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 Lambdas 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 Lambdas 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.

Table de Matières

Listes des Abréviations

API	Application Programming Interface
ARN	Amazon Resource Names
AWS	Amazon Web Services
BASCAC	Blu Age Serverless COBOL Administration console
COBOL	Common Business Oriented Language
CSD	Customer Service Description
DevOps	Software Development (*Dev*) IT operations (*Ops*)
DSL	Domain-Specific Language
IDE	Integrated Development Environment
MDA	Model Driven Architecture
SPA	Single-Page Application
UX	User Experience
VM	Virtual Machine
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]. Composée d'une extension VSC permettant de compiler une application COBOL en java. Le résultat de cette compilation pourra alors être déployé de manière « serverless » sur AWS. C'est le premiers produit de Blu Age à traduire du code, sans nécessiter 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 technologies en obsolescence qui s'opposent aux standards actuels et futurs . La plupart des solutions de Blu Age sont des outils pour assister les développeurs dans la traduction de codes legacy(1) 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 60 ans) de se reconvertir vers le java. Les entreprises peuvent faire le choix d'accompagner ces développeurs en maintenant leurs activitées jusqu'à la retraite. C'est pour cela qu'a été conçu le produit « Serverless COBOL for AWS ». Le développeur COBOL pourra continuer à coder en COBOL, tout en produisant une application compatible avec le déploiement « serverless », sur les microenvironnements d'Amazon que l'on appelle les Lambdas. [3]

Pour fonctionner, ces Lambdas devront être configurées avec un Runtime personnalisé développé par Blu Age. Il devra être fournis aux clients en tant que Layer sur Amazon Web Services. De cette manière les utilisateurs seront en mesure de configurer leurs Lambdas en y ajoutant ce 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. À partir de la console graphique de Amazon, il est déployé manuellement dans toutes 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. À l'issue de ce traitement, on obtient les fichiers du Runtime sous forme de zip. C'est la deuxième partie de la chaîne, chargée du déploiement sur Amazon qu'il faudra concevoir.

(1) Traduit de l'anglais 'héritage', ce dit du matériels, logiciels et système qu'un organisme utilise depuis longtemps, et continue d'utiliser malgré qu'ils soit supplantés par des outils plus modernes.

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. 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' ingénieurs. 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, charger de la technologie de migration ainsi que de la gestion du cycle de vie des logiciels modernisés. Et d'autre part Opteams qui est spécialisée dans la formation et le consulting pour les architecture Cloud

<table>
<tr><td>Age moyen</td><td>Chiffre d'affaire</td></tr>
<tr><td rowspan="2">Pays implanté

France (Pessac, Surenne)

Maroc (Casablanca, Rabat)

États-Unis (Dallas)</td><td>Nombre de Collaborateurs

160</td></tr>
<tr><td>Turnover

5%</td></tr>
</table>

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 suites logiciels 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 génération 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, elle 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 lequel le client pourra déployer son application. Il a la particularité d'être « sans état » et de ne nécessiter ni mise en service, ni gestion de serveur. C'est le fournisseur d'accès (c.-à-d. Amazon) qui se charge de l'administration des ressources, de la maintenance des servers, du dimensionnement/mise l'échelles de la capacité, de la surveillance et de 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 de la plateforme.

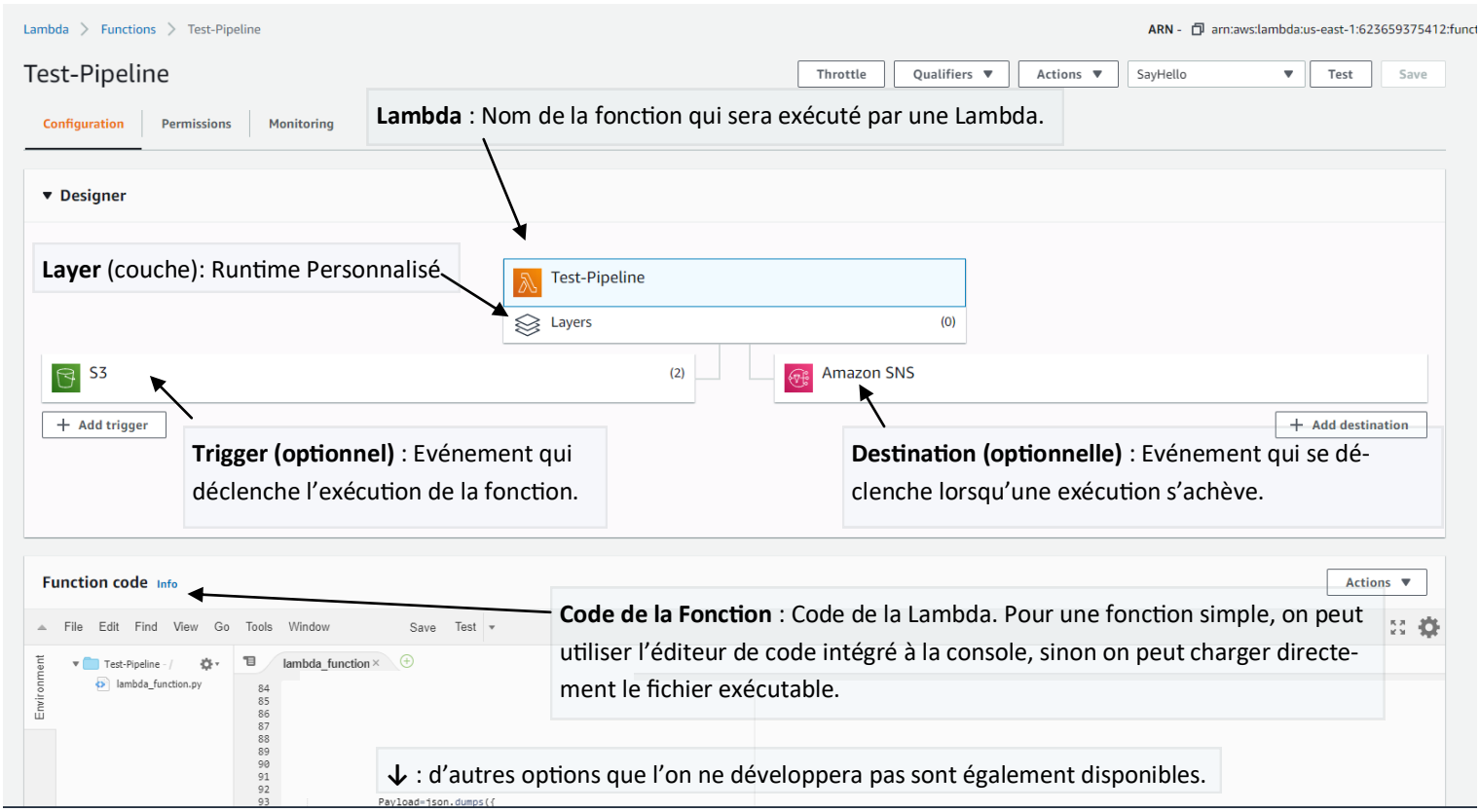


Figure 4 : Capture d'écran de la console AWS pour le paramétrage d'une Lambda

Ensuite les Layers. C'est une couche supplémentaire que l'on peut ajouter à une Lambda : C'est une archive ZIP un environnement d'exécution personnalisé, et en particulier des bibliothèques. Ainsi, il n' est plus nécessaire d'inclure toute les dépendances dans le package qui sera déployer en tant que code de la fonction Lambda. La raison de cette mécanique vient du fait qu'une lambda est limité en taille de code. De ce fait, l'inclusion de toutes les dépendances peut s'avérer être un problème. Cette couche de publication existe indépendamment de la Lambda. Elle peut être utilisé par plusieurs fonctions Lambdas simultanément. Enfin, tout comme n'importe quelle ressource d'Amazon, on peut configurer ses permissions pour qu'elle soit partagée avec d'autres comptes AWS.

Enfin, les buckets. Ce sont des compartiments de stockage fournit par Amazon Simple Storage Services (Amazon S3). Un compartiment permet de stocker des objets; un objet est la somme d'un fichier et de toutes les méta datas qui le décrivent.

Pour finir, il est important de préciser la spécificité technique suivante : Amazon est composé de plusieurs régions. Une ressources (Lambda, Layer, Bucket…) lorsqu'elle est créée 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 aux clients 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). Elle fait appel, par le billet d'une requête http, à un server de Blu Age déployé sur AWS. Celui-ci héberge le compilateur. En retour, le server 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 laquelle il déploiera le produit de la compilation en tant que code de la fonction. Puis pour finaliser le déploiement, il devra ajouter une Layer contenant le Runtime à cette Lambda. Ainsi, il ajoute les dépendances manquantes. Il réalisera cette étape en fournissant l'ARN de la Layer déployée sur le compte Amazon de Blu Age, dont il aura au préalable reçu la permission.[6] (cf Annexe 1).

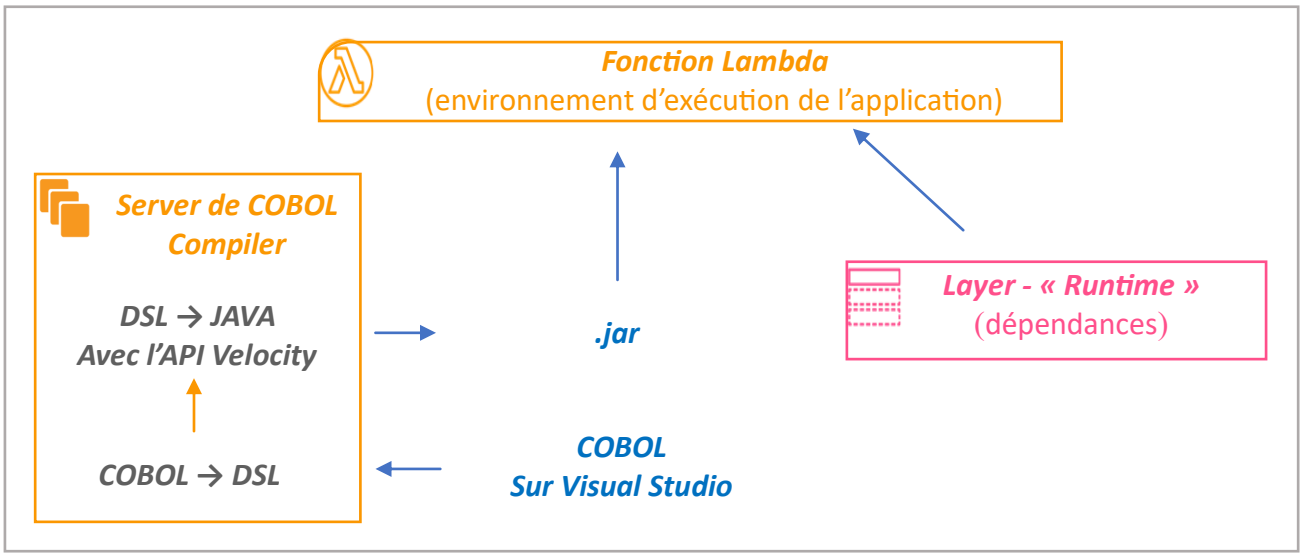


Figure 3 : Schéma fonctionnel de la solution serverless COBOL for AWS solution

III. Réalisation de chaîne de déploiement

1) Étape de recherche

Livrables

La chaîne de déploiement sera en charge de crée une nouvelle version d'une Layer (ou la créer si elle n' existe pas) contenant la dernière release du Runtime de Serverless COBOL for AWS.

Elle devra également :

- S'assurer que l'accès à la Layer soit configuré correctement et que seules les personnes habilitées aient accès au produit.
- Tester la Layer sur une Lambda.
- Faire l'ensemble de ces étapes, dans toute les régions du Cloud ou l'ont souhaite que la Layer soit disponible.

Le prototype devra être sécurisé, facilement paramétrable, 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 du Runtime (compilation, test...). Une première solution envisagée à été d'utiliser le même outil pour la chaîne de déploiement. Cela est possible si l'on utilise 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 opérations 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 crée un pipeline. (services qui permet d'effectuer 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 services rajoute du code supplémentaire pour lier les Lambdas entre elle (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 ajoute 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.

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'atout de cette méthode par rapport à Jenkins est que l'on utilise quand même un outils 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 (cf. figure 5)

Le flow d'exécution du pipeline est codé par 3 fonctions Lambdas codées en python. [7] [8] Grâce à la bibliothèque d'Amazon Boto 3, elles peuvent interagir avec les autre ressources et reproduire toute 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. A l'issue de ce traitement, la nouvelle version du Runtime, va être chargé dans le *« Bucket Source »,* dans la première région. Cette évènement va faire office de trigger de la fonction principal 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 syncrone 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 droit d'accès à partir des ACL.

Une fois le déploiement de la Layer effectuer 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 nommé *« Deployer »*. Une pour chaque région. Elles ont pour rôle de déployer le Runtime dans la région qui leurs a été attitré.

Elle devrons 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 appel 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 seront 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.

Additionnellement, 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. Elle est nomée *« Notification-Manager »* Cette dernière permet de centraliser les informations des différents maillions de la chaîne qui s'exécutent indépendamment.

N.B : L'annexe 2 fournis un récapitulatifs des fonctions créées et de leur rôle.

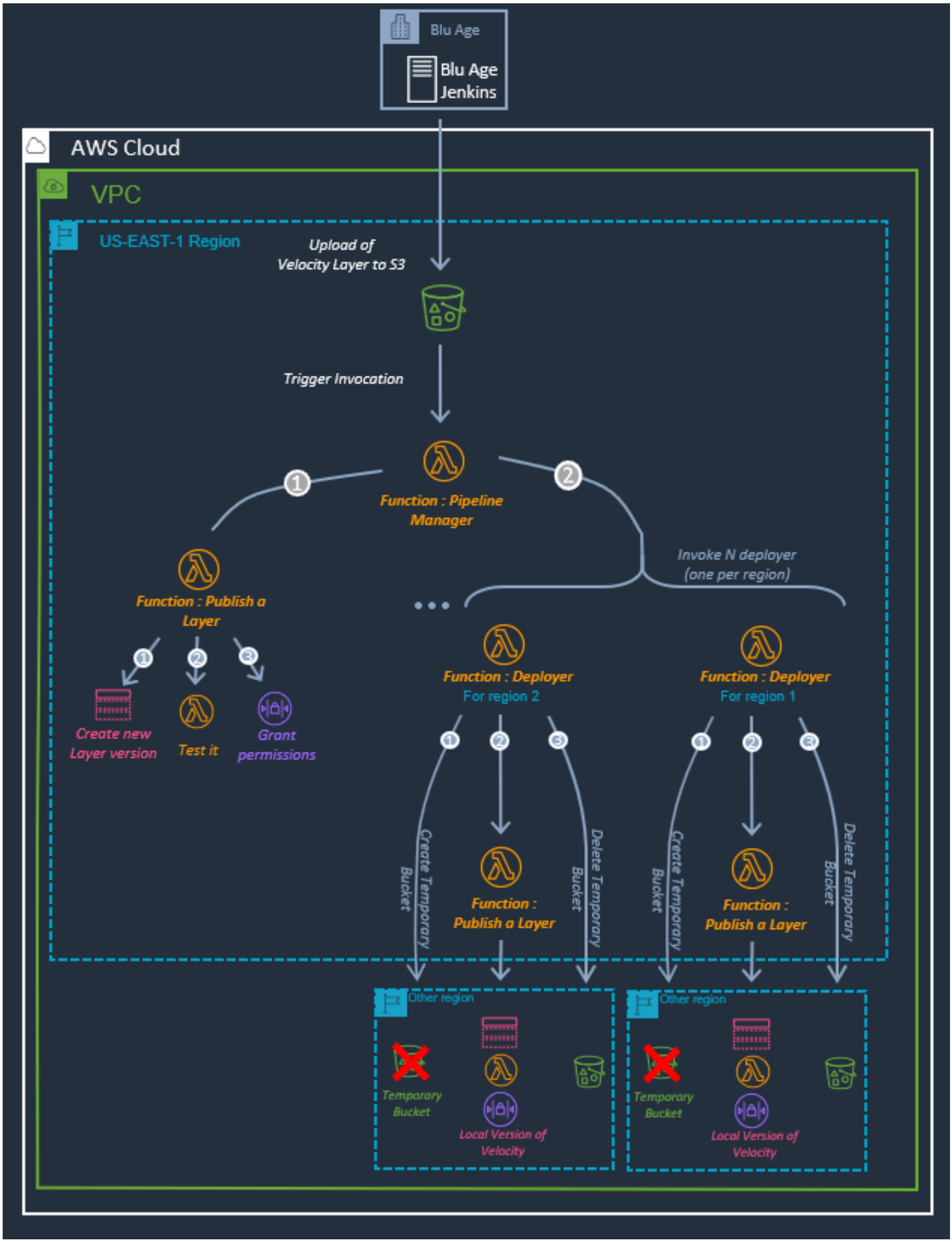


Figure 5 : Schéma de l'architecture de la chaîne de déploiement

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_REQUIRMENT : Règle sur le nom du fichier qui permettra de filtrer les fichiers qui déclencherons effectivement le pipeline.
- FILE_EXTENTION : 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.

Pré requis pour le fonctionnement du pipeline :

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.

```
{
    "client_list" :
    [
        {
          "CLIENT_NAME": "Client1",
          "CLIENT_ID": 623659375412
        },
        {
          "CLIENT_NAME": "Client2",
          "CLIENT_ID": 623659375412
        }
    ]
}
```

Figure 6 : Exemple de ACL.json

Journaux d'exécution

Par défaut, Amazon stocke les journaux d'exécution à travers un service nommé Cloud Watch. Le principal inconvénient de ces journaux pour notre pipeline multifonctions est que les logs de chaques fonctions se retrouvent séparées les uns des autres. Pour faciliter la lecture, un système de journaux qui centralise les logs de toutes les fonctions de la chaîne à été mise en place. Dans le bucket source va se trouver un dossier Log (ou crée s'il n'existe pas déjà). À 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 1er 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 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).

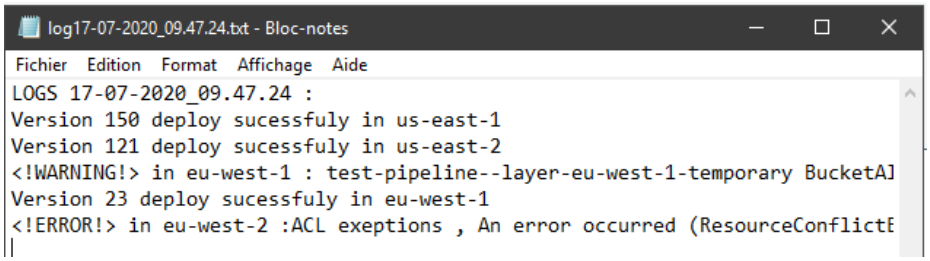


Figure 7 : Résultat du journal d'exécution du pipeline

Système de Notification

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

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

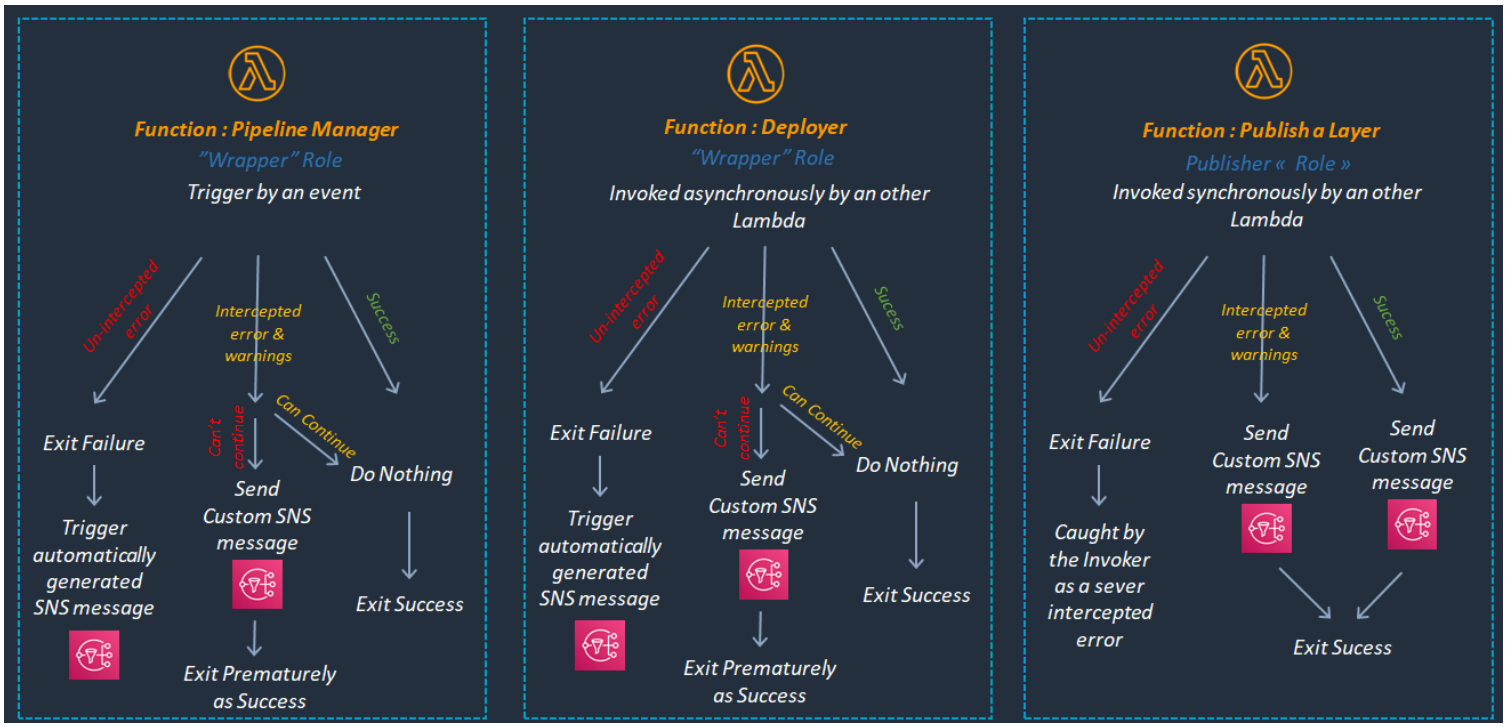


Figure 8 : Schéma de la gestion des Notification SNS selon la fonction

Dans le cas où l'erreur n'as pas été interceptée ou si l'invocation de la Lambda *Notification-Manager* échoue, un mécanisme mise en place par Amazon prend le relais et envoie un message SNS généré automatiquement. (C'est l'option *Destination, cf.* II-1- L'environnement AWS). Ce message contiendra du texte au format json des informations sur l'exécution de la Lambda dont il provient. Il n'est pas vraiment adapté pour la lecture par un humain. C'est pour cela qu'on préfèrera l'utiliser comme mécanisme de secours uniquement.

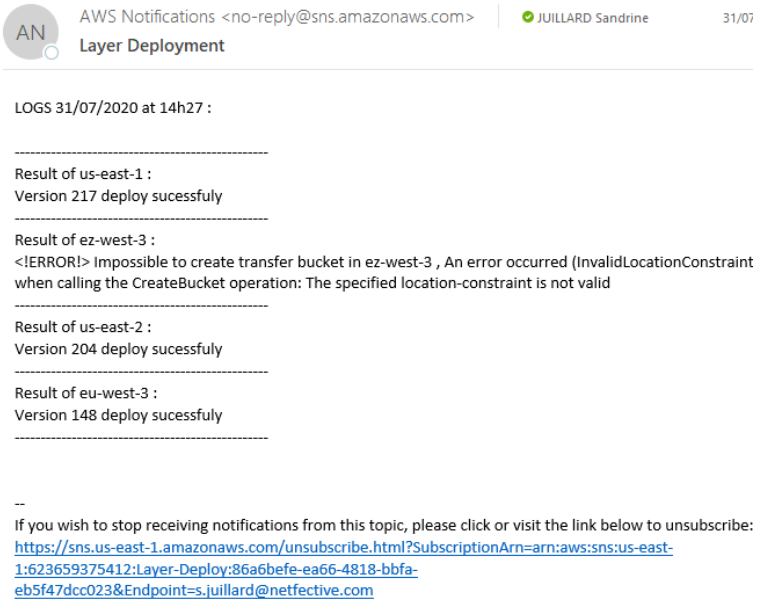


Figure 9 : Résultat de la notification mail reçu lorsque l'on est abonné au topic SNS de la chaîne de déploiement

Gestion des ACL

Pour éviter de devoir éditer à la main le fichier ACL.json, une fonction Lambda annexe a été créé. Celle-ci gérera automatiquement la mise à jour ce fichier. Blu Age stocke les informations de ses clients dans une ressource Amazon sous forme de base de données Le services en question s'appelle 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 identifiants, puis met à jour le fichier ACL contenu dans le « *bucket source* ».

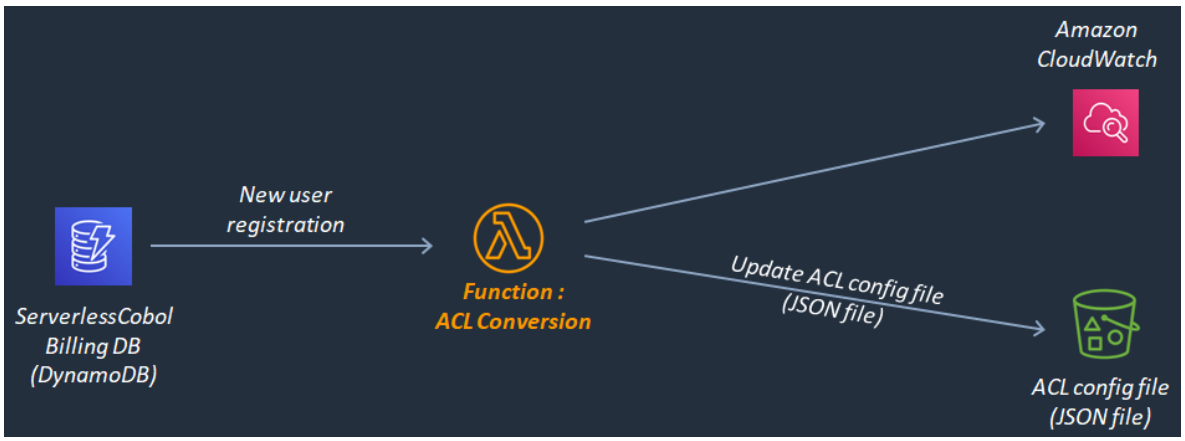


Figure 10 : Schéma fonctionnel de la mise à jour automatiquement le fichier ACL

Génération de bucket temporaire

Un bucket doit avoir un nom unique. Pour évité que la pipeline échoue lors de la génération des buckets temporaires, ils sont nommés de la manière suivante :

Le nom du bucket prendra la forme : [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 ajouté au nom défini plus tôt jusqu'à trouver un nom valide.

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 superviser le pourcentage de réduction à accorder à un utilisateur. 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.

L'outils 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 réalisée 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 fallu 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 exigé l'utilisation de 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 à partir de la console graphique d' Amazon. Puis le services 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 à été abandonné. 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.

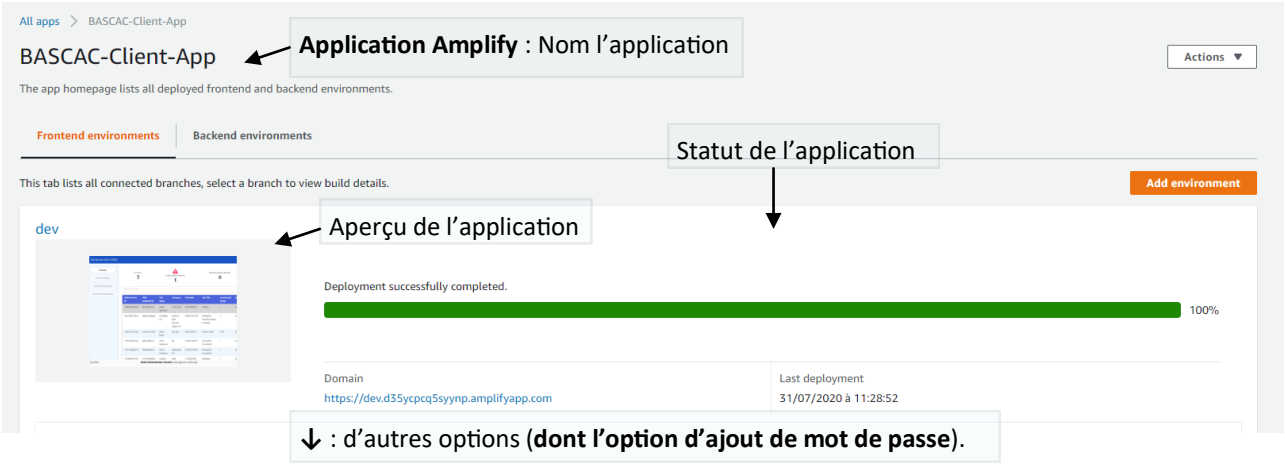


Figure 11 : Capture d'écran de la console Amazon Amplify

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 Velocity pourra 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 issues 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, commandés 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 créée par Blu Age, c'est une vue dont le code est interne à Eclipse, il est donc impossible d'en récupérer le contenu directement. Pour récupérer ce contenu, il a fallu aller à la source : On va récupérer les informations dans les *markers* des projets. Un *marker* est un ensemble de donnée généré dans notre cas à la racine du projet dans lesquels les informations relatives au projet sont stockées. En récupérant ces informations on peut reconstituer la vue problème, puis l'exporter au 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 spécifique: en faisant une clique droite sur celui-ci dans la vue navigation, l'option proposais d'exporter les problèmes, à la racine du projet, dans plusieurs formats possible. Il était donc judicieux d'adapter la solution précédente pour qu'elle s'intègre au travail déjà effectué, 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 situé dans l'onglet navigation. (cf Annexe 5).

Une fois la tâche terminée, 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 présenter l'option implémenté.

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

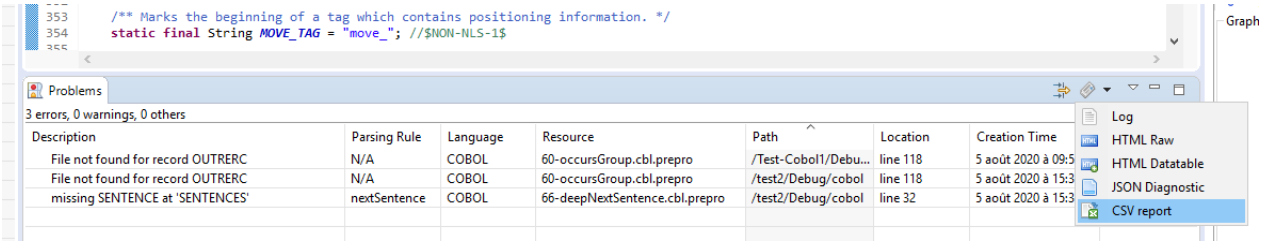


Figure 12 : Vue problème et son menu déroulant permettant l'export des problèmes

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 à disposition de leur clients un Plugin VSC. Ce plugin ajoute à l'IDE plusieurs commandes pour compiler leur code, en faisant appel à un server par l'intermédiaire d'une requête http (cf II-2- Serverless 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 ressources 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'a pas une extension COBOL, et ce, peu-importe la nature de la commande invoquée. Le problème est que pour la compilation d'un fichier BMS, la présence d'un fichier COBOL n'est pas nécessaire.

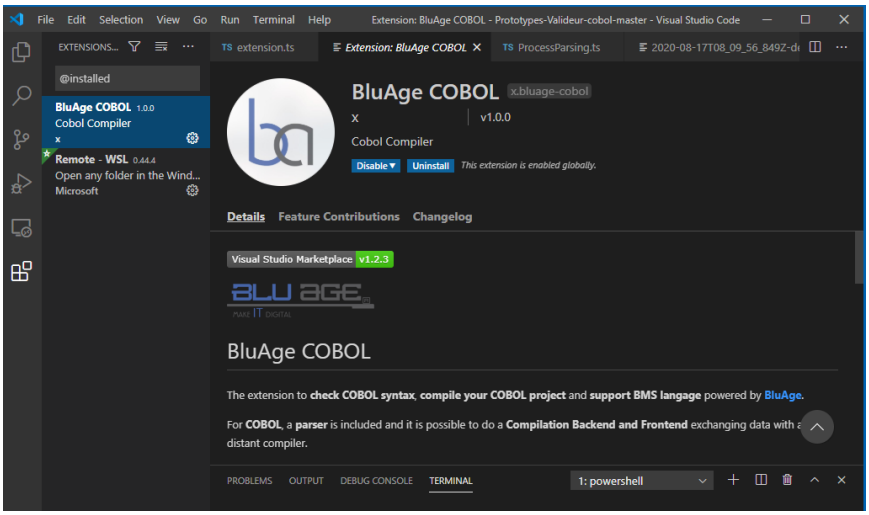


Figure 13 : Capture d'écran de l'extension BluAge COBOL dans l'IDE VSC

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

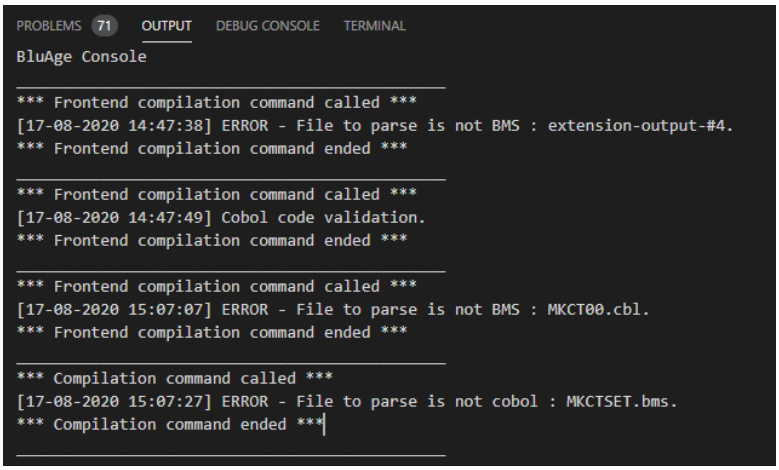


Figure 14 : Résultat dans la console de la phase de vérification, on voit que lorsque le type de fichier sélectionné n'est pas bon, la phase de validation échoue

Plugin Blu Age COBOL — Tâche 2: Ajout d'une commande

Les applications CICS (Customer Information Control System) sont des systèmes qui permettent d'effectuer des opérations transactionnelles. Généralement codé en COBOL, c'est couramment ce type d'application qui doit être transcrite 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 liée 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.

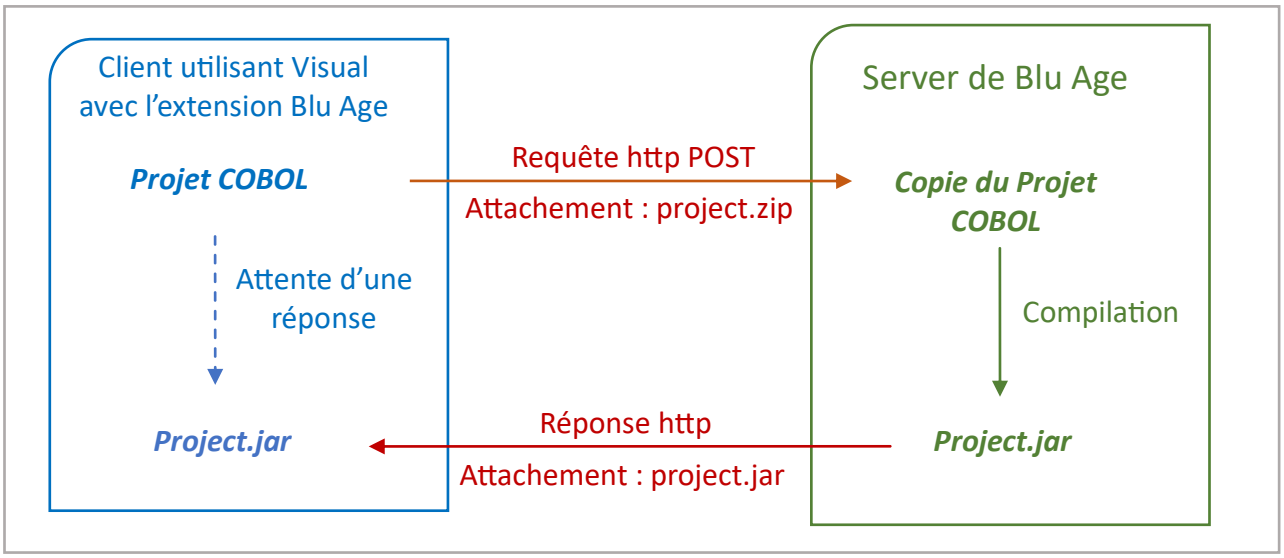


Figure 15 : Schéma fonctionnel de la compilation par l'extension Blu Age COBOL (architecture de type REST [12])

Le développement des mécanismes qui vont être présentés a été inspiré des commandes de compilations Blu Age déjà présente qui figure sur le schéma ci-dessus. L'objectif de cette tâche a 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 TypeScript 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

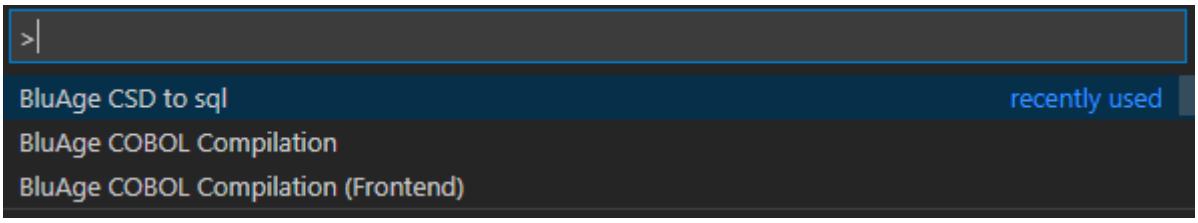


Figure 16a : Capture de la commande CSD to SQL dans VSC

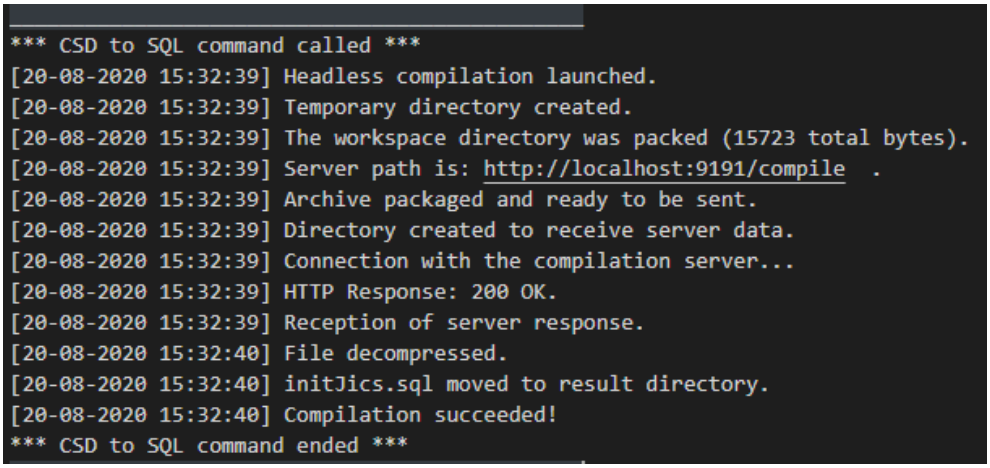


Figure 16b : Résultat de la commande CSD to SQL dans les logs dans la consol sur VSC

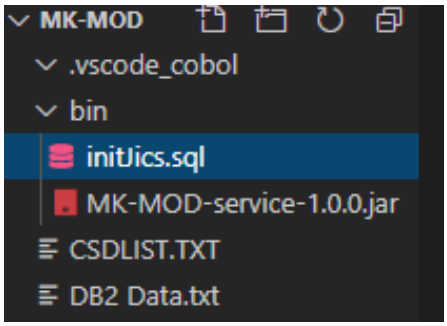


Figure 16c : Fichier SQL généré par la commande CSD to SQL

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 était testé manuellement au fur et à mesure de son développement.

Le pipeline n'a pas été paramétré et mis en service pour les vrais fichiers du Runtime, mais tout à été mise en place pour rendre ce travail le plus simple possible. Une documentation détaillée, en anglais a été créée dans cette optique. De plus, une attention toute particulière a été portée à la clarté du code.

En ce qui concerne les 4 tâches supplémentaires, elles ont toutes été terminées également. Parfois, les contributions apportées ont créées des problèmes au moment du packaging, en raison de nouvelles dépendances ajoutées. Heureusement cela fut rapidement résolu.

D'un point de vu personnel, ce stage m'a permis de gagner en compétence, et ce dans diverses domaines : devOps, devWeb et dev logiciel. J'ai codé dans plusieurs langages différents, utilisé divers outils. Cette expérience m'a permis de développer mes capacités d'adaptation et d'autonomie.

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. Enfin, j'ai eu l'immense satisfaction d'avoir terminée les travaux que l'on m'a 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=jhB39NlgGl4&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

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

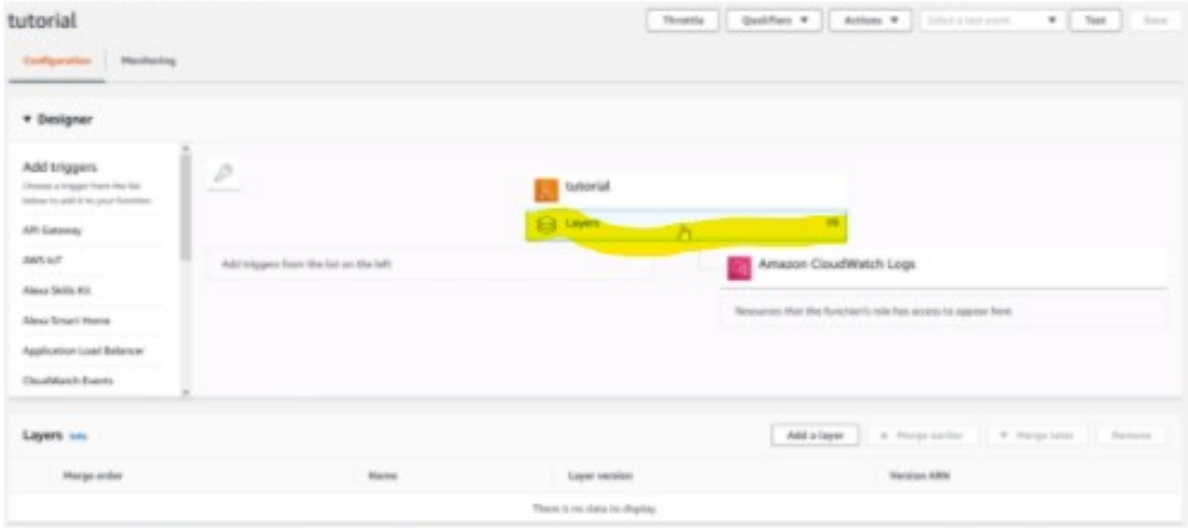
A. **Create a new lambda function**

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**

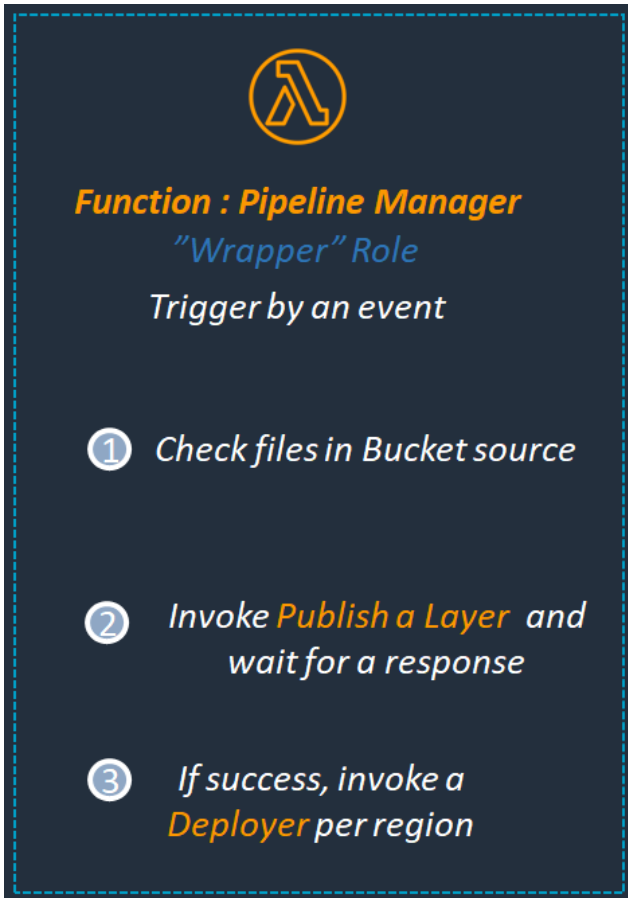
1. Go to the function editing page;
2. Click on the Layer zone to make the Layer management zone visible;
3. Press the button "Add a layer";

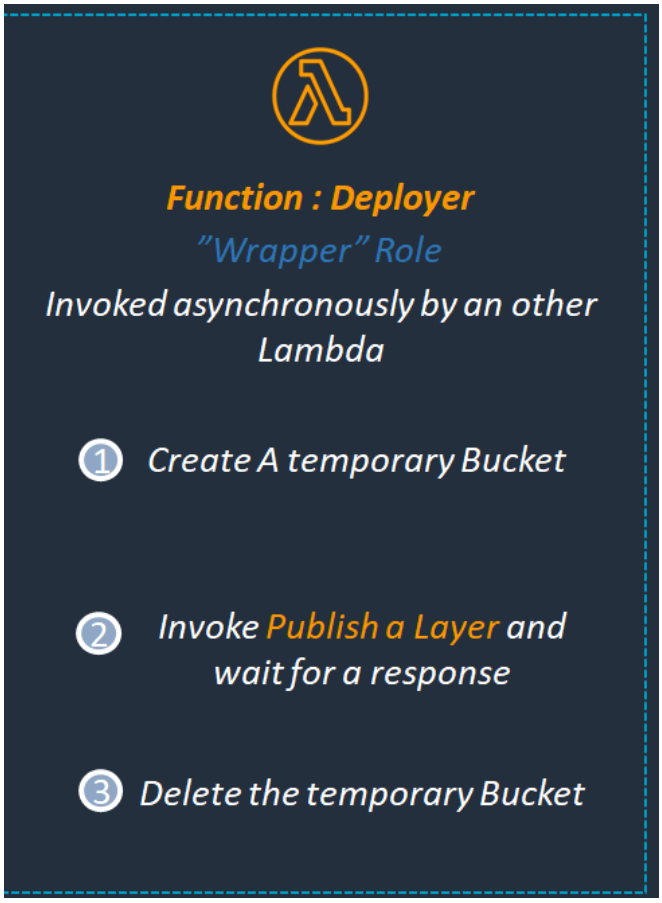


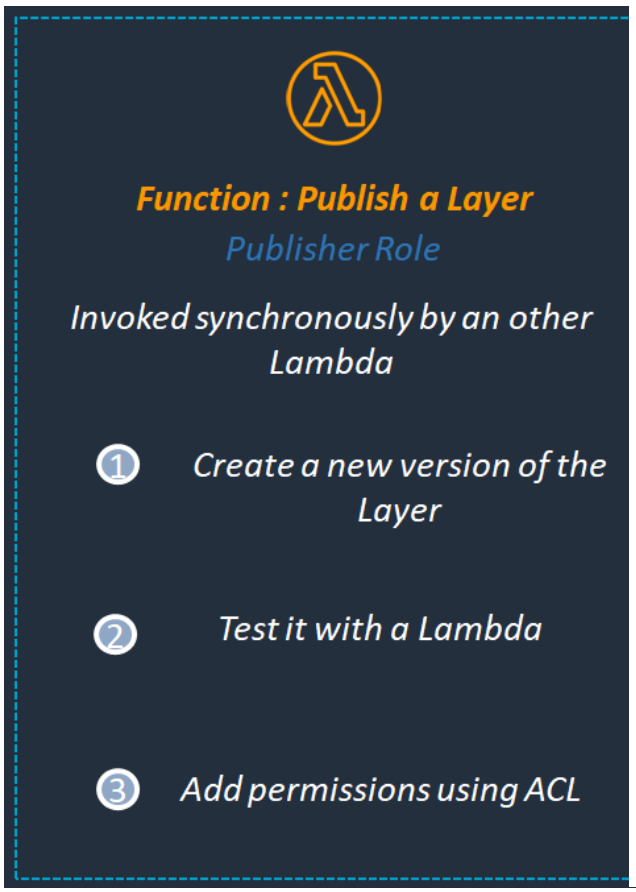
4. Select the option "Provide a layer version ARN";
5. In the Layer version ARN textbox, paste the layer version ARN you have been granted use permission;
6. Click on the "Add" button;
7. Click on the function name, to make the standard Configuration visible again;
8. Browse to the function code section and upload the jar file generated previously;
9. In the Handler textbox, replace the existing default value by com.netfective.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;

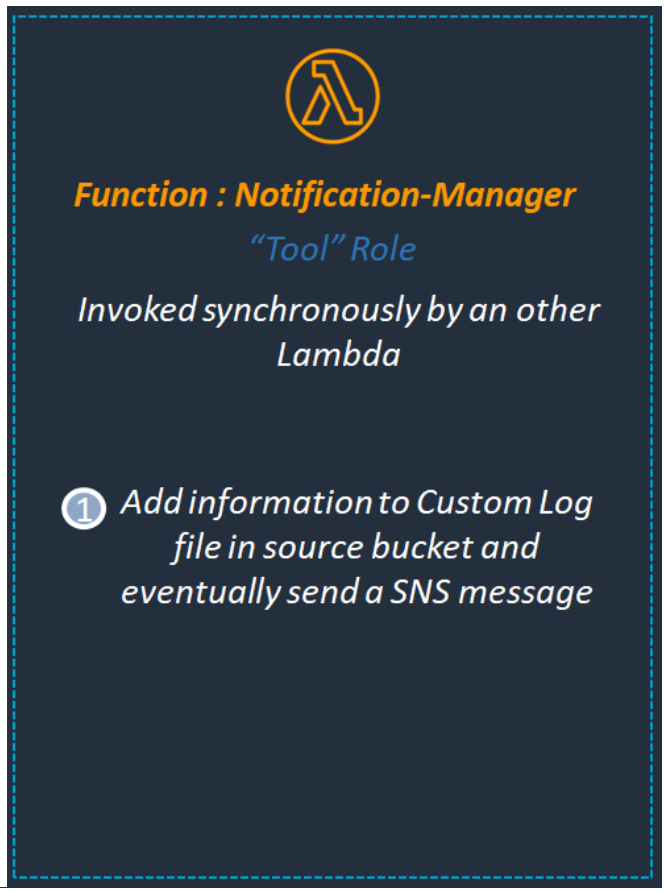
Annexe 1 : Extrait du tutoriel Blu Age Serverless. [6]

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

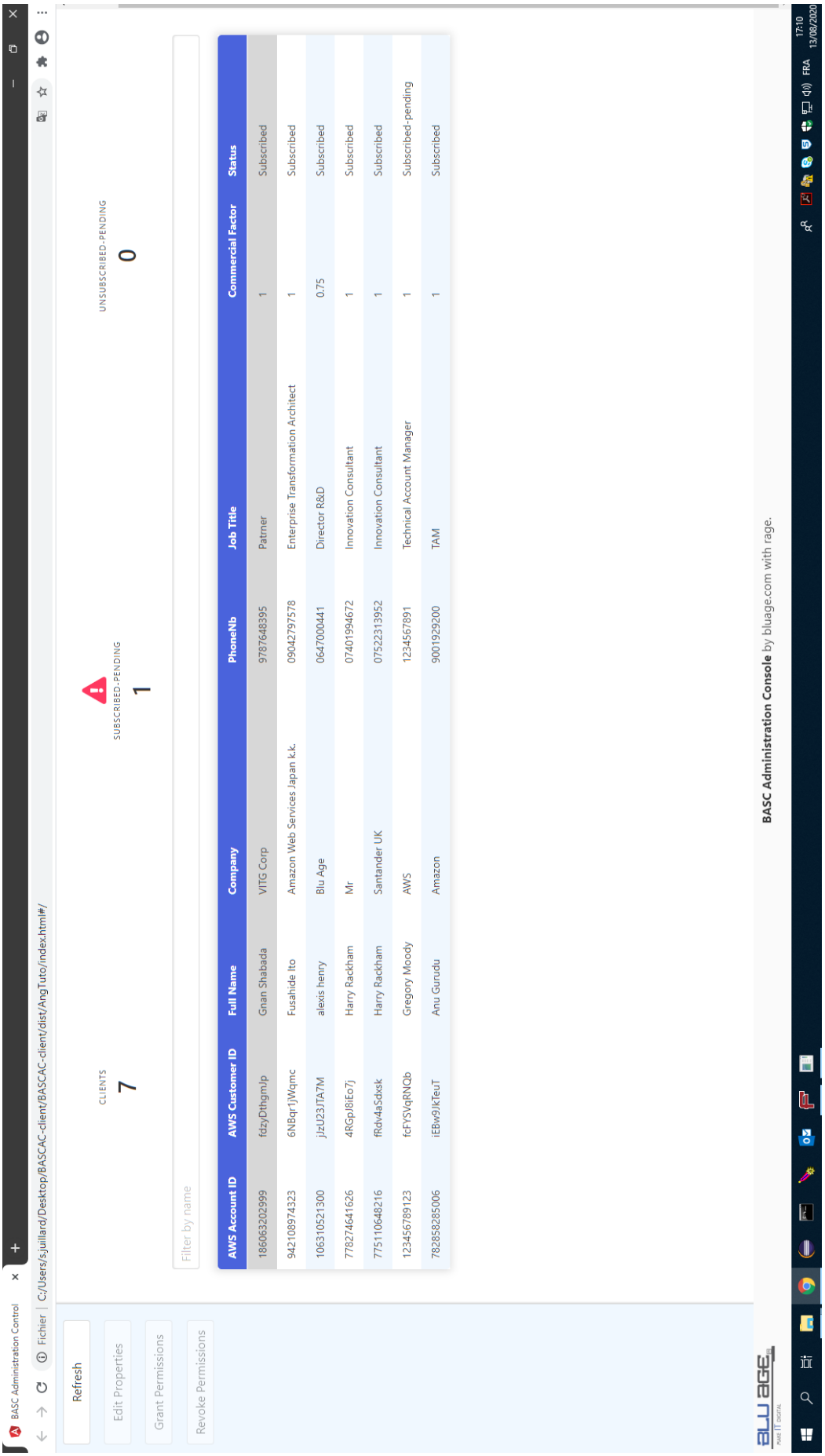




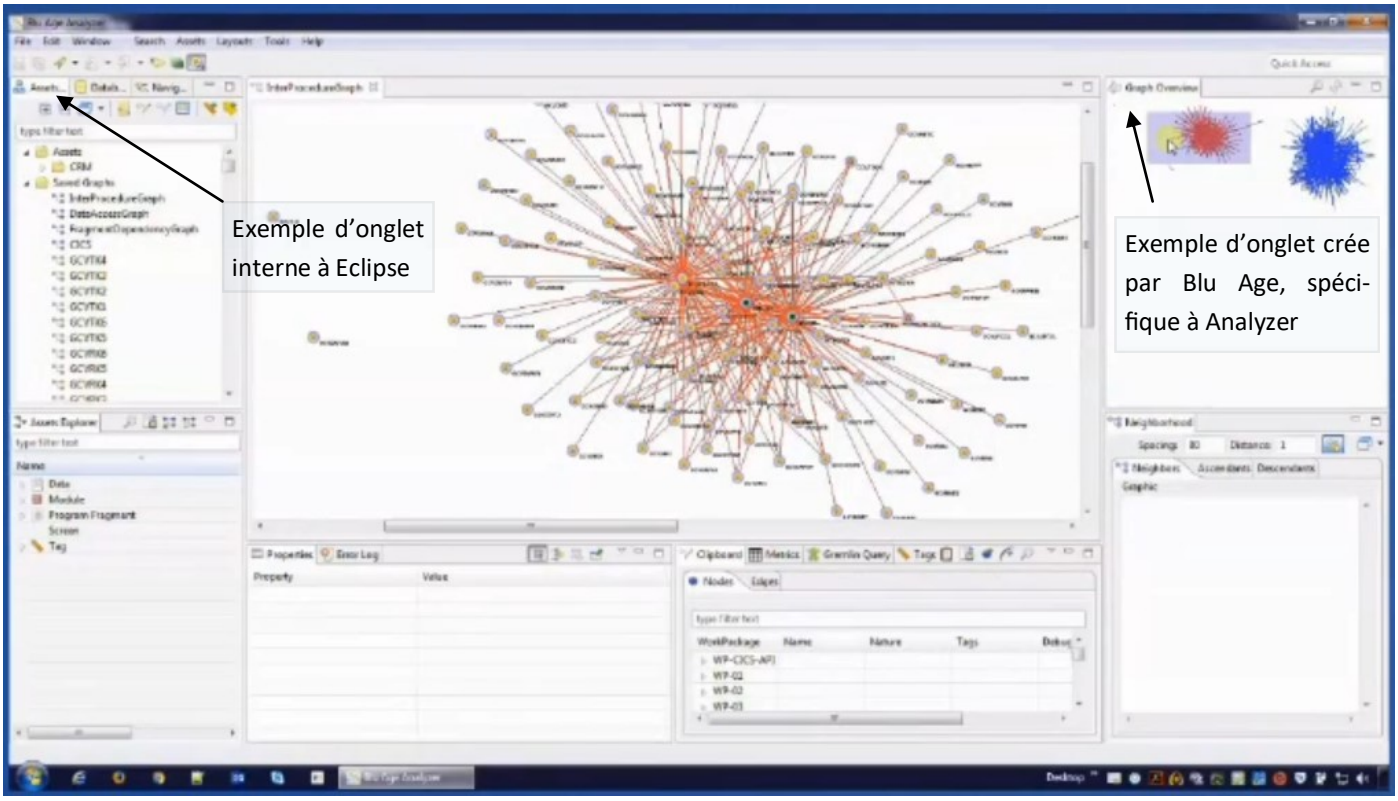




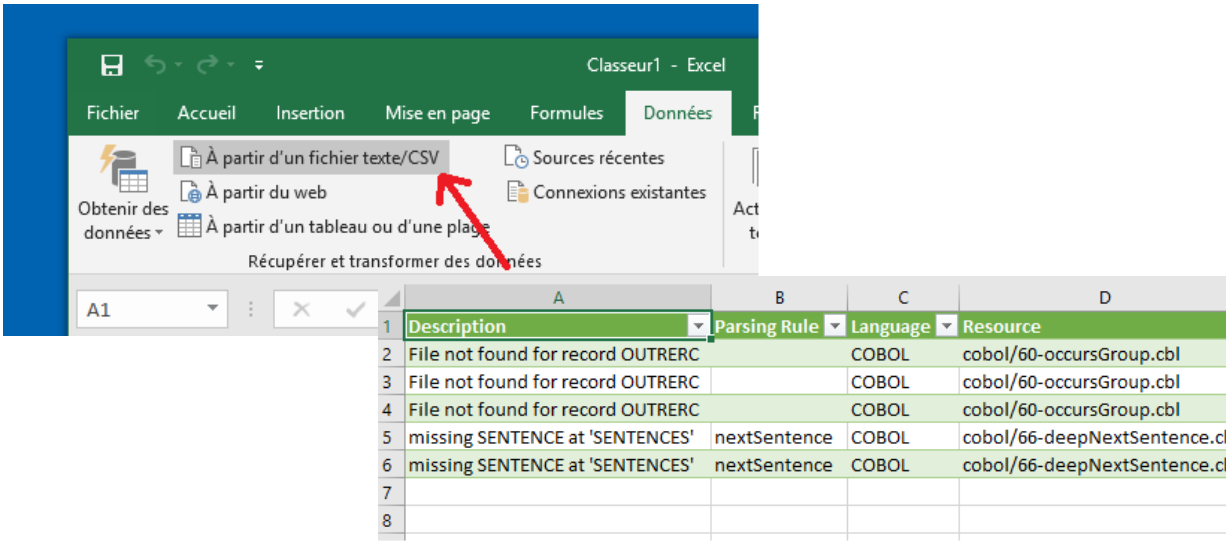
Annexe 3 : Capture d'écran de BASCAC



Annexe 4 : Analyzer, Export CSV

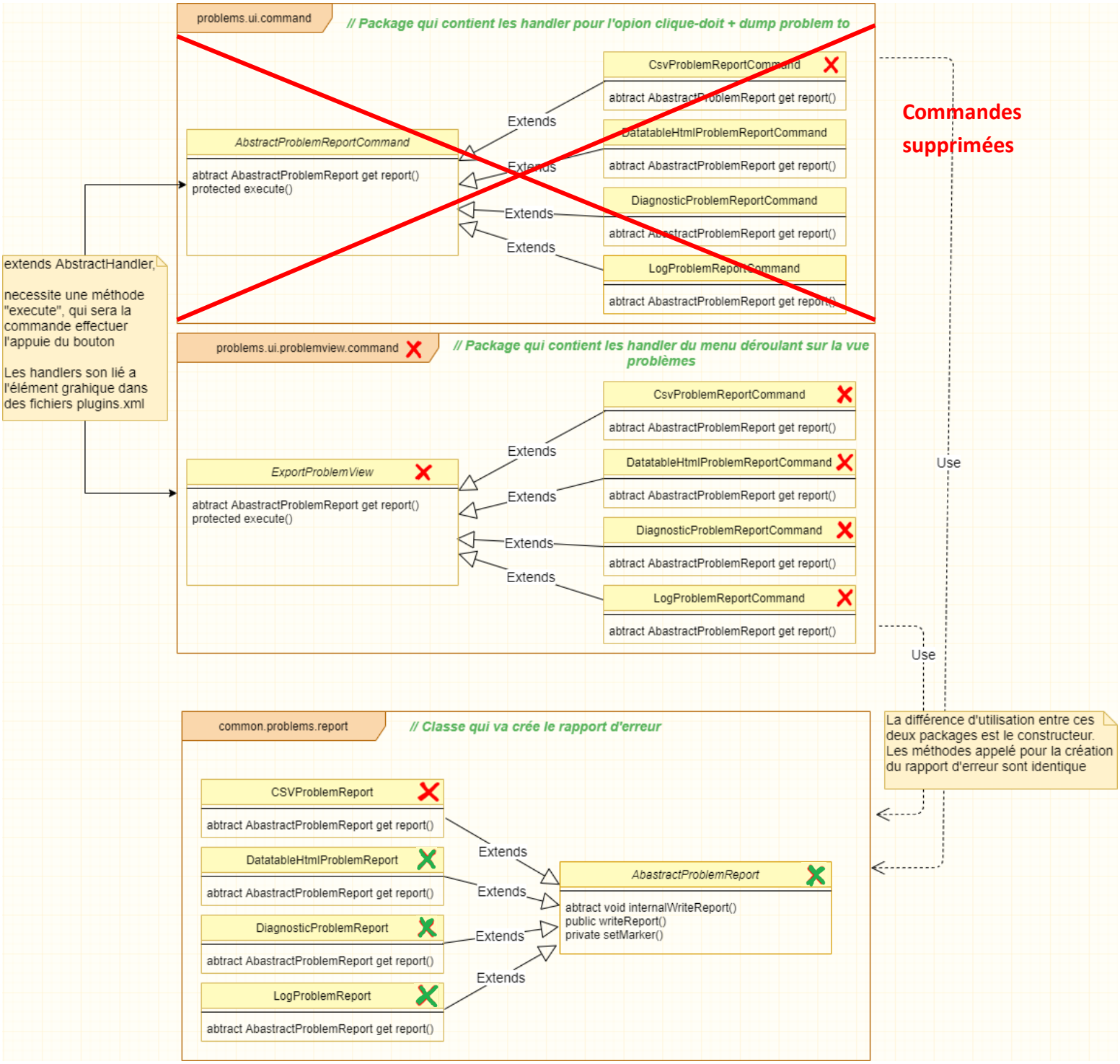


Annexe 4A : Capture d'écran de Analyzer
La figure montre le résultat de l'analyse des dépendances d'une application COBOL



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

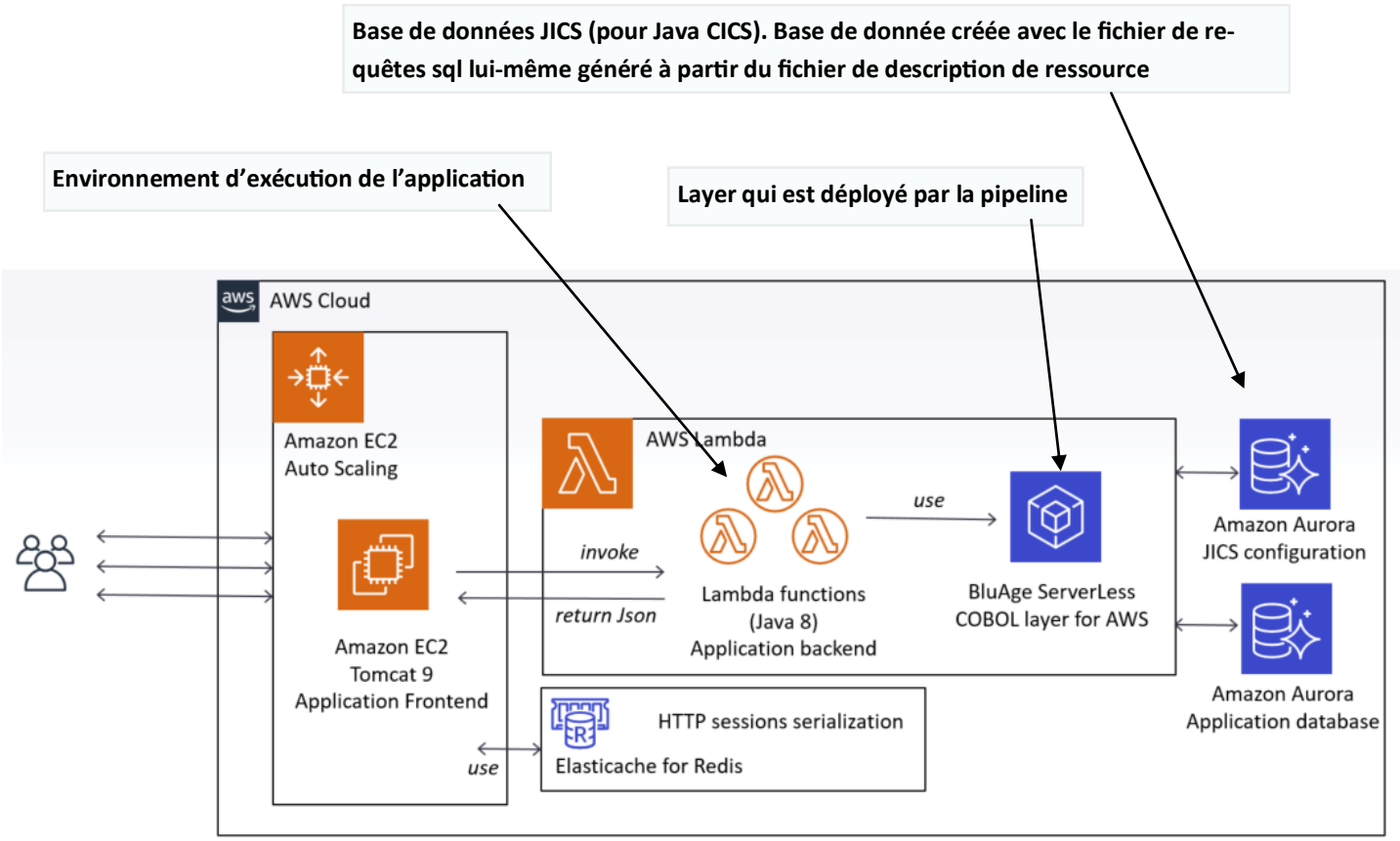
Annexe 5 : Diagramme UML des classes utilisées pour l'export de la vue « problème »



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 étés modifié.

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.