

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.6
II. Contexte Technique	p.7
1) Serverless COBOL for AWS solution	p.7
2) L'environnement Amazon Web Services	p.7
III. Réalisation de chaîne de déploiement	p.9
1) Etape de recherche	p.9
2) Solution retenue	p.10
1. Aperçu global	p.10
2. Détails de la solution	p.12
IV. Travaux complémentaires	p.15
1) Blu Age Serverless Cobol Administration Cobol (BASCAC)	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	Amazon Web Services	DSL	Domain-specific language
Cobol	Common Business Oriented Language	BASCAC	Blu Age Serverless Cobol Administration Cobol
API	Application Programming Interface		
DevOps	Software <u>Development</u> (<i>Dev</i>) IT <u>operations</u> (<i>Ops</i>)		
VM	Virtual Machine		
IDE	Integrated development environment		
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](#)", Composé d'une extension VSC servant de compilateur COBOL vers Java faisant appel à l'API « Velocity », le résultat de cette compilation pourra être utilisé pour déployer de manière « serverless » une application legacy. [1] [2] En ciblant plus particulièrement les micro conteneurs éphémères en surchargeant environnement d'exécution des Lambdas avec le Runtime Blu Age Velocity via la technologie AWS Layers.

Pourquoi la nécessité de compiler en Java un code écrit en COBOL ? Car le COBOL est un langage très présent dans les entreprises qui ont construits des logiciels métiers critiques avant l'apparition des langages de 4 ième génération. De ce fait ces logiciels ont accumulés de la dette technique, furent conçus sans design pattern et reposent sur des architectures technologiques en obsolescence et qui s'opposent au standards actuels et futurs (microservices, consistance éventuelles, bases de données propriétaires etc) de plus le cout de possession de ces applications souffrent de la comparaison avec des applications écrites dans des langages comme Java et .Net surtout quand elles sont déployées sur des environnements auto scalable et facturé à la demande comme le proposent les architecture Cloud. C'est ce que désigne le terme de « legacy » (héritage en anglais). Les logiciels Blu Age permettent d'automatiser la conversion de ces applications complexes et critiques vers une nouvelle cible d'architecture en mettant en œuvre des refactoring complexes afin que la nouvelle architecture soit utilisée à son plein potentiel technologique, financier, opérationnel et en terme de performance et de scalabilité afin d'accompagner les entreprises dans leur transitions digitale. La plupart de leurs solutions sont des outils pour assister les développeurs dans la traduction de leurs codes legacy vers des langages plus moderne comme Java ou .Net. Néanmoins, il est parfois difficile pour des développeurs COBOL de se reconvertis vers le Java ou l'entreprise peut choisir d'accompagner plusieurs centaines voir milliers de programmeurs Cobol vers une retraite très proche (la moyenne d'âge de ces programmeurs dépassant 60 ans) ce qui permet de plus une conduite du changement faite dans la douceur et sans provoquer de crise sociale

Introduction

Le compilateur proposé par Blu Age, permet d'éviter les licenciements en permettant aux développeurs de continuer à écrire en COBOL, tout en générant des applications en Java. C'est le 1er produit Blu Age à traduire du code, sans nécessité aucune intervention humaine.

A l'heure actuelle, [le Runtime servant au déploiement serverless des applications COBOL 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. De plus il faut pouvoir mettre ce Runtime à disposition dans l'ensemble des datacenter AWS à travers le monde, gérer le versionning de cet environnement distribué et gérer les droits d'accès de tous les clients selon les régions et versions déployées.

Il semble alors intéressant de réfléchir à l'automatisation de cette tâche.

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, les fichiers de la couche sont archiver 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. Ce faisant il devra s'assurer de tester, et d'accorder les droits d'accès au Runtime, et ce dans toute les régions voulues. En résumé, le travail à effectuer est de concevoir, réaliser et finaliser un pipeline qui effectuera de manière 100% automatique les tâches présentées plus tôt.

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 [3]. Cette suite fournira en plus des outils de générations automatique de code des outils de visualisation du code et des outils d'analyse. [4]

II - Contexte technique

1) Serverless COBOL for AWS solution

La solution serverless [5] de COBOL for aws solution a pour but de procurer au client une solution pour déployer, sur une Lambda Amazon, une application java initialement codées en COBOL.

Il est constituée d'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. L'extension fait elle-même appel à un serveur de Blu Age déployé sur AWS. Entre autre, il fera appelle à l'API Velocity pour convertir le DSL en Java. Le fichier généré à l'issue de la compilation contient le code minimum de l'application : Ces dépendances sont fournies par le « Runtime » [3]. Ainsi, pour déployer son application, le client devra créer une Lambda, à laquelle il devra ajouter le Runtime en tant que Layer, puis il pourra y exécuter le fichier jar généré à partir de son code COBOL par l'extension VSC.

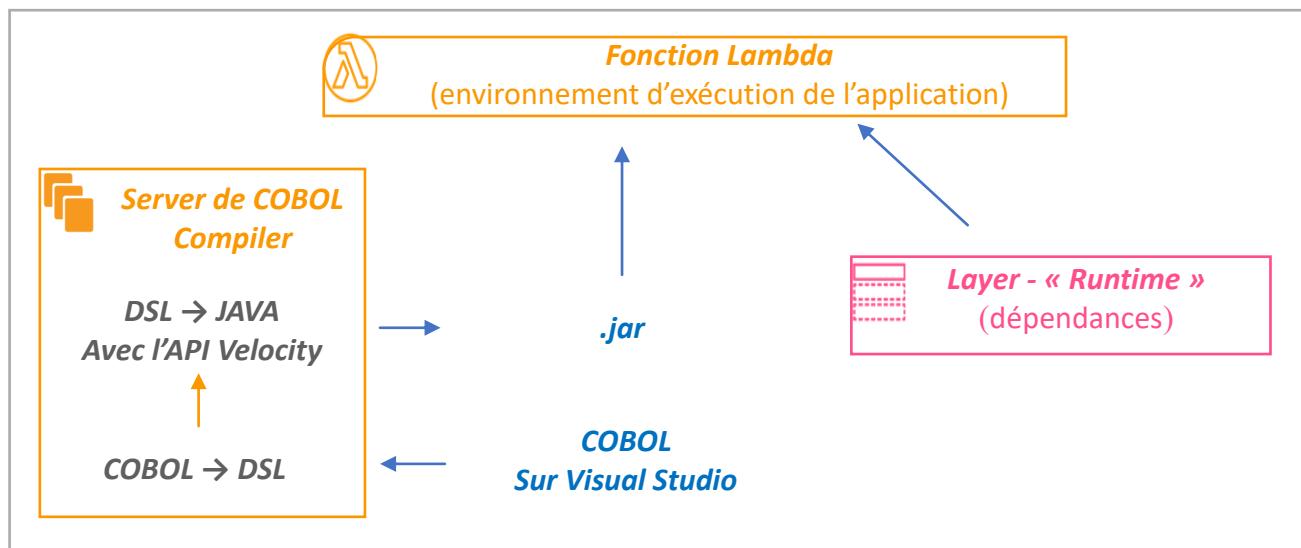


Figure 3 : Schéma fonctionnel de la solution serverless COBOL for aws solution

2) 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 :

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 l'échelles 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 facturer 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 ressource 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. (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. 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 fournit 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.

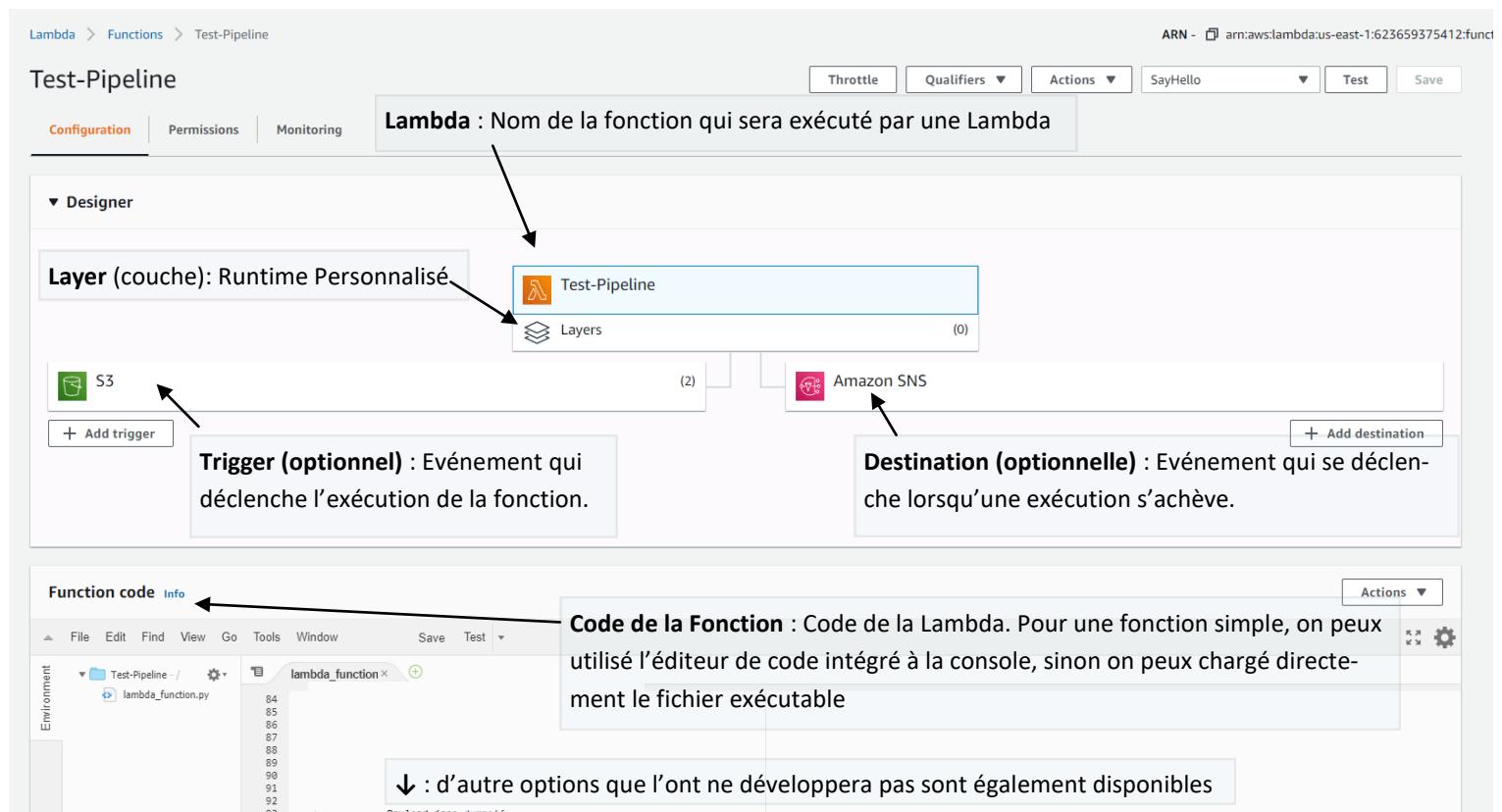


Figure 4 : Capture d'écran de la console AWS pour le paramétrage d'une Lambda

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'Amazon, les buckets s3.

Elle devra également :

- S'assurer que l'accès à ces espaces 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 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 devra également facilement paramétrables et sécurisé.

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 a 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 workflow. 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 permettent d'héberger toutes les étapes : 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 vocation à simplifier la gestion de workflow complexes notamment pour des architectures distribuées ou faisant usage de consistance éventuelle, il nous a donc paru trop complexe pour nos besoins. On évite ainsi un cout 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.

2) Prototype retenue

Pourquoi choisir cette solution ?

Le choix de cette solution est motivé d'un part, par sa flexibilité, mais également pour son cout très faible comparé au autre solution proposé. Enfin, l'atout de cette méthode comparativement l'utilisation de Jenkins par exemple est que l'on utilise quand même des outils Amazon, et les opération sont effectuer 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. Cette solution propose le meilleurs compromis entre sécurité, flexibilité, simplicité de maintenance, intégration au sein d'AWS et cout.

Aperçu global du prototype (figure 5)

Le déploiement démarre lorsque la chaîne d'intégration se termine. A l'issue de ce traitement, la nouvelle version du Framework, sous forme de jar va être chargé dans le Bucket Source. Ce fichier, c'est celui indiquer 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 aux Framework. Toute ces opérations sont effectuées dans la région par default (canoniquement, en Virginie du Nord us-east-1). Une fois le déploiement de la Layer effectuer dans la région par défaut, et si l'opération c'est déroulé correctement, alors le déploiement sur toutes les régions 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 nommé "Déployer". Chacune avec comme argument une région différente. Elles ont pour rôle de réaliser d'effectuer le déploiement de la layer dans la région qui leurs a été attribué. Ces fonctions vont créer une copie du contenu du Bucket Source dans un bucket temporaire, pour pouvoir ensuite reproduire les étapes de déploiement (publié, tester, ajouter les permissions) dans leur région.

Les étapes de déploiement, étant commune à la fois de la fonction Main Pipeline et Déployer seront effectués par une autre Lambda afin de factoriser ce code. Cette fonction sera lancé de manière synchrone par leurs mères.

En résumé, le flow d'exécution du pipeline est codés par 3 fonctions Lambdas codé 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.

Additionnellement, une fonction Lambda codée en Java devra se charger de tester la Layer (fonction que je n'ai pas codée, elle est matérialisée dans le prototype par une fonction bouchon, qui ne fait rien). Et enfin, une dernière fonction qui 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 et en parallèle.

N.B : L'annexe 1 fournit un récapitulatif des fonctions créées et de leurs rôles.

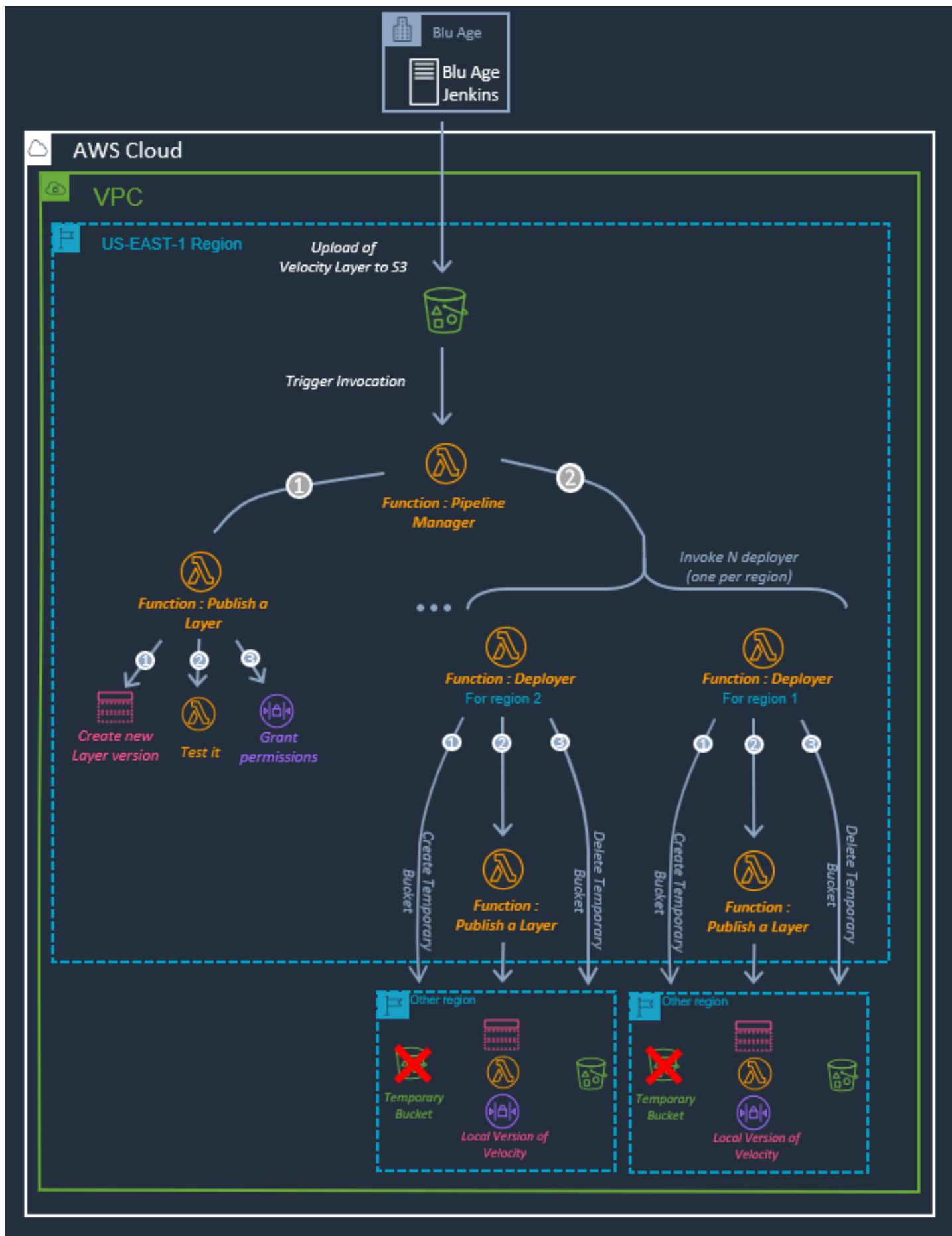


Figure 5 : Schéma de l'architecture de la chaîne de déploiement

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.

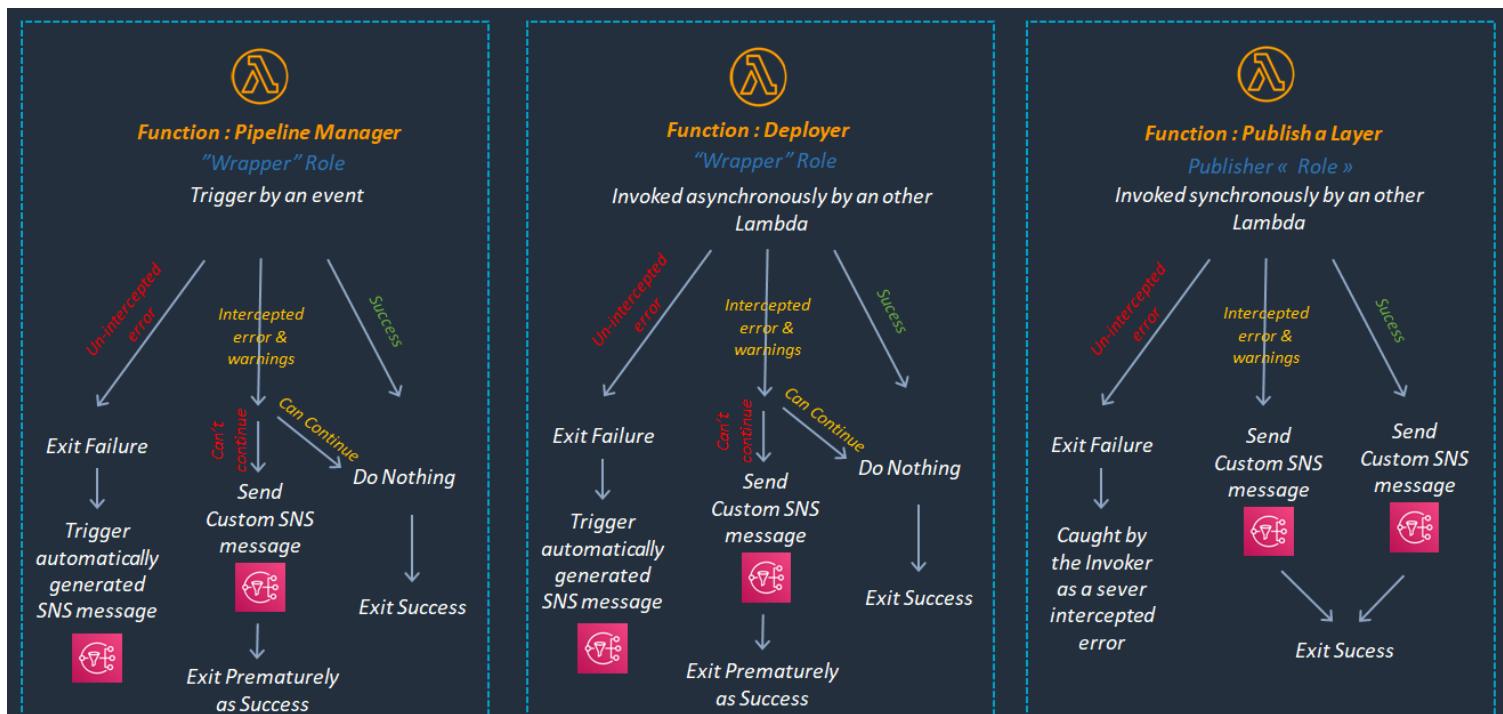
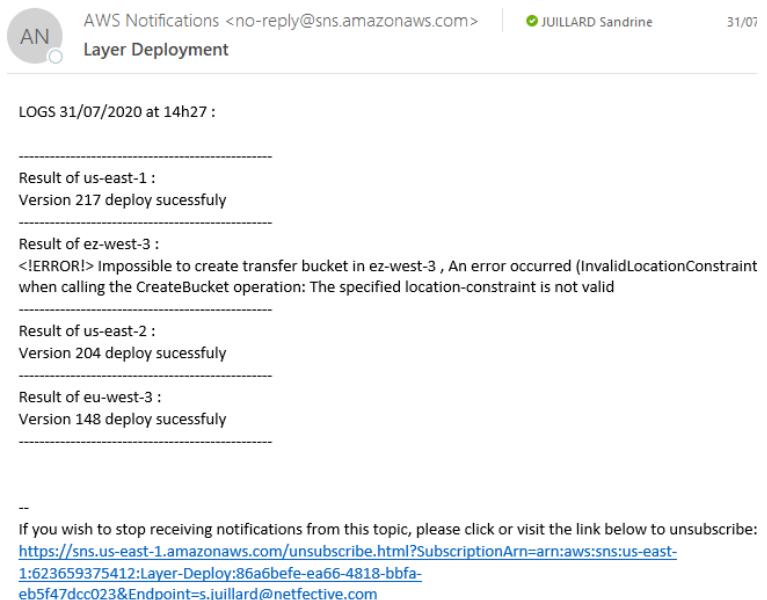


Figure 8 : Schéma de la gestion des Notifications SNS selon la fonction

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).

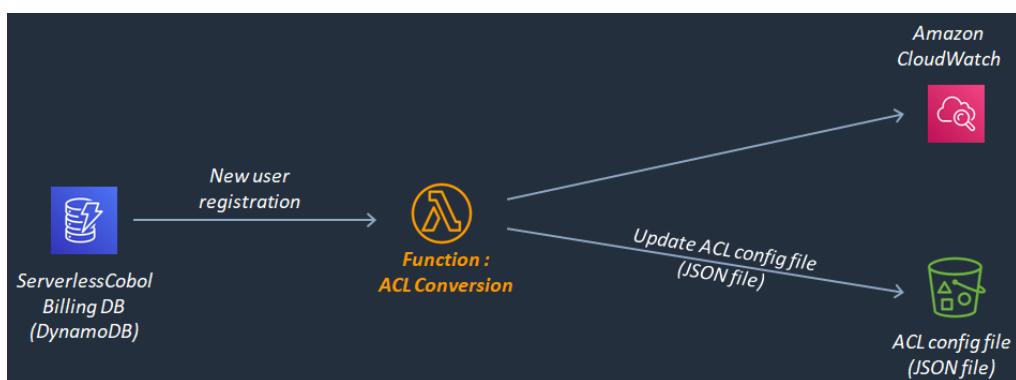


Figure 10 : Schéma fonctionnel de la mise à jour automatiquement le fichier ACL

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 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 « serverless COBOL for aws » 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 le runtime, ou encore de gérer le pourcentage de réduction accorder au client. 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. J'ai utilisé l'IDE Visual Studio Code pour y ajouter mes modifications.

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

L'application a été réalisé avec Angular, TypeScript, html et css. L'application est une "single-page application". J'ai dû rajouté un mécanisme de routage afin d'y ajouter une page d'authentification. Une fois cela terminé, j'ai essayé d'intégrer cogito à ma page d'authentification. C'est lors de cette étape que j'ai décidé d'intégrer 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 car 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. Cela a été réalisable grâce à la console graphique d'Amazon pour le déploiement d'application avec Amplify.

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.

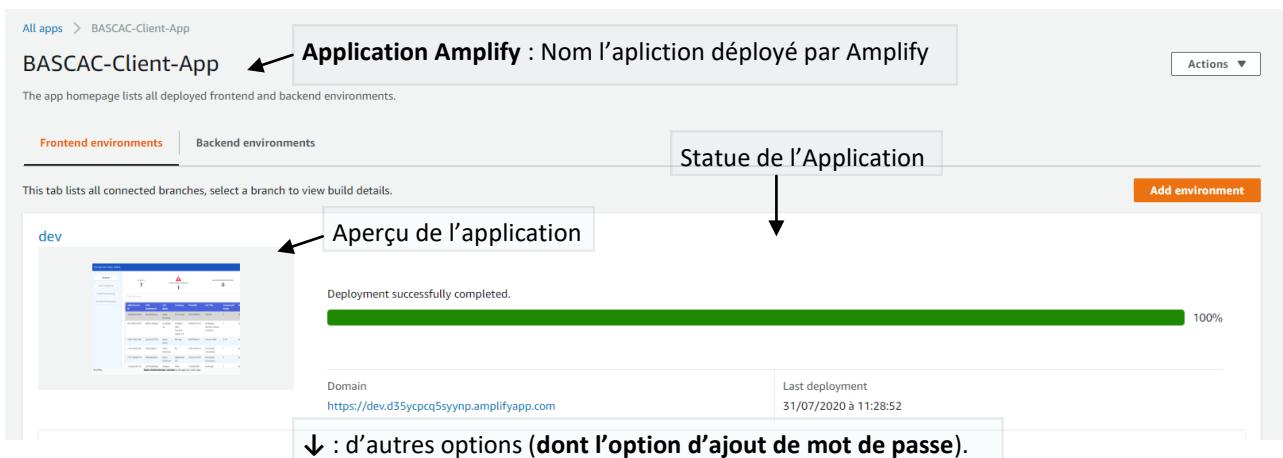


Figure 11 : Capture d'écran de la console Amazon Amplify

Exportation de la Vue Problème de Blu Age Analyzer.

Analyzer est un produit de Blu Age de la V7 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 », l'onglet dans lesquels est affiché les erreurs et les warnings issue de la 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 onglet, 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 ouverts), dans plusieurs formats différents, dont le csv, et de l'enregistre 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. 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.

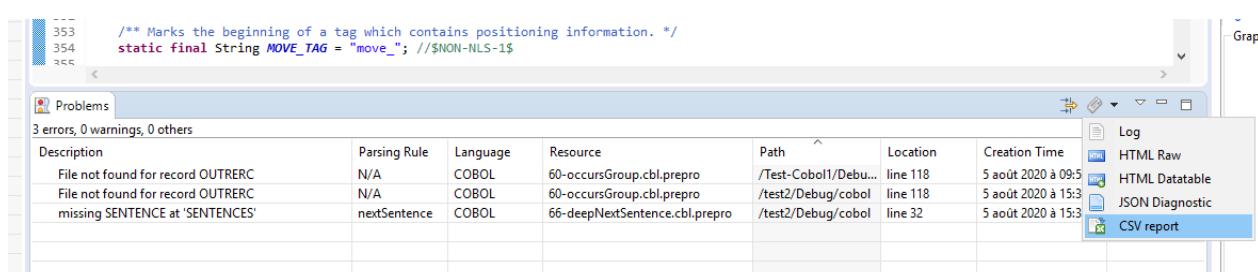


Figure 12 : Vue problème et son menu déroulant permettant l'export des problèmes

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 (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.

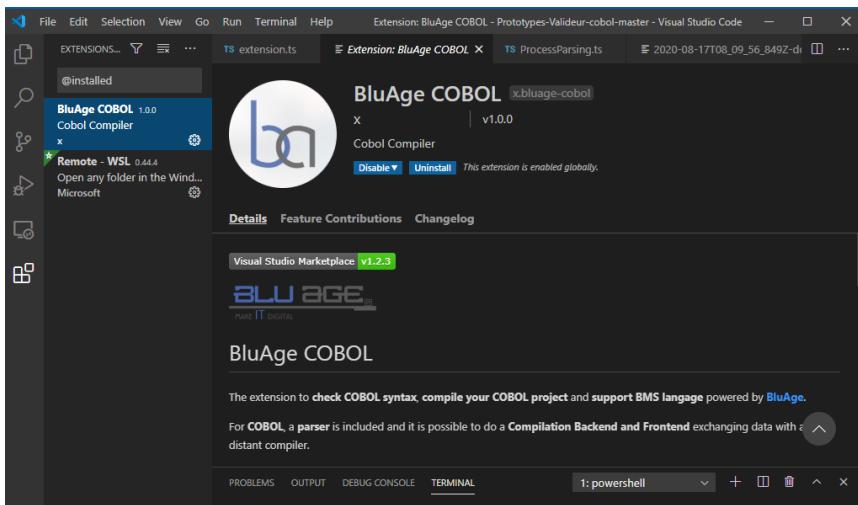


Figure 13 : Capture d'écran de l'extension BluAge COBOL dans l'IDE VSC

L'objectif de cette première tâche est de séparer les phases de vérification pour qu'elle soit 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 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

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 fichier textuel de description de ressource est parse pour en faire une suite de requêtes SQL qui permet de former et alimenter une base de données liée à l'application. L'annexe 5 montre comment cette ressource est utilisée.

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.

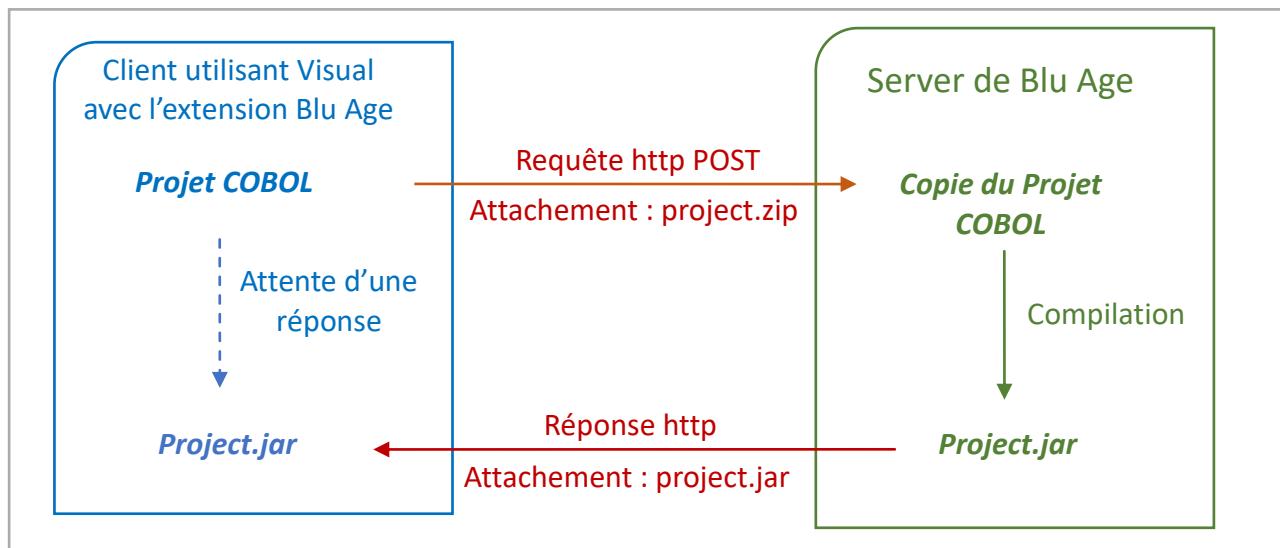


Figure 15 : Schéma fonctionnel de la compilation par l'extension Blu Age Cobol (architecture REST [11])

Le développement des mécanismes présentés a été inspiré des commandes de compilations Blu Age déjà présente. La figure ci-dessus montre comment fonctionne l'architecture du compilateur. Mon travail a été d'ajouter à cette 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, zippé, 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.

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

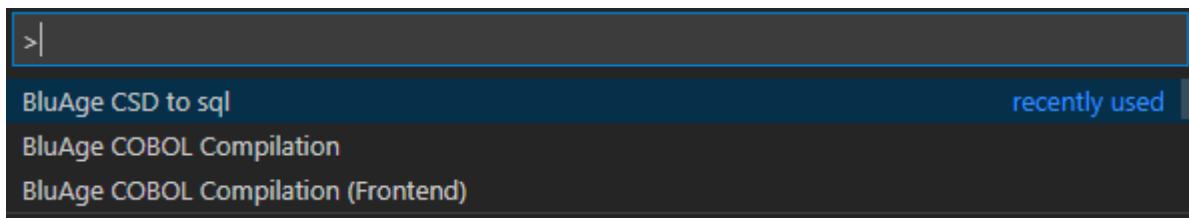


Figure 16a : Capture de la commande CSD to SQL dans VSC

```
*** 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

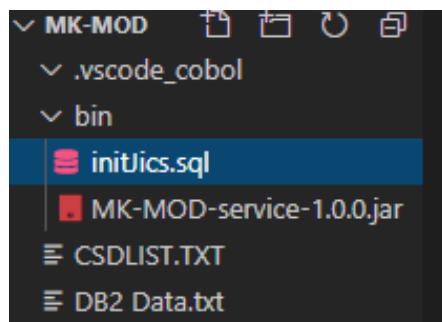


Figure 16c : Fichier SQL généré par la commande CSD to SQL

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.

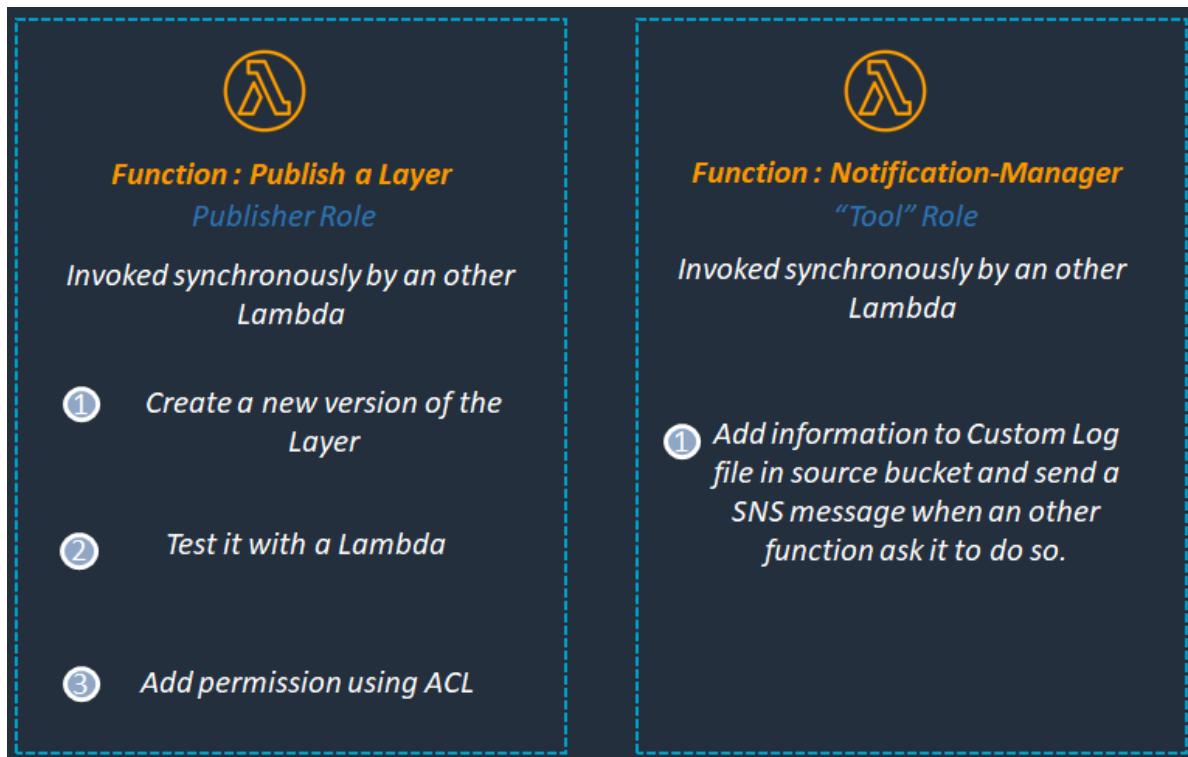
D'un point de vu personnelle, ce stage m'a permis de prendre en compétence, et ce dans diverse domaine : devOps, devWeb et dev logiciel. J'ai codé dans plusieurs langages différents, utilisé et tester divers outils. Cette expérience m'as permis de développer ma capacité d'adaptation et ma flexibilité. 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

- [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] 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>
- [4] « Atout et limites du serverless computing », ITForBuisness, Laurent Delattre , 2018
Lien : <https://www.itforbusiness.fr/atouts-et-limites-du-serverless-computing-18219>
- [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] « A Streamlined Journey from Legacy to Microservices with Blu Age », BluAge, Avril 2019
Lien : <https://www.youtube.com/watch?v=jhB39NIgG14&feature=youtu.be&t=2806>
- [8] « AWS Lambda Language Comparison : Pros and Cons », Yan Cui, Octobre 2018
Lien : <https://epsagon.com/development/aws-lambda-programming-language-comparison/>
- [9] 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>
- [10] « Managing AWS Lamnda fucntion Concurrency», Chris Munns , Decembre 2017
Lien : <https://aws.amazon.com/fr/managing-aws-lambda-function-concurrency/#>
- [11] L'architecture REST expliquée en 5 règles, Nicolas Hachet, Juin 2012
Lien : <https://blog.nicolashachet.com/larchitecture-rest-expliquee-en-5-regles/>
- [12] Basic Mapping Support, IBM Knowledge Center, aout 2020
Lien : https://www.ibm.com/support/knowledgecenter/SSGMCP_5.6.0/dfhp370.html

Annexe 1 : Récapitulatifs des fonctions du pipeline



Annexe 2 : Capture d'écran de BASCAC

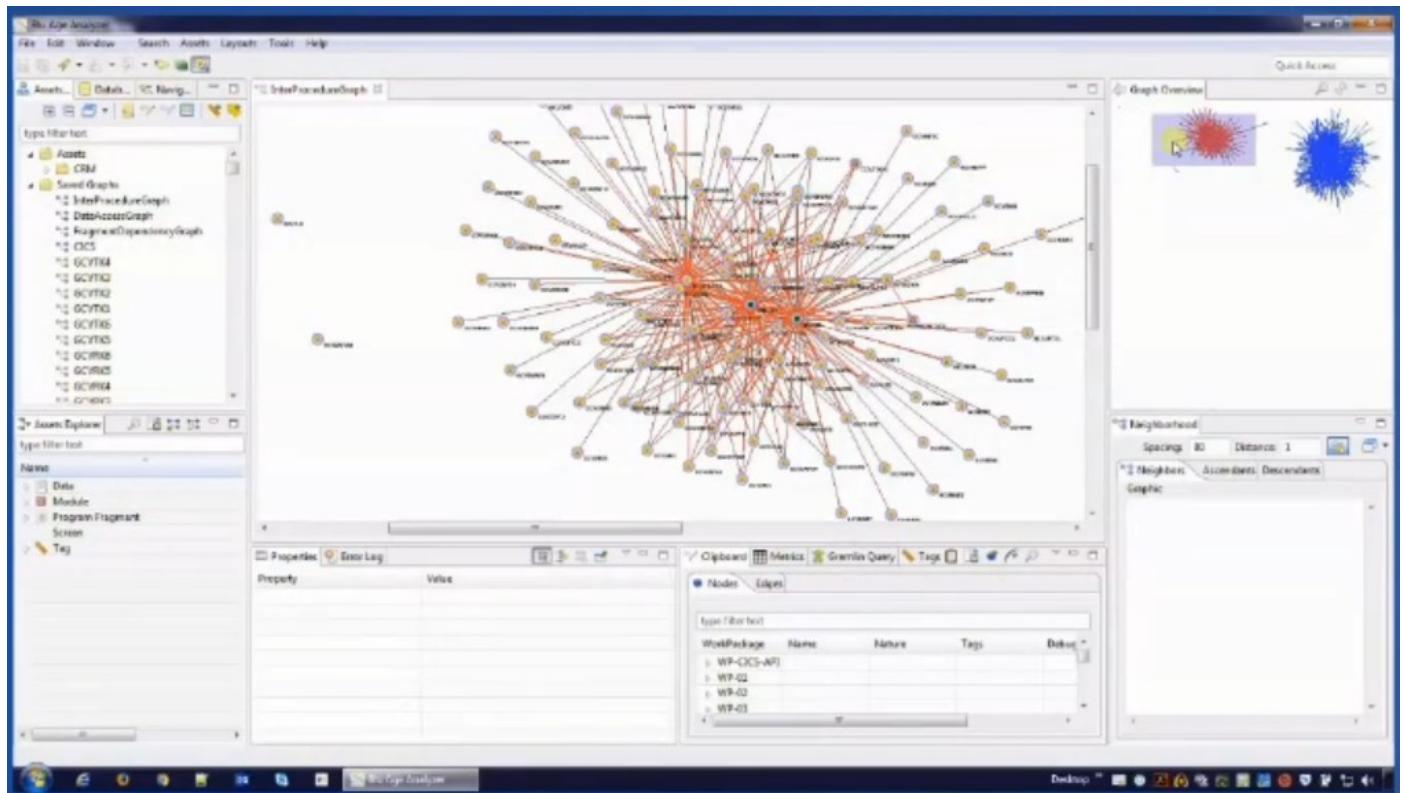
The screenshot shows the BASCAC Administration Control interface. At the top, there are navigation buttons for back, forward, refresh, and file operations (New, Open, Save, Print, etc.). The title bar displays the path: C:\Users\sjulliard\Desktop\BASCAC-client\dist\AngTuto\index.html#/CLIENTS.

The main area is titled "CLIENTS" and shows a table of client information. The columns are: AWS Account ID, AWS Customer ID, Full Name, Company, PhoneNb, Job Title, Commercial Factor, and Status. There are 7 clients listed:

AWS Account ID	AWS Customer ID	Full Name	Company	PhoneNb	Job Title	Commercial Factor	Status
18606320999	f3zyD0tgmjp	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
778274641626	4RGpJ8fO7J	Harry Rackham	Mr	07401994672	Innovation Consultant	1	Subscribed
775110548216	fRdyaDaDisk	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 of the screen, there is a footer bar with various icons and the text "BASC Administration Console by bluage.com with rage." The status bar at the bottom right shows the date and time: 17:10 13/08/2020.

Annexe 3 : Analyzer, Export CSV

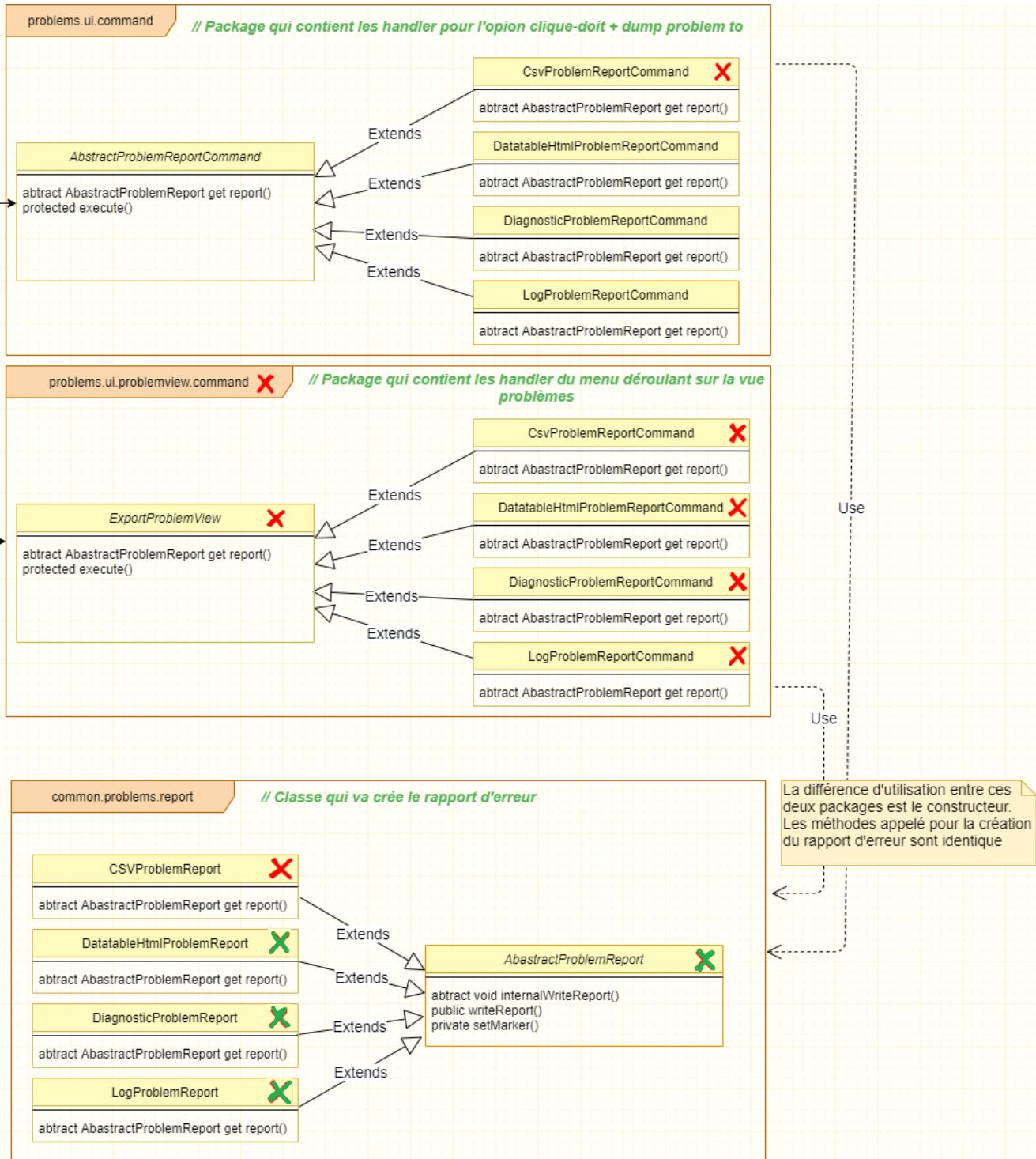


Annexe 2A : Capture d'écran de Analyzer
La figure montre le résultat de l'analyse des dépendance d'une application en COBOL

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

Annexe 4 : Diagramme UML des classes utilisées pour l'export de la vue problème



Légende :

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

Annexe 5 : Schéma illustrant l'utilisation de la base SQL d'une application CICS COBOL traduite en Java.

