



الْمَهْدِيُّ الْوَطَنِيُّ لِلْبَرِيدِ وَالْمَوَاسِلَاتِ
•٢٤٠٦٠٩٠ | +٢٣٥٧٥٨ | ٩٢٣٥٧٥٨
Institut National des Postes et Télécommunications

INSTITUT NATIONAL DES POSTES ET TÉLÉCOMMUNICATIONS

FILIÈRE : SMART ICT

RAPPORT DE PROJET D'INNOVATION

Building Microservices and a CI/CD Pipeline with AWS

Année académique : 2023-2024

Realisé par :
Ayman HAKIM

Encadrante :
Pr. AIT MANSOUR ASMAE

Table des matières

1	Introduction générale et contexte du projet	6
2	L'importance de l'architecture microservices	6
3	Le Sujet	6
4	La Problématique	6
5	AWS et ses services	6
5.1	Introduction aux services AWS	7
5.2	Amazon Virtual Private Cloud (Amazon VPC)	7
5.3	Amazon EC2 et Load Balancer	7
5.4	AWS CodeCommit, CodeDeploy, et CodePipeline	7
5.5	Amazon ECS et ECR	7
5.6	AWS Cloud9	7
5.7	AWS IAM	7
5.8	Amazon RDS et CloudWatch	8
6	Phase 1 : Planification de la conception et estimation des coûts	8
6.1	Diagramme d'architecture	8
6.2	Estimation des coûts	10
7	Phase 2 : Analyse de l'infrastructure de l'application monolithique	13
7.1	Analyse des Ports et Processus	13
7.2	Structure de l'Application	13
7.3	Réflexions	14
7.4	Connexion à la base de données RDS et observation des données	15
8	Phase 3 : Mise en place de l'environnement de développement AWS Cloud9	16
8.1	Création de l'IDE AWS Cloud9	16
8.2	Copie du code de l'application vers l'IDE	16
8.3	Création de l'environnement de développement pour les microservices	17
8.4	Mise en Place du Dépôt Git et Synchronisation avec AWS CodeCommit	17
9	Phase 4 : Configuration de l'application en deux microservices et tests dans des conteneurs Docker	19
9.1	Ajustement des paramètres du groupe de sécurité de l'instance AWS Cloud9	19
9.2	Modification du code source du microservice customer	19
9.3	Création du Dockerfile du microservice customer et lancement d'un conteneur de test	22
9.4	Vérification du fonctionnement du microservice customer	23
9.5	Commit et push des changements de code dans CodeCommit	24
9.6	Observation des détails du commit dans CodeCommit	24
9.7	Modification du code source du microservice employee	24
9.8	Ajustements du microservice employee	25

9.9	Création du Dockerfile pour le microservice employee et lancement d'un conteneur de test	25
9.10	Création d'une Image Docker	26
9.11	Vérification de Fonctionnement	26
9.12	Ajustement du Port et Reconstruction de l'Image	27
9.13	Intégration du Code dans CodeCommit	27
10	Phase 5 : Création de dépôts ECR, un cluster ECS, des définitions de tâches et des fichiers AppSpec	28
10.1	Création des dépôts ECR et téléversement des images Docker	28
10.2	Création d'un Cluster ECS	29
10.3	Création d'un Répertoire CodeCommit pour les Fichiers de Déploiement	30
10.4	Création et Enregistrement des Définitions de Tâches	30
10.5	Création des fichiers AppSpec pour chaque microservice	31
10.5.1	Fichier AppSpec pour le microservice customer	31
10.5.2	Fichier AppSpec pour le microservice employee	32
10.6	Mise à jour et Validation des Fichiers de Définition de Tâches	32
10.7	Intégration des Changements dans CodeCommit	32
11	Phase 6 : Création de groupes cibles et d'un Application Load Balancer	33
11.1	Création de quatre groupes cibles	33
11.1.1	Groupes cibles pour le microservice customer	33
11.1.2	Groupes cibles pour le microservice employee	34
11.2	Création d'un Load Balancer et Configuration des Règles de Trafic	34
11.2.1	Configuration des Auditeurs	35
12	Phase 7 : Création de deux services Amazon ECS	35
12.1	Création du service ECS pour le microservice customer	35
12.2	Création du service ECS pour le microservice employee	36
13	Phase 8 : Configuration de CodeDeploy et CodePipeline	36
13.1	Création d'une application CodeDeploy et de groupes de déploiement	37
13.2	Création d'un pipeline pour le microservice customer	38
13.3	Test du pipeline CI/CD pour le microservice customer	39
13.4	Surveillance des tâches dans Amazon ECS	40
13.5	Vérification des configurations du Load Balancer et des groupes cibles	41
13.6	Création d'un pipeline pour le microservice employee	41
13.7	Test du pipeline CI/CD pour le microservice employee	42
13.8	Vérification des configurations du Load Balancer et des groupes cibles	43
14	Phase 9 : Ajustement du code des microservices pour relancer le pipeline	43
14.1	Limitation de l'accès au microservice employee	44
14.2	Modification de l'interface utilisateur du microservice employee et mise à jour de l'image Docker	44
14.3	Test du pipeline CI/CD pour le microservice employee	45
14.4	Test de l'accès au microservice employee	45
14.5	Augmentation du nombre de conteneurs pour le microservice customer	46

15 Conclusion	47
----------------------	-----------

Table des figures

1	Diagramme d'architecture des microservices et du pipeline CI/CD avec AWS	9
2	Résumé de l'estimation des coûts pour l'architecture proposée.	11
3	Détails de l'estimation des coûts démontrant les coûts mensuels et totaux pour chaque service AWS utilisé.	12
4	Résultat de la commande lsof montrant que le daemon Node.js utilise le port 80.	13
5	Contenu du répertoire de codebase indiquant la présence du fichier index.js.	14
6	Capture d'écran montrant le contenu du fichier package.json confirmant les dépendances de l'application.	14
7	Résultats de la vérification de la connectivité de la base de données RDS.	15
8	Interactions avec la base de données RDS via le client MySQL.	15
9	Résultats du transfer des fichiers.	16
10	Structure des répertoires pour les microservices customer et employee.	17
11	Commandes Git utilisées pour initialiser le dépôt	18
12	Commandes Git utilisées pour pousser le code vers AWS CodeCommit.	18
13	Confirmation de la branche dev dans AWS CodeCommit.	18
14	Ajustements du groupe de sécurité pour l'instance AWS Cloud9.	19
15	Nouveau code du fichier supplier.controller.js après modifications.	20
16	Modification du fichier index.js pour exécuter l'application sur le port 8080.	21
17	Suppression du bouton "Add a new supplier" dans le microservice customer.	22
18	Liste des images Docker, incluant l'image customer créée.	23
19	Vérification des conteneurs Docker actuellement en exécution, montrant le microservice customer actif.	23
20	Aperçu de l'application après modifications, affichant la liste des fournisseurs.	24
21	Enregistrement des modifications dans le dépôt Git.	24
22	Envoi des changements à la branche dev dans AWS CodeCommit.	24
23	Modification des fichiers.	25
24	Construction de l'image docker	26
25	Lancement du conteneur.	26
26	Vérification du site	27
27	Le commit et le push dans dev après la vérification	27
28	Authentification du client docker et Creation du dépôt ECR	28
29	Résultat de la commande pour pousser les images ECR.	29
30	Vérification des images dans Aws ECR	29
31	Création du cluster ECS dans la console AWS.	29
32	Création du répertoire Deployment.	30
33	Configuration des fichiers de définition de tâches.	30
34	Enregistrement des définitions de tâches dans ECS.	31
35	Vérification des définitions de tâches dans ECS.	31
36	Le fichier Customer.Yaml	32
37	Commit et push des changements dans CodeCommit.	33
38	Liste des groupes cibles créés pour les microservices.	34
39	l'ALB qui a été créé.	34
40	Règles configurées pour les auditeurs de l'ALB.	35
41	Le fichier de configuration.	36

42	La commande AWS CLI pour créer le service dans ECS.	36
43	Création de l'application CodeDeploy microservices	37
44	Création du groupe de déploiement.	38
45	Création du pipeline avec intégration des sources ECR.	39
46	Ajout d'une nouvelle source d'image Docker dans le pipeline.	39
47	Le microservice customer fonctionnant via le DNS du load balancer.	40
48	Confirmation du succès du déploiement dans CodeDeploy.	40
49	Surveillance des tâches dans le cluster ECS.	41
50	Modification des règles d'écouteur de l'Application Load Balancer.	41
51	Configuration du pipeline pour le microservice employee avec intégration des sources ECR.	42
52	Confirmation du succès du déploiement du microservice employee dans CodeDeploy.	42
53	Succès de l'affichage du site pour le microservice employee.	43
54	Mise à jour des règles d'écouteur de l'Application Load Balancer pour le microservice employee.	43
55	Modification des règles de l'écouteur pour l'adresse IP spécifiée.	44
56	Poussée de la nouvelle image Docker vers Amazon ECR.	44
57	Vérification des images docker dans ECR.	45
58	Exécution réussie du pipeline après mise à jour de l'image Docker.	45
59	Modification réussie du banner de notre site web.	46
60	Augmentation du nombre de conteneurs pour le microservice customer.	46

1 Introduction générale et contexte du projet

Dans un monde où la technologie évolue à une vitesse fulgurante, les entreprises doivent constamment innover pour rester compétitives. Ce projet explore la transformation numérique d'une application traditionnelle en une architecture moderne basée sur des microservices, en tirant parti de l'infrastructure flexible et évolutive proposée par Amazon Web Services (AWS). Nous aborderons la problématique posée par l'application monolithique existante, qui souffre de problèmes de fiabilité et de performance, et proposerons une solution modulaire et élastique capable de répondre aux demandes dynamiques du marché.

2 L'importance de l'architecture microservices

Dans un système informatique, la manière dont les composantes sont organisées et interconnectées a un impact significatif sur la performance, la maintenance, et l'évolutivité de l'application. L'architecture microservices représente une méthode structurale qui divise une application en petites parties indépendantes, chacune ayant une fonction spécifique. Cette approche offre une flexibilité inégalée par rapport aux anciennes architectures monolithiques, où toute l'application est conçue comme un seul et unique bloc.

3 Le Sujet

L'objectif de ce projet est de concevoir et de déployer une architecture microservices pour une application de gestion de fournisseurs de café. En se détachant d'une structure monolithique rigide, nous visons à créer une nouvelle architecture qui remédie aux défaillances actuelles et introduit une scalabilité et une résilience accrues.

4 La Problématique

La principale difficulté réside dans le découplage des composantes de l'application existante en services indépendants qui peuvent être déployés et mis à l'échelle de manière autonome. Ce projet vise également à établir un pipeline CI/CD automatisé pour mettre en œuvre une stratégie de déploiement bleu/vert, où deux environnements identiques permettent de passer de l'ancienne à la nouvelle version de l'application sans interruption de service.

5 AWS et ses services

AWS est à la pointe du cloud computing, offrant un éventail de services qui permettent de construire des solutions informatiques de haute qualité, accessibles de partout dans le monde. Ce projet s'appuie sur plusieurs de ces services pour construire une architecture fiable et évolutive.

5.1 Introduction aux services AWS

Avant de plonger dans le cœur du sujet, il est essentiel de comprendre les services AWS que nous allons utiliser et comment ils s'intègrent dans notre solution. Chaque service AWS joue un rôle spécifique dans l'architecture, collaborant avec les autres pour créer un système cohérent et robuste.

5.2 Amazon Virtual Private Cloud (Amazon VPC)

Amazon VPC nous permet de lancer des ressources AWS dans un réseau virtuel que nous avons défini. Ce réseau isolé et sécurisé est la base de notre architecture cloud, garantissant à la fois la sécurité et la séparation nette de notre environnement de développement et de production.

5.3 Amazon EC2 et Load Balancer

Les instances EC2 nous fournissent la capacité de calcul nécessaire pour exécuter notre application avec flexibilité et efficacité. L'Application Load Balancer répartit automatiquement le trafic entrant entre les instances EC2, assurant la haute disponibilité et la résilience de l'application.

5.4 AWS CodeCommit, CodeDeploy, et CodePipeline

AWS CodeCommit est utilisé comme un référentiel de code source versionné, sécurisé et intégré. AWS CodeDeploy automatise les déploiements d'applications, tandis que AWS CodePipeline orchestre les différentes étapes du processus d'intégration et de livraison continues, de la mise à jour du code à son déploiement en production.

5.5 Amazon ECS et ECR

Amazon ECS et ECR sont au cœur de notre architecture microservices. ECS gère et orchestre les conteneurs Docker, permettant un déploiement et une gestion aisés des microservices, tandis qu'ECR fournit un registre de conteneurs Docker, facilitant le stockage et la gestion des images de conteneurs.

5.6 AWS Cloud9

AWS Cloud9 est l'environnement de développement intégré dans le cloud qui permet aux développeurs de rédiger, exécuter et déboguer leur code avec juste un navigateur.

5.7 AWS IAM

AWS IAM joue un rôle essentiel dans la gestion des accès et des autorisations, assurant que seules les entités authentifiées et autorisées peuvent interagir avec les ressources AWS.

5.8 Amazon RDS et CloudWatch

Amazon RDS simplifie la mise en place, l'exploitation et l'échelle d'une base de données relationnelle dans le cloud, fournissant un stockage sécurisé et performant pour nos données. Amazon CloudWatch surveille notre application et les ressources AWS, collectant des métriques et des logs pour une visibilité complète sur la performance et la santé de l'application.

6 Phase 1 : Planification de la conception et estimation des coûts

6.1 Diagramme d'architecture

Après avoir introduit les services individuels, le diagramme d'architecture suivant (voir Figure 1) représente la manière dont ces services sont interconnectés pour former une solution complète et cohérente répondant aux exigences de notre application de fournisseurs de café.

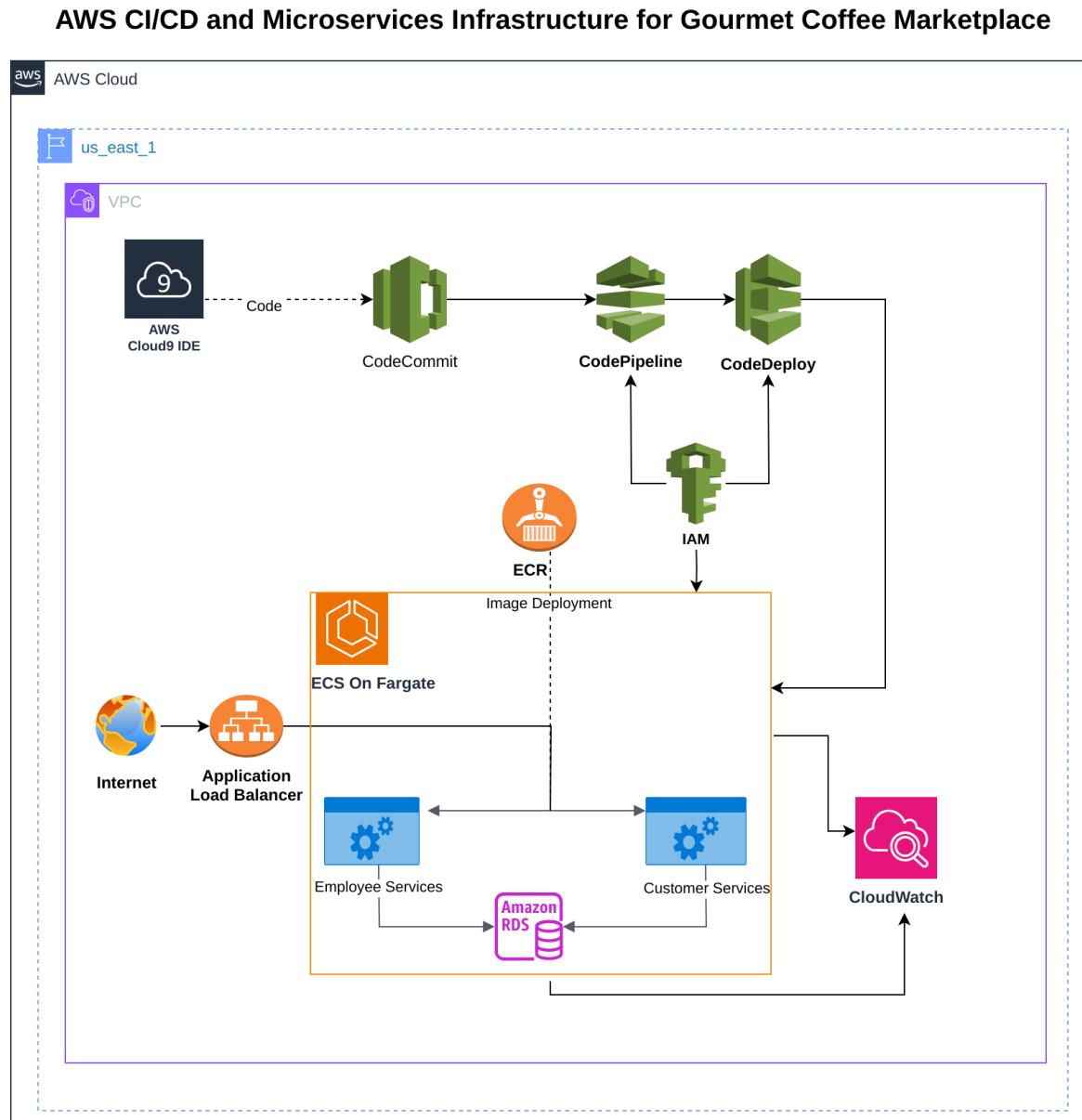


FIGURE 1 – Diagramme d’architecture des microservices et du pipeline CI/CD avec AWS

Le diagramme présente une orchestration fine entre les services AWS pour le développement, le déploiement, et la surveillance de notre application microservices. En voici un résumé :

- **Développement** : AWS Cloud9 sert de plateforme de développement, où le code est généré et versionné via CodeCommit.
- **Intégration Continue** : Les mises à jour du code déclenchent des pipelines dans CodePipeline, automatisant les tests et les builds.
- **Déploiement Continu** : CodeDeploy orchestre le déploiement des applications vers ECS, utilisant des images Docker stockées dans ECR.
- **Équilibrage de Charge** : L'Application Load Balancer distribue le trafic réseau de manière intelligente entre les microservices déployés.
- **Base de Données et Surveillance** : Les microservices interagissent avec RDS pour la persistance des données, tandis que CloudWatch surveille la performance et la santé du système.
- **Gestion des Accès** : IAM assure une gestion sécurisée des permissions à travers l'écosystème AWS.

Ce flux crée une chaîne d'intégration et de déploiement continue où chaque service renforce la robustesse et l'efficacité de l'infrastructure cloud.

6.2 Estimation des coûts

L'estimation des coûts pour le déploiement et l'exploitation de cette architecture sur une période de 12 mois dans la région *us-east-1* est présentée ci-dessous. Cette estimation est fondée sur les calculs effectués à l'aide de l'outil AWS Pricing Calculator et prend en compte les ressources nécessaires pour maintenir une performance optimale et une haute disponibilité.

Estimate summary		
Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	90.89 USD	1,090.68 USD
Includes upfront cost		

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	11.72 USD
Status: -				
Description:				
Config summary: Tenancy (Shared Instances), Operating system (Linux), Workload (Daily, (Workload days: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Baseline: 1, Peak: 2, Duration of peak: 8 Hr 30 Min)), Advance EC2 instance (t3a.small), Pricing strategy (Compute Savings Plans 3yr No Upfront), Enable monitoring (disabled), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)				
Elastic Load Balancing	No group applied	US East (N. Virginia)	0.00 USD	17.43 USD
Status: -				
Description:				
Config summary: Number of Application Load Balancers (1)				
AWS CodeDeploy	No group applied	US East (N. Virginia)	0.00 USD	2.00 USD
Status: -				
Description:				
Config summary: Number of on-premise instances (10), Number of deployments (10 per month)				

FIGURE 2 – Résumé de l'estimation des coûts pour l'architecture proposée.

AWS CodePipeline	No group applied	US East (N. Virginia)	0.00 USD	1.00 USD
Status: -				
Description:				
Config summary: Number of active pipelines used per account per month (2)				
AWS Fargate	No group applied	US East (N. Virginia)	0.00 USD	0.00 USD
Status: -				
Description:				
Config summary: Operating system (Linux), CPU Architecture (x86), Average duration (1 minutes), Number of tasks or pods (2 per day), Amount of ephemeral storage allocated for Amazon ECS (20 GB)				
Amazon RDS for MariaDB	No group applied	US East (N. Virginia)	0.00 USD	53.22 USD
Status: -				
Description:				
Config summary: Storage volume (General Purpose SSD (gp2)), Storage amount (20 GB), Quantity (1), Instance type (db.t3.micro), Utilization (On-Demand only) (100 %Utilized/Month), Deployment selection (Multi-AZ), Pricing strategy (OnDemand), Additional backup storage (20 GB)				
Amazon CloudWatch	No group applied	US East (N. Virginia)	0.00 USD	5.52 USD
Status: -				
Description:				
Config summary: Number of Metrics (includes detailed and custom metrics) (10), Standard Logs: Data Ingested (5 GB), Number of Dashboards (1)				

FIGURE 3 – Détails de l'estimation des coûts démontrant les coûts mensuels et totaux pour chaque service AWS utilisé.

Note : Les images ci-dessus illustrent le coût total sur 12 mois pour l'ensemble des services AWS nécessaires à notre architecture microservices. Elles montrent que l'utilisation d'Amazon EC2, Elastic Load Balancing et AWS CodeDeploy constitue une partie significative de l'investissement total. Ces services sont essentiels pour garantir que notre application soit évolutive, résiliente et qu'elle bénéficie d'une intégration et d'un déploiement continu efficaces.

7 Phase 2 : Analyse de l'infrastructure de l'application monolithique

Pour comprendre en détail comment l'application monolithique est configurée et exécutée, nous avons utilisé EC2 Instance Connect pour nous connecter à l'instance MonolithicAppServer. Cette étape nous permet d'analyser directement l'application en cours d'exécution.

7.1 Analyse des Ports et Processus

En utilisant la commande `sudo lsof -i :80`, j'ai identifié que le daemon Node.js est à l'écoute sur le port 80, qui est le port standard pour le trafic HTTP. Ce port est utilisé par défaut pour les applications web qui ne nécessitent pas de connexion chiffrée SSL/TLS.

```
*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/**/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-16-10-164:~$ sudo lsof -i :80
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
node    15103 root    18u  IPv6  51085      0t0  TCP *:http (LISTEN)
ubuntu@ip-10-16-10-164:~$ █
```

FIGURE 4 – Résultat de la commande `lsof` montrant que le daemon Node.js utilise le port 80.

Ensuite, la commande `ps -ef | head -1; ps -ef | grep node` nous a permis de confirmer que le processus Node.js est exécuté par l'utilisateur `ubuntu` et de repérer le PID du processus Node, qui correspond aux PIDs identifiés précédemment par `lsof`.

7.2 Structure de l'Application

En naviguant dans le répertoire `/resources/codebase_partner`, j'ai constaté la présence du fichier `index.js`, qui contient la logique principale de l'application (voir Figure

5). Cela indique que l'application utilise un environnement Node.js pour exécuter son serveur back-end.

```
Last login: Tue Apr 16 09:17:13 2024 from 18.206.107.27
ubuntu@ip-10-16-10-164:~$ sudo lsof -i :80
COMMAND PID USER FD      TYPE DEVICE SIZE/OFF NODE NAME
node    452 root  18u   IPv6  22479      0t0  TCP *:http (LISTEN)
ubuntu@ip-10-16-10-164:~$ ps -ef | head -1; ps -ef | grep node
UID      PID      PPID  C STIME TTY          TIME CMD
root     444      442  0 20:33 ?        00:00:00 sudo node index.js
root     452      444  0 20:33 ?        00:00:01 node index.js
ubuntu   1402     1382  0 21:39 pts/0    00:00:00 grep --color=auto node
ubuntu@ip-10-16-10-164:~$ cd ~/resources/codebase_partner
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ ls
app index.js node_modules package-lock.json package.json public views
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$
```

FIGURE 5 – Contenu du répertoire de codebase indiquant la présence du fichier index.js.

7.3 Réflexions

L'exécution de la commande `ls node_modules` et l'examen du fichier `package.json` révèlent les dépendances spécifiques utilisées par l'application, notamment `express` pour la structure de l'application web, `body-parser` pour l'analyse des requêtes, et `mysql` pour l'interaction avec les bases de données MySQL. De plus, le script de démarrage défini dans `package.json` suggère que l'application se connecte à une base de données via une variable d'environnement `APP_DB_HOST`, ce qui implique l'utilisation d'Amazon RDS pour la persistance des données.

```
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ ls node_modules
accepts           cors            express          ipaddr.js      mime-db       on-finished    safer-buffer  type-is
array-flatten    debug           express-validator  is-property  mime-types  parseurl      send         unpipe
async             dengue          finalhandler    lodash       ms           path-to-regexp seq-queue   utils-merge
body-parser      depd            forwarded      long         mustache   proxy-addr    serve-favicon  validator
bytes             destroy         fresh          lru-cache   mustache-express pseudomap    serve-static  vary
content-disposition ee-first      generate-function media-type mysql2       qs           setprototypeof yallist
content-type      encodeurl     http-errors    merge-descriptors named-placeholders range-parser  sqlstring
cookie            escape-html   iconv-lite    methods      negotiator  raw-body      statuses
cookie-signature etag            inherits      mime         object-assign  safe-buffer  toIdentifier
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ ls package.json
package.json
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ cd package.json
-bash: cd: package.json: Not a directory
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ cat package.json
{
  "name": "coffee_api",
  "version": "1.0.0",
  "description": "simple coffee partners API",
  "main": "index.js",
  "scripts": {
    "start": "APP_DB_HOST=SAPP_DB_HOST node index.js"
  },
  "author": "awt6c",
  "license": "MIT",
  "dependencies": {
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "express-validator": "^6.10.0",
    "mustache-express": "^1.3.0",
    "mysql2": "^2.2.5",
    "serve-favicon": "^2.5.0"
  }
}
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ ^C
```

FIGURE 6 – Capture d'écran montrant le contenu du fichier `package.json` confirmant les dépendances de l'application.

7.4 Connexion à la base de données RDS et observation des données

Dans cette section, nous avons établi une connexion au serveur de base de données RDS qui héberge les données de l'application Node.js. Le processus est décrit ci-dessous avec les résultats observés.

1. Accès au tableau de bord RDS depuis la console de gestion AWS pour récupérer le point de terminaison de la base de données.
2. Utilisation de l'outil `nmap` pour confirmer la connectivité depuis l'instance EC2 vers le port MySQL standard (3306) de la base de données RDS.
3. Connexion au serveur de base de données avec le client MySQL installé sur l'instance EC2 à l'aide des identifiants fournis (nom d'utilisateur : `admin` et mot de passe : `lab-password`).
4. Exécution des commandes SQL pour interagir avec la base de données. Il a été confirmé que la base de données `COFFEE` contient bien une table nommée `suppliers`, qui à son tour contient les entrées ajoutées lors des tests de l'application web.

Les résultats de ces interactions sont illustrés dans les images ci-dessous :

```
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ nmap -Pn supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com -p 3306
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-21 22:01 UTC
Nmap scan report for supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com (10.16.40.203)
Host is up (0.00066s latency).
rDNS record for 10.16.40.203: ip-10-16-40-203.ec2.internal

PORT      STATE SERVICE
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ mysql -u admin -p'lab-password' -h [RDS-endpoint]
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 2005 (HY000): Unknown MySQL server host '[RDS-endpoint]' (-2)
ubuntu@ip-10-16-10-164:~/resources/codebase_partner$ mysql -u admin -p'lab-password' -h supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1555
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE COFFEE;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_COFFEE |
+-----+
| suppliers        |
+-----+
```

FIGURE 7 – Résultats de la vérification de la connectivité de la base de données RDS.

```
mysql> SELECT * FROM suppliers;
+----+----+-----+-----+-----+-----+-----+
| id | name          | address           | city   | state  | email            | phone  |
+----+----+-----+-----+-----+-----+-----+
| 1  | Ayman Hakim  | Lot Talhaoui, Rue El Ouafae 2, Nr 55, Sidi Yahya, Oujda | Oriental | aymanhakim@hotmail.fr | 0700301809 |
+----+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> EXIT;
Bye
```

FIGURE 8 – Interactions avec la base de données RDS via le client MySQL.

Ces observations confirment que l'application stocke et gère ses données à l'aide du service Amazon RDS, qui est accessible depuis l'instance EC2, comme prévu dans l'architecture de l'application.

8 Phase 3 : Mise en place de l'environnement de développement AWS Cloud9

8.1 Création de l'IDE AWS Cloud9

J'ai configuré l'environnement de développement intégré, AWS Cloud9, comme suit :

- Nom de l'environnement : *MicroservicesIDE*.
- Type d'instance : t3.small exécutant Amazon Linux 2.
- Configuration réseau : exécuté dans *LabVPC* sur *Public Subnet1*, autorisant les connexions SSH.

Ce nouvel IDE servira de plateforme principale pour le développement de l'architecture microservices.

8.2 Copie du code de l'application vers l'IDE

La phase suivante a consisté à copier le code source de l'application monolithique existante vers notre environnement de développement AWS Cloud9 :

1. Téléchargement et configuration des permissions du fichier de clé SSH `labsuser.pem` pour permettre les connexions sécurisées.
2. Création d'un répertoire temporaire à l'emplacement `/home/ec2-user/environment/temp` sur l'instance Cloud9.
3. Utilisation de la commande `scp` pour transférer le code source de l'application depuis l'instance *MonolithicAppServer* vers le répertoire temporaire de l'instance Cloud9.

Le transfert a été vérifié et tous les fichiers source sont maintenant disponibles dans le répertoire `temp` de l'environnement AWS Cloud9. Après la commande suivante :

```
scp -r -i ~/environment/labsuser.pem ubuntu@44.192.120.164:/home/ubuntu/resources/codebase_partner/* ~/environment/temp/
```



```
voclabs:~/environment $ ls
labsuser.pem  microservices  README.md  temp
voclabs:~/environment $ ls
labsuser.pem  microservices  README.md  temp
voclabs:~/environment $
```

FIGURE 9 – Résultats du transfert des fichiers.

Avec l'environnement Cloud9 configuré et le code de l'application transféré, nous sommes bien positionnés pour commencer l'analyse et la refonte de l'application en services microservices indépendants.

8.3 Crédation de l'environnement de développement pour les microservices

Pour répondre aux besoins du projet, J'ai organisé l'application monolithique en deux microservices distincts, nommés `customer` et `employee`.

Dans l'environnement de développement AWS Cloud9, des répertoires séparés pour chaque microservice ont été créés :

```
— $ mkdir -p /environment/microservices/customer  
— $ mkdir -p /environment/microservices/employee
```

Le code source de l'application monolithique a été copié dans ces nouveaux répertoires, et le répertoire temporaire initial a été vidé et supprimé pour maintenir l'ordre dans l'environnement de travail :

L'état final de l'organisation des fichiers reflète la séparation des services, comme le montre l'image ci-dessous :

```
voclabs:~/environment $ ls  
labsuser.pem microservices README.md  
voclabs:~/environment $ cd microservices  
voclabs:~/environment/microservices $ ls  
customer employee  
voclabs:~/environment/microservices $  
voclabs:~/environment/microservices $ ls customer  
app index.js node_modules package.json package-lock.json public views  
voclabs:~/environment/microservices $ ls employee  
app index.js node_modules package.json package-lock.json public views  
voclabs:~/environment/microservices $ █
```

FIGURE 10 – Structure des répertoires pour les microservices `customer` et `employee`.

8.4 Mise en Place du Dépôt Git et Synchronisation avec AWS CodeCommit

La mise en place du dépôt Git dans l'environnement AWS Cloud9 et la synchronisation avec AWS CodeCommit se sont déroulées en plusieurs étapes clés, décrites ci-dessous et accompagnées de captures d'écran pour illustrer le processus.

1. Initialisation du répertoire `microservices` comme dépôt Git et configuration des informations de l'utilisateur Git.
2. Création de la branche `dev` et ajout de tous les fichiers au dépôt pour préparation de la première validation.
3. Validation des fichiers dans la branche `dev` avec un message descriptif des changements effectués.
4. Ajout du référentiel AWS CodeCommit en tant qu'origine distante et poussée des modifications de la branche `dev` vers le référentiel distant.

La séquence de commandes utilisée pour effectuer ces opérations est illustrée dans la capture d'écran suivante :

```
cd ~/environment/microservices
git init
git config --global user.name "Hakim"
git config --global user.email "aymanhakim@hotmail.fr"
git branch -m dev
git add .
git commit -m 'two unmodified copies of the application code'
```

FIGURE 11 – Commandes Git utilisées pour initialiser le dépôt .

```
vocabs:~/environment/microservices (dev)$ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Microservices
vocabs:~/environment/microservices (dev)$ git push -u origin dev
Enumerating objects: 2191, done.
Counting objects: 100% (2191/2191), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2083/2083), done.
Writing objects: 100% (2191/2191), 1.94 MiB | 5.09 MiB/s, done.
Total 2191 (delta 547), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Microservices
 * [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
vocabs:~/environment/microservices (dev)$
```

FIGURE 12 – Commandes Git utilisées pour pousser le code vers AWS CodeCommit.

Après avoir poussé le code, la confirmation que la branche `dev` a été correctement configurée dans AWS CodeCommit peut être observée dans la capture d'écran suivante, prise de la console CodeCommit :

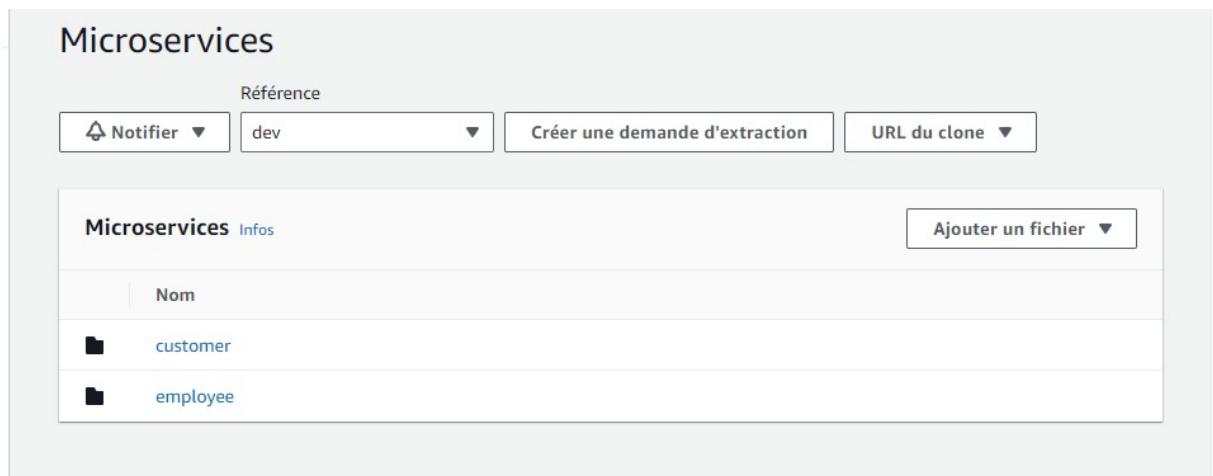


FIGURE 13 – Confirmation de la branche `dev` dans AWS CodeCommit.

Enfin, la présence du code dans le référentiel CodeCommit est vérifiée, confirmant la réussite de l'opération.

9 Phase 4 : Configuration de l'application en deux microservices et tests dans des conteneurs Docker

Cette phase implique la modification des deux copies du code source de l'application monolithique afin d'implémenter les fonctionnalités sous forme de deux microservices distincts. Pour les tests initiaux, ces microservices seront exécutés dans des conteneurs Docker sur la même instance EC2 qui héberge l'IDE AWS Cloud9 utilisé. Cet IDE sera utilisé pour construire les images Docker et lancer les conteneurs Docker.

9.1 Ajustement des paramètres du groupe de sécurité de l'instance AWS Cloud9

Afin de permettre l'accès aux conteneurs depuis un navigateur sur Internet, j'ai ajusté le groupe de sécurité de l'instance AWS Cloud9 pour autoriser le trafic réseau entrant sur les ports TCP 8080 et 8081.

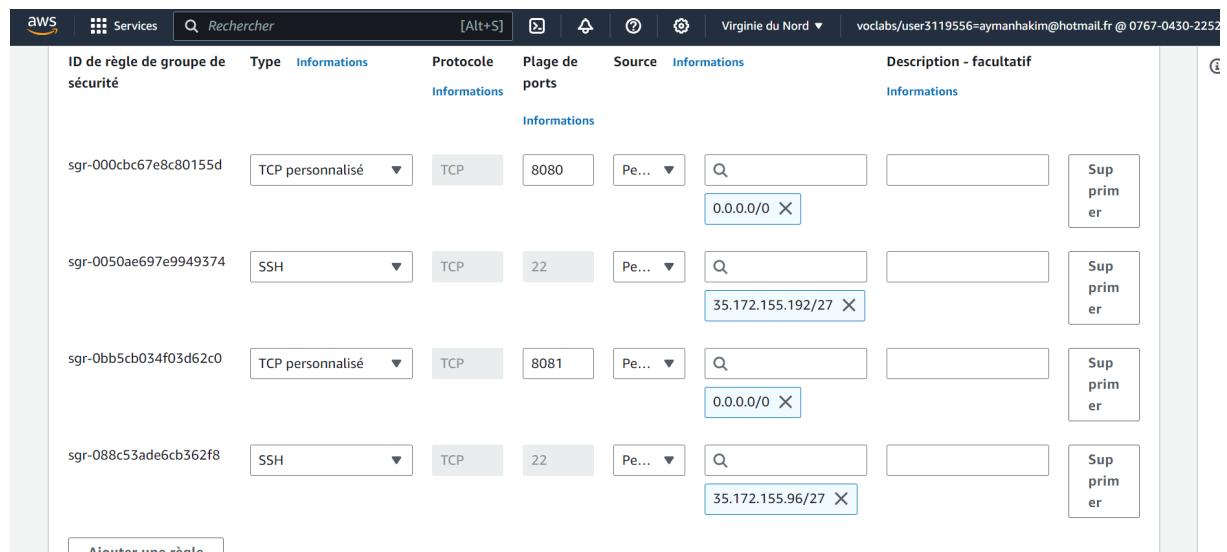


FIGURE 14 – Ajustements du groupe de sécurité pour l'instance AWS Cloud9.

9.2 Modification du code source du microservice customer

J'ai modifié le code source pour le microservice customer, en éliminant les fonctionnalités qui seront prises en charge par le microservice employee. Après les modifications, le code source ne contient que les lignes nécessaires pour les actions en lecture seule.

```
        database: dbConfig.APP_DB_NAME
    });

Supplier.getAll = result => {
    db_connection.query("SELECT * FROM suppliers", (err, res) => {
        if (err) {
            console.log("error: ", err);
            result(err, null);
            return;
        }
        console.log("suppliers: ", res);
        result(null, res);
    });
};

Supplier.findById = (supplierId, result) => {
    db_connection.query(`SELECT * FROM suppliers WHERE id = ${supplierId}`, (err, res) => {
        if (err) {
            console.log("error: ", err);
            result(err, null);
            return;
        }
        if (res.length) {
            console.log("found supplier: ", res[0]);
            result(null, res[0]);
            return;
        }
        result({kind: "not_found"}, null);
    });
};

module.exports = Supplier;
```

FIGURE 15 – Nouveau code du fichier `supplier.controller.js` après modifications.

```
app.set("view engine", "html")
app.set("views", __dirname + "/views")
app.use(express.static('public'));
app.use(favicon(__dirname + "/public/img/favicon.ico"));

// list all the suppliers
app.get("/", (req, res) => {
  res.render("home", {});
});

app.get("/suppliers/", supplier.findAll);
// show the add supplier form
//app.get("/supplier-add", (req, res) => {
//  res.render("supplier-add", {});
//});
// receive the add supplier POST
//app.post("/supplier-add", supplier.create);
// show the update form
//app.get("/supplier-update/:id", supplier.findOne);
// receive the update POST
//app.post("/supplier-update", supplier.update);
// receive the POST to delete a supplier
//app.post("/supplier-remove/:id", supplier.remove);
// handle 404
app.use(function (req, res, next) {
  res.status(404).render("404", {});
})

// set port, listen for requests
const app_port = process.env.APP_PORT || 8080
app.listen(app_port, () => {
  console.log(`Server is running on port ${app_port}.`);
});
"index.js" 48L, 1571B
```

FIGURE 16 – Modification du fichier `index.js` pour exécuter l'application sur le port 8080.

```
{${nav}}
<h1>All suppliers</h1>
<table class="table table-hover">
  <thead>
    <tr>
      <th scope="col">Name</th>
      <th scope="col">Address</th>
      <th scope="col">City</th>
      <th scope="col">State</th>
      <th scope="col">Email</th>
      <th scope="col">Phone</th>
      <th scope="col"></th>
    </tr>
  </thead>
  <tbody>
    {{#suppliers}}
    <tr>
      <th scope="row">{{name}}</th>
      <td>{{address}}</td>
      <td>{{city}}</td>
      <td>{{state}}</td>
      <td>{{email}}</td>
      <td>{{phone}}</td>
      <td><h4><a href="/supplier-update/{{id}}">Edit</a></h4></td>
    </tr>
    {{/suppliers}}
  </tbody>
</table>
<h4><a href="/supplier-add">Add a new supplier</a></h4>

</div>
${{#footer}}
-- VISUAL LINE --
```

FIGURE 17 – Suppression du bouton "Add a new supplier" dans le microservice customer.

9.3 Création du Dockerfile du microservice customer et lancement d'un conteneur de test

Un Dockerfile pour le microservice customer a été créé, et un conteneur Docker a été lancé pour exécuter le microservice sur le port 8080.

```

Step 2/7 : RUN mkrui -p /usr/src/app
--> Running in e1be3c9eef25
Removing intermediate container e1be3c9eef25
--> aa6c038dfc4a
Step 3/7 : WORKDIR /usr/src/app
--> Running in 400b65e32ddb
Removing intermediate container 400b65e32ddb
--> 5afbb0df2fea
Step 4/7 : COPY . .
--> 553459aa32e1
Step 5/7 : RUN npm install
--> Running in da66788c6280
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.717s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container da66788c6280
--> dbe86058c7c0
Step 6/7 : EXPOSE 8080
--> Running in e4e46b5f00db
Removing intermediate container e4e46b5f00db
--> 14a04db501d6
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in b1d61ad44a7c
Removing intermediate container b1d61ad44a7c
--> ab63ca56b788
Successfully built ab63ca56b788
Successfully tagged customer:latest
voclabs:~/environment/microservices/customer (dev) $ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
customer        latest        ab63ca56b788   6 minutes ago  82.7MB
node            11-alpine    f18da2f58c3d   4 years ago   75.5MB
voclabs:~/environment/microservices/customer (dev) $ 

```

FIGURE 18 – Liste des images Docker, incluant l'image customer créée.

```

voclabs:~/environment/microservices/customer (dev) $ docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint" customer
71128068659ef5d6d7c9bfb67d0b9190485f6f0f98d16c5c92ae6c2ad138ace1
voclabs:~/environment/microservices/customer (dev) $ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS        PORTS     NAMES
71128068659e   customer   "docker-entrypoint.s..."  5 seconds ago  Up 3 seconds  0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   customer_1
voclabs:~/environment/microservices/customer (dev) $ 

```

FIGURE 19 – Vérification des conteneurs Docker actuellement en exécution, montrant le microservice customer actif.

9.4 Vérification du fonctionnement du microservice customer

Le microservice customer a été chargé dans le navigateur en utilisant l'adresse IPv4 publique de l'instance AWS Cloud9 sur le port 8080, et le fonctionnement a été confirmé.



FIGURE 20 – Aperçu de l'application après modifications, affichant la liste des fournisseurs.

9.5 Commit et push des changements de code dans CodeCommit

Les modifications apportées au code source ont été enregistrées (commit) et envoyées (push) avec succès dans CodeCommit.

```
voclabs:~/environment/microservices/customer (dev) $ git add .
voclabs:~/environment/microservices/customer (dev) $ git commit -m "Update microservice for deployment with Docker"
[dev 8f2127b] Update microservice for deployment with Docker
 9 files changed, 46 insertions(+), 288 deletions(-)
  create mode 100644 customer/Dockerfile
  delete mode 100644 customer/views/supplier-add.html
  delete mode 100644 customer/views/supplier-form-fields.html
  delete mode 100644 customer/views/supplier-update.html
```

FIGURE 21 – Enregistrement des modifications dans le dépôt Git.

```
voclabs:~/environment/microservices/customer (dev) $ git push origin dev
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 2 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 1.40 KiB | 715.00 KiB/s, done.
Total 13 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Microservices
 642744e..8f2127b dev -> dev
```

FIGURE 22 – Envoi des changements à la branche dev dans AWS CodeCommit.

9.6 Observation des détails du commit dans CodeCommit

Les détails du commit ont été observés dans la console CodeCommit, révélant les lignes supprimées et ajoutées, ce qui permet de voir tous les détails de chaque changement de chaque fichier modifié.

9.7 Modification du code source du microservice employee

Dans cette tâche, j'ai apporté des modifications au code source du microservice employee, de manière similaire aux ajustements effectués pour le microservice customer. Les

clients (gérants de franchises de café) doivent avoir un accès en lecture seule aux données de l'application, tandis que les employés du siège social du café doivent pouvoir ajouter de nouvelles entrées ou modifier les entrées existantes dans la liste des fournisseurs de café.

Plus tard dans ce projet, les microservices seront déployés derrière un Application Load Balancer qui acheminera le trafic vers les microservices en fonction du chemin contenu dans l'URL de la requête. Si le chemin de l'URL inclut '/admin/', le trafic sera dirigé vers le microservice employee. Autrement, si le chemin de l'URL n'inclut pas '/admin/', le trafic sera dirigé vers le microservice customer.

En raison de la nécessité de router le trafic, une grande partie du travail dans cette tâche consiste à configurer le microservice employee pour ajouter '/admin/' au chemin des pages qu'il sert.

9.8 Ajustements du microservice employee

Dans l'IDE AWS Cloud9, après avoir réduit le répertoire customer et développé le répertoire employee, j'ai mis à jour le fichier `supplier.controller.js` pour que tous les appels de redirection commencent par '/admin/'.

Pour toutes les appels à `app.get` et `app.post` dans le fichier `index.js`, j'ai ajouté '/admin/' au premier paramètre et j'ai changé le numéro de port en 8081 pour éviter les conflits de port avec le microservice customer lors des tests.

Dans les fichiers `supplier-add.html` et `supplier-update.html`, j'ai modifié les chemins d'action des formulaires pour qu'ils débutent par '/admin/'. De même, j'ai ajusté les chemins HTML dans les fichiers `supplier-list-all.html` et `home.html`.

Enfin, dans le fichier `nav.html`, j'ai modifié le lien pour qu'il indique « Home » de l'administrateur, et j'ai ajouté un lien permettant aux employés de naviguer vers la partie customer de l'application web.

```
vocabs:~/environment/microservices/employee (dev) $ cd ~/environment/microservices/employee
vocabs:~/environment/microservices/employee (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
vocabs:~/environment/microservices/employee (dev) $ grep -n 'redirect' app/controller/supplier.controller.js
25:           else res.redirect('/suppliers');
86:           } else res.redirect('/suppliers');
103:       } else res.redirect('/suppliers');
vocabs:~/environment/microservices/employee (dev) $ grep -n 'app.get\|app.post' index.js
22:app.get("/", (req, res) => {
25:app.get("/suppliers/", supplier.findAll);
27:app.get("/supplier-add", (req, res) => {
31:app.post("/supplier-add", supplier.create);
33:app.get("/supplier-update/:id", supplier.findOne);
35:app.post("/supplier-update", supplier.update);
37:app.post("/supplier-remove/:id", supplier.remove);
vocabs:~/environment/microservices/employee (dev) $ grep -n 'action' views/*
views/supplier-add.html:11:    <form action="/supplier-add" method="POST">
views/supplier-update.html:12:    <form action="/supplier-update" method="POST">
views/supplier-update.html:38:                <form action="/supplier-remove/{id}" method="POST">
vocabs:~/environment/microservices/employee (dev) $ grep -n 'href' views/supplier-list-all.html views/home.html
views/supplier-list-all.html:27:          href="/supplier-update/{id}"/>edit</a></span></h4></td>
views/supplier-list-all.html:32:          <h4><a class="badge badge-success" href="/supplier-add">Add a new supplier</a></h4>
views/home.html:7:          <p><a href="/suppliers">List of suppliers</a></p>
vocabs:~/environment/microservices/employee (dev) $ docker ps
```

FIGURE 23 – Modification des fichiers.

9.9 Crédation du Dockerfile pour le microservice employee et lancement d'un conteneur de test

Dans cette tâche, j'ai créé une image Docker pour le microservice employee et lancé un conteneur de test à partir de cette image pour vérifier que le microservice fonctionne comme prévu.

9.10 Crédation d'une Image Docker

J'ai dupliqué le Dockerfile du microservice customer dans la zone du microservice employee et modifié le fichier `employee/Dockerfile` pour changer le numéro de port sur la ligne `EXPOSE` en 8081. Après avoir construit l'image Docker pour le microservice employee avec le tag `employee`, j'ai lancé un conteneur nommé `employee_1`.

```

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
71128068659e customer "docker-entrypoint.s..." 23 hours ago Up 44 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
vclabs:~/environment/microservices/employee (dev) $ docker build --tag employee .
Sending build context to Docker daemon 9.01MB
Step 1/7 : FROM node:11-alpine
--> f18da2f5bc3d
Step 2/7 : RUN mkdir -p /usr/src/app
--> Using cache
--> aa6c038dfc4a
Step 3/7 : WORKDIR /usr/src/app
--> Using cache
--> 5afbb0df2fea
Step 4/7 : COPY . .
--> e35085eed7c0
Step 5/7 : RUN npm install
--> Running in 96b39be3143b
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.68s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 96b39be3143b
--> c9083ff59298
Step 6/7 : EXPOSE 8080
--> Running in 8974321937b3
Removing intermediate container 8974321937b3
--> dae775c09356
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in c7b4dd5dbc44
Removing intermediate container c7b4dd5dbc44
--> 473b3a4a2ea5
Successfully built 473b3a4a2ea5
Successfully tagged employee:latest
vclabs:~/environment/microservices/employee (dev) $ 

```

FIGURE 24 – Construction de l'image docker

9.11 Vérification de Fonctionnement

```

npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.839s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 0dc45e9e4000
--> e9f903ec905b
Step 6/7 : EXPOSE 8081
--> Running in 45505b6735fb
Removing intermediate container 45505b6735fb
--> 66a7a4696cae
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in a13a5fb4be65
Removing intermediate container a13a5fb4be65
--> 3f9a9d8c48bc
Successfully built 3f9a9d8c48bc
Successfully tagged employee:latest
vclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep "APP_DB_HOST" | cut -d '"' -f2)
vclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com
vclabs:~/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
f47eb72f1a07280670167ed247e028e473e7aa21ac5491a66a61bac373e442e
vclabs:~/environment/microservices/employee (dev) $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f47eb72f1a07 employee "docker-entrypoint.s..." 5 minutes ago Up 5 minutes 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp employee_1
71128068659e customer "docker-entrypoint.s..." 23 hours ago Up 14 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp customer_1
vclabs:~/environment/microservices/employee (dev) $ 

```

FIGURE 25 – Lancement du conteneur.

Le microservice a été chargé dans un nouvel onglet de navigateur . J'ai vérifié que cette vue montre les boutons pour éditer les fournisseurs existants et ajouter un nouveau

fournisseur. J'ai également testé l'ajout, l'édition et la suppression de fournisseurs et confirmé que les modifications étaient reflétées sur la page des fournisseurs.

Name	Address	City	State	Email	Phone
Ayman Hakim	Lot Talhaoui, Rue El Ouafae 2, Nr 55, Sidi Yahya, Oujda	Oujda	Oriental	aymanhakim@hotmail.fr	0700301809
AymanV2	rabat inpt	Oujda	Oriental	aymanhakim@hotmail.fr	07 00 30 18 09

FIGURE 26 – Vérification du site .

9.12 Ajustement du Port et Reconstruction de l'Image

Lors des tests sur l'instance AWS Cloud9, le microservice employee a été exécuté sur le port 8081. Cependant, pour le déploiement sur Amazon ECS, il est souhaité qu'il s'exécute sur le port 8080. Pour cela, j'ai modifié les fichiers `employee/index.js` et `employee/Dockerfile` et reconstruit l'image Docker pour le microservice employee.

9.13 Intégration du Code dans CodeCommit

Enfin, j'ai examiné les mises à jour apportées au code source et engagé mes changements dans CodeCommit. Ce processus comprenait la revue des fichiers modifiés, tels que `index.js`, en utilisant l'interface de contrôle de source Git dans AWS Cloud9, et en soulignant les modifications avant de procéder au commit et au push dans le terminal.

```
voclabs:~/environment/microservices/employee (dev) $ git add .
voclabs:~/environment/microservices/employee (dev) $ git commit -m "Update employee microservice configurations"
[dev 4b202ad] Update employee microservice configurations
 9 files changed, 29 insertions(+), 20 deletions(-)
  create mode 100644 employee/Dockerfile
voclabs:~/environment/microservices/employee (dev) $ git push origin dev
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 2 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 1.63 KiB | 1.63 MiB/s, done.
Total 15 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Microservices
 8f2127b..4b202ad dev -> dev
voclabs:~/environment/microservices/employee (dev) $
```

FIGURE 27 – Le commit et le push dans dev après la vérification .

10 Phase 5 : Création de dépôts ECR, un cluster ECS, des définitions de tâches et des fichiers AppSpec

Cette phase consiste à mettre en place une architecture de déploiement évolutive pour les microservices en utilisant Amazon ECS et ECR. Cela permettra d'ajuster dynamiquement le nombre de conteneurs en fonction des besoins et de gérer efficacement le routage du trafic via un équilibrEUR de charge.

10.1 CrÉation des dépôts ECR et tÉlÉversement des images Docker

Pour débuter, j'ai configuré l'authentification de mon client Docker avec Amazon ECR pour pouvoir téléverser les images Docker des microservices.

Un message dans la sortie de la commande confirme que la connexion a réussi. J'ai ensuite créé un dépôt ECR privé pour chacun des microservices, nommés `customer` et `employee`, et configuré les permissions nécessaires pour ces dépôts.

Les images Docker ont été marquées avec l'ID de registre unique pour faciliter leur gestion :

```
voclabs:~/environment/microservices (dev) $ account_id=$(aws sts get-caller-identity | grep Account|cut -d '"' -f4)
voclabs:~/environment/microservices (dev) $ echo $account_id
076704302252
voclabs:~/environment/microservices (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
voclabs:~/environment/microservices (dev) $ aws ecr create-repository --repository-name customer --region us-east-1
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:076704302252:repository/customer",
    "registryId": "076704302252",
    "repositoryName": "customer",
    "repositoryUrl": "076704302252.dkr.ecr.us-east-1.amazonaws.com/customer",
    "createdAt": "2024-04-29T20:19:27.525000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
voclabs:~/environment/microservices (dev) $ aws ecr create-repository --repository-name employee --region us-east-1
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:076704302252:repository/employee",
    "registryId": "076704302252",
    "repositoryName": "employee",
    "repositoryUrl": "076704302252.dkr.ecr.us-east-1.amazonaws.com/employee",
    "createdAt": "2024-04-29T20:19:28.593000+00:00",
  }
}
```

FIGURE 28 – Authentification du client docker et Creation du dépôt ECR

Après avoir vérifié l'existence des images et l'application des tags avec la commande `docker images`, j'ai poussé chaque image dans les dépôts ECR correspondants.

Chaque image est maintenant stockée dans Amazon ECR avec l'étiquette `latest` appliquée.

10 Phase 5 : Création de dépôts ECR, un cluster ECS, des définitions de tâches et des fichiers AppSpec

```
vclabs:~/environment/microservices (dev) $ docker tag customer:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
vclabs:~/environment/microservices (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vclabs:~/environment/microservices (dev) $ docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
076704302252.dkr.ecr.us-east-1.amazonaws.com/employee    latest    473b3a4a2ea5  58 minutes ago  82.7MB
employee            latest    473b3a4a2ea5  58 minutes ago  82.7MB
<none>              <none>   3f9a908c48bc  2 hours ago   82.7MB
076704302252.dkr.ecr.us-east-1.amazonaws.com/customer    latest    f97df85bd600  24 hours ago  82.7MB
customer            latest    f97df85bd600  24 hours ago  82.7MB
node                11-alpine  f18da2af98c3d  4 years ago   75.5MB
vclabs:~/environment/microservices (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
The push refers to repository [076704302252.dkr.ecr.us-east-1.amazonaws.com/customer]
a45472d113d: Pushed
c7ffb9a27357: Pushed
233fb8025230: Pushed
d810715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb779824: Pushed
fib993fe4ab5: Pushed
latest: digest: sha256:e56ff15c674423600aa98244b9580f29981b5dd9ff8d8e2d29db9462a05122f5 size: 1783
vclabs:~/environment/microservices (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [076704302252.dkr.ecr.us-east-1.amazonaws.com/employee]
d642ec0dbe02: Pushed
97699c030c94: Pushed
233fb8025230: Pushed
d810715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb779824: Pushed
fib993fe4ab5: Pushed
latest: digest: sha256:8c26f43fd71373dc3f6408b87dce58ac403716ca27b702764da5b23975e8e447 size: 1783
vclabs:~/environment/microservices (dev) $
```

FIGURE 29 – Résultat de la commande pour pousser les images ECR.

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
customer	076704302252.dkr.ecr.us-east-1.amazonaws.com/customer	29 avril 2024, 21:19:27 (UTC+01)	Disabled	Manual	AES-256
employee	076704302252.dkr.ecr.us-east-1.amazonaws.com/employee	29 avril 2024, 21:19:28 (UTC+01)	Disabled	Manual	AES-256

FIGURE 30 – Vérification des images dans Aws ECR

10.2 Crédation d'un Cluster ECS

Dans cette tâche, j'ai créé un cluster Amazon ECS utilisant AWS Fargate, sans dépendre des instances EC2 ou de ECS Anywhere, nommé `microservices-serverlesscluster`.

Cluster	Services	Tasks	Container instances	Cloud
microservices-serverlesscluster	0	No tasks running	0 EC2	Def

FIGURE 31 – Crédation du cluster ECS dans la console AWS.

10.3 Crédit d'un Répertoire CodeCommit pour les Fichiers de Déploiement

J'ai également créé un nouveau dépôt CodeCommit nommé `deployment` pour stocker les fichiers de configuration de déploiement des microservices, y compris les fichiers de spécification des tâches et les fichiers AppSpec pour CodeDeploy.

```
voclabs:~/environment $ ls
labsuser.pem  microservices  README.md
voclabs:~/environment $ mkdir deployment
voclabs:~/environment $ cd deployment
voclabs:~/environment/deployment $ git init -b dev
Initialized empty Git repository in /home/ec2-user/environment/deployment/.git/
voclabs:~/environment/deployment (dev) $ █
```

FIGURE 32 – Crédit du répertoire Deployment.

Dans l'environnement AWS Cloud9, j'ai initialisé ce dépôt comme un dépôt Git avec une branche nommée `dev`.

10.4 Crédit et Enregistrement des Définitions de Tâches

J'ai créé des fichiers de définition de tâches pour chaque microservice dans le nouveau répertoire `deployment`. Ces fichiers, nommés `taskdef-customer.json` et `taskdef-employee.json`, contiennent des spécifications pour déployer les microservices dans le cluster ECS via Fargate.

```
1  {
2      "containerDefinitions": [
3          {
4              "name": "employee",
5              "image": "employee",
6              "environment": [
7                  {
8                      "name": "APP_DB_HOST",
9                      "value": "supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com"
10                 }
11            ],
12            "essential": true,
13            "portMappings": [
14                {
15                    "hostPort": 8080,
16                    "protocol": "tcp",
17                    "containerPort": 8080
18                }
19            ],
20            "logConfiguration": {
21                "logDriver": "awslogs",
22                "options": {
23                    "awslogs-create-group": "true",
24                    "awslogs-group": "awslogs-capstone",
25                    "awslogs-region": "us-east-1",
26                    "awslogs-stream-prefix": "awslogs-capstone"
27                }
28            }
29        ]
30    }
```

FIGURE 33 – Configuration des fichiers de définition de tâches.

Ces fichiers spécifient les paramètres nécessaires pour que les microservices fonctionnent correctement sous la gestion d'AWS Fargate, y compris les configurations des ports, des variables d'environnement pour les bases de données, et des logs.

J'ai enregistré ces définitions de tâches dans Amazon ECS, et j'ai vérifié que les définitions de tâches apparaissaient correctement dans le panneau des définitions de tâches de la console ECS.

```
voclabs:~/environment/deployment (dev) $ aws ecs register-task-definition --cli-input-json "file:///home/ec2-user/environment/deployment/taskdefinition/customer.json"
{
  "taskDefinition": {
    "name": "customer",
    "image": "customer",
    "cpu": 0,
    "portMappings": [
      {
        "containerPort": 8080,
        "hostPort": 8080,
        "protocol": "tcp"
      }
    ],
    "essential": true,
    "environment": [
      {
        "name": "APP_DB_HOST",
        "value": "supplierdb.cmrfb4udjw0e.us-east-1.rds.amazonaws.com"
      }
    ]
  }
}
```

FIGURE 34 – Enregistrement des définitions de tâches dans ECS.

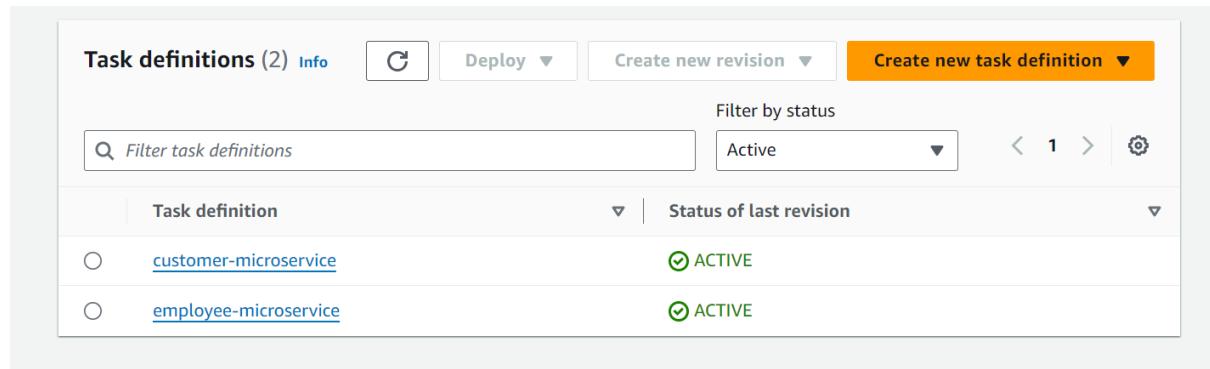


FIGURE 35 – Vérification des définitions de tâches dans ECS.

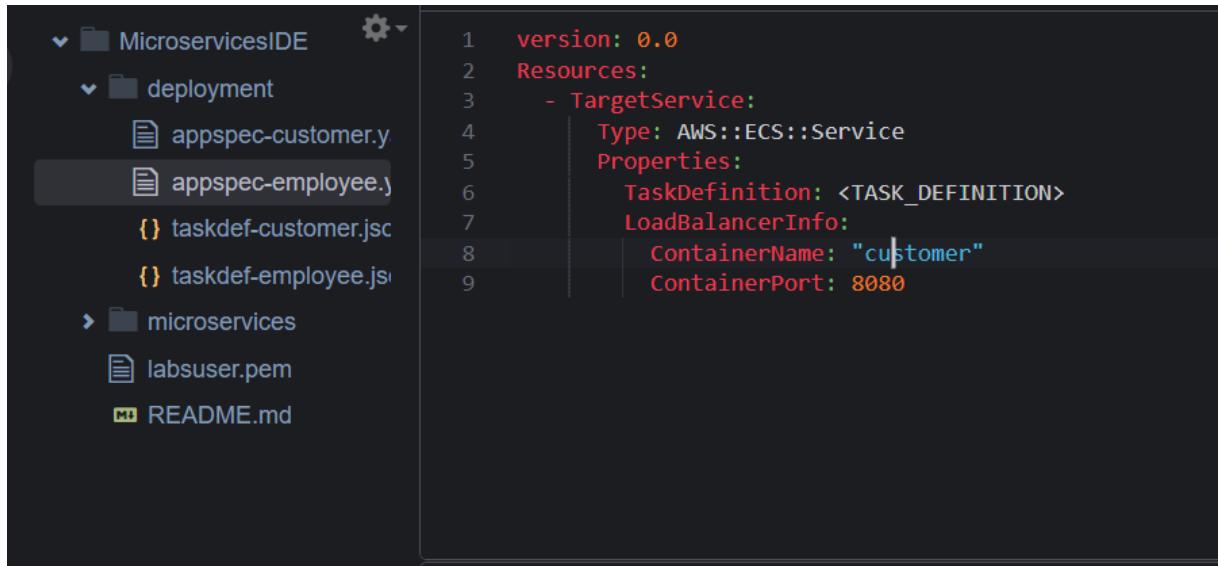
10.5 Crédation des fichiers AppSpec pour chaque microservice

Dans cette tâche, j'ai créé des fichiers AppSpec, qui fournissent les instructions nécessaires à AWS CodeDeploy pour déployer les microservices sur l'infrastructure Amazon ECS utilisant AWS Fargate.

10.5.1 Fichier AppSpec pour le microservice customer

J'ai créé un fichier nommé `appspec-customer.yaml` dans le répertoire de déploiement. Ce fichier contient des instructions spécifiques pour le service ECS, y compris les informations de configuration du load balancer.

Ce fichier YAML, sensible à l'indentation, configure le service ECS pour utiliser la définition de tâche spécifiée et les informations du load balancer pour le microservice customer.



The screenshot shows the AWS Lambda & Serverless IDE interface. On the left, there's a file tree with the following structure:

- MicroservicesIDE
- deployment
- appspec-customer.yaml
- appspec-employee.yaml
- taskdef-customer.json
- taskdef-employee.json
- microservices
- labsuser.pem
- README.md

The right pane displays the content of the `appspec-customer.yaml` file:

```
1 version: 0.0
2 Resources:
3   - TargetService:
4     Type: AWS::ECS::Service
5     Properties:
6       TaskDefinition: <TASK_DEFINITION>
7       LoadBalancerInfo:
8         ContainerName: "customer"
9         ContainerPort: 8080
```

FIGURE 36 – Le fichier Customer.Yaml .

10.5.2 Fichier AppSpec pour le microservice employee

De manière similaire, j'ai créé `appspec-employee.yaml` pour le microservice `employee`, en ajustant le nom du conteneur pour correspondre à ce service.

10.6 Mise à jour et Validation des Fichiers de Définition de Tâches

Pour aligner les fichiers de définition de tâches avec les besoins actuels, j'ai modifié les fichiers `taskdef-customer.json` et `taskdef-employee.json`, en changeant le nom de l'image en un placeholder `<IMAGE1_NAME>`, qui sera mis à jour dynamiquement lors de l'exécution par CodePipeline.

```
"image": "<IMAGE1_NAME>" ,
```

Ces changements permettent à la pipeline CI/CD de mettre à jour dynamiquement le nom de l'image au moment de l'exécution, assurant ainsi que les microservices utilisent la version la plus récente du code lors du déploiement.

10.7 Intégration des Changements dans CodeCommit

Les quatre fichiers mis à jour ont été commit et poussés dans le dépôt CodeCommit pour assurer que la pipeline CI/CD puisse les récupérer et utiliser ces informations pour déployer les mises à jour des microservices sur le cluster ECS.

```

vocabs:~/environment $ cd deployment
vocabs:~/environment $ cd deployment
vocabs:~/environment/deployment (dev) $ git add .
vocabs:~/environment/deployment (dev) $ git commit -m "Initial commit with task definitions and AppSpec files"
[ dev (root-commit) cc1cccd] Initial commit with task definitions and AppSpec files
 4 files changed, 96 insertions(+)
create mode 100644 appspec-customer.yaml
create mode 100644 appspec-employee.yaml
create mode 100644 taskdef-customer.json
create mode 100644 taskdef-employee.json
vocabs:~/environment/deployment (dev) $ git push origin dev
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
vocabs:~/environment/deployment (dev) $ git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
vocabs:~/environment/deployment (dev) $ git push origin dev
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.06 KiB | 1.06 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/deployment
 * [new branch]      dev -> dev
vocabs:~/environment/deployment (dev) $ 

```

FIGURE 37 – Commit et push des changements dans CodeCommit.

Cette étape assure que tous les composants nécessaires sont en place pour que la pipeline CI/CD fonctionne efficacement, en tirant les configurations à partir de CodeCommit pour déployer les microservices de manière dynamique et gérée sur ECS.

11 Phase 6 : Création de groupes cibles et d'un Application Load Balancer

Cette phase implique la création d'un Application Load Balancer (ALB) qui agira comme point d'entrée HTTPS pour que les clients et les employés accèdent à votre application via un navigateur web. L'ALB utilisera des auditores qui auront des règles de routage et d'accès pour déterminer vers quel groupe cible de conteneurs en cours d'exécution la requête de l'utilisateur doit être dirigée.

11.1 Crédation de quatre groupes cibles

Pour supporter une stratégie de déploiement bleu/vert, j'ai créé deux groupes cibles pour chaque microservice, ce qui nécessite que CodeDeploy utilise deux groupes cibles pour chaque groupe de déploiement.

11.1.1 Groupes cibles pour le microservice customer

Dans la console Amazon EC2, sous l'onglet Groupes cibles, j'ai créé deux groupes cibles :

- Le premier, nommé **customer-tg-one**, configuré pour le protocole HTTP sur le port 8080 et un chemin de vérification de santé sur /.
- Le deuxième, nommé **customer-tg-two**, utilisant les mêmes configurations que le premier mais avec un nom différent pour permettre la séparation des environnements bleu et vert.

11.1.2 Groupes cibles pour le microservice employee

De manière similaire, j'ai configuré deux autres groupes cibles pour le microservice employee, avec des chemins de vérification de santé spécifiques à leurs fonctions administratives :

- `employee-tg-one` et `employee-tg-two`, tous deux configurés pour surveiller le chemin `/admin/suppliers`.

Ces configurations sont illustrées dans l'image suivante, qui montre la liste des quatre groupes cibles et leurs numéros de port.

Target groups (4) Info					
<input type="checkbox"/>	Name	ARN	Port	Protocol	Target type
<input type="checkbox"/>	employee-tg-two	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/employee-tg-two/5678901234567890	8080	HTTP	IP
<input type="checkbox"/>	employee-tg-one	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/employee-tg-one/5678901234567890	8080	HTTP	IP
<input type="checkbox"/>	customer-tg-two	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/customer-tg-two/5678901234567890	8080	HTTP	IP
<input type="checkbox"/>	customer-tg-one	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/customer-tg-one/5678901234567890	8080	HTTP	IP

FIGURE 38 – Liste des groupes cibles créés pour les microservices.

11.2 Crédit d'un Load Balancer et Configuration des Règles de Trafic

J'ai également créé un Application Load Balancer nommé `microservicesLB`, configuré pour être accessible via Internet et utiliser les sous-réseaux publics ainsi que le groupe de sécurité `microservices-sg`.

Summary			
Review and confirm your configurations. Estimate cost			
Basic configuration Edit microservicesLB • Internet-facing • IPv4	Security groups Edit • microservices-sg sg-05654c80eb5af80a6	Network mapping Edit VPC vpc-07d0079f426b4e9f6 LabVPC • us-east-1a subnet-0f3dcba570ee1efab Public Subnet1 • us-east-1b subnet-0ddc98b30675ad3a6 Public Subnet2	Listeners and routing Edit • HTTP:80 defaults to customer-tg-two • HTTP:8080 defaults to customer-tg-one
Service integrations Edit AWS WAF: None AWS Global Accelerator: None	Tags Edit None		

FIGURE 39 – l'ALB qui a été créé.

11.2.1 Configuration des Auditeurs

Deux auditeurs ont été configurés sur l'ALB :

- Un auditeur sur le port HTTP 80, configuré pour rediriger le trafic par défaut vers `customer-tg-two`.
- Un auditeur sur le port HTTP 8080, configuré pour rediriger le trafic par défaut vers `customer-tg-one`.

Pour chacun des auditeurs, une règle supplémentaire a été ajoutée pour diriger le trafic vers les groupes cibles des employés si le chemin URL contient `/admin/*`.

Protocol:Port	Default action	Rules	ARN	Security
HTTP:80	Forward to target group <ul style="list-style-type: none"> customer-tg-two: 1 (100%) Group-level stickiness: Off 	2 rules	ARN	Not applied
HTTP:8080	Forward to target group <ul style="list-style-type: none"> customer-tg-one: 1 (100%) Group-level stickiness: Off 	2 rules	ARN	Not applied

FIGURE 40 – Règles configurées pour les auditeurs de l'ALB.

Cette configuration permet de s'assurer que le trafic est correctement acheminé vers le microservice approprié en fonction du chemin de l'URL, facilitant ainsi la gestion des différentes fonctions de l'application au sein de l'environnement ECS.

12 Phase 7 : Création de deux services Amazon ECS

Dans cette phase, ne vais créer un service Amazon ECS pour chaque microservice. car pour ce projet, il sera plus facile de gérer les microservices indépendamment si chacun est déployé sur son propre service ECS.

12.1 Crédit à la création du service ECS pour le microservice customer

Pour le microservice customer, j'ai créé un nouveau fichier JSON dans le répertoire de déploiement sur AWS Cloud9, nommé `create-customer-microservice-tg-two.json`, et j'ai inséré la configuration nécessaire pour le service ECS.

J'ai adapté le fichier JSON pour correspondre aux configurations actuelles de l'infrastructure, y compris les ARN du groupe cible, les ID des sous-réseaux, et l'ID du groupe de sécurité. Après avoir sauvegardé les modifications, j'ai exécuté la commande AWS CLI pour créer le service ECS pour le microservice customer.

```

1 {
2     "taskDefinition": "customer-microservice:1",
3     "cluster": "microservices-serverlesscluster",
4     "loadBalancers": [
5         {
6             "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:076704302252:targetgroup/customer-tg-two/11f8dd7b2cab808",
7             "containerName": "customer",
8             "containerPort": 8080
9         }
10    ],
11    "desiredCount": 1,
12    "launchType": "FARGATE",
13    "schedulingStrategy": "REPLICA",
14    "deploymentController": {
15        "type": "CODE_DEPLOY"
16    },
17    "networkConfiguration": {
18        "awsvpcConfiguration": {
19            "subnets": [
20                "subnet-0f3cba570ee1efab",
21                "subnet-0ddc98b30675ad3a6"
22            ],
23            "securityGroups": [
24                "sg-05654c80eb5af80a6"
25            ],
26            "assignPublicIp": "ENABLED"
27        }
28    }
29 }

```

FIGURE 41 – Le fichier de configuration.

12.2 Crédation du service ECS pour le microservice employee

De manière similaire, j'ai créé un service ECS pour le microservice employee en copiant et en modifiant le fichier JSON utilisé pour le service customer. J'ai renommé ce fichier en `create-employee-microservice-tg-two.json` et apporté les modifications nécessaires.

Après avoir mis à jour le fichier pour le microservice employee, j'ai exécuté la commande AWS CLI appropriée pour créer le service dans Amazon ECS. La vérification dans la console Amazon ECS a confirmé la création des services et la configuration correcte des révisions de définition des tâches.

```

bash - "ip-10-16-10-4.ec2 x" + 
voclabs:~/environment $ cd ~/environment/deployment
voclabs:~/environment/deployment (dev) $ aws ecs create-service --service-name customer-microservice --cli-input-json file://./create-employee-microservice-tg-two.json
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:076704302252:service/microservices-serverlesscluster/customer-microservice",
        "serviceName": "customer-microservice",
        "clusterArn": "arn:aws:ecs:us-east-1:076704302252:cluster/microservices-serverlesscluster",
        "loadBalancers": [
            {
                "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:076704302252:targetgroup/customer-tg-two/11f8dd7b2cab808",
                "containerName": "customer",
                "containerPort": 8080
            }
        ]
    }
}

```

FIGURE 42 – La commande AWS CLI pour créer le service dans ECS.

13 Phase 8 : Configuration de CodeDeploy et CodePipeline

Après avoir défini l'Application Load Balancer, les groupes cibles et les services Amazon ECS qui constituent l'infrastructure pour le déploiement de nos microservices, l'étape suivante consiste à définir le pipeline CI/CD pour déployer l'application.

13.1 Création d'une application CodeDeploy et de groupes de déploiement

J'ai utilisé la console CodeDeploy pour créer une application CodeDeploy nommée **microservices**, utilisant Amazon ECS comme plateforme de calcul.

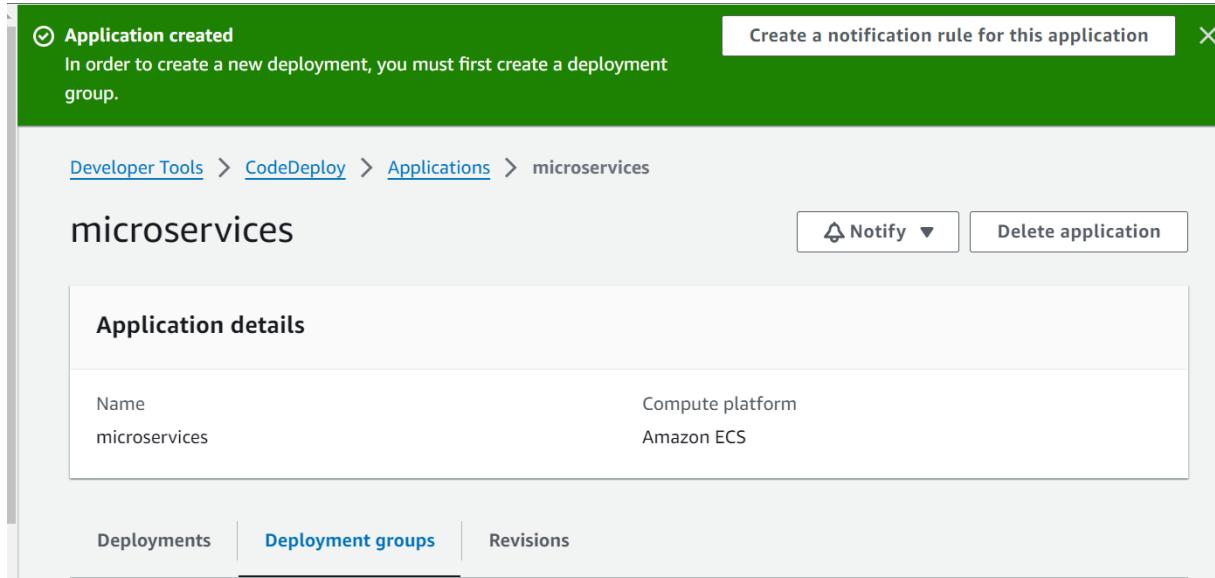
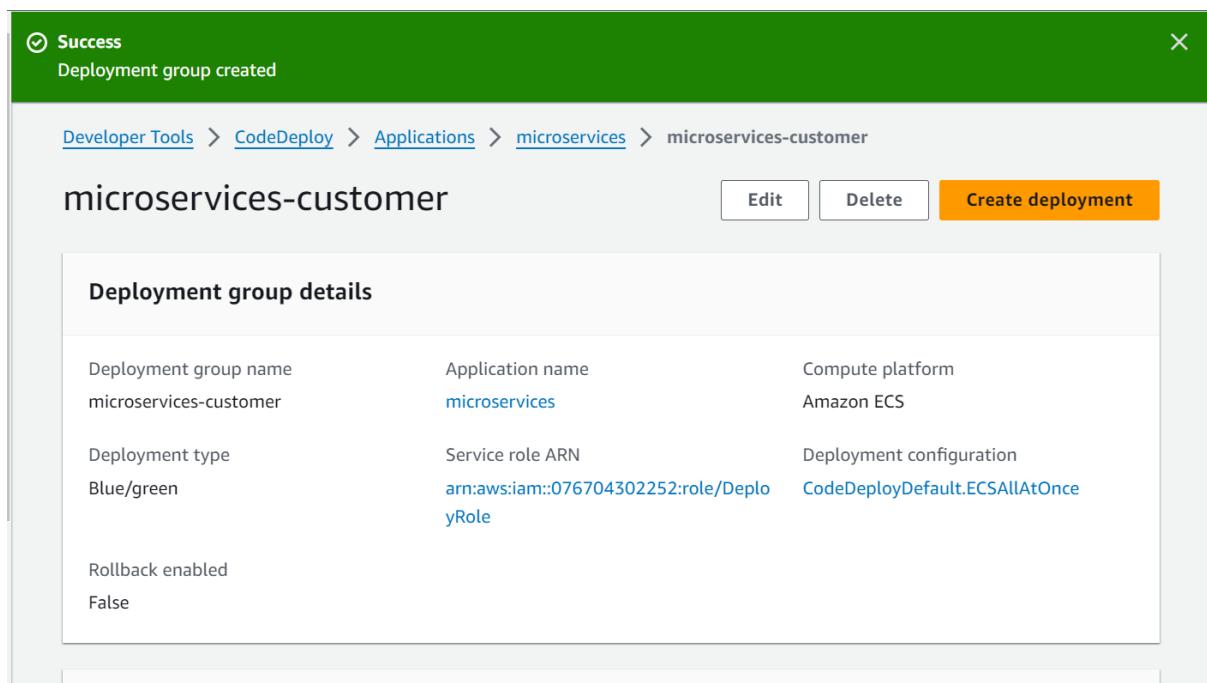


FIGURE 43 – Création de l'application CodeDeploy **microservices**.

- Pour le microservice customer, j'ai créé un groupe de déploiement nommé **microservices-customer**, spécifiant un service Amazon ECS, un load balancer, un port d'écoute de production sur HTTP :80, et deux groupes cibles.
- Pour le microservice employee, j'ai créé un groupe de déploiement avec des paramètres ajustés pour ce service.

FIGURE 44 – Crédit de la figure : [AWS Documentation](#).

13.2 Crédit de la figure : [AWS Documentation](#)

Dans la console CodePipeline, j'ai configuré un pipeline avec les paramètres suivants :

- Nom du pipeline : update-customer-microservice.
- Fournisseur de source : AWS CodeCommit avec le dépôt deployment et la branche dev.
- Fournisseur de déploiement : Amazon ECS (Blue/Green) avec des configurations spécifiques pour la gestion des définitions de tâches et des fichiers AppSpec.

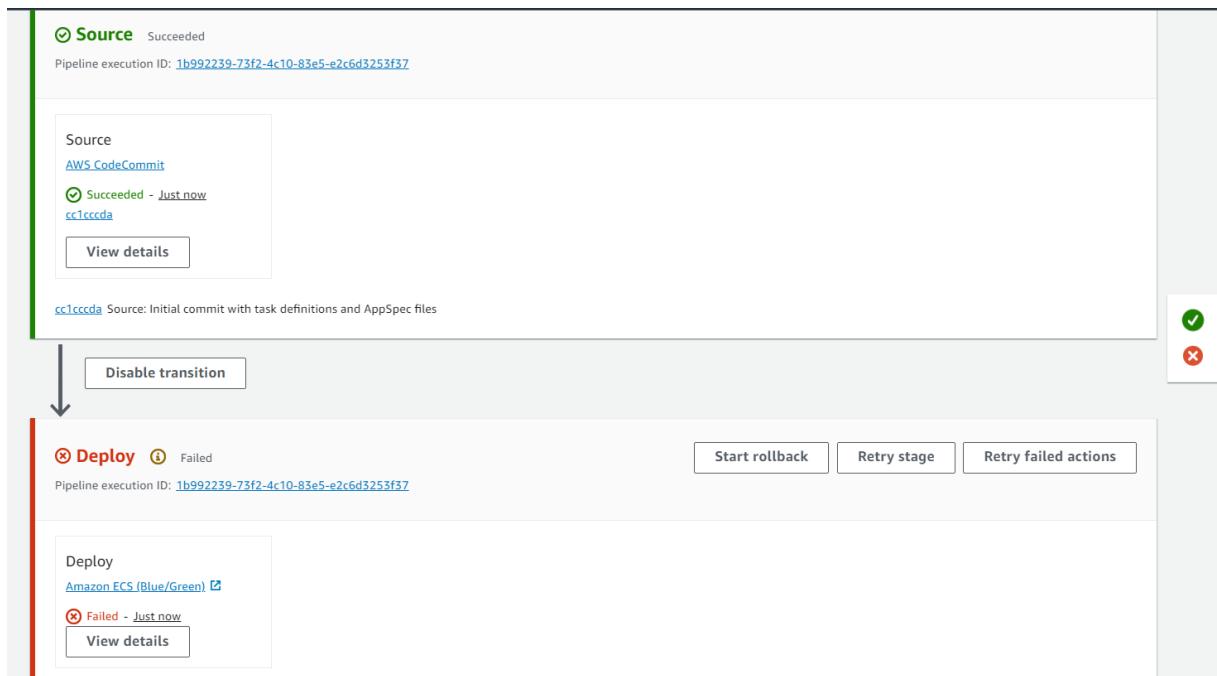


FIGURE 45 – Création du pipeline avec intégration des sources ECR.

13.3 Test du pipeline CI/CD pour le microservice customer

Après avoir configuré le pipeline initial, j'ai procédé à une mise à jour pour inclure Amazon ECR comme une nouvelle source pour les images Docker, ce qui est crucial pour les déploiements dynamiques des mises à jour des microservices.

The screenshot shows the "Edit action" dialog box for a new action being added to the pipeline. The action is currently titled "Image".

Action name: "Image".
Action provider: "Amazon ECR".
Repository name: "Q customer".
Image tag - optional: "latest".
Variable namespace - optional: An empty input field.
Output artifacts: "image-customer".

At the bottom right, there are "Cancel" and "Done" buttons.

FIGURE 46 – Ajout d'une nouvelle source d'image Docker dans le pipeline.

Dans la section **Edit: Source** de la console CodePipeline, j'ai ajouté une action pour récupérer l'image Docker la plus récente du dépôt Amazon ECR ;, j'ai lancé un déploiement pour tester les réglages actuels du pipeline et observé le processus dans CodeDeploy.

Le microservice customer a été testé en utilisant le nom DNS du load balancer microservicesLB.

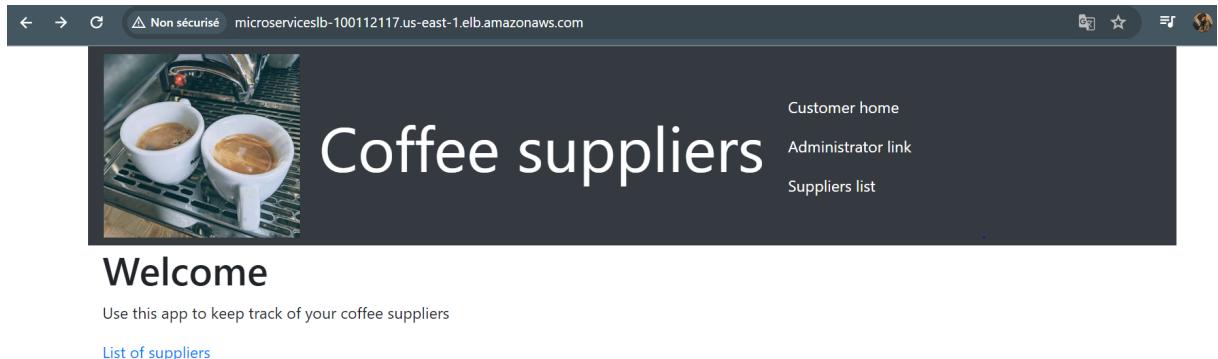


FIGURE 47 – Le microservice customer fonctionnant via le DNS du load balancer.

Ce déploiement réussi marque une étape significative dans l'adoption des pratiques de développement moderne et de l'automatisation pour la gestion des applications cloud à grande échelle.

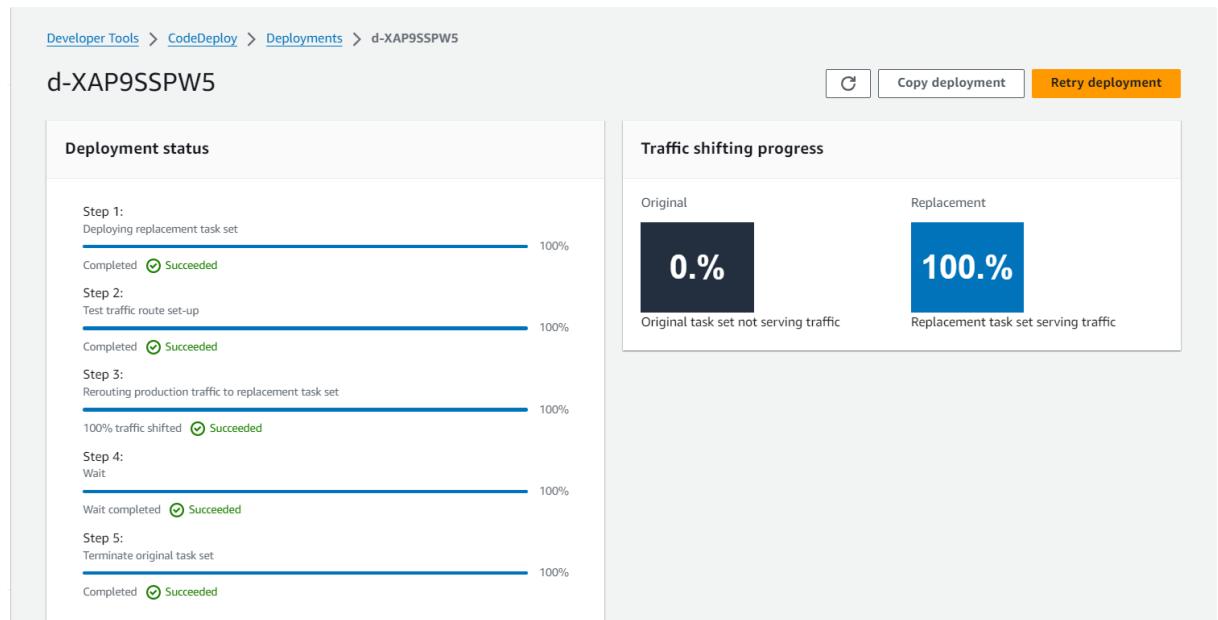


FIGURE 48 – Confirmation du succès du déploiement dans CodeDeploy.

13.4 Surveillance des tâches dans Amazon ECS

J'ai navigué vers la console Amazon ECS pour surveiller les tâches du cluster `microservices-serverlesscluster`.

ARN arn:aws:ecs:us-east-1:076704302252:task/k/microservices-serverlesscluster/645fb3d3f264de7ad6e37a945df5118	Last status Running	Desired status Running	Started/created at 2024-05-03T02:21:40.420Z 2024-05-03T02:21:05.122Z
Configuration			
Operating system/Architecture Linux/X86_64	Capacity provider -	ENI ID eni-03fb71811851be56	Public IP 3.237.182.59 open address
CPU Memory .5 vCPU 1 GB	Launch type FARGATE	Network mode awsvpc	Private IP 10.16.10.13
Platform version 1.4.0	Container instance IDs: -	Subnet ID subnet-0f3dcba570ee1efab	MAC address 02:17:3c:05:a0:f3
Task definition revision			

FIGURE 49 – Surveillance des tâches dans le cluster ECS.

L'onglet Tâches a permis d'observer les détails des tâches en cours, y compris les adresses IP associées aux conteneurs actifs et les révisions des définitions de tâches.

13.5 Vérification des configurations du Load Balancer et des groupes cibles

Pour assurer une gestion adéquate du trafic réseau, j'ai vérifié les configurations de l'Application Load Balancer et des groupes cibles dans la console Amazon EC2.

Listener rules (2) <small>Info</small>					
Traffic received by the listener is routed according to the default action and any additional rules. Rules are evaluated in priority order from the lowest value to the highest value.					
Rule limits Edit Actions ▼ Add rule					
Filter rules ✖					
Name tag	Priority	Conditions (If)	Actions (Then)	ARN	Tags
-	1	Path Pattern is /admin/*	Forward to target group <ul style="list-style-type: none"> employee-tg-two: 100 (100%) Group-level stickiness: Off 	ARN	0 tags
Default	Last (default)	If no other rule applies	Forward to target group <ul style="list-style-type: none"> customer-tg-one: 1 (100%) Group-level stickiness: Off 	ARN	0 tags

FIGURE 50 – Modification des règles d'écouteur de l'Application Load Balancer.

La mise à jour des règles d'écouteur confirme la direction correcte du trafic vers les nouveaux conteneurs, essentielle pour assurer que les nouvelles versions des microservices reçoivent le trafic prévu.

13.6 Créeation d'un pipeline pour le microservice employee

Pour le microservice employee, j'ai configuré un pipeline similaire à celui du microservice customer, adapté aux besoins spécifiques de ce service. Les détails sont les suivants :

- **Nom du pipeline** : update-employee-microservice.
- **Fournisseur de source** : AWS CodeCommit avec le dépôt deployment et la branche dev.
- **Fournisseur de déploiement** : Amazon ECS (Blue/Green), avec des configurations spécifiques pour gérer les définitions des tâches et les fichiers AppSpec.

Une source supplémentaire pour les images Docker stockées dans Amazon ECR a été intégrée pour être utilisée lors des déploiements.

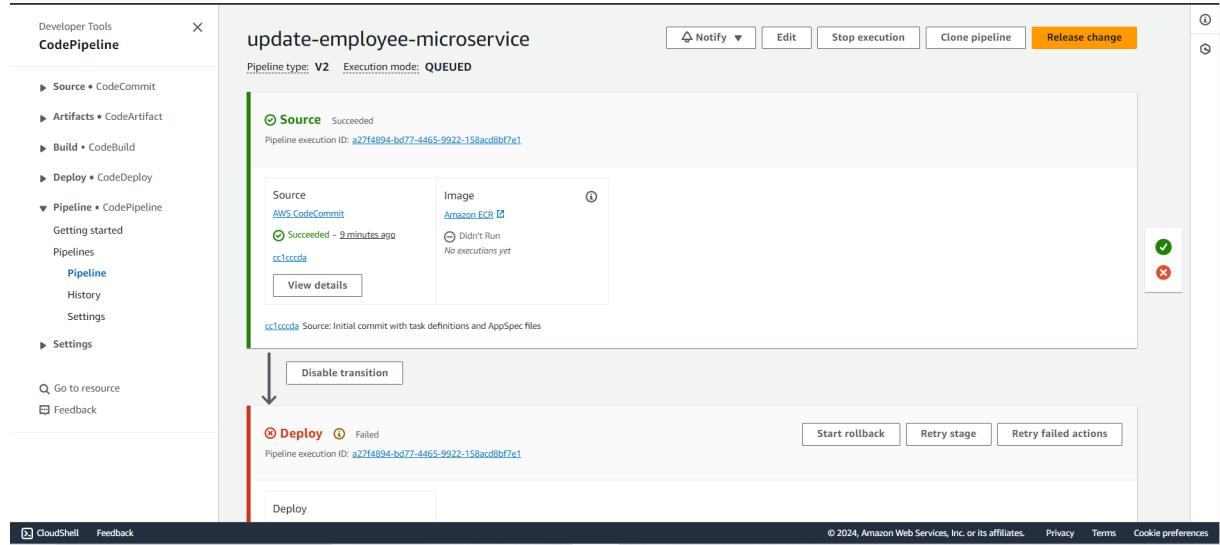


FIGURE 51 – Configuration du pipeline pour le microservice employee avec intégration des sources ECR.

13.7 Test du pipeline CI/CD pour le microservice employee

Après la configuration du pipeline, j'ai lancé un déploiement pour tester les réglages actuels du pipeline et j'ai observé le processus dans CodeDeploy. Le microservice employee a été testé en utilisant le nom DNS du load balancer microservicesLB. Ce déploiement réussi pour le microservice employee, similaire à celui du microservice customer, démontre l'efficacité de notre pipeline CI/CD pour la mise à jour et la maintenance des applications distribuées sur Amazon ECS.

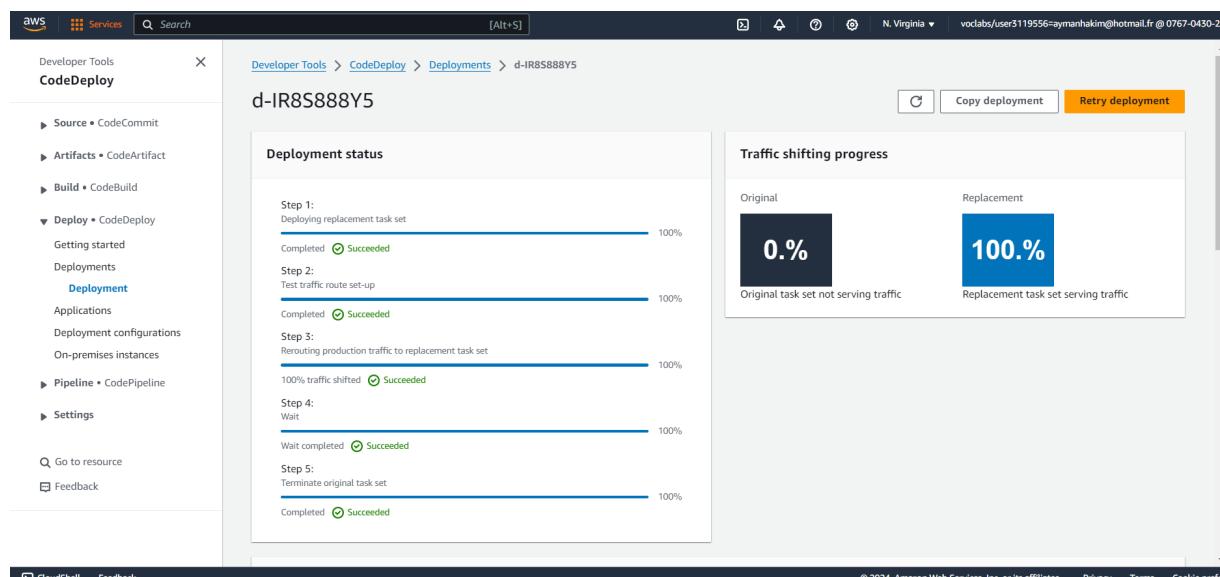


FIGURE 52 – Confirmation du succès du déploiement du microservice employee dans CodeDeploy.

The screenshot shows a web browser window with the URL <https://microserviceslb-100112117.us-east-1.elb.amazonaws.com/admin/suppliers>. The page title is "Manage coffee suppliers". On the left, there is a small image of two cups of coffee. On the right, there is a sidebar with links: "Administrator home", "Suppliers list", and "Customer home". The main content area is titled "All suppliers" and lists two entries:

Name	Address	City	State	Email	Phone	
Ayman Hakim	Lot Talhaoui, Rue El Ouafae 2, Nr 55, Sidi Yahya, Oujda	Oujda	Oriental	aymanhakim@hotmail.fr	0700301809	edit
AymanV2	rabat inpt	Oujda	Oriental	aymanhakim@hotmail.fr	07 00 30 18 09	edit

At the bottom left, there is a green button labeled "Add a new supplier".

FIGURE 53 – Succès de l'affichage du site pour le microservice employee.

13.8 Vérification des configurations du Load Balancer et des groupes cibles

Pour garantir une gestion adéquate du trafic réseau, j'ai vérifié les configurations de l'Application Load Balancer et des groupes cibles. Les règles d'écouteur ont été mises à jour pour assurer que le trafic soit correctement dirigé vers les nouveaux conteneurs.

The screenshot shows the "Listener rules" section of the AWS Application Load Balancer configuration. It displays two rules:

- Path Pattern is /admin/***: Priority 1, Forward to target group "employee-tg-one" (100%), Group-level stickiness: Off.
- Default**: Last (default), If no other rule applies, Forward to target group "customer-tg-one" (100%), Group-level stickiness: Off.

A blue arrow points to the "Default" rule.

FIGURE 54 – Mise à jour des règles d'écouteur de l'Application Load Balancer pour le microservice employee.

14 Phase 9 : Ajustement du code des microservices pour relancer le pipeline

Dans cette phase, nous allons ajuster le code du microservice employee pour relancer le pipeline de déploiement et mettre à jour l'application en production. Nous allons également augmenter le nombre de conteneurs supportant le microservice customer.

14.1 Limitation de l'accès au microservice employee

Pour sécuriser l'accès au microservice employee, j'ai modifié les règles du load balancer pour limiter l'accès à une adresse IP spécifique. Cette étape garantit que seules les requêtes provenant de cette adresse IP puissent atteindre les pages de gestion des fournisseurs.

Name tag	Priority	Conditions (If)	Actions (Then)	ARN	Tags
-	1	• Source IP is 105.71.16.38/32, AND • Path Pattern is /admin/*	Forward to target group • employee-tg-two 1 (100%) • Group-level stickiness: Off	ARN	0 tags
Default	Last (default)	If no other rule applies	Forward to target group • customer-tg-two 1 (100%) • Group-level stickiness: Off	ARN	0 tags

FIGURE 55 – Modification des règles de l'écouteur pour l'adresse IP spécifiée.

14.2 Modification de l'interface utilisateur du microservice employee et mise à jour de l'image Docker

J'ai ajusté l'interface utilisateur du microservice employee et généré une nouvelle image Docker. Les commandes suivantes ont été utilisées pour construire l'image et la pousser vers Amazon ECR, déclenchant ainsi le pipeline de mise à jour.

```

--> Using cache
--- 3f4bf4d7e9f18
Step 5/7 : RUN npm install
---> Running in c5546aec02b1
npm WARN coffee_api@1.0.0 No repository field.

audited 78 packages in 0.773s
found 10 vulnerabilities (5 moderate, 3 high, 2 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
Removing intermediate container c5546aec02b1
--- 4efda28a1e73
Step 6/7 : EXPOSE 8080
---> Running in e8cd549a035a
Removing intermediate container e8cd549a035a
--- 2bbbaad79337
Step 7/7 : CMD ["npm", "run", "start"]
---> Running in b14a8c5c784c
Removing intermediate container b14a8c5c784c
--- 9d6d4acc2730
Successfully built 9d6d4acc2730
Successfully tagged employee:latest
vclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat config/config.js | grep 'APP_DB_HOST' | cut -d '' -f2)
cat: config/config.js: No such file or directory
vclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity | grep Account | cut -d '' -f4)
vclabs:~/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
vclabs:~/environment/microservices/employee (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
vclabs:~/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [076704302252.dkr.ecr.us-east-1.amazonaws.com/employee]
b1785ad7cf50: Pushed
9d6d4acc2730: Pushed
235f0025252b: Layer already exists
d0f71509b7: Layer already exists
1d7f3bb09a4: Layer already exists
deaceb72924: Layer already exists
f1b5933fe4b5: Layer already exists
latest: digest: sha256:400ce54abfd654558372d44d86468f319febe3dc45e8ce8d9f56cb79103c30a3 size: 1783
vclabs:~/environment/microservices/employee (dev) $ 

```

FIGURE 56 – Poussée de la nouvelle image Docker vers Amazon ECR.

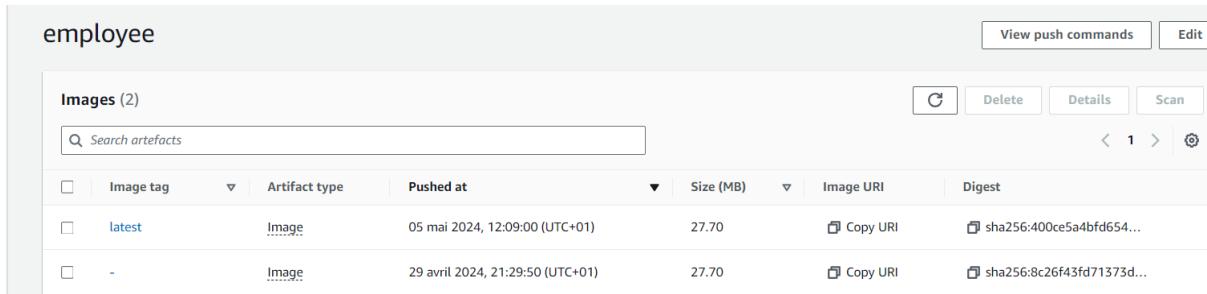


FIGURE 57 – Vérification des images docker dans ECR.

14.3 Test du pipeline CI/CD pour le microservice employee

Après la mise à jour de l'image Docker, le pipeline CI/CD a été relancé automatiquement. J'ai observé le déploiement via CodeDeploy pour confirmer que les modifications étaient bien appliquées en production.

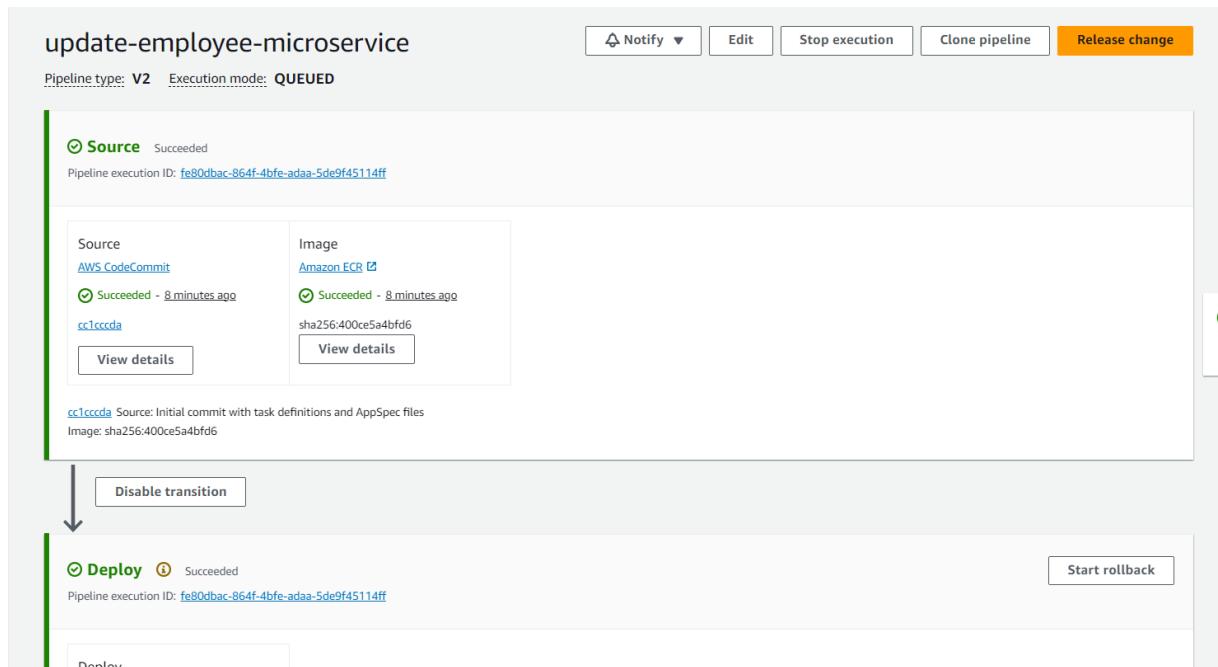


FIGURE 58 – Exécution réussie du pipeline après mise à jour de l'image Docker.

14.4 Test de l'accès au microservice employee

J'ai testé l'accès aux pages du microservice employee pour confirmer que les modifications de sécurité et de l'interface utilisateur étaient effectives. Seules les requêtes provenant de l'adresse IP spécifiée pouvaient accéder aux pages administratives.

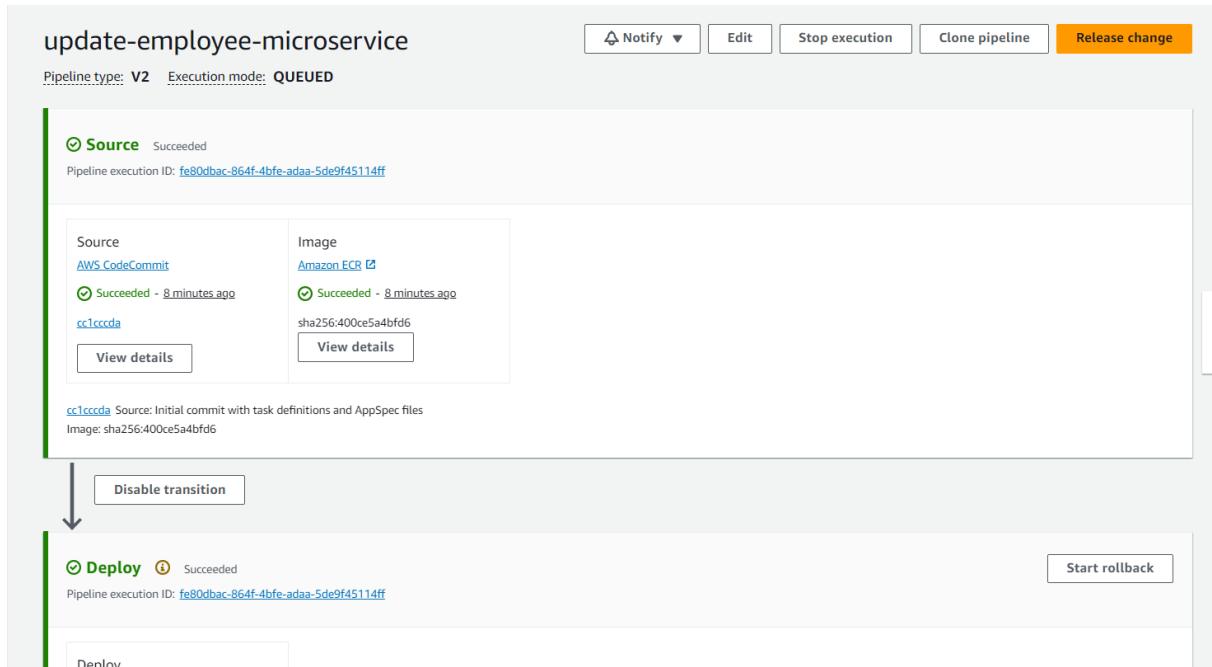


FIGURE 59 – Modification réussie du banner de notre site web.

14.5 Augmentation du nombre de conteneurs pour le microservice customer

Pour répondre à une charge accrue, j'ai augmenté le nombre de conteneurs exécutant le microservice customer. Cette opération démontre la flexibilité et l'indépendance des microservices dans notre architecture.

Services		Tasks	
Draining	-	Pending	-
Active	2	Running	4

FIGURE 60 – Augmentation du nombre de conteneurs pour le microservice customer.

15 Conclusion

Ce projet a été une exploration approfondie des principes d'ingénierie des microservices et de l'automatisation des déploiements via des pipelines CI/CD, en utilisant une suite d'outils et de services AWS. En décomposant une application monolithique en microservices, ce projet a non seulement amélioré la maintenabilité et la scalabilité de l'application, mais a également permis une plus grande flexibilité dans le développement et le déploiement des différentes composantes de l'application.

La mise en œuvre de l'architecture microservices a démontré comment les services individuels peuvent être développés, testés et déployés de manière indépendante, réduisant ainsi les risques associés aux changements et permettant une mise sur le marché plus rapide des nouvelles fonctionnalités. L'intégration de pipelines CI/CD avec CodeDeploy et CodePipeline a automatisé les processus de test et de déploiement, ce qui a renforcé la fiabilité des déploiements et amélioré l'efficacité opérationnelle globale.

En outre, l'utilisation d'Amazon ECS et Fargate pour gérer les déploiements de conteneurs a simplifié l'orchestration des services et assuré une haute disponibilité et une gestion des ressources efficace. L'application de stratégies de déploiement Blue/Green via le load balancer a également permis de minimiser les interruptions pendant les mises à jour, garantissant ainsi une expérience utilisateur sans interruption.

Ce projet a non seulement renforcé les compétences techniques en matière de cloud computing et de gestion de microservices, mais a également fourni des connaissances précieuses sur l'optimisation des architectures d'applications pour le cloud. Les leçons apprises ici seront sans aucun doute un atout précieux pour les futurs projets et initiatives de développement.

En conclusion, ce projet a été une grande réussite , démontrant l'efficacité des pratiques modernes de développement et d'infrastructure as code dans la réalisation d'une architecture robuste et évolutive. Les compétences et expériences acquises tout au long de ce projet seront bénéfiques pour moi pour aborder des défis encore plus grands à l'avenir.