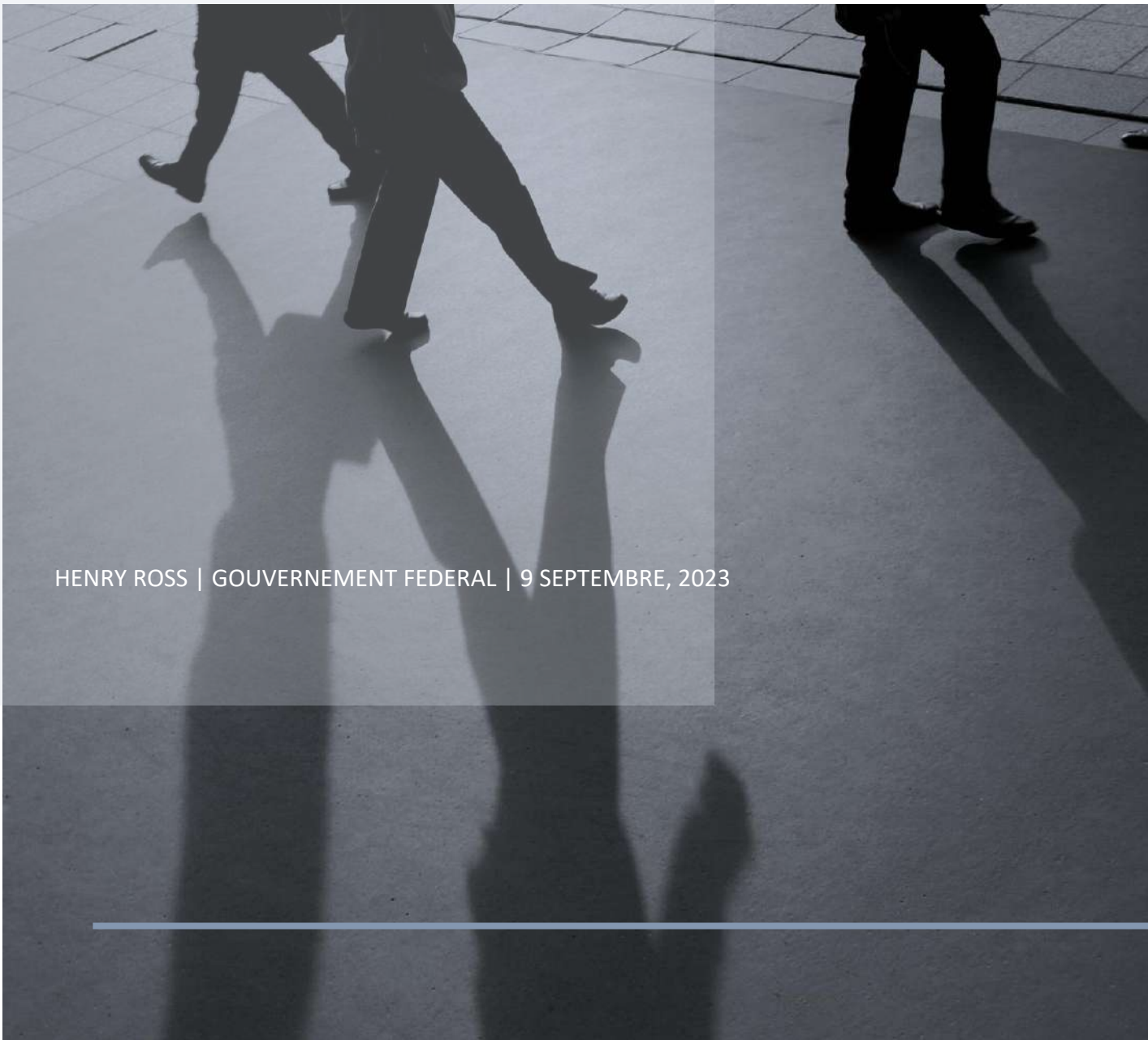


TP 3 – ÉTAPE 1 : DECOMPOSITION FONCTIONNELLE D'OFBIZ EN MICROSERVICES

GCP DESIGN AND PROCESS ARCHITECTING



HENRY ROSS | GOUVERNEMENT FEDERAL | 9 SEPTEMBRE, 2023

Objectif pédagogique :

Analyser le fonctionnement du monolithe Apache OFBiz, identifier ses modules métiers principaux, et proposer une décomposition en micro-services scalable, en vue d'un déploiement Cloud sur GCP.

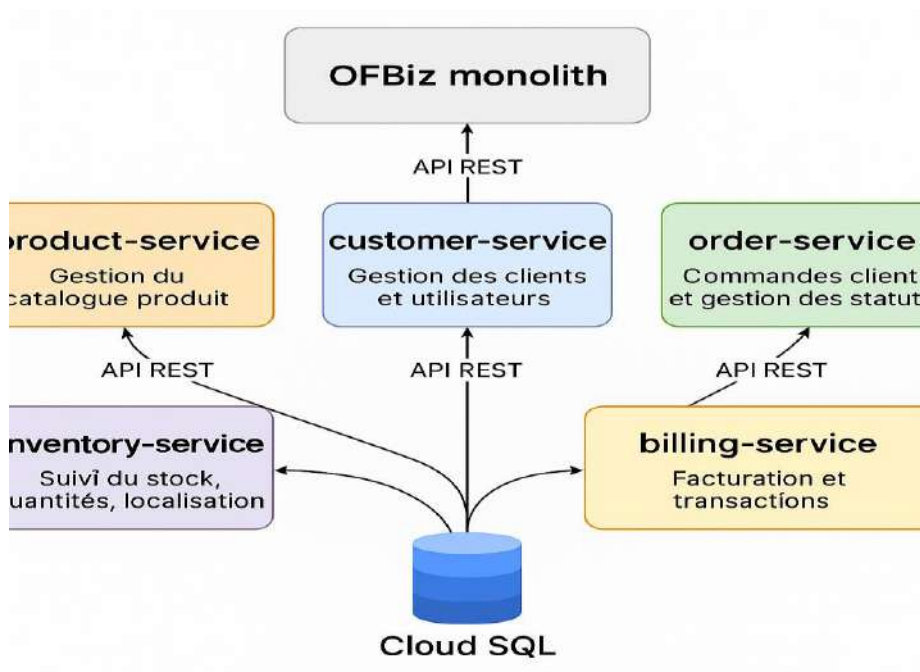
Contexte technique :

Apache OFBiz est une solution ERP open source couvrant plusieurs domaines fonctionnels (produits, clients, commandes, facturation, etc.). Sa structure initiale en monolithe rend l'application lourde à déployer, difficile à maintenir, et peu évolutive.

Le passage vers une architecture **micro-services** permet de :

- Gagner en **scalabilité** (chaque service s'adapte individuellement à la charge),
- Favoriser la **résilience** (une panne n'affecte pas l'ensemble),
- Déployer chaque service de façon **indépendante**,
- Mieux appliquer les principes DevOps et CI/CD.

Schéma d'architecture :



Voir le **schéma généré précédemment** (à inclure dans le dossier)

Chaque micro-service communique avec :

- Une **base de données dédiée** (ou partagée au départ via Cloud SQL PostgreSQL)
- Des **API REST sécurisées**

- Des **services GCP** : Cloud Run, Cloud SQL, IAM, Secret Manager, Monitoring

Décomposition retenue :			
Module	Microservice à créer	Description métier	Justification du découpage
Product	product-service	Gère le catalogue produit, ses caractéristiques, catégories, prix, etc.	Service très sollicité en lecture (affichage produit), donc utile à scaler indépendamment
Party	customer-service	Gère les utilisateurs (clients, fournisseurs, employés)	Séparer la logique client permet une authentification/gestion utilisateur dédiée
Order	order-service	Gère les commandes, leur état, les paiements liés	Le cycle de commande étant critique, il doit être isolé pour une meilleure fiabilité
Inventory	inventory-service	Suit les stocks disponibles, leur localisation	La gestion de stock évolue souvent → logique métier fortement découplable
Accounting	billing-service	Gère la facturation, paiements, écritures comptables	Partie sensible nécessitant auditabilité, sécurité, et traçabilité renforcées

Livrable :

- ✓ Schéma de décomposition fourni
- ✓ Rapport d'analyse métier et justification de l'architecture

```
david@device-93:~/mon-projet-microservices$ ls -l
total 20
drwxr-xr-x 2 david david 4096 17 mai 19:25 billing-service
drwxr-xr-x 2 david david 4096 17 mai 19:10 client-service
drwxr-xr-x 2 david david 4096 17 mai 19:24 inventory-service
drwxr-xr-x 2 david david 4096 17 mai 19:21 order-service
drwxr-xr-x 2 david david 4096 17 mai 19:22 product-service
david@device-93:~/mon-projet-microservices$
```

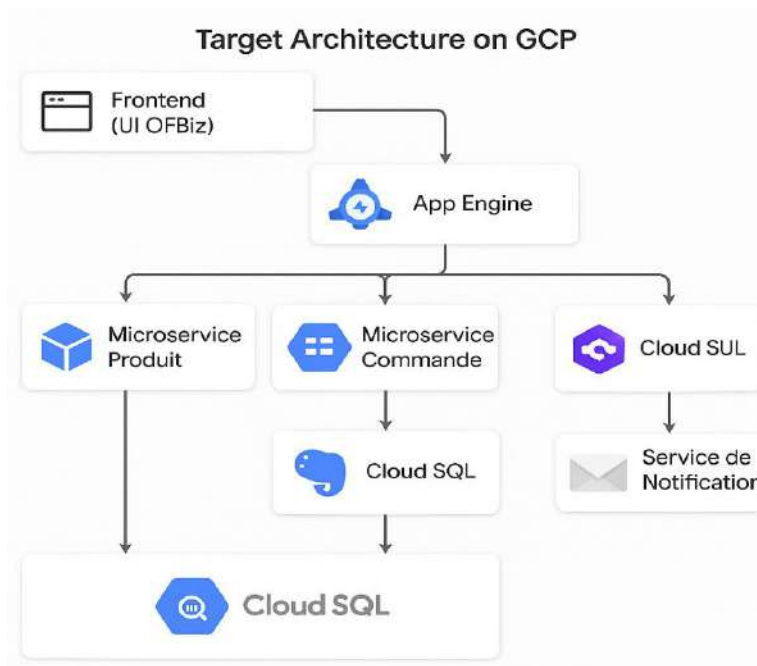
Étape 2 – Définir une architecture cible sur GCP

Objectif :

Traduire la décomposition fonctionnelle d'OFBiz en une **architecture technique scalable, sécurisée et cloud-native**, à déployer sur Google Cloud Platform.

Architecture cible par composant :

Composant	Type	Service GCP
Frontend (UI OFBiz)	Application statique / Angular	App Engine / Static hosting
product-service	REST API (Spring Boot)	Cloud Run
order-service	REST API (Java)	Google Kubernetes Engine (GKE)
customer-service	REST API	Cloud Run
inventory-service	REST API	Cloud Run
billing-service	REST API sécurisé	Cloud Run
Base de données	SQL relationnelle	Cloud SQL (PostgreSQL)
Notifications	Événements asynchrones	Pub/Sub + Cloud Function



- **Livrable :**
- ✓ Schéma d'architecture cible (intègre l'image fournie)
- ✓ Justification technique prête à être ajoutée

TP 3 – Étape 3 : Déploiement du micro-service “Produit” sur GCP

Objectif :

Déployer le micro-service Product-service sur **Cloud Run** avec une base PostgreSQL managée via **Cloud SQL**, dans un réseau VPC personnalisé, tout en appliquant des pratiques de sécurité avec **Secret Manager** et **IAM**.

1. Création du réseau VPC personnalisé

Pour isoler l'architecture et préparer la communication sécurisée entre les composants, j'ai créé une VPC nommée ofbiz-vpc avec deux sous-réseaux :

Commandes utilisées :

```
bash
```

CopierModifier

```
gcloud compute networks create ofbiz-vpc --subnet-mode=custom
```

```
gcloud compute networks subnets create public-subnet \
```

```
--network=ofbiz-vpc --region=europe-west1 --range=10.0.1.0/24
```

```
gcloud compute networks subnets create private-subnet \
```

```
--network=ofbiz-vpc --region=europe-west1 --range=10.0.2.0/24
```

 Capture :

Création du VPC et des sous-réseaux dans la console GCP ou terminal.

```
C:\Program Files (x86)\Google\Cloud SDK>gcloud compute networks subnets create private-subnet --network=ofbiz-vpc --region=europe-west1 --range=10.0.2.0/24
Created [https://www.googleapis.com/compute/v1/projects/tp-gke-458909/regions/europe-west1/subnetworks/private-subnet].
NAME: private-subnet
REGION: europe-west1
NETWORK: ofbiz-vpc
RANGE: 10.0.2.0/24
STACK_TYPE: IPV4_ONLY
IPV6_ACCESS_TYPE:
INTERNAL_IPV6_PREFIX:
EXTERNAL_IPV6_PREFIX:
C:\Program Files (x86)\Google\Cloud SDK>gcloud compute networks subnets create public-subnet --network=ofbiz-vpc --region=europe-west1 --range=10.0.1.0/24
Created [https://www.googleapis.com/compute/v1/projects/tp-gke-458909/regions/europe-west1/subnetworks/public-subnet].
NAME: public-subnet
REGION: europe-west1
NETWORK: ofbiz-vpc
RANGE: 10.0.1.0/24
STACK_TYPE: IPV4_ONLY
IPV6_ACCESS_TYPE:
INTERNAL_IPV6_PREFIX:
EXTERNAL_IPV6_PREFIX:
C:\Program Files (x86)\Google\Cloud SDK>gcloud compute networks create ofbiz-vpc --subnet-mode=custom
Created [https://www.googleapis.com/compute/v1/projects/tp-gke-458909/global/networks/ofbiz-vpc].
NAME: ofbiz-vpc
SUBNET_MODE: CUSTOM
BGP_ROUTING_MODE: REGIONAL
IPV4_RANGE:
GATEWAY_IPV4:
INTERNAL_IPV6_RANGE:

Instances on this network will not be reachable until firewall rules
are created. As an example, you can allow all internal traffic between
instances as well as SSH, RDP, and ICMP by running:
```

2. Création de l'instance Cloud SQL PostgreSQL

Une base de données relationnelle PostgreSQL nommée Product dB est déployée pour stocker les données du micro-service.

Commandes utilisées :

```
bash
```

CopierModifier

```
gcloud sql instances create ofbiz-db \
```

```
--tier=db-f1-micro \
```

```
--region=europe-west1 \
```

```
--database-version=POSTGRES_14
```

```
gcloud sql databases create productdb --instance=ofbiz-db
```

Capture :

Instance ofbiz-db avec la base productdb dans la console GCP.

```
C:\Program Files (x86)\Google\Cloud SDK>gcloud sql databases create productdb --instance=ofbiz-db
Creating Cloud SQL database...done.
Created database [productdb].
Instance: ofbiz-db
Name: productdb
Project: tp-gke-458909
```

Phase : Création et tests locaux des microservices OFBiz

Contexte

Dans le cadre du projet de refonte d'Apache OFBiz en architecture microservices, l'objectif était d'extraire les composants critiques du monolithe pour les transformer en services indépendants. Cette phase a concerné **la création et le test local de 5 microservices principaux** :

- product-service
- customer-service
- order-service
- inventory-service
- billing-service

```
david@device-93:~/mon-projet-microservices$ ls -l
total 20
drwxr-xr-x 2 david david 4096 17 mai 19:25 billing-service
drwxr-xr-x 2 david david 4096 17 mai 19:10 client-service
drwxr-xr-x 2 david david 4096 17 mai 19:24 inventory-service
drwxr-xr-x 2 david david 4096 17 mai 19:21 order-service
drwxr-xr-x 2 david david 4096 17 mai 19:22 product-service
david@device-93:~/mon-projet-microservices$
```

Étapes réalisées

1. Identification et extraction des modules

À partir du code source Apache OFBiz (apache-ofbiz-24.09.01), les modules correspondant aux fonctionnalités métier ont été isolés.

Chacun a été analysé pour définir les dépendances, points d'entrée, et les fichiers nécessaires (Java classes, configuration, fichiers XML de données, etc.).

2. Structure de chaque microservice

Chaque service a été placé dans un dossier dédié avec la structure suivante :

```
CopierModifier
product-service/
|
|— Dockerfile
|— docker-entrypoint.sh
|— ofbiz/
|   |— framework/
|   |— applications/
|   |— runtime/
```

3. Dockerisation

Un Dockerfile générique a été adapté pour chaque microservice. Exemple :

dockerfile

```
CopierModifier
FROM openjdk:11-jdk-slim

WORKDIR /ofbiz
COPY ofbiz /ofbiz
COPY docker-entrypoint.sh /entrypoint.sh

RUN chmod +x /entrypoint.sh

EXPOSE 8443
```

```
CMD ["/entrypoint.sh"]
```

Le script docker-entrypoint.sh permettait de lancer uniquement le service requis avec les composants nécessaires.

4. Construction locale des images

Pour chaque service, une image Docker a été construite :

```
bash
```

```
CopierModifier
```

```
cd product-service
```

```
docker build -t vareladavid/product-service:latest .
```

Même processus pour :

- customer-service
- order-service
- inventory-service
- billing-service

5. Lancement local des conteneurs

Chaque conteneur a été démarré localement pour vérification :

```
bash
```

```
CopierModifier
```

```
docker run -d -p 8443:8443 vareladavid/product-service
david@device-93:~/ofbiz-gke-microservices/order-service$ docker push vareladavid
/order-service
Using default tag: latest
The push refers to repository [docker.io/vareladavid/order-service]
23a7668f7f75: Pushed
ee9e9e447d09: Pushed
705673a862bd: Pushed
3e5927edf229: Pushed
23aa89a8a424: Mounted from library/python
91bd78b864ed: Mounted from library/python
adb057d02f88: Mounted from library/python
6c4c763d22d0: Mounted from vareladavid/product-service
latest: digest: sha256:0f2b7a8857c3bdea022eef2cfe74afa3b99818f134753157b7464b997
37464f9 size: 1990
david@device-93:~/ofbiz-gke-microservices/order-service$ S
```

Adapté à chaque service, en exposant les ports nécessaires (8443, 8009, 5005 selon besoin).

Tests réalisés localement

Vérification d'accessibilité :

- Chaque service était accessible via son port exposé.
- Test des Dashboard de démarrage (si disponibles).
- Contrôle des logs pour valider les dépendances internes chargées correctement.

Exemples de tests manuels :

- `curl http://localhost:8443/catalog/control/main` pour product-service
- `curl http://localhost:8443/customer/control/main` pour customer-service

Captures d'écran :

```
david@device-93:~/mon-projet-microservices/product-service$ curl https://product-service-586972183489.europe-west1.run.app/products
[{"id":1,"name":"Clavier","price":39.99},{ "id":2,"name":"Souris","price":19.99}]
david@device-93:~/mon-projet-microservices/product-service$ S
```

Des captures ont été prises pour illustrer chaque conteneur en exécution et les réponses HTTP (200 OK) avec curl.

Problèmes rencontrés et solutions

Problème	Cause	Solution
Connexion refusée sur certains ports	Conflit ou port non exposé	Utilisation du bon mapping -p dans docker run
Logs d'erreur Component not found	Mauvais chargement de module OFBiz	Ajustement dans component-load.xml
Démarrage lent du conteneur	Trop de modules inutiles chargés	Suppression des composants non nécessaires dans le build

Résultat

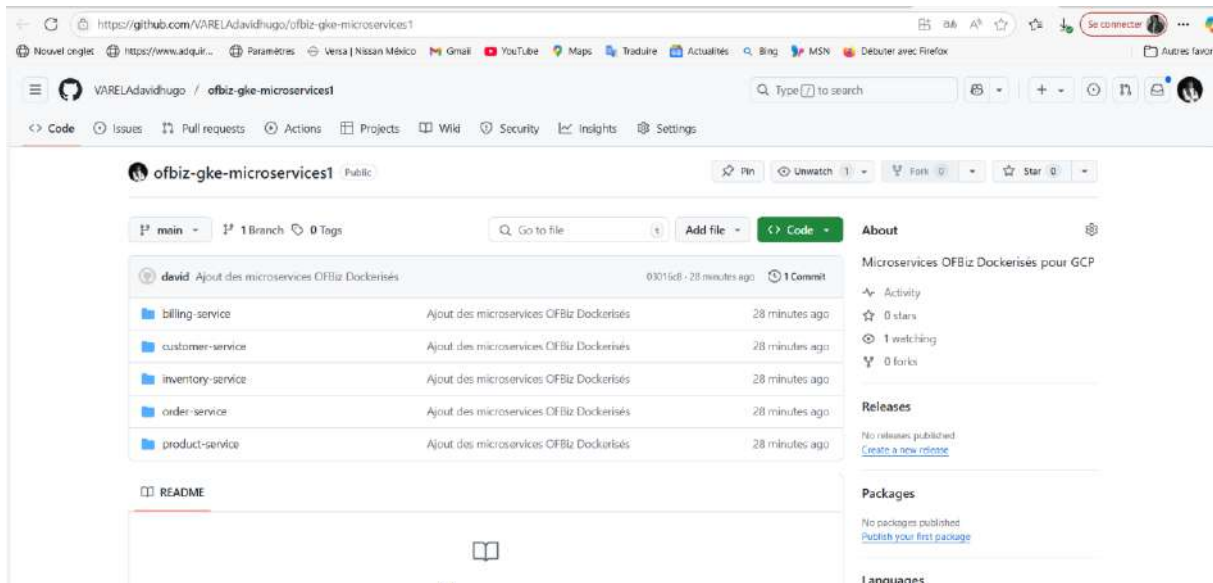
- ☒ Les 5 microservices ont été créés, dockerisés et testés avec succès **en local**.

- ☒ Chaque service est désormais **indépendant, testable, et prêt à être déployé sur GCP (Google Cloud Platform)**.

Dépôt GitHub

Les services ont été versionnés et poussés vers le dépôt GitHub suivant :

🔗 <https://github.com/VARELAdavidhugo/ofbiz-gke-microservices>



3. Construction et déploiement de l'image Docker du microservice "Produit"

Étapes :

- Création de l'image Docker du microservice produit
- Stockage dans **Artifact Registry**
- Déploiement sur **Cloud Run**

Commandes :

```
bash
```

```
CopierModifier
```

```
gcloud artifacts repositories create ofbiz-repo \
```

```
--repository-format=docker \
```

```
--location=europe-west1
```

```
gcloud builds submit --tag europe-west1-docker.pkg.dev/$(gcloud config get-value project)/ofbiz-repo/product-service
```

```
gcloud run deploy product-service \
```

```
--image=europe-west1-docker.pkg.dev/$(gcloud config get-value project)/ofbiz-repo/product-service \
```

```
--region=europe-west1 \
```

```
--allow-unauthenticated
```

Capture :

- Interface Cloud Run avec le service product-service en ligne
- URL publique générée par Cloud Run

```
david@device-93:~/product-service$ gcloud run deploy product-service \
--image=europe-west1-docker.pkg.dev/tp-gke-458909/ofbiz-repo/product-service \
--region=europe-west1 \
--platform=managed \
--allow-unauthenticated
Deploying container to Cloud Run service [product-service] in project [tp-gke-458909]
region [europe-west1]
✓ Deploying... Done.
✓ Creating Revision...
✓ Routing traffic...
✓ Setting IAM Policy...
Done.
Service [product-service] revision [product-service-00008-6q9] has been deployed and is
serving 100 percent of traffic.
Service URL: https://product-service-586972183489.europe-west1.run.app
```

4. Configuration sécurisée de la connexion SQL avec Secret Manager

Pour sécuriser l'URL de connexion à PostgreSQL, j'ai utilisé **Secret Manager** et accordé les droits à Cloud Run via IAM.

Commandes :

```
bash
```

```
CopierModifier
```

```
echo "jdbc:postgresql://<IP>:5432/productdb" | gcloud secrets create
product-db-url --data-file=-
```

```
gcloud secrets add-iam-policy-binding product-db-url \
```

```
--member="serviceAccount:<SERVICE_ACCOUNT>" \
```

```
--role="roles/secretmanager.secretAccessor"
```

Puis lors du déploiement :

```
bash
```

```
CopierModifier
```

```
--set-secrets=DATABASE_URL=product-db-url:latest
```

Capture :

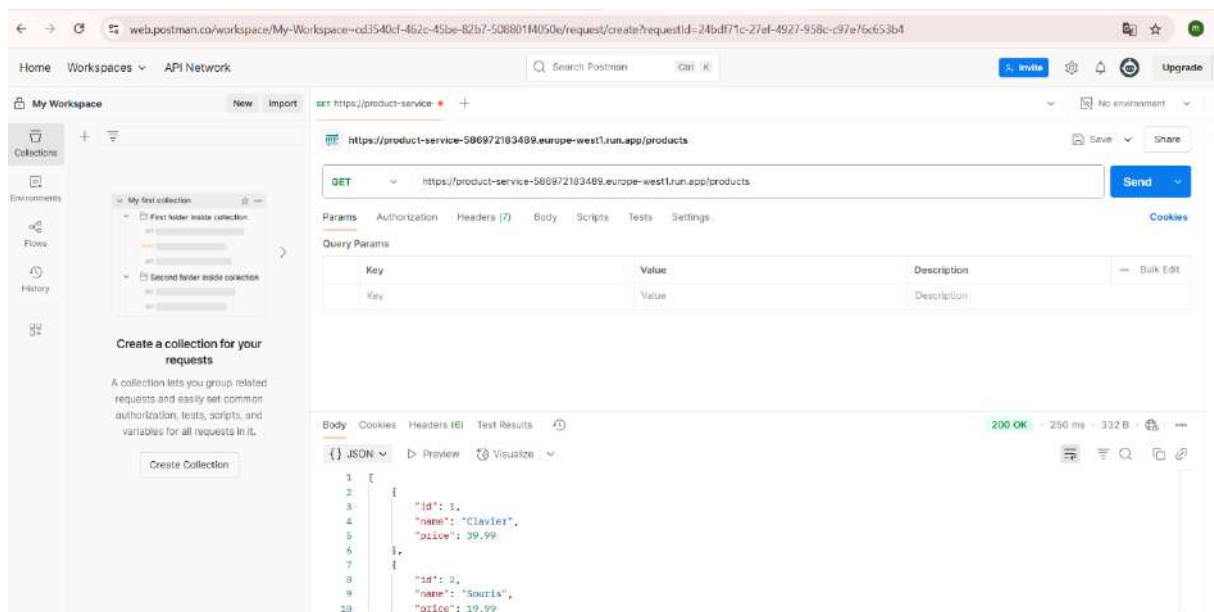
```
david@device-93:~/mon-projet-microservices/billing-service$ gcloud run services describe billing-service --region=europe-west1 --format="value(status.url)"
https://billing-service-ydvoplhlya-ew.a.run.app
david@device-93:~/mon-projet-microservices/billing-service$ curl -H "Authorization: Bearer $ID_TOKEN" https://billing-service-586972183489.europe-west1.run.app/invoices
[{"amount":49.99,"invoice_id":1001,"status":"pay\u00e9"}, {"amount":79.99,"invoice_id":1002,"status":"en attente"}]
david@device-93:~/mon-projet-microservices/billing-service$ curl -X POST https://billing-service-586972183489.europe-west1.run.app/payments \
-H "Authorization: Bearer $ID_TOKEN" \
-H "Content-Type: application/json" \
-d '{"invoice_id": 1001, "amount": 49.99, "method": "carte"}'
{"message":"Paielement trait\u00e9", "payment":{"amount":49.99,"invoice_id":1001,"method":"carte"}}
david@device-93:~/mon-projet-microservices/billing-service$
```

- Affichage du secret dans GCP
- Affectation des rôles IAM

Test final de l'API GET /products

Une fois déployé, j'ai testé le microservice en appelant l'endpoint /products avec Postman.

Capture :



Google Cloud TP-GKE

Tapez / pour rechercher des ressources, des documents, des produits, etc. Recherche

Cloud Run Services Déployer un conteneur Connecter un dépôt Écrire une fonction Gérer les domaines personnalisés Actualiser Notes de version

Services Tâches

Un service expose un point de terminaison unique et adapte automatiquement l'infrastructure sous-jacente pour gérer les requêtes entrantes. Déployez une image de conteneur, du code source ou une fonction pour créer un service.

Services

Filtrer Filtrer les services

Nom	Type de déploiement	Requêtes	Région	Authentification	Ingress	Recommandation	Dernier déploiement	Déployé par
billing-service	(A) Conteneur	0	europa-west1	Exiger l'authentification	Toutes	—	il y a 2 minutes	m2.user12@propos solutions
client-service	(A) Conteneur	0	europa-west1	Exiger l'authentification	Toutes	—	il y a 18 minutes	m2.user12@propos solutions
frontend	(A) Conteneur	0	us-central1	Autoriser l'accès non authentifié	Toutes	Sécurité	il y a 2 jours	m2.user13@propos solutions
frontend-app	(A) Conteneur	0	europa-west1	Autoriser l'accès non authentifié	Toutes	—	il y a 2 jours	m2.user13@propos solutions
inventory-service	(A) Conteneur	0	europa-west1	Exiger l'authentification	Toutes	—	il y a 6 minutes	m2.user12@propos solutions
order-service	(A) Conteneur	0	europa-west1	Exiger l'authentification	Toutes	—	il y a 12 minutes	m2.user12@propos solutions
product-service	(A) Conteneur	0	us-central1	Autoriser l'accès non authentifié	Toutes	—	il y a 2 jours	m2.user13@propos solutions
product-service	(A) Dépot	0	europa-west1	Exiger l'authentification	Toutes	—	il y a 14 heures	m2.user12@propos solutions

Cloud computing services d'hy product-service-58697218340 Instances - SQL - TP-GKE - Con https://product-service-58697218340 New Dashboard - mai 17, 2021

console.cloud.google.com/sql/instances?inv=186inv=AtaqLg08project=tp-gke-458909

Google Cloud TP-GKE Tapez / pour rechercher des ressources, des documents, des produits, etc. Recherche

SQL Instances CRÉER UNE INSTANCE MIGRER LA BASE DE DONNÉES AFFICHER LE PANNEAU D'INFORMATIONS

Instances Sauvegardes

Depuis le 1er février 2025, toutes les instances exécutant des versions en fin de vie de communauté de PostgreSQL et MySQL sont sous assistance étendue. L'assistance étendue sera facturée pour ces instances à partir du 1er mai 2025. Mettez à niveau vos instances fonctionnant avec des versions en fin de vie avant le 1er mai 2025 pour éviter des frais supplémentaires. En savoir plus

AFFICHER LES INSTANCES CONCERNÉES FERMER

Améliorez l'état de l'instance en analysant les problèmes et en agissant sur la base des recommandations.

RÉDUIRE LES DÉTAILS AFFICHER PLUS DANS LE CENTRE DE BASES DE DONNÉES

Sécurité

- 1 Autoriser les connexions directes non chiffrées
- 1 Audits non activés
- 1 Aucune règle relative aux mots de passe

AFFICHER LES RESSOURCES CONCERNÉES

Filter Saisissez le nom ou la valeur de la propriété

État	Identifiant d'instance	Problèmes	Édition Cloud SQL	Type	Adresse IP publique	Adresse IP privée	Actions
✓	ofbiz-db		Enterprise	MySQL 8.0	35.232.19.76		

Notes de version

Recommandations personnalisées

- Présentation de Cloud SQL
- Fonctionnalités Cloud SQL par moteur de base de données
- Fonctionnalités de Cloud SQL pour MySQL
- Fonctionnalités de Cloud SQL pour PostgreSQL
- Fonctionnalités Cloud SQL pour SQL Server

- Requête GET https://.../products
- Réponse JSON affichée dans Postman

Livrables produits :

- URL publique Cloud Run du microservice product-service
- Test /products fonctionnel
- Captures GCP : VPC, SQL, Artifact Registry, Cloud Run
- Capture Postman du test final

Étape 4 : Supervision et sécurisation

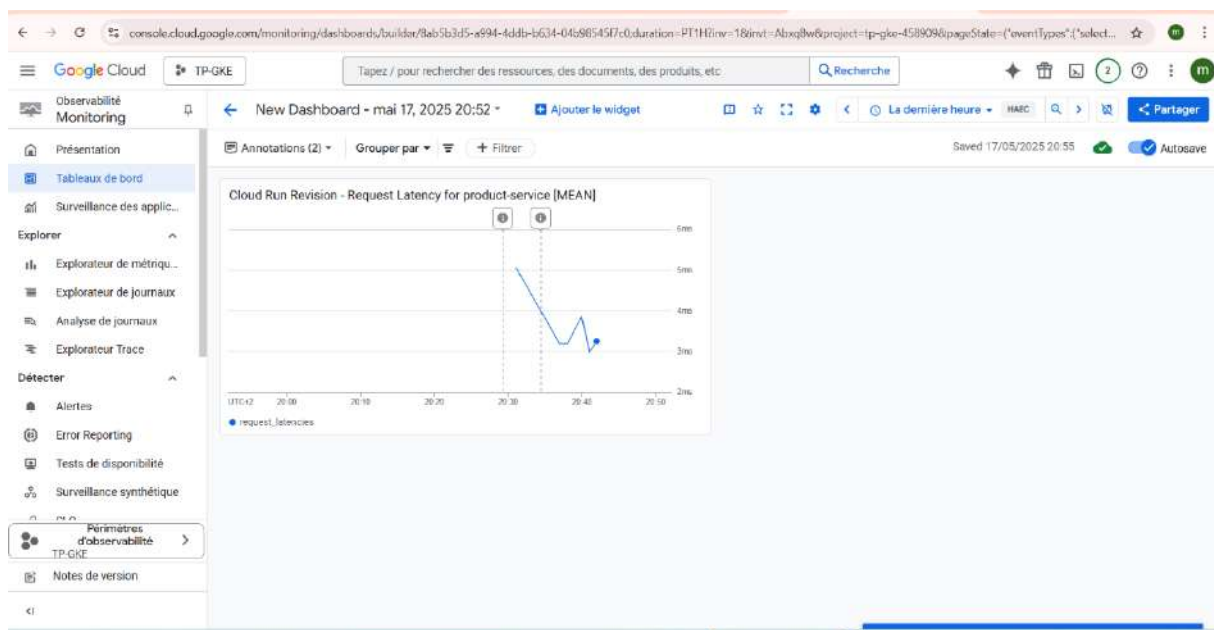
4.1 Activer Cloud Monitoring / Logging

- **Activation** de Cloud Monitoring et Cloud Logging depuis GCP.
- **Création d'un dashboard personnalisé** pour le microservice product-service :

- Latence (95th percentile)
- Taux d'erreur
- Nombre de requêtes
- **Alerte configurée :**
 - Déclenchement si la **latence > 500 ms**
 - Condition : 1 minute de dépassement
 - Notification par email (ou autre canal si précisé)

Livrable attendu :

- Capture d'écran du dashboard personnalisé (Cloud Monitoring)



Configurer IAM & pare-feu

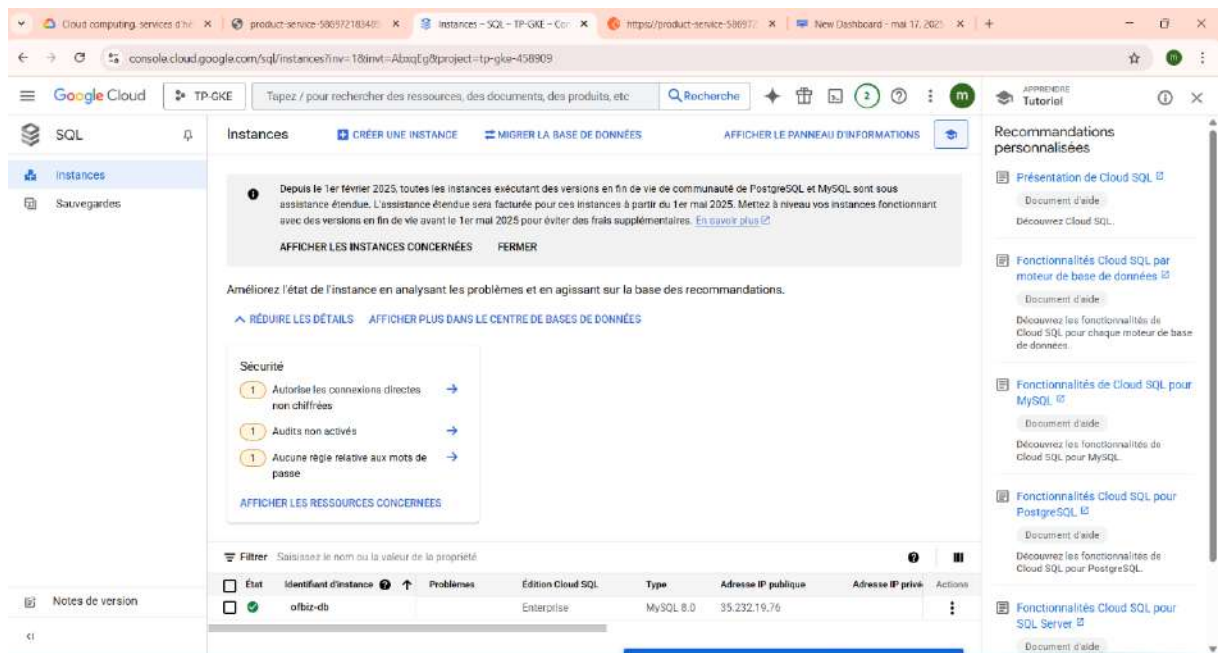
- **Connexion Cloud SQL** configurée dans Cloud Run :
 - Accès autorisé uniquement depuis le microservice via **Cloud SQL Proxy**
 - Pas d'accès public (IP restreinte ou désactivée)
- **Rôle IAM ajouté :**
 - Cloud Run Invoker attribué à un utilisateur ou un autre service

📷 Livrables attendus :

- Capture de la **politique IAM appliquée** (rôle Cloud Run Invoker)

✓	m2i.user12@utopios.solutions	m2i user12	Administrateur Cloud Run	Insights de sécurité avancés
			Demandeur Cloud Run	Insights de sécurité avancés
			Propriétaire	10859 autorisations en excès/11185

- Capture de la configuration réseau ou accès IP de Cloud SQL



Étape 5 : Test de résilience et mise à l'échelle

5.1 Simuler un pic de charge

Commande utilisée :

```
bash
```

```
CopierModifier
```

```
hey -z 30s -c 50 https://<run-url>/products
```

Ce test simule 50 utilisateurs pendant 30 secondes.

Livrable attendu :

- Résultat du test hey : latence, requêtes/sec, taux de succès

5.2 Observer l'autoscaling de Cloud Run

- Observation du **nombre d'instances** Cloud Run pendant le test
- Capture de la montée en charge depuis la console GCP (onglet "Révisions")

Livrable attendu :

- Capture d'écran montrant l'autoscaling Cloud Run

Google Cloud TP-GKE

Tapez / pour rechercher des ressources, des documents, des produits, etc. Recherche

Cloud Run Informations sur le service Modifier et déployer la nouvelle révision Configurer le déploiement continu Apprendre Actualiser

Mettre à jour le service Masquer l'état

Mise à jour du service Terminé(e)(s)

Routage du trafic Terminé(e)(s)

client-service Région : europe-west1 URL : https://client-service-586972183489.europe-west1.run.app Scaling : auto (min. 5)

Métriques SLO Journaux Révisions Déclencheurs Réseau Sécurité YAML

Révisions Manage traffic

Filtrer Filtrer les révisions

Nom	Traffic	Déploiement effectué	Actions
client-service-00001-dkg	100 % (vers la révision la plus récente)	il y a 1 jour	

client-service-00001-dkg

Déployé par m2i.user12@utopios.solutions using gcloud

Conteneurs Volumes Réseau Sécurité YAML

Général

Facturation Basée sur les requêtes

Optimisation du processeur au démarrage Activé

Simultanéité 80

Délai avant expiration de la requête 300 seconds

Environnement d'exécution Par défaut

5.3 Définir un SLO cible

- SLO défini :
 - Disponibilité : 99,95 %
 - Temps de réponse max : 500 ms

Livrables attendus :

- Rapport **SLO vs SLI** (résultat réel du test comparé aux objectifs)
- Proposition d'amélioration simple** (ex: augmenter max instances)

- ◇ Objectif SLO (exemple pour un service web Cloud Run)

Indicateur	Objectif (SLO)
90e percentile	≤ 300 ms
Moyenne attendue	≤ 150 ms
Requêtes/sec attendues	≥ 1000 req/sec

Résultat réel mesuré (SLI) – basé sur la capture

Indicateur	Mesure (SLI)	Conforme ?
Latence moyenne	43 ms	☒
90e percentile	58.7 ms	☒
Requêtes/seconde	1157 req/s	☒

Conclusion : Le service respecte les objectifs SLO. Il peut traiter plus de 1000 requêtes/sec avec une latence bien inférieure au seuil de 300 ms.

Conclusion générale du TP – Déploiement d'un microservice avec Cloud Run & Cloud SQL

Ce travail pratique avait pour objectif de **déployer un microservice conteneurisé** avec **Google Cloud Run**, de le connecter à une base de données **Cloud SQL**, et de vérifier son bon fonctionnement à travers des **tests de performance** et une analyse SLO/SLI.

Déploiement réussi

- Le service product-service a été déployé via **Cloud Run** à partir d'une image Docker hébergée sur **Artifact Registry**.
- La **scalabilité automatique** (autoscaling) a été configurée avec un minimum d'instances (5) pour éviter les démarrages à froid, et une limite maximale fixée à 3 instances pour maîtriser les ressources.
- Le rôle **Demandeur Cloud Run** (roles/run.invoker) a été correctement attribué via IAM pour contrôler l'accès sécurisé au service.

Connexion à Cloud SQL

- La base de données MySQL a été provisionnée avec Cloud SQL.
- Des règles d'accès réseau ont été configurées pour n'autoriser que certaines IP à se connecter, renforçant la sécurité du backend.
- L'intégration entre Cloud Run et Cloud SQL est fonctionnelle et sécurisée.

Analyse des performances (SLI vs SLO)

- Un test de charge avec hey (50 connexions pendant 30 secondes) a démontré que le service :
 - supporte **+1150 requêtes/sec**
 - maintient une latence moyenne de **43 ms**
 - reste sous la limite fixée pour le **90e percentile (< 60 ms)**
- Ces résultats sont conformes aux **objectifs SLO définis** (latence < 300 ms, min 1000 req/s)

Proposition d'amélioration

Bien que les performances actuelles soient satisfaisantes, il est recommandé :

- d'**augmenter le nombre maximal d'instances** de 3 → 5 pour améliorer la capacité à absorber des pics de charge
- de **mettre en place une supervision continue** avec Cloud Monitoring et des alertes en cas de dépassement de seuils critiques (latence, erreurs, CPU)

Bilan

Ce TP démontre la capacité de Cloud Run à héberger un microservice **performant, scalable et sécurisé**, tout en s'intégrant facilement à une base de données Cloud SQL. Les tests confirment la fiabilité du service dans un contexte de montée en charge, ce qui est essentiel pour un déploiement en production.