affiche les logs de la commande CSD to SQL et la structure de l'explorateur de fichiers.

```

*** CSD to SQL command called ***
[20-08-2020 15:32:39] Headless compilation launched.
[20-08-2020 15:32:39] Temporary directory created.
[20-08-2020 15:32:39] The workspace directory was packed (15723 total bytes).
[20-08-2020 15:32:39] Server path is: http://localhost:9191/compile .
[20-08-2020 15:32:39] Archive packaged and ready to be sent.
[20-08-2020 15:32:39] Directory created to receive server data.
[20-08-2020 15:32:39] Connection with the compilation server...
[20-08-2020 15:32:39] HTTP Response: 200 OK.
[20-08-2020 15:32:39] Reception of server response.
[20-08-2020 15:32:40] File decompressed.
[20-08-2020 15:32:40] initJics.sql moved to result directory.
[20-08-2020 15:32:40] Compilation succeeded!
*** CSD to SQL command ended ***

```

A droite, l'explorateur de fichiers montre les résultats générés :

- Un dossier "MK-MOD" contenant :
 - Un dossier ".vscode_cobol"
 - Un dossier "bin" contenant :
 - Le fichier "initJics.sql" (surligné en bleu)
 - Le fichier "MK-MOD-service-1.0.0.jar"
 - Les fichiers "CSDLIST.TXT" et "DB2 Data.txt"

Figure 13 : Résultat de la commande CSD to SQL : A droite, les logs dans la console, à gauche le fichier SQL généré

Conclusion

La forme finale du pipeline a pu être développé jusqu'au bout. Tous les aspects de la chaîne ont pu être traité : De la gestion des erreurs en passant par les notifications, la récupération des objets nécessaires au déploiement (fichier ACL, fonction teste), ainsi que la documentation de la chaîne.

Le dernier aspect qui pourrait éventuellement manquer est la réalisation de teste unitaire pour vérifier le bon fonctionnement du pipeline. La fiabilité du pipeline n'a à ce jour qu'était tester par des teste manuellement fait au fur es à mesure de son développement. A cause de problématiques de droit d'accès, mon pipeline est encore en stade de prototype (elle n'a pas été paramétré pour déployer les vrais fichiers de Velocity) mais tout à été mise en place pour que ce travail soit simple. Une documentation détaillée de 14 pages, en anglais é été crée dans cette optique. De plus, j'ai porté une attention toute particulière à la clarté de mon code.

Les 4 tâches supplémentaires que j'ai pu effectuer j'ai pu toute les terminer également. J'ai pu push mon travail sur le git de Blu Age. Parfois, mes push on crée des problèmes en particulier au niveau des importation de packages, lorsque j'ajouter des nouvelles dépendances. Mon code à également pu être revu par les personnes qui m'encadrées.

D'un point de vu personnelle, ce stage m'a permis de prendre en compétence, et ce dans diverse do-maine : devOps, devWeb et dev logiciel. J'ai codé dans plusieurs langages différents, utilisé et tester divers outils. Cette expérience m'as permis de développer ma capacité d'adaptation et ma flexibilité. Grâce au revus de code, j'ai eu également appris des bonnes pratiques de développement que je ne connaissais pas.

J'ai dû moi-même concevoir des solutions, chercher par moi-même les outils qui pourrais me servir et faire le choix des plus adéquats. Pour la tâche d'implémentation d'une fonctionnalité d'export, j'ai pu me familiariser avec les problématiques de la programmation d'un grand projet, ou plusieurs acteurs différent commit du code. Ce que l'on ne rencontre pas forcément dans un projet en milieu scolaire. J' ai aussi du crée de la documentation pour le pipeline que j'ai créée, pour qu'elle puisse être reprise plus tard. Enfin, j'ai eu l'immense satisfaction de terminer mon travail et d'en être fière.

Références

- [1] « Ce qu'il faut retenir d'AWS re:Invent 2018», ITForBuisness, Laurent Delattre , 2018
Lien : <https://www.itforbusiness.fr/ce-qu-il-faut-retenir-d-aws-re-invent-2018-18837>
- [2] « Atout et limites du serverless computing », ITForBuisness, Laurent Delattre , 2018
Lien : <https://www.itforbusiness.fr/atouts-et-limites-du-serverless-computing-18219>
- [3] « Démo - Modernisation d'application avec Blu Age », BluAge, 2017
Lien : <https://www.youtube.com/channel/UCREdw7o0hKmqWFt1BjUmTuQ>
- [4] Serverless COBOL - Quickstart,, BluAge
Lien : <https://www.bluage.com/products/serverless-cobol-quickstart>
- [5] « A short story of serverless COBOL for AWS », BluAge, 2019
Lien : <https://github.com/BluAge/ServerlessCOBOLforAWS>
- [6] « A Streamlined Journey from Legacy to Microservices with Blu Age », BluAge, Avril 2019
Lien : <https://www.youtube.com/watch?v=jhB39NIgGl4&feature=youtu.be&t=2806>
- [7] « AWS Lambda Language Comparison : Pros and Cons », Yan Cui, Octobre 2018
Lien : <https://epsagon.com/development/aws-lambda-programming-language-comparison/>
- [8] Invocation Asynchrone et Invocation Syncrone , Documentation Amazon Web Services, 2020
Liens : <https://docs.aws.amazon.com/lambda/latest/dg/invocation-async.html>
<https://docs.aws.amazon.com/lambda/latest/dg/invocation-sync.html>
- [9] « Managing AWS Lamnda fucntion Concurrency», Chris Munns , Decembre 2017
Lien : [https://aws.amazon.com/fr/blogs/compute/managing-aws-lambda-function-concurrency/#:](https://aws.amazon.com/fr/blogs/compute/managing-aws-lambda-function-concurrency/#:~:text=AWS%20Lambda%20functions%20are%20designed%20to%20execute%20asynchronous%20code%20in%20a%20serverless%20environment%20and%20scale%20automatically%20based%20on%20the%20volume%20of%20events%20they%20receive.)

Annexe 1 : Capture d'écran de BASCAC

The screenshot shows the BASCAC Administration Control interface. At the top, there are tabs for 'CLIENTS' (selected), 'AWS Accounts', and 'AWS Services'. Below the tabs, there are buttons for 'Refresh', 'Edit Properties', 'Grant Permissions', and 'Revoke Permissions'. On the left, there is a sidebar with a red warning icon and the number '1'. The main area displays two tables: one for 'CLIENTS' and one for 'AWS Accounts'.

CLIENTS

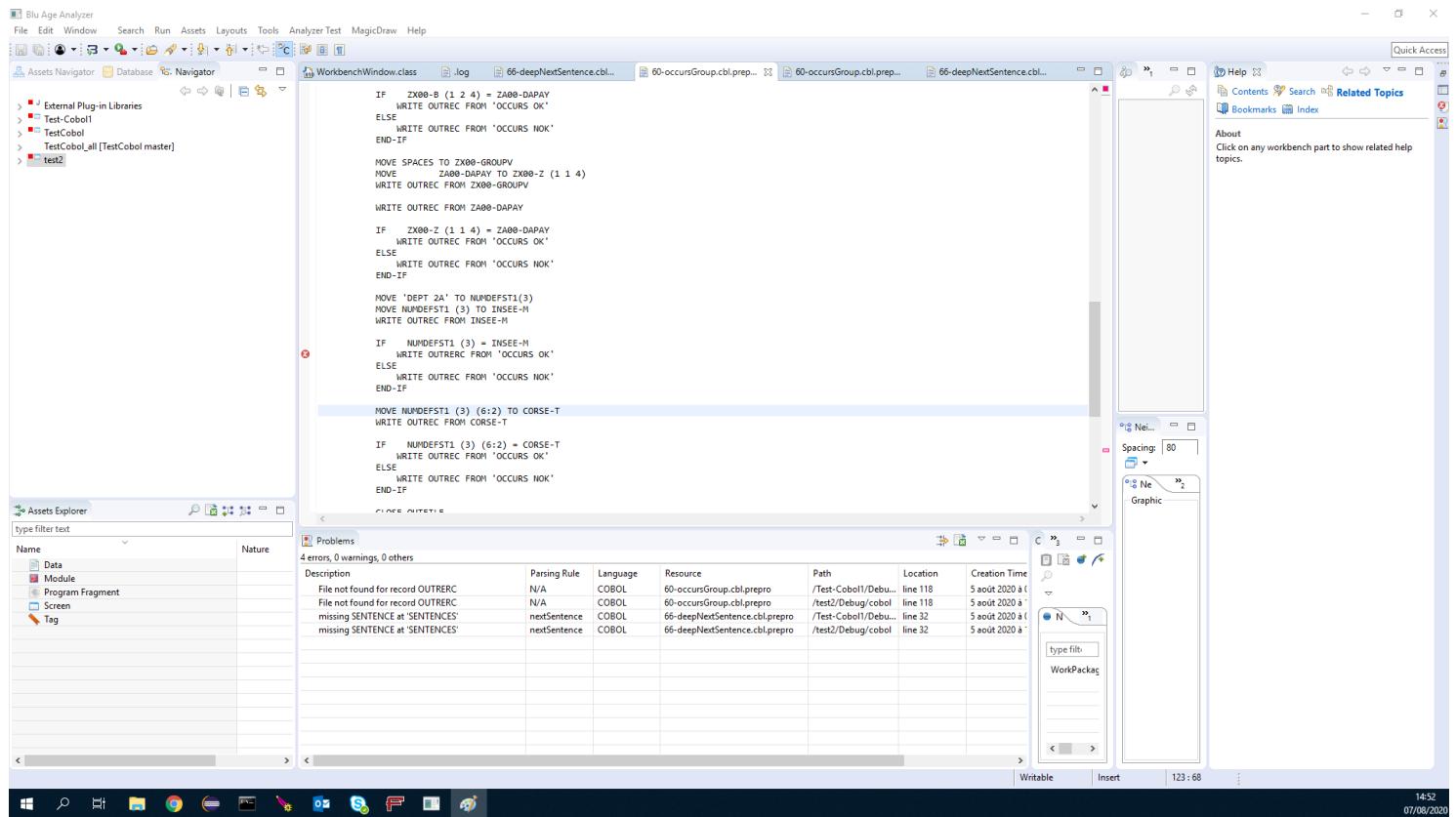
Client ID	Name	Status
UNSUBSCRIBED-PENDING	0	
SUBSCRIBED-PENDING	1	

AWS Accounts

AWS Account ID	AWS Customer ID	Full Name	Company	PhoneNb	Job Title	Commercial Factor	Status
18606320999	f3zy0Dtgmlp	Gnat Shabada	VITG Corp	9787648395	Partner	1	Subscribed
942108974323	6NBq1jWqmc	Fusahide Ito	Amazon Web Services Japan k.k.	09042797578	Enterprise Transformation Architect	1	Subscribed
106310521300	jzU23jTA7M	alexis henry	Blu Age	0647000441	Director R&D	0.75	Subscribed
778224641626	4RGp8jEoTJ	Harry Rackham	Mr	07401994672	Innovation Consultant	1	Subscribed
775110548216	fRdydaDisk	Harry Rackham	Santander UK	07523313952	Innovation Consultant	1	Subscribed
123456789123	fCFYSvqRNQb	Gregory Moody	AWS	1234567891	Technical Account Manager	1	Subscribed-pending
78285828506	iEBw9jkTeuT	Anu Gurudu	Amazon	9001323200	TAM	1	Subscribed

At the bottom right, there is a footer with the text 'BASC Administration Console by bluage.com with rage.' and various system icons.

Annexe 2 : Analyzer, Export CSV



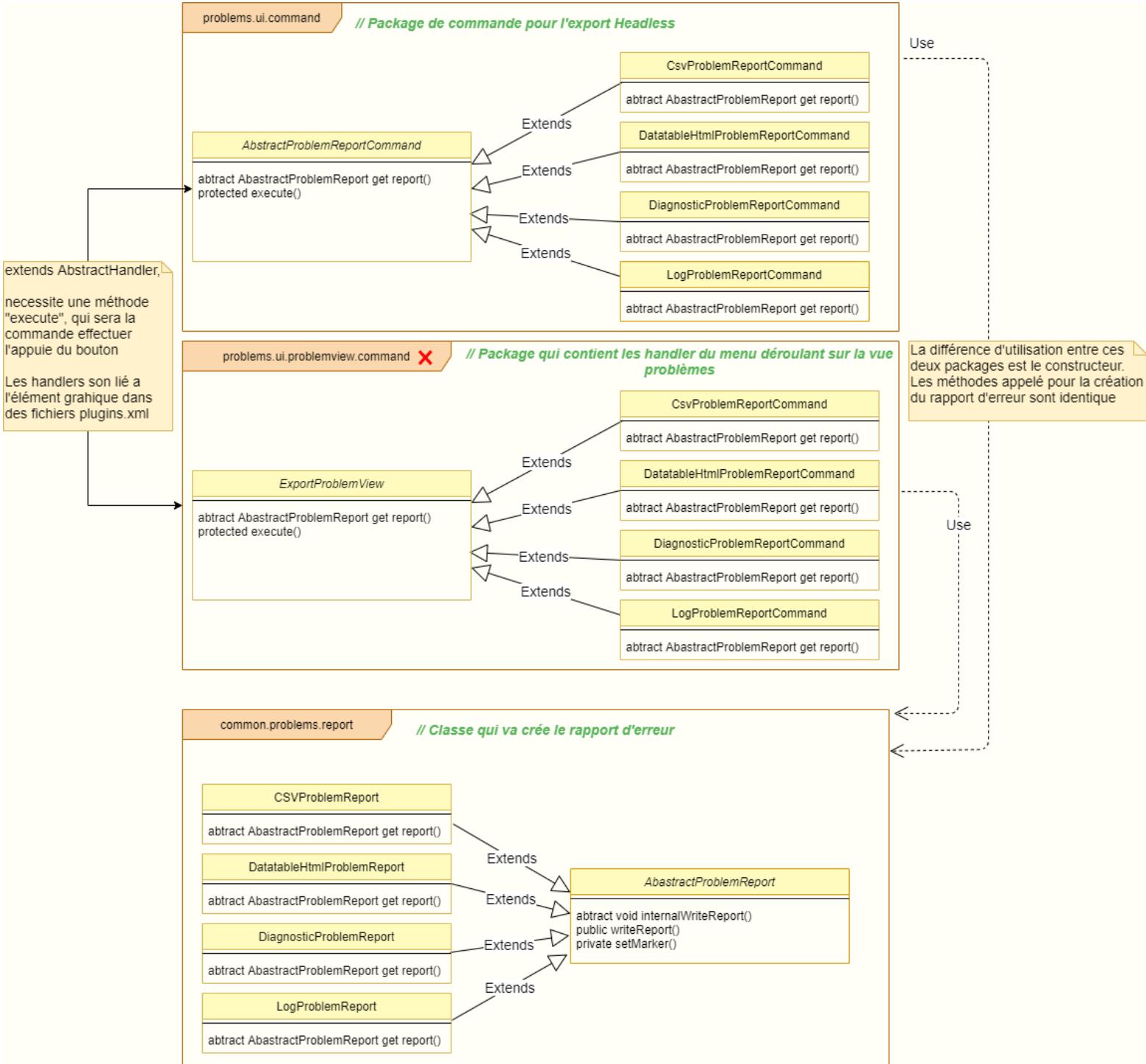
Annexe 2A : Capture d'écran de Analyzer

The screenshot shows an Excel spreadsheet titled 'Classeur1 - Excel'. The ribbon is visible with the 'Données' tab selected. In the 'Obtenir des données' section of the ribbon, the 'À partir d'un fichier texte/CSV' button is highlighted with a red arrow. The main area of the screen shows a table with columns: Description, Parsing Rule, Language, and Resource. The data rows correspond to the errors listed in the Analyzer's Problems view.

Description	Parsing Rule	Language	Resource
File not found for record OUTRERC	COBOL	cobol/60-occursGroup.cbl	
File not found for record OUTRERC	COBOL	cobol/60-occursGroup.cbl	
File not found for record OUTRERC	COBOL	cobol/60-occursGroup.cbl	
missing SENTENCE at 'SENTENCES'	nextSentence	COBOL	cobol/66-deepNextSentence.cbl
missing SENTENCE at 'SENTENCES'	nextSentence	COBOL	cobol/66-deepNextSentence.cbl

Annexe 2B : Résultat de l'export CSV, lorsqu'il est ouvert avec Excel

Annexe 3 : Diagramme SQL des classes utilisé pour l'export de la vus problème



Légende :

Les éléments marqué d'une croix rouge correspond au classe/Package créé.
Les éléments marqué d'une croix verte correspond à ceux qui à été modifié.