

NOM : Juillard

PRENOM : Sandrine

FILLIÈRE : Télécommunication



Création d'une chaîne de déploiement dans l'environnement Amazon Web Services

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

DENOMINATION DE L'ORGANISME D'ACCUEIL : Blu Age

ADRESSE : 32 av Léonard de Vinci, 33600 PESSAC

PAYS : FRANCE

Abstract

Au sein de l'environnement Amazon Web Service, les microenvironnements appelés Lambda peuvent être programmés pour construire un flow d'exécution capable d'interagir avec les autres ressources que la plateforme propose. Ce rapport détaillera la conception et la réalisation d'une architecture basée sur les fonctions Lambda permettant le déploiement d'un runtime sous forme de Layer AWS. Ce layer sera utilisé par des applications COBOL traduites en JAVA et déployées sur des fonctions Lambda AWS.

Ayant terminé la réalisation du pipeline plus tôt, ce rapport contiendra également une présentation du déploiement d'une application avec Amplify ainsi que des tâches de développements sur le logiciel Analyzer de Blu Age et l'extension Blu Age Cobol sur Visual Studio Code.

Tables de Matière

| | |
|---|-------------|
| Introduction | p.4 |
| I. Blu Age | p.5 |
| II. Contexte Technique | p.6 |
| 1) L'environnement Amazon Web Services | p.6 |
| 2) Serverless COBOL for AWS solution | p.7 |
| III. Réalisation de chaîne de déploiement | p.8 |
| 1) Etape de recherche | p.8 |
| 2) Solution retenue | p.9 |
| 1. Aperçu global | p.9 |
| 2. Détails de la solution | p.11 |
| IV. Travaux complémentaires | p.14 |
| 1) Blu Age Serverless Cobol Administration Cobol (BASCAC) | p.14 |
| 2) Export de la vue problème d'Analyzer | p.15 |
| 3) Plugin Blu Age Cobol — Tâche 1 : Phase de vérification | p.16 |
| 4) Plugin Blu Age Cobol — Tâche 2: Ajout d'une commande | p.17 |
| Conclusion | p.19 |
| Références | p.20 |
| Annexes | p.21 |

Listes des Abréviations

| | | | |
|--------|---|--------|---|
| AWS | Amazon Web Services | DSL | Domain-Specific Language |
| Cobol | Common Business Oriented Language | BASCAC | Blu Age Serverless Cobol Administration Cobol |
| API | Application Programming Interface | ARN | Amazon Resource Names |
| DevOps | Software <u>Development</u> (<i>Dev</i>) IT <u>operations</u> (<i>Ops</i>) | UX | User Experience |
| VM | Virtual Machine | CSD | Customer Service Description |
| IDE | Integrated Development Environment | MDA | Model Driven Architecture |
| SPA | Single-Page Application | | |
| 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" [1] [2]. Composé d'une extension VSC permettant de compiler une application COBOL en java. Le résultat de cette compilation pourra alors être déployer de manière « serverless » sur AWS. C'est le premiers produit de Blu Age à traduire du code, sans nécessité aucune intervention humaine

Pourquoi la nécessité de compiler en Java un code écrit en COBOL ? Car encore aujourd'hui, l'activité de beaucoup d'entreprises reposent sur des technologie en obsolescence et qui s'opposent au standards actuels et futurs . La plupart des solutions de Blu Age sont des outils pour assister les développeurs dans la traduction de codes legacy vers des langages plus moderne comme Java ou .Net.

Néanmoins, des enjeux sociaux peuvent freiner la transition digital des l'entreprises. En effet, il peut être difficile pour des développeurs COBOL (dont la moyenne d'âge est de 60ans) de se reconvertir vers le Java. Les entreprises peuvent faire le choix d'accompagner ces développeurs en maintenant leurs activité jusqu'à la retraite. C'est pour cela qu'a été conçu le produit « Serverless COBOL for AWS».

Le développeur COBOL pourras continuer à codé en COBOL, tout en produisant une application compatible avec le déploiement « serverless », sur les microenvironnements d'Amazon que l'ont appelle les Lambdas. [3]

Pour fonctionner, ces Lambdas devront être configurer avec un Runtime personnalisé développer par Blu Age. Il devra être fournis aux clients en tant que Layer sur Amazon Web Services., De cette manière il sera en mesure de configurer sa Lambda en y ajoutant la Layer, c'est-à-dire en ajoutant une couche supplémentaire pour personnalisé l'environnement d'exécution. [4] (cf II-2- L'environnement Amazon Web Services)

A l'heure actuelle, **ce Runtime est mise à disposition du client de manière manuelle**. A partir de la console graphique de Amazon, il est déployé manuellement dans toute les régions souhaitées. L'opération est répétitive et rébarbative. Il semble alors intéressant de l'automatiser.

Objectif : Crée un pipeline pour le déploiement du Runtime de serverless COBOL for AWS.

Il existe déjà une chaîne d'intégration en charge de compiler et tester le Runtime. A l'issue de ce traitement, ont obtient les fichiers de la couche sont sous forme de zip. C'est la deuxième partie de la chaîne, chargé du déploiement sur Amazon qu'il faudra concevoir.

I - 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 nouveaux besoins, émergeant des entreprises, aussi bien privé que public, ont vu le jour. En particulier, les évolutions très rapides de ces technologies demandent de se renouveler constamment. C'est pour répondre à cette demande qu'a été créé Netfective Technology, et la suite logicielle Blu Age. Crée en 2000 et dirigé 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 grandes entreprises tel que Amazon, BNP PARIBAS, Orange, Spora Steria, Accenture.. Mais aussi des administrations gouvernementales 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 activités sont séparées dans ces deux filiales : Blu Age et OptTeam. Blu age d'une part, chargé de la technologie de migration ainsi que de la gestion du cycle de vie des logiciels modernisés, Optteams est spécialisée dans la formation et le consulting pour les architectures Cloud

| Age moyen | Chiffre d'affaire |
|---------------------------|--------------------------|
| | |
| Pays implanté | Nombre de Collaborateurs |
| France (Pessac, Surenne) | 160 |
| Maroc (Casablanca, Rabat) | Turnover |
| États-Unis (Dallas) | 5% |

Figure 1 : Tableau informatif sur Netfective

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 logicielle de Blu Age va automatiquement extraire la logique de l'application, et automatisera le travail de réécriture et de refactoring. Cette suite fournira en plus des outils de générations automatique de code des outils de visualisation du code et des outils d'analyse. [5]

II - Contexte technique

1) 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.

Parmi les services qui vont être utilisés, il est essentiel d'en introduire trois :

Tous d'abord, **les Lambda AWS**. C'est l'environnement d'exécution sur laquelle le client pourra déployer son application. Cet environnement à la particularité d'être « sans état » et de ne nécessiter ni mise en services, ni gestion de serveur. C'est le fournisseur d'accès (c.-à-d. Amazon) qui se charge de l'administration des ressources; entre autres, la maintenance des servers, le dimensionnement et la mise à l'échelle de la capacité, la surveillance et la journalisation des exécutions.

Pour l'utilisateur, ce service permet de déployer simplement n'importe quel type d'application en étant facturé uniquement pour le temps de calcul utilisé. Lorsque la fonction Lambda ne s'exécute pas, l'utilisateur ne débourse rien. Ce service d'Amazon est également naturellement intégré avec la plupart des autres ressources et services que Amazon propose.

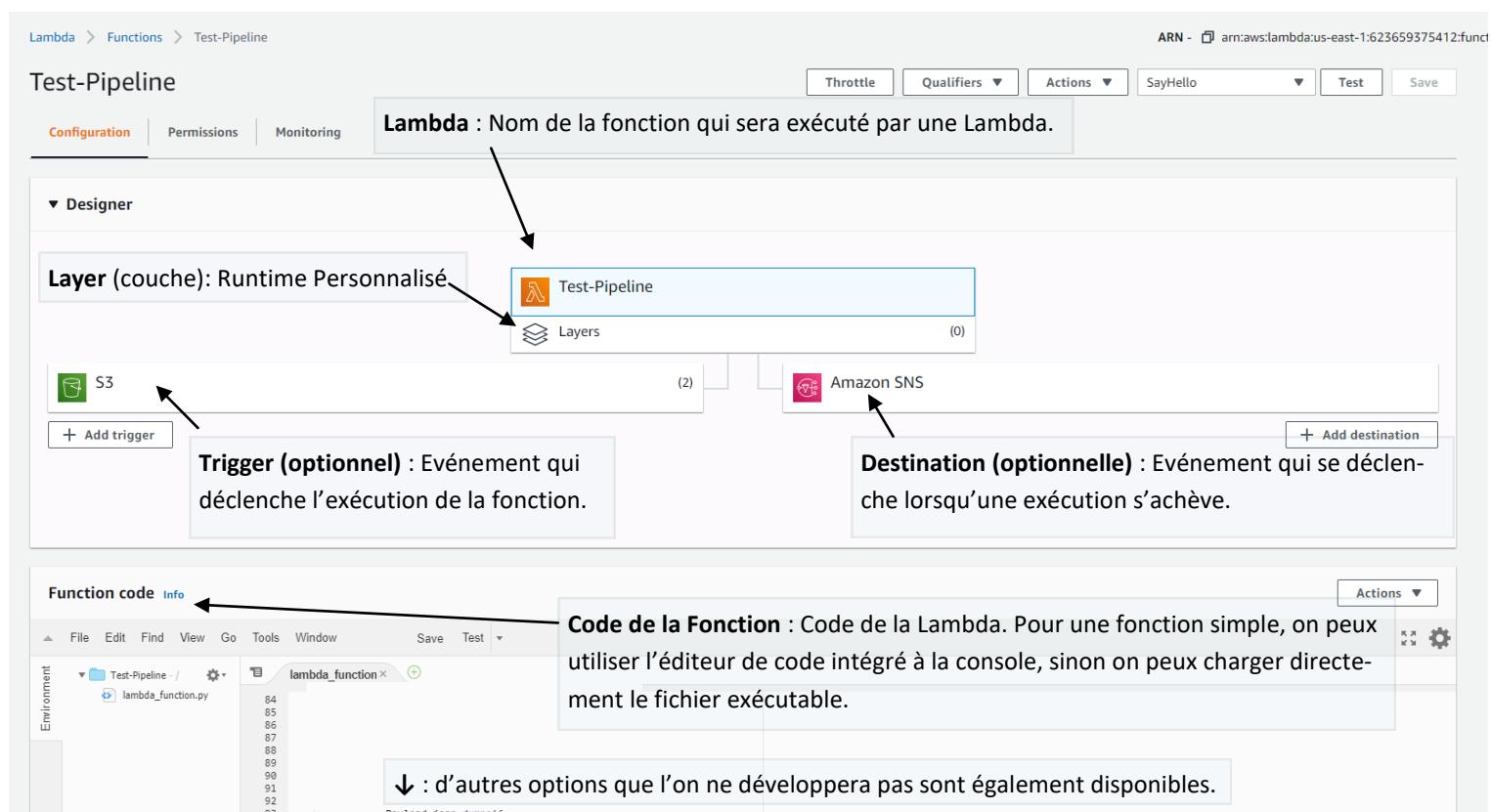


Figure 4 : Capture d'écran de la console AWS pour le paramétrage d'une Lambda

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. (car une lambda est limité en taille de code et l'inclusion de toutes les librairies en dépendance peut s'avérer un problème). AWS Layer est donc une couche de publication de framework dont le contrôle d'accès peut être paramétré.

Enfin, les **buckets**. Ce sont des compartiments de stockage fournis par 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 décrivent.

Pour finir, il est important de préciser la spécificité technique suivante : Amazon est composé de plusieurs régions. Un object (Lambda, Layer, Bucket...) lorsqu'il est créé n'existe que dans une seule région. Pour que deux ressources interagissent entre elles, elles doivent se trouver sur la même région.

2) Serverless COBOL for AWS solution

La solution *COBOL for aws* a pour but de procurer au client une solution pour déployer, sur une Lambda Amazon, une application java initialement codées en COBOL.

Elle prend la forme d'une extension VSC : *Blu Age COBOL*. (cf IV-3- Plugin Blu Age Cobol). Cette extension fait appel, par le billeter d'une requête http, à un serveur de Blu Age déployé sur AWS. Celui-ci héberge le compilateur. En retour, le serveur enverra un fichier .jar contenant le code minimum de l'application : C'est-à-dire le code de l'application moins ces dépendances.

Pour déployer son application, l'utilisateur devra créer une Lambda sur son compte Amazon, sur lequel il déployera le produit de la compilation en tant que code de la fonction. Puis pour finaliser le déploiement, il devra ajouter une Layer à cette Lambda pour y ajouter les dépendances manquantes. Il réalisera cette étapes en fournissant l'ARN de la Layer déployée sur le compte d'Amazon de Blu Age, dont il aura au préalable reçu la permission.[6] (cf Annexe 1).

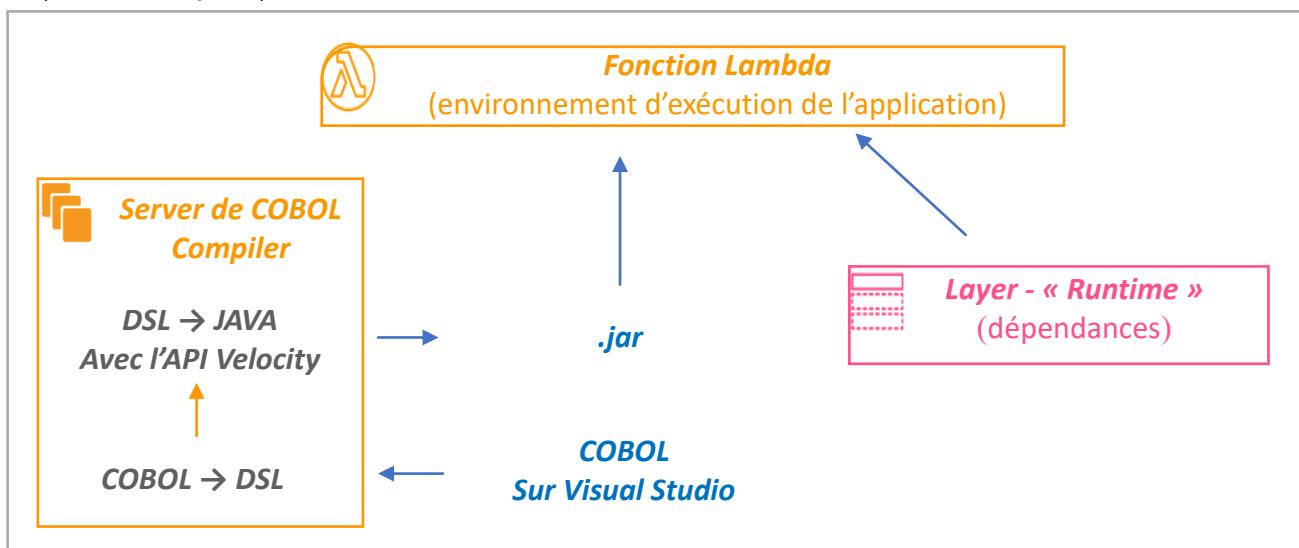


Figure 3 : Schéma fonctionnel de la solution serverless COBOL for aws solution

III. Réalisation de chaîne de déploiement

1) Etape de recherche

Livrables

La chaîne de déploiement sera en charge déployer le Runtime de Blu Age en tant que Layer sur AWS.

Elle devra également :

- S'assurer que l'accès à ces espace de stockage est configuré correctement et que seul les personnes habilitées aient accès au produit.
- Tester l'API en tant que Layer, (soit en l'ajoutant en tant que Layer à une Lambda programmer pour faire des tests).
- Faire l'ensemble de ces étapes, dans toute les régions du Cloud ou l'on souhaite que l'API soit disponible.

Le prototype devra être sécurisé, facilement paramétrables et le bon déroulement (ou pas) de l'exécution de la pipeline doit être observable.

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 Runtime. Une première solution envisagée à été d'utiliser le même outil pour la chaîne de déploiement. Cela est possible 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.

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 viens 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 pour créer un pipeline. (services qui permet d'héberger toutes les étapes : git, intégration, déploiement) tandis qu'on ne souhaite l'utiliser que pour le déploiement. Amazon Step Fonction quant à lui, utilise les lambdas comme environnement d'exécution, ce service rajoute du code supplémentaire pour lier les Lambdas entre elles (en un langage propre à Amazon, proche du json). Ce service a vocation à simplifier la gestion de workflow complexes; trop complexe pour nos besoins. L'utiliser reviens à ajouter du code supplémentaire à l'architecture qui n'aurais pas grand intérêt dans notre cas, en plus du surcoût lié à l'utilisation du service.

N.B : Plus que de m'être renseigné sur ces outils, je les ai testés en réalisant des petites ébauches de pipelines.

2) Prototype retenue

Pourquoi choisir cette solution ?

Le choix de cette solution est motivé en grande partie par sa flexibilité comparativement aux autres solutions. De plus, l'avantage de cette méthode par rapport à Jenkins est que l'on utilise quand même un outil interne à Amazon, et les opérations de la chaîne sont effectuées de manière interne au compte Amazon de l'entreprise. En d'autre terme, on élimine la nécessité de devoir se logger et de transmettre des identifiants. Cette solution propose le meilleurs compromis entre sécurité, flexibilité, coût, et simplicité de maintenance.

Aperçu global du prototype (figure 5)

Le flow d'exécution du pipeline est codé par 3 fonctions Lambdas codé en python. [7] [8] Grâce à la bibliothèque d'Amazon Boto 3, elles peuvent interagir avec les autres ressources et reproduire toutes les commandes que l'on peut effectuer à partir de la console d'Amazon de manière programmatique.

Le déploiement démarre lorsque la chaîne d'intégration se termine. À l'issue de ce traitement, la nouvelle version du Runtime, va être chargé dans le « *Bucket Source* », dans la première région. Cette événement va faire office de trigger de la fonction principale de la chaîne : « *Main Pipeline* ».

Sa première tâche va être de déployer le Runtime dans la première région. Pour cela, elle fera appelle de manière synchrone à une autre fonction [9] : « *Publish a Layer* ». Cette fonction mettra à jour la nouvelle version de la Layer (ou créera la Layer si elle n'existe pas) à partir des fichiers contenu dans le bucket. Puis testera la Layer et y ajoutera les droits d'accès à partir des ACL.

Une fois le déploiement de la Layer effectué dans la première région, et si l'opération c'est déroulé correctement, alors la fonction "Main Pipeline" effectuera sa deuxième tâche, lancé le déploiement sur toutes les régions.

Pour cela, elle déclenchera de manière asynchrone N fonction lambda nommée « *Deployer* ». Une pour chaque région. Elles ont pour rôle de déployer le Runtime dans la région qui leur a été attribué.

Elle devra commencer par créer une copie du contenu du Bucket Source dans un bucket temporaire qui devra se trouver dans leur région. Cette étape est nécessaire pour pouvoir ensuite faire appelle à la fonction « *Publish a Layer* », qui pourra reproduire les étapes de déploiement en utilisant ce bucket temporaire comme source.

Pour ce qui est de l'étape de test, il sera réalisé par une fonction Lambda codée en Java. Dans le prototype que j'ai conçu, cette fonction elle est matérialisée par une fonction *bouchon*, qui ne fait rien.

Ensuite, une dernière fonction qui ne figure pas sur le schéma figure 5, sera en charge de gérer les journaux d'exécution et l'envoie des notifications du statut du déploiement. Cette dernière permet de centraliser les informations des différents maillons de la chaîne qui s'exécute indépendamment.

N.B : L'annexe 2 fournit un récapitulatif des fonctions créées et de leurs rôles.

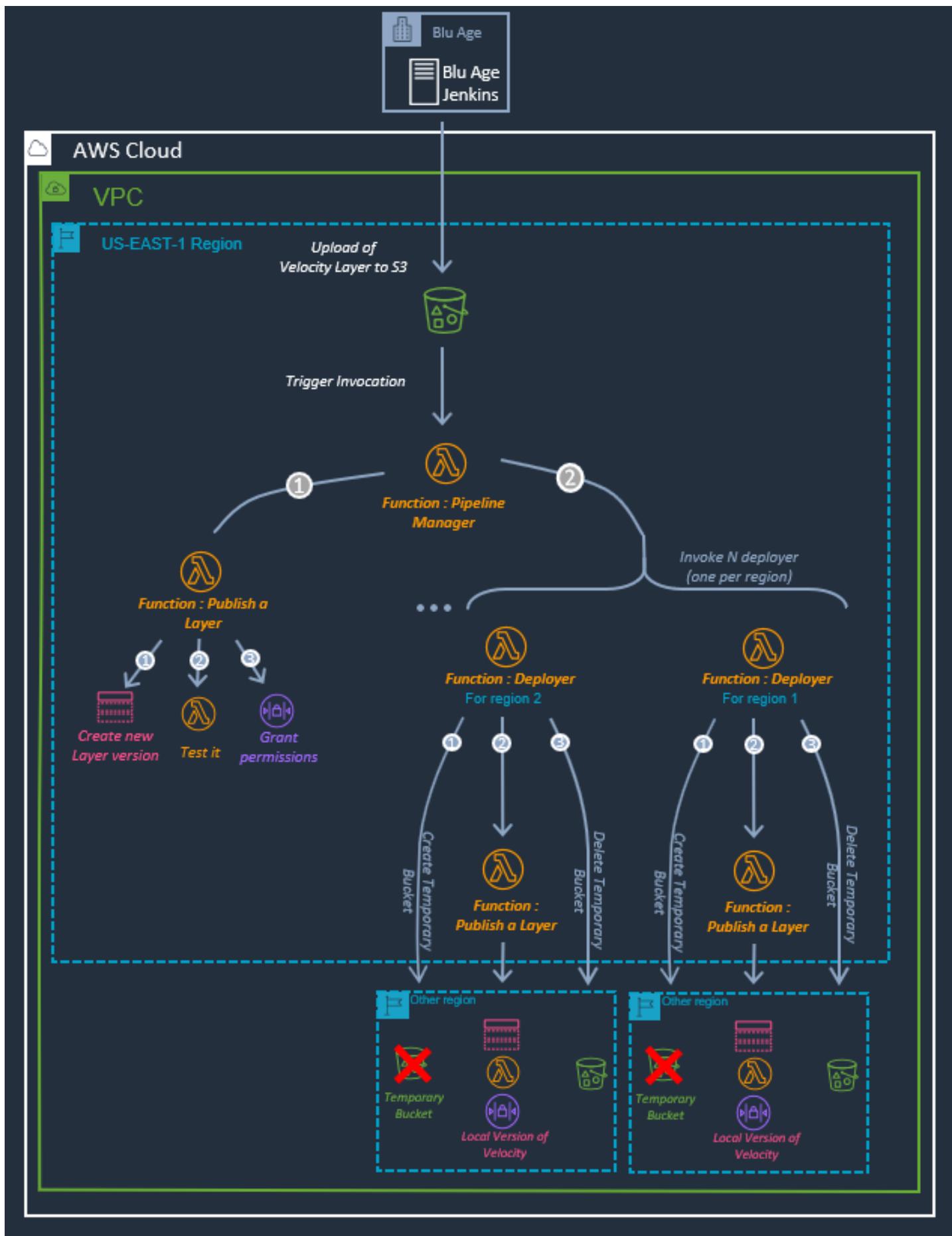


Figure 5 : Schéma de l'architecture de la chaîne de déploiement

2) Détails de la chaîne de déploiement

Paramétrage des fonctions

Tous les paramètres susceptibles de devoir être modifiés sont entrés 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 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 lequel s'exécute le Pipeline
- FILE_NAME_REQUIREMENT : Règle sur le nom du fichier qui permettra de filtrer les fichiers qui déclencheront effectivement le pipeline.
- FILE_EXTENSION : Règle sur l'extension du fichier (soit zip soit jar pour qu'elle soit reconnue 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.

Pré requis pour le fonctionnement du pipeline : La pipeline requiert que le bucket source doit contenir une fichier ACL.json (pour la gestion des permissions). Ce fichier d'ACL (Access Control) devra suivre le template figure 5. Il servira à l'administration des permissions de la layer.

```

1  {
2   "client_list" :
3   [
4     {
5       "CLIENT_NAME": "Client1",
6       "CLIENT_ID": 623659375412
7     },
8     {
9       "CLIENT_NAME": "Client2",
10      "CLIENT_ID": 623659375412
11    }
12  ],
13 }
```

Figure 6 : Exemple de ACL.json

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 retrouvent séparés 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 créé si il n'existe pas déjà). A chaque exécution, un nouveau fichier texte contenant les logs est créé. Pour différencier 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 toutes les autres fonctions invoquées.

Une fonction Lambda que l'on nommera Notification-Manager sera chargée 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 mentionnée 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 7 : 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 récupéré 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.

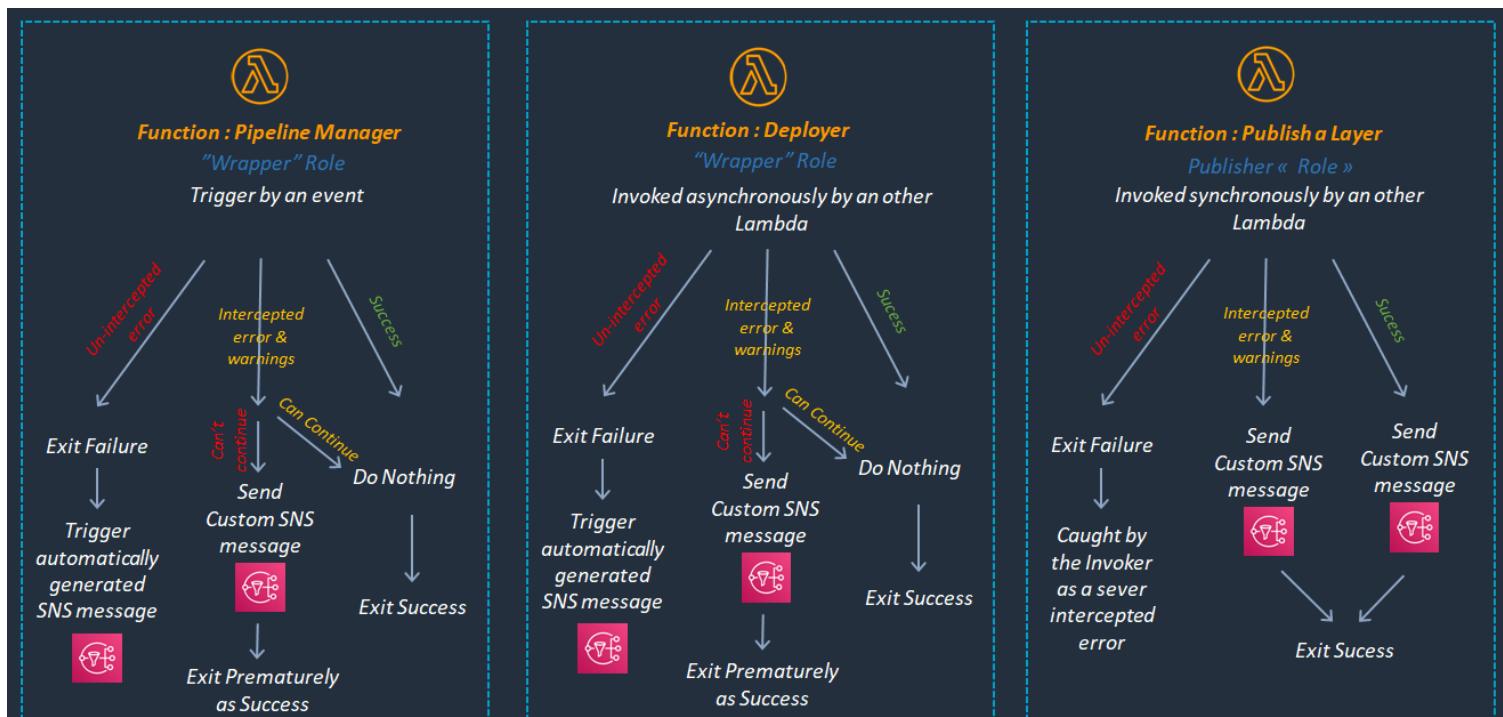
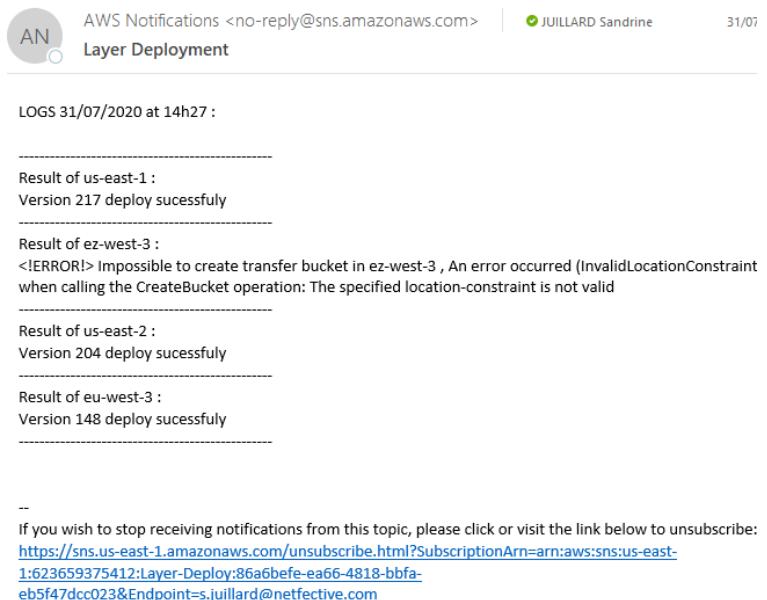


Figure 8 : Schéma de la gestion des Notifications SNS selon la fonction

Dans le cas où l'erreur n'est pas interceptée 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).

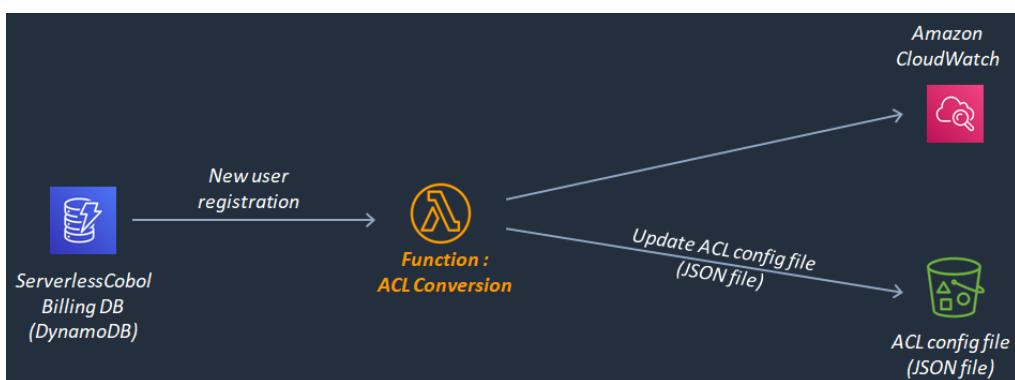


Figure 10 : Schéma fonctionnel de la mise à jour automatiquement le fichier ACL

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 → 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 (Annexe 3)

BASCAC est une petite application web qui permet de simplifier la gestion de la facturation de « serverless COBOL for AWS ». 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 le runtime, ou encore de gérer le pourcentage de réduction accorder à l'utilisateur. En conséquence, son accès doit être limité car elle gère des données de facturations.

L'objectif de cette tâche est de mettre en ligne BASCAC de manière sécurisée, en limitant son accès.

L'outil qui m'a été préconisé est AWS Cognito, un service d'Amazon, pour les applications web, qui est la solution de fédération d'identité, authentification et autorisation d'AWS. [10]

L'application a été réalisé avec Angular, TypeScript, html et css sur VSC. L'application est une "single-page application". Pour y ajouter une page d'authentification, il a d'abord fallut implémenter un système de routage pour créer une navigation entre pages.

Vient ensuite l'étape d'intégration de Cognito à la page d'authentification. Cette étape exige l'utilisation d'Amplify : Un service d'Amazon qui facilite la mise en service d'applications. L'utilisation d'Amplify a permis de repenser la gestion de l'authentification. En effet, il est possible grâce à Amplify de déployer BASCAC, après l'avoir compilé, de manière non-programmatique sur la console graphique d'Amazon. Puis le service propose des options pour la gestion de l'accès à l'application et des mots de passe.

Au final, la page d'authentification et l'utilisation de Cognito a été abandonnée. J'ai fait le choix d'opter pour un déploiement et une gestion de la sécurité non-programmatique avec Amplify, ce qui simplifie énormément le travail de gestion et sécurité de l'application.

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

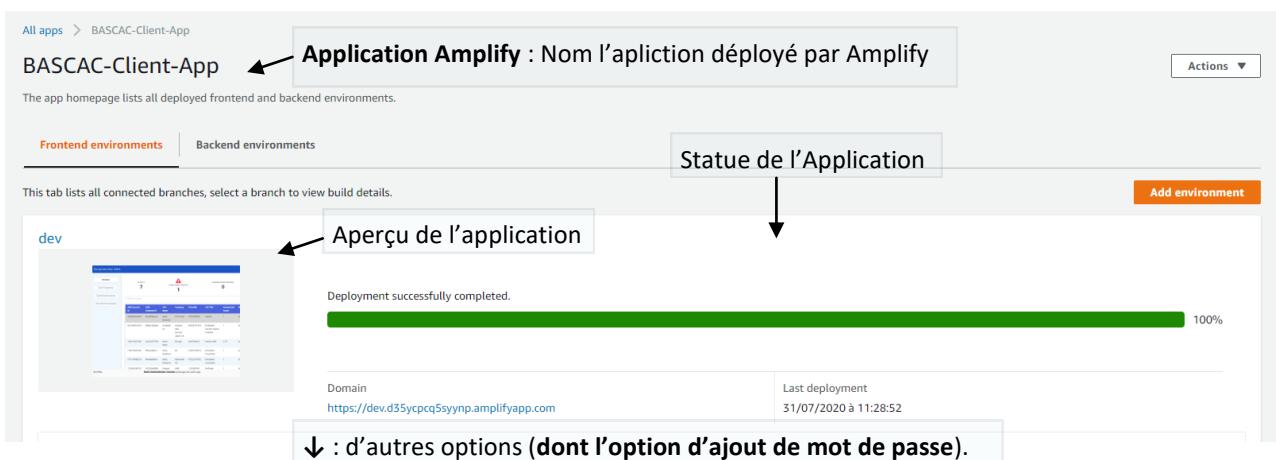


Figure 11 : Capture d'écran de la console Amazon Amplify

Exportation de la Vue Problème de Blu Age Analyzer.

Analyzer est un produit de Blu Age qui permet la Visualisation des dépendances entre les différentes parties du code COBOL. Il permet entre-autre de traduire du code legacy en DSL. Ensuite, l'API Vélocity pourras permettre de traduire le DSL en Java. 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 », onglet dans lesquels est affiché les erreurs et les warnings issue de la compilations du/ des projets en cours. (cf Annexe 4A)

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 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, n'est pas une fonctionnalité crée par Blu Age, c'est une vue dont le code est interne à Eclipse, 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 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 stockées. En récupérant ces informations on peux reconstituer la vue problème, puis l'exporter sous le format csv.

Après l'implémentation de cette solution, j'ai pris connaissance d'une option similaire, qui avait déjà été implémenté auparavant par un développeur de Blu Age, mais qui avais un usage différent (usage plus *programmatique que UX*). Cette option proposait d'exporter les problèmes relatifs à un projet en particulier : en faisant une clique droite sur celui-ci dans la vue navigation, l'option proposais d'exporter les problèmes dans plusieurs format possible, à la racine du projet. Il était donc judicieux d'adapter la solution précédente pour quelle s'intègre au travail déjà effectuer, en réutilisant et en adaptant les classes et les méthodes déjà créées. Au final, j'ai réutilisé la structure des classes déjà implémentées, et l'option d'export directement sur la vue problème remplace désormais l'option précédente. (cf Annexe 5).

Une fois la tâche terminer, j'ai pris connaissance des protocoles de Blu Age afin de manager le suivi des commit. 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 que j'utilisais pour écrire dans écrire dans un fichier. Le premier jet que j'ai rendu était trop lent.

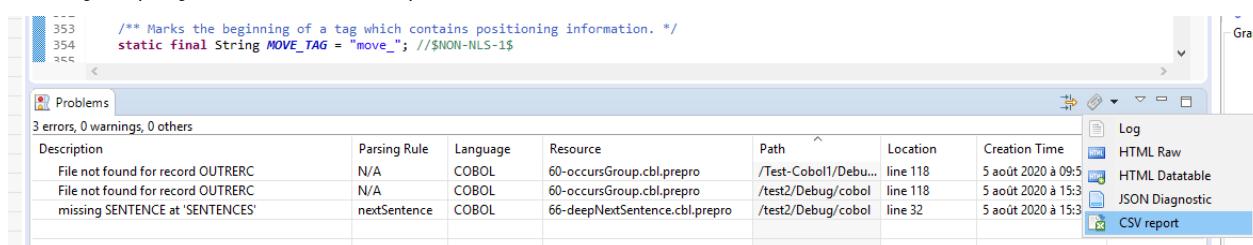


Figure 12 : Vue problème et son menu déroulant permettant l'export des problèmes

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 (cf II-2- Server-less COBOL for AWS). Plusieurs types d'opérations sont proposées par l'extension en plus de la compilation COBOL vers Java : une commande « frontend », par exemple, permet 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-importer la nature de la commande invoquée. Or, pour la compilation d'un fichier BMS, ne nécessite pas la présence d'un fichier Cobol.

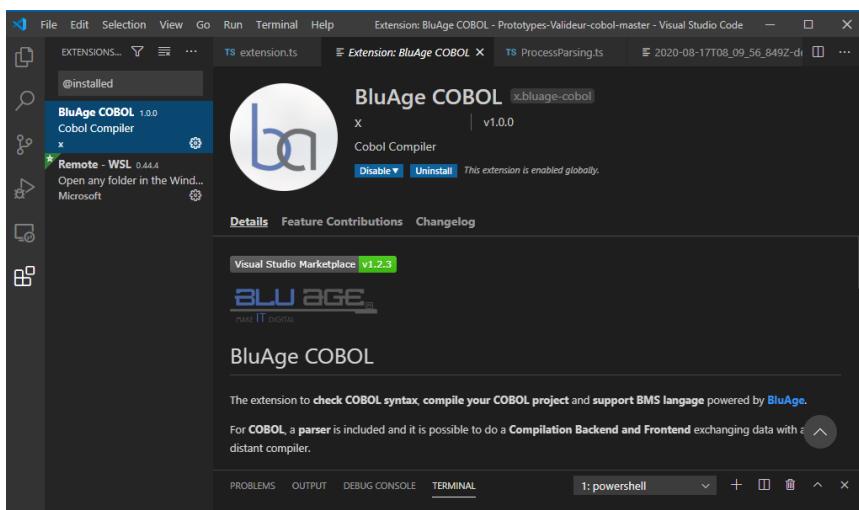


Figure 13 : Capture d'écran de l'extension BluAge COBOL dans l'IDE VSC

L'objectif de cette première tâche est de séparer les phases de vérification pour qu'elle soit adapté au type de compilation invoqué. L'opération une fois effectué, il a fallut générer à partir du code un fichier 'vsix'. Celui-ci correspond au format de fichier qui permet de publier une extension VSC.

 A screenshot of the Visual Studio Code terminal window. The title bar shows 'PROBLEMS 71', 'OUTPUT' (selected), 'DEBUG CONSOLE', and 'TERMINAL'. The terminal output shows several error messages from the 'BluAge Console' extension. It includes log entries like '*** Frontend compilation command called ***', '[17-08-2020 14:47:38] ERROR - File to parse is not BMS : extension-output-#4.', and '[17-08-2020 14:47:49] Cobol code validation.' followed by '*** Frontend compilation command ended ***'. Other entries mention 'MKCT00.cbl' and 'MKCTSET.bms' as non-BMS files. The terminal also shows '*** Compilation command ended ***' at the end.

Figure 14 : 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

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

Les applications CICS (Customer Information Control System) sont des systèmes qui permet 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 (fichier CSD) sont généralement présent. Lors de la compilation, le fichier textuel de description de ressource est parsé pour en faire une suite de requêtes SQL qui permet de former et alimenter une base de données lié à l'application. (cf Annexe 6)

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.

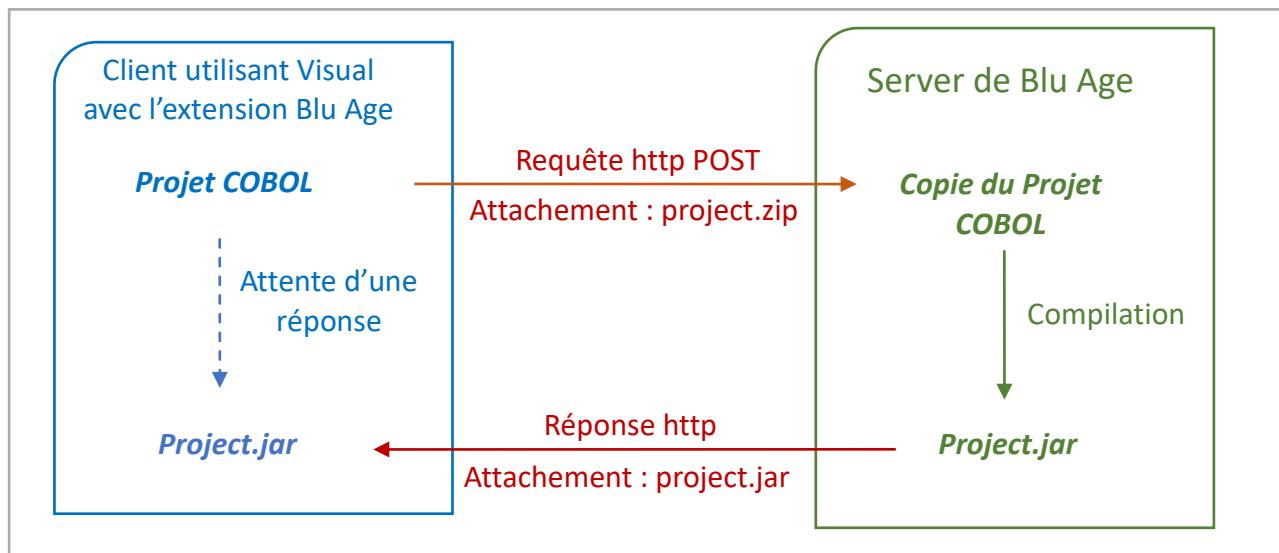


Figure 15 : Schéma fonctionnel de la compilation par l'extension Blu Age Cobol (architecture de type REST [12])

Le développement des mécanismes qui vont être présenter a été inspiré des commandes de compilations Blu Age déjà présente qui figure sur le schéma ci-dessus. L'objectif de cette tâche à été d'ajouter à une architecture déjà présente, une fonctionnalités supplémentaire.

Tous d'abord, coté 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 server 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, zippé, puis renvoyé à l'utilisateur. (Développement en java, avec Eclipse).

Ensuite, à nouveau côté client, la réception des réponses du server a été adapté pour que le fichier soit récupéré correctement.

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

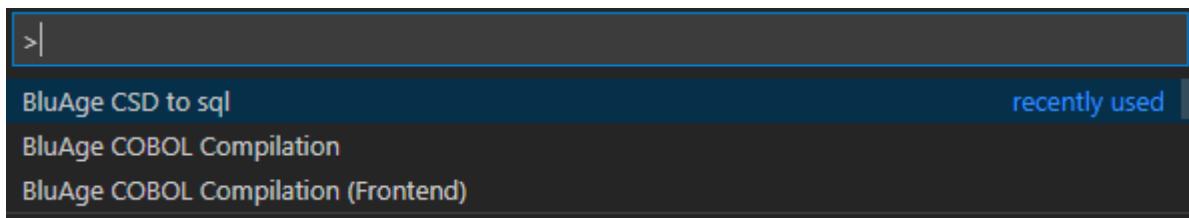


Figure 16a : Capture de la commande CSD to SQL dans VSC

```
*** 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 ***
```

Figure 16b : Résultat de la commande CSD to SQL dans les logs dans la consol sur VSC

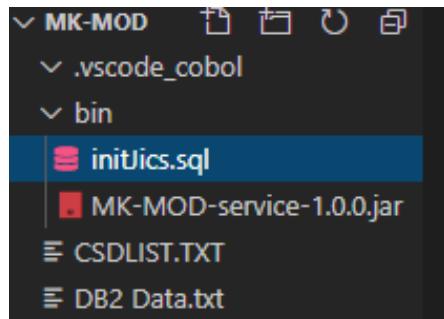


Figure 16c : Fichier SQL généré par la commande CSD to SQL

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 de test), ainsi que la documentation de la chaîne.

Le dernier aspect qui pourrait éventuellement manquer est la réalisation de test unitaire pour vérifier le bon fonctionnement du pipeline. La fiabilité du pipeline n'a à ce jour qu'était testé par des tests manuellement fait au fur et à mesure de son développement. Le pipeline n'a pas été paramétré pour déployer les vrais fichiers de du Runtime, mais tout à été mise en place pour que ce travail soit le ^mmus simple possible. Une documentation détaillée, en anglais é été créee dans cette optique. De plus, j'ai porté une attention toute particulière à la clarté de mon code.

Des 4 tâches supplémentaires que j'ai effectuer, j'ai pu toute les terminer également. J'ai pu push mon travail sur le git de Blu Age. Parfois, mes contribution on crée des problèmes au moment du packaging en raison de nouvelles dépendances que j'ajouter. Mais ce problème à pu être résolu. Mon code à également pu être revu par les personnes qui m'ont encadrées.

D'un point de vu personnelle, ce stage m'a permis de prendre en compétence, et ce dans diverse domaine : 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 aux 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 pourraient 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 écrite pour qu'elle puisse être reprise plus tard. Enfin, j'ai eu l'immense satisfaction d'avoir finalisé les travaux que l'on m'as confié.

Références

- [0] Documentation - Chaîne de Déploiement pour serverless COBOL solution, Sandrine Julliard, 2020
- [1] « A short story of serverless COBOL for AWS », BluAge, 2019
Lien : <https://github.com/BluAge/ServerlessCOBOLforAWS>
- [2] « 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>
- [3] « Atout et limites du serverless computing », ITForBuisness, Laurent Delattre , 2018
Lien : <https://www.itforbusiness.fr/atouts-et-limites-du-serverless-computing-18219>
- [4] « 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>
- [5] « Démo - Modernisation d'application avec Blu Age », BluAge, 2017
Lien : <https://www.youtube.com/channel/UCREdw7o0hKmqWFt1BjUmTuQ>
- [6] Serverless COBOL - Quickstart,, BluAge
Lien : <https://www.bluage.com/products/serverless-cobol-quickstart>
- [7] « AWS Lambda Language Comparison : Pros and Cons », Yan Cui, Octobre 2018
Lien : <https://epsagon.com/development/aws-lambda-programming-language-comparison/>
- [8] « Managing AWS Lamnda fucntion Concurrency», Chris Munns , Decembre 2017
Lien : <https://aws.amazon.com/fr/managing-aws-lambda-function-concurrency/#:>
- [9] Invocation Asyncrone 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>
- [10] Amazon Cognito, AWS Inc, 2020
Lien : <https://aws.amazon.com/fr/cognito/>
- [11] Basic Mapping Support, IBM Knowledge Center, aout 2020
Lien : https://www.ibm.com/support/knowledgecenter/SSGMCP_5.6.0/dfhp370.html
- [12] L'architecture REST expliquée en 5 règles, Nicolas Hachet, Juin 2012
Lien : <https://blog.nicolashachet.com/larchitecture-rest-expliquee-en-5-regles/>
- [13] IBM Rational COBOL Runtime provides the run-time libraries for programs that execute on z/OS
Lien : <https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an=ENUSA06-0557>

Annexe 1 : Etape de configuration d'une Lambda

Serverless Cobol for AWS - Quickstart



Blu Age Serverless

Thank you for your interest in Blu Age Serverless Cobol! Please follow the following steps for your trial.
If you like to dive into the full documentation, please click here.

Step 3 of 4

1 - Setup environment 2 - Compile source code 3 - Setup lambda function 4 - Deploy / run

A. Create a new lambda function
 This step is fully made on your AWS account.

1. Connect to AWS;
2. Go to the AWS Lambda service;
3. Click on the "Create function" button;
4. Give a name to your function;
5. Select "Java 8" as the Runtime to be used;
6. Pick "Create a new role with basic Lambda permissions" for the Execution role;
7. Click on the "Create function" button.

B. Setup lambda function

1. Go to the function editing page;
2. Click on the Layer zone to make the Layer management zone visible;
3. Press the button "Add a layer";



4. Select the option "Provide a layer version ARN";
 5. In the Layer version ARN textbox, paste the layer version ARN you have been granted use permission;
 6. Click on the "Add" button;
 7. Click on the function name, to make the standard Configuration visible again;
 8. Browse to the function code section and upload the jar file generated previously;
 9. In the Handler textbox, replace the existing default value by com.neteffecte.blu.age.GWLambdaRequestHandler::handleRequest;
 10. Set the environment variables as shown in the image below. The BA_RUN_UNIT_ENTRYPOINT environment variable permits to specify the run unit program entry point;

Annexe 1 : Extrait du tutorial Blu Age Serverless. [6]

Annexe 2 : Récapitulatifs des fonctions de la chaîne de déploiement.



Function : Pipeline Manager

"Wrapper" Role

Trigger by an event

- ① Check files in Bucket source
- ② Invoke Publish a Layer and wait for a response
- ③ If success, invoke a Deployer per region



Function : Deployer

"Wrapper" Role

Invoked asynchronously by an other Lambda

- ① Create A temporary Bucket
- ② Invoke Publish a Layer and wait for a response
- ③ Delete the temporary Bucket



Function : Publish a Layer

Publisher Role

Invoked synchronously by an other Lambda

- ① Create a new version of the Layer
- ② Test it with a Lambda
- ③ Add permissions using ACL



Function : Notification-Manager

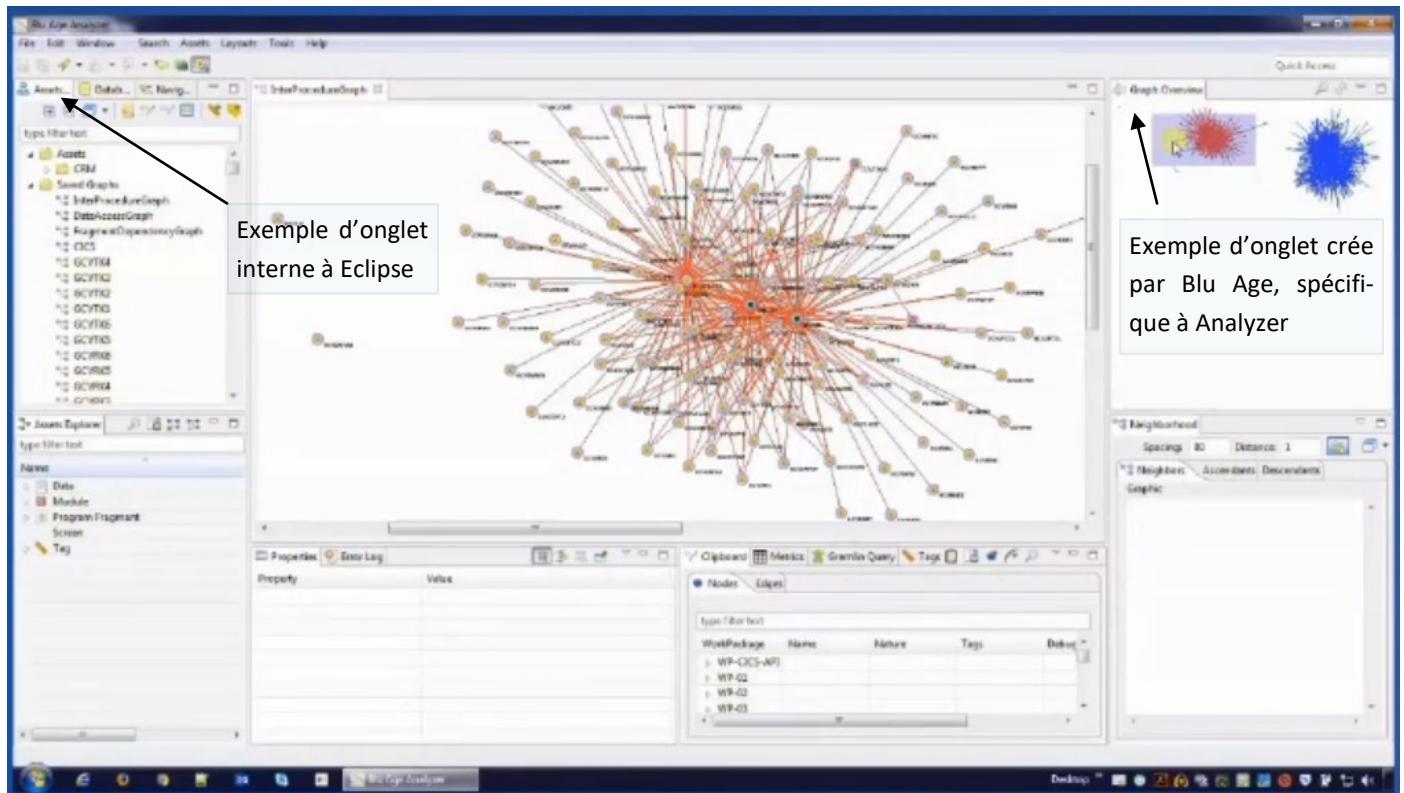
"Tool" Role

Invoked synchronously by an other Lambda

- ① Add information to Custom Log file in source bucket and eventually send a SNS message

Annexe 3 : Capture d'écran de BASCAC

Annexe 4 : Analyzer, Export CSV



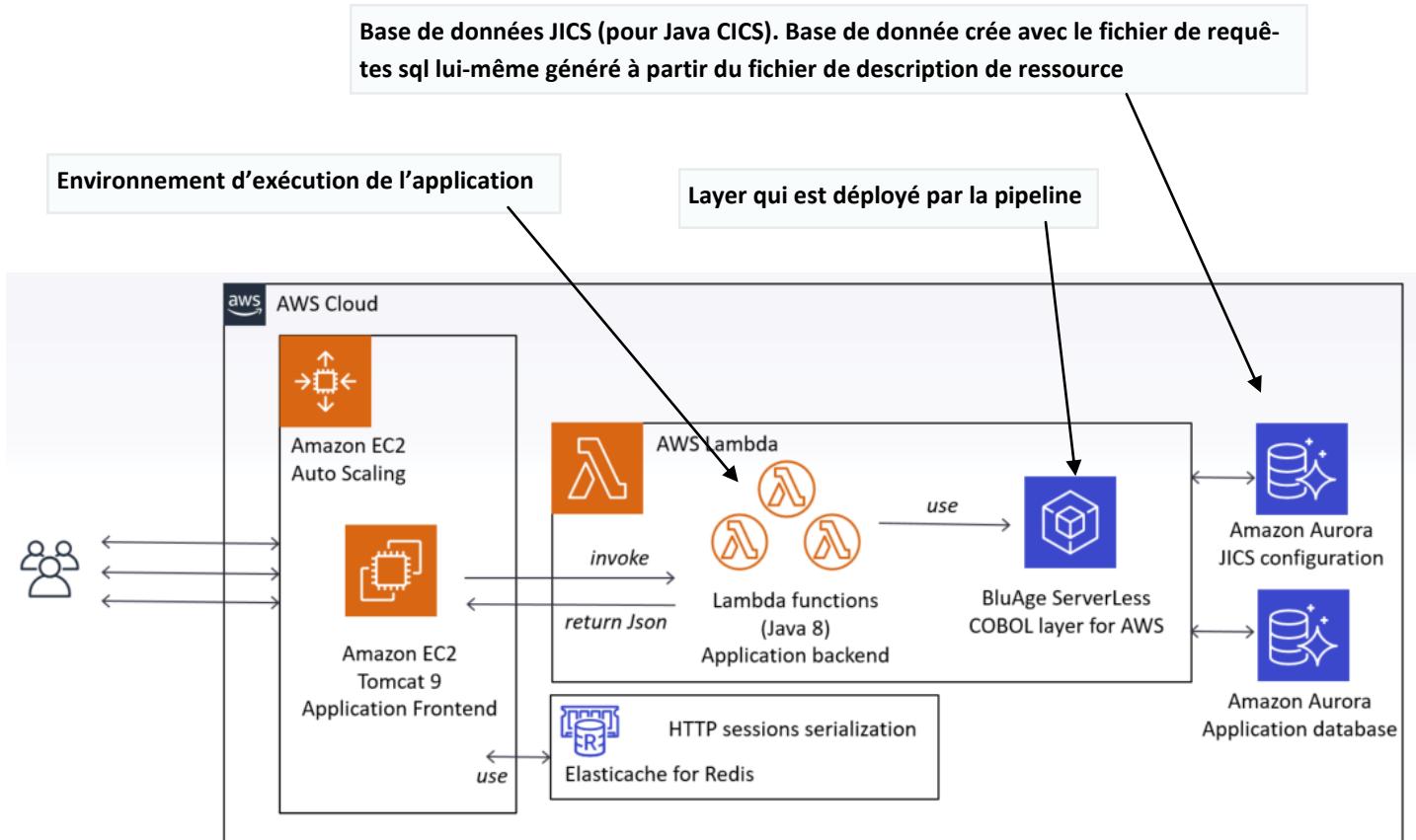
Annexe 4A : Capture d'écran de Analyzer
La figure montre le résultat de l'analyse des dépendances d'une application COBOL

The figure shows a screenshot of an Excel spreadsheet titled "Classeur1 - Excel". The ribbon menu is visible at the top. In the "Obtenir des données" section of the ribbon, the "À partir d'un fichier texte/CSV" option is highlighted with a red arrow. The main area of the spreadsheet displays a table with four columns: Description, Parsing Rule, Language, and Resource. The data in the table is as follows:

| | Description | Parsing Rule | Language | Resource |
|---|-----------------------------------|--------------|----------|-----------------------------|
| 2 | File not found for record OUTRERC | | COBOL | cobol/60-occursGroup.cbl |
| 3 | File not found for record OUTRERC | | COBOL | cobol/60-occursGroup.cbl |
| 4 | File not found for record OUTRERC | | COBOL | cobol/60-occursGroup.cbl |
| 5 | missing SENTENCE at 'SENTENCES' | nextSentence | COBOL | cobol/66-deepNextSentence.c |
| 6 | missing SENTENCE at 'SENTENCES' | nextSentence | COBOL | cobol/66-deepNextSentence.c |
| 7 | | | | |
| 8 | | | | |

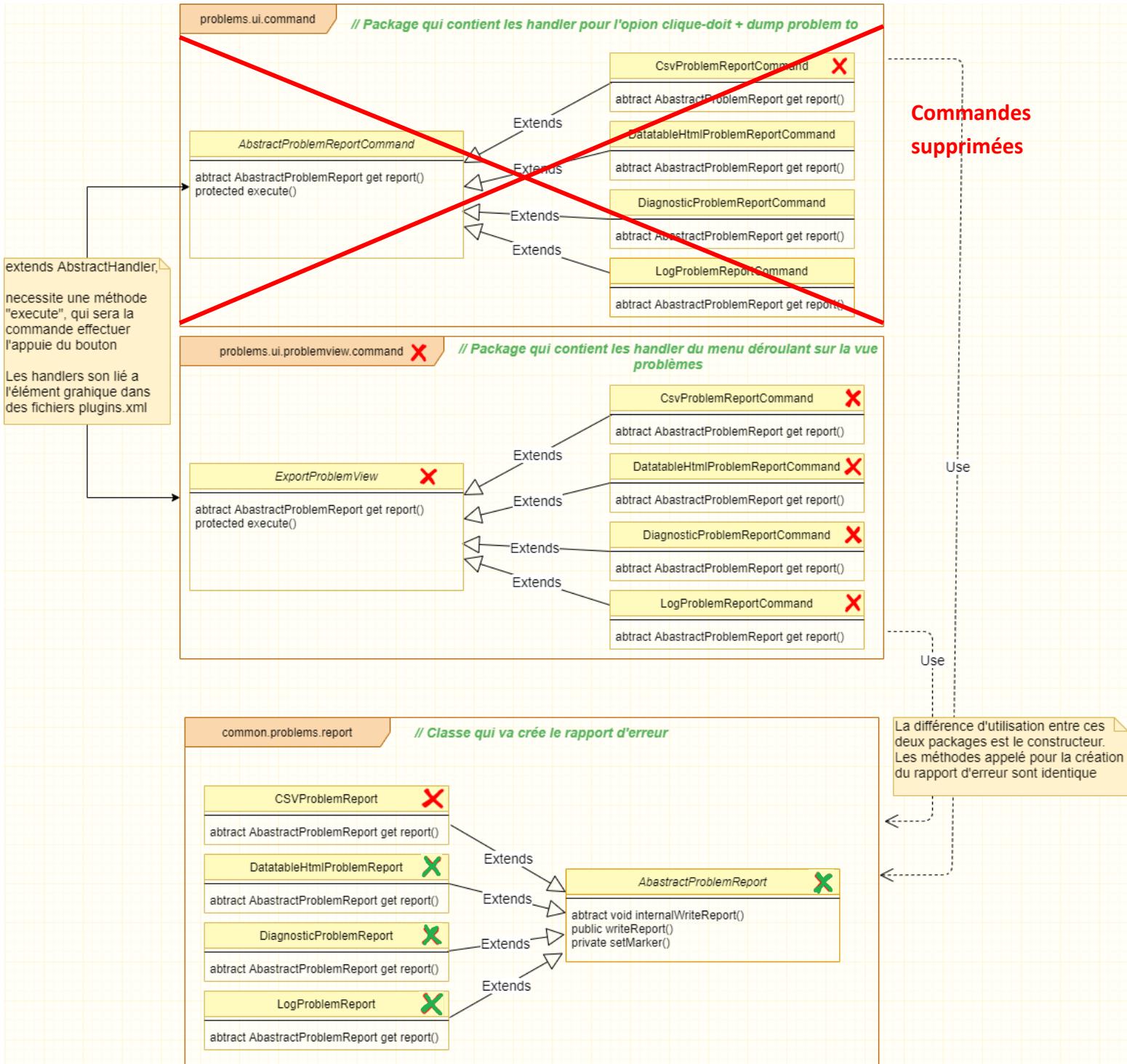
Annexe 4B : Résultat de l'export CSV, lorsqu'il est ouvert avec Excel

Annexe 6 : Schéma illustrant l'utilisation de la base SQL d'une application CICS COBOL traduite en Java.



Annexe 5 : Schéma tiré de la présentation de Serverless COBOL for AWS, the Accenture application Architecture.

Annexe 5 : Diagramme UML des classes utilisées pour l'export de la vue problème



Légende :

Les éléments marqués d'une croix rouge correspondent aux classes/Packages créés.
Les éléments marqués d'une croix verte correspondent à ceux qui ont été modifiés.