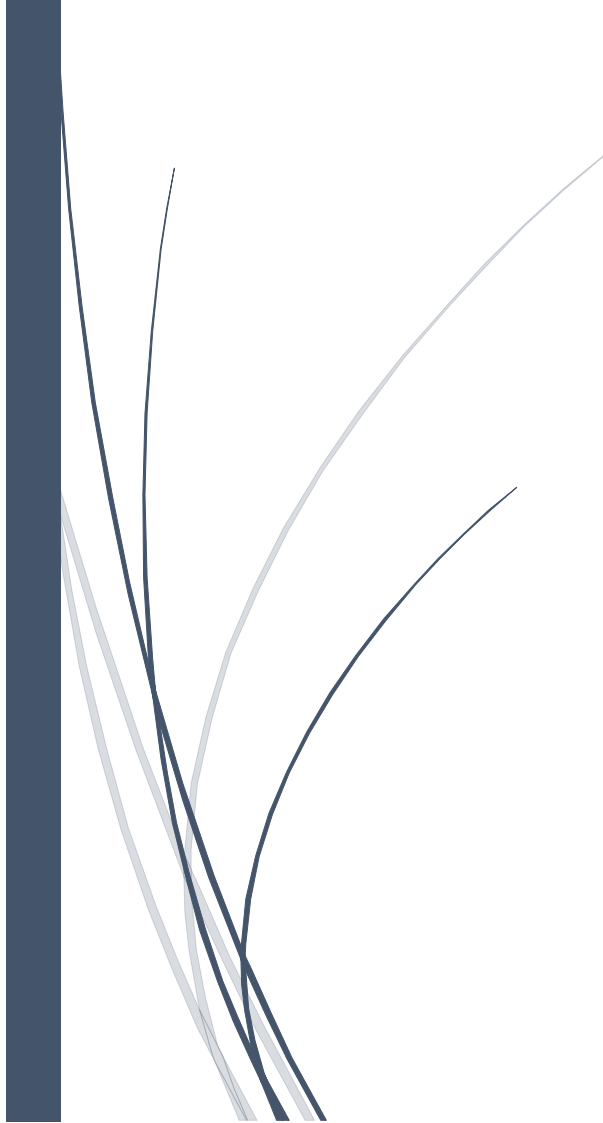


A dark blue vertical bar runs along the left edge of the page. A blue arrow points to the right from the bar, containing the date.

09/10/2021

# Software Architecture for Cloud Computing

Rapport Intermédiaire

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right across the page.

Anas FRANCIS  
Jonas ANIGLO  
Soulaïman ZABOURDINE  
Patrick ANAGONOU

POLYTECH'NICE SOPHIA

# Table des matières

I.	Introduction .....	2
II.	Personas.....	2
III.	Architecture du système .....	3
A.	Diagramme d'architecture logicielle.....	3
B.	Diagramme de classes (UML).....	4
1.	Accounting service .....	4
2.	Catalogue service .....	5
3.	CustomerCare service .....	6
IV.	Scénarios .....	7
V.	Supervision du système .....	10
VI.	Système de déploiement .....	11
A.	Intégration continue .....	11
B.	Diagramme de déploiement de Polypet sur Google cloud Platform.....	12
VII.	Scénario de démonstration MVP .....	14
VIII.	Estimation du prix de déploiement.....	15
A.	Google Cloud.....	15
B.	Amazon Web Services.....	17
IX.	Conclusion & répartition des points .....	19

## I. Introduction

Dans le cadre du cours Cloud Computing, nous avons eu à faire une analyse de besoin et mis en place une architecture pour l'implémentation d'une application appelée PolyPet. PolyPet répondra aux besoins de la société PolyPet, il s'agit d'une société qui développe une large variété de produits orientés NAC (Nouveaux Animaux de Compagnie).

PolyPet aura pour but de permettre à ses clients de découvrir les différents produits présents sur le catalogue et d'en passer une commande. Il permettra aussi la mise à jour de ce catalogue par les gestionnaires du site.

## II. Personas

Gilbert : visiteur du site PolyPet.

Besoin 1 : **En tant que** client de PolyPet, **je veux** voir les produits triés par catégories, **afin de** trouver le produit qui m'intéresse.

Besoin 2 : **En tant que** client de PolyPet ayant passé une commande sur le site internet, **je veux** que les produits de ma commande soient livrés par drone, **afin de** clôturer mon achat.

Alexandre : manager d'une industrie pharmaceutique

Besoin : **En tant que** manager d'une industrie pharmaceutique, **je voudrais** ajouter mes produits sur le site de PolyPet **afin que** les clients du site puissent voir mon produit.

Marion : cliente qui s'est connecté sur le site de PolyPet et qui veut ajouter des éléments au panier

Besoin : **En tant que** cliente qui a ajouté des produits au panier **je veux** procéder à l'achat en utilisant ma carte bancaire, **afin de** finaliser l'achat.

Frédéric : Manager du site PolyPet

Besoin 1 : **En tant que** manager **je souhaite** gérer les produits sur les sites, **afin de** mettre à jour le site avec les nouveaux produits.

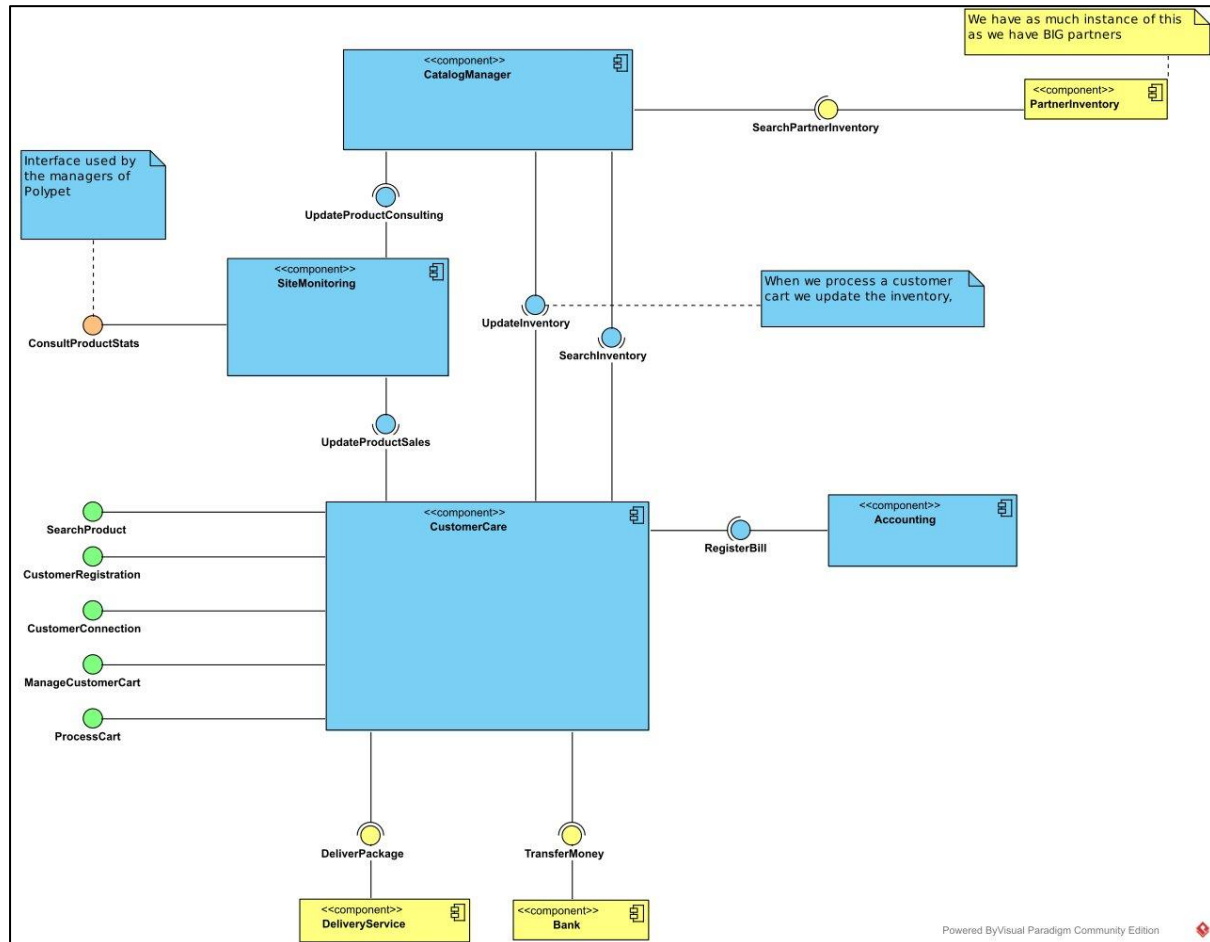
Besoin 2 : **En tant que** manager **je souhaite** mettre à jour les stocks **afin de** m'assurer qu'il n'y a pas de pénurie dans un/une entrepôt/région pendant trop longtemps.

Valérie : Membre de l'équipe Marketing

Besoin : **En tant que** membre de l'équipe Marketing de PolyPet **je souhaite** voir les statistiques des visites et les achats sur les différentes pages des produits **afin de** mettre en place une nouvelle stratégie de vente.

### III. Architecture du système

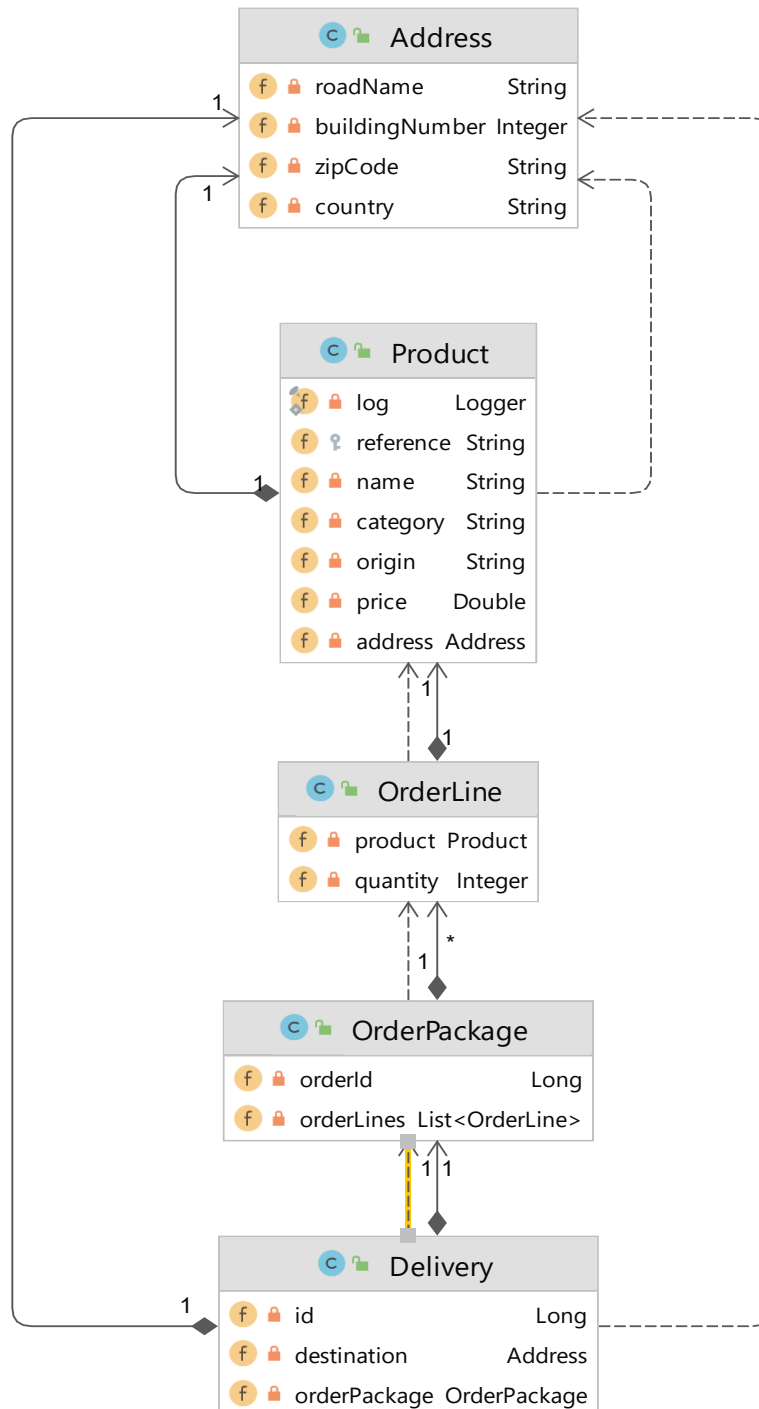
#### A. Diagramme d'architecture logicielle



Nous avons découpé notre application en services. Notre découpage a été fait de manière fonctionnelle : on a en effet avec ce découpage les différents départements qui pourraient exister au sein d'une entreprise de vente c'est-à-dire la comptabilité, le service client, et l'inventaire. Ce découpage permet aussi de dégager des différences de besoin (de ressources matérielles d'exécutions) entre nos différents services. Par exemple, notre service le plus important est le `CustomerCare` qui est appelé par tous les clients de Polypet le déploiement doit donc attribuer plus de ressources et d'élasticité à ce service. Le `SiteMonitoring` enregistre les informations de statistiques (ventes, recherche d'articles, des autres services ...). Dans le rapport initial nous comptons implémenter des intercepteurs sur les autres services qui capturerait les fonctions clés et enverrait des messages vers le `SiteMonitoring`. Mais nous n'avons pas pu utiliser les intercepteurs dans l'implémentation à cause de difficultés techniques.

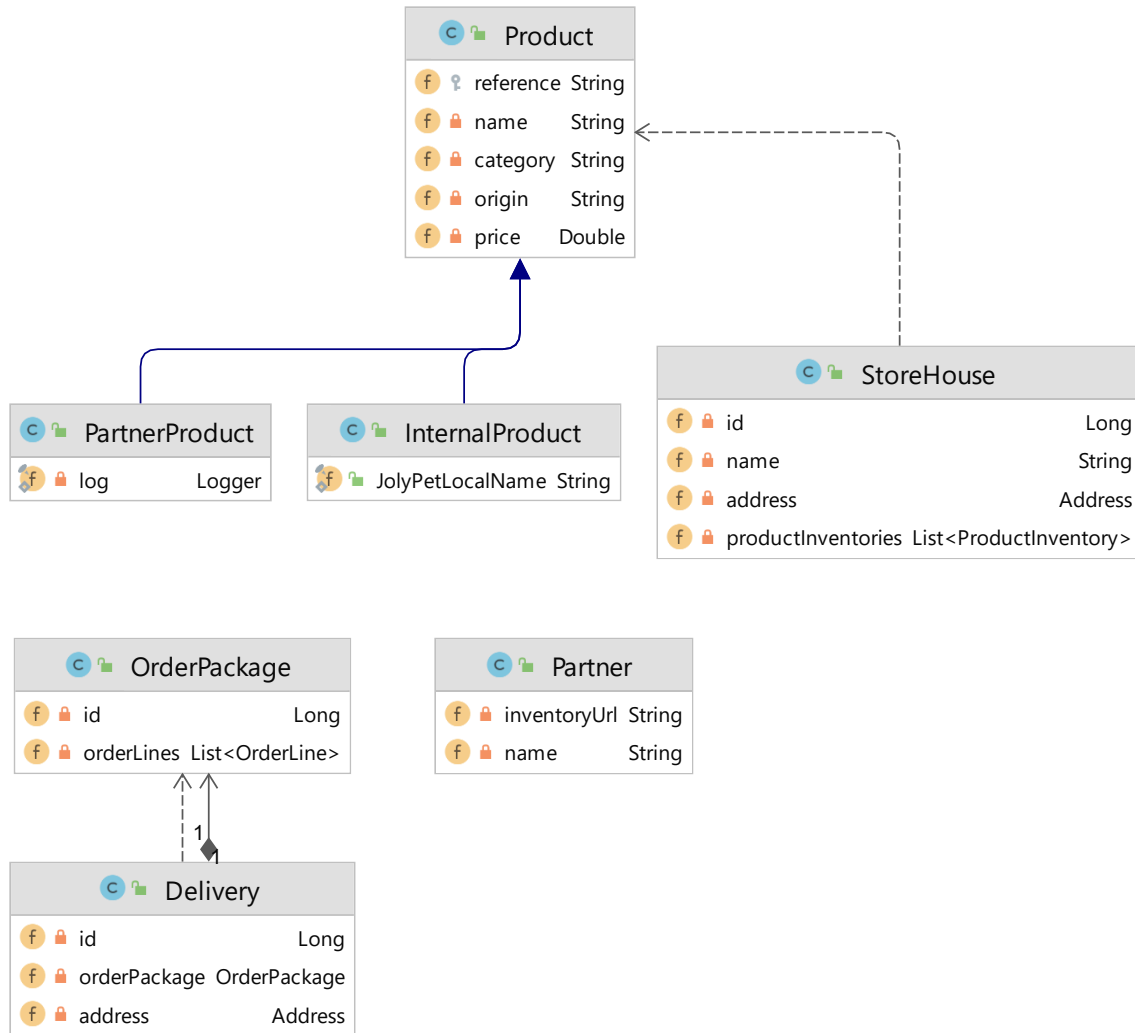
## B. Diagramme de classes (UML)

### 1. Accounting service



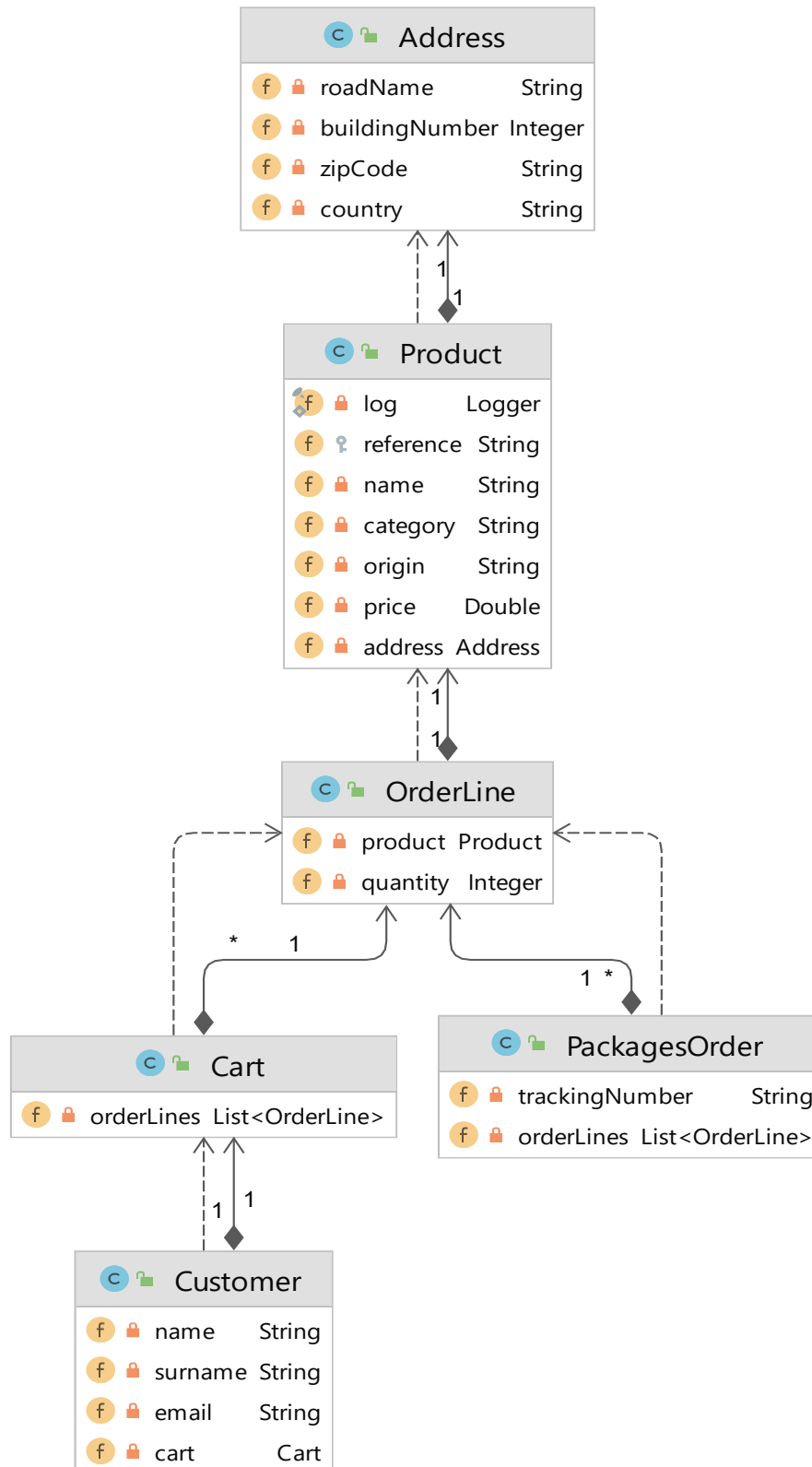
Powered by yFiles

## 2. Catalogue service



Powered by J2U

### 3. CustomerCare service



## IV. Scénarios

### - Achat/Recherche de produit

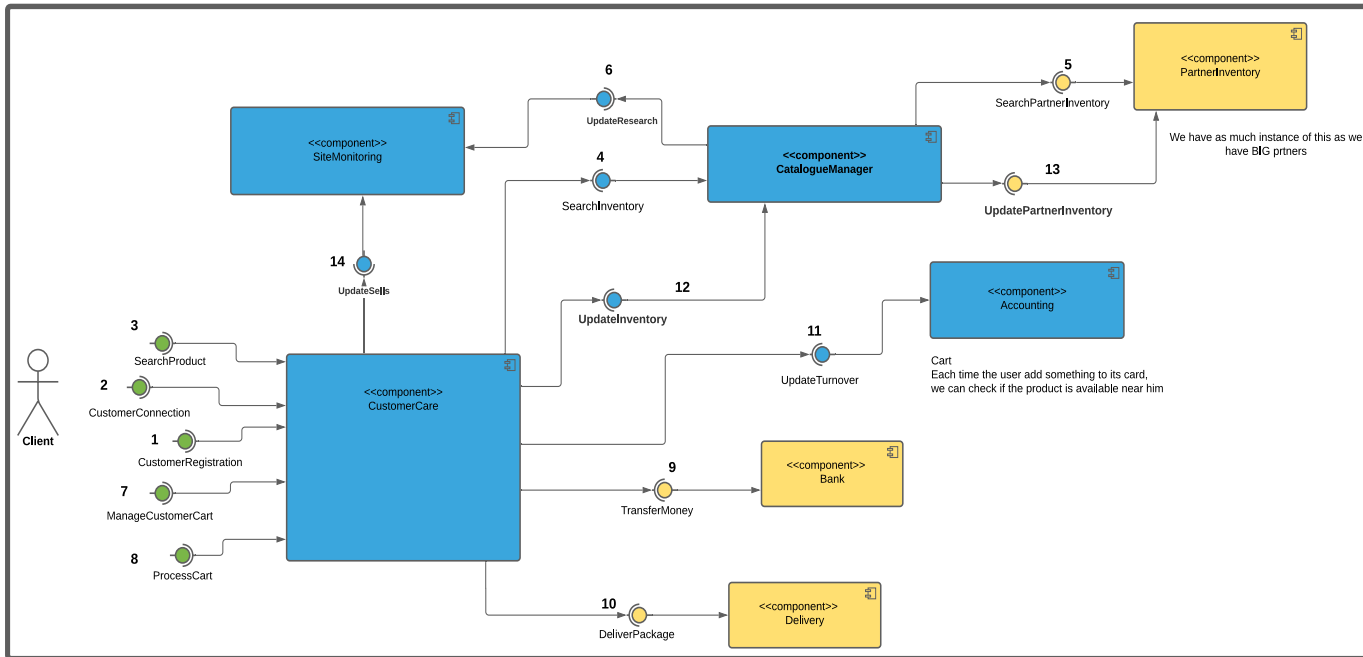


Figure 1 Recherche et achat de produit

Le déroulement de ce scénario commence par une initialisation. Pendant cette phase nous créons les produits 'Laisse' vendu par PolyPet et 'Caoutchouc' vendu par le petit partenaire (ZooPlus) tous les deux de la catégorie « Caniche » ; ils seront stockés dans un magasin(store) que nous créons également. Ensuite nous créons le produit Medium Maxi Adult dans le grand partenaire (Ultima). Et pour finir la phase d'initialisation nous souscrivons le partenaire au catalogue en envoyant l'adresse du service catalogue au partenaire, ce dernier contactera alors le service catalogue en faisant un appel Post contenant l'adresse à laquelle il est accessible ; une fois l'appel effectué le partenaire est connu du catalogue en enregistrant son nom et adresse dans la table 'Partner'. Après s'être inscrit au catalogue, le partenaire envoie les données liées à ses produits (référence, nom...) au catalogue.

Après la phase d'initialisation l'utilisateur s'enregistre (**1&2**) en envoyant un appel Post contenant son adresse email vers le **CustomerCare service**. Le retour de cet appel est une chaîne de caractères contenant un token qui sera plus tard utilisé pour ajouter les produits au panier et à procéder à un achat. Une fois enregistré et connecté l'utilisateur effectue une recherche de produits. Il recherche (**3**) alors dans un premier temps la catégorie caniche et ensuite par nom « Medium Maxi Adult ». Le service CustomerCare sera alors chargé de relayer cette requête au service **CatalogueManager (4)**. Ce dernier doit retourner les données concernant un produit ainsi que l'adresse à laquelle il est stocké. Il va alors immédiatement retrouver les produits de la catégorie « Caniche » car les deux produits sont stockés dans un des entrepôts de Polypet. Cependant n'ayant seulement le nom ainsi que la référence du produit (l'adresse est manquante), il va alors faire un appel *GET* avec la référence du produit vers le partenaire (**5**) (origine du produit) afin de retrouver cette information. Une fois récupérés les résultats (Produits et adresses) sont retournés au CatalogueManager, puis au CustomerCare et finalement au client. Chaque recherche effectuée est interceptée (**6**) par au niveau du catalogue service pour mettre à jour les données servant de statistiques dans le service SiteMonitoring ; des appels



Posts sont envoyés pour faire la mise à jour. Les produits étant récupérés le client peut alors procéder à leur ajout dans son panier(7), un appel post contenant le token la référence du produit ainsi que la quantité désirée est envoyé au service CustomerCare qui ajoute le *OrderLine* dans le panier du client correspondant. A cette é tape nous récupérons aussi la quantité courante de chaque produit pour faire la différence avant et après achat. Lorsque le client valide son achat (8), un appel Post contenant les coordonnées bancaires du client et l'adresse de livraison est envoyé au service CustomerCare. Il demande au catalogue s'il possède la quantité nécessaire pour l'achat qui veut être valider, lui à son tour demande au partenaire si nécessaire. Il se charge ensuite de contacter la banque (9) dans avec les coordonnées bancaires reçues. Après réponse positive de la banque, le service CustomerCare va se charger de transformer le panier en commande(*OrderPackage*) pour ensuite envoyer une requête Post contenant la commande et la destination au service externe Delivery (10). La même requête sera envoyée au service Accounting(11). Après cela le stock est mis à jour par un appel envoyé au catalogue (12) qui à son tour demande aux partenaires (13) de mettre à jour les produits achetés chez eux. Une interception est faite au niveau du service CustomerCare pour mettre à jour les statistiques (14).

- Mettre à jour le dépôt du site web (stockage)

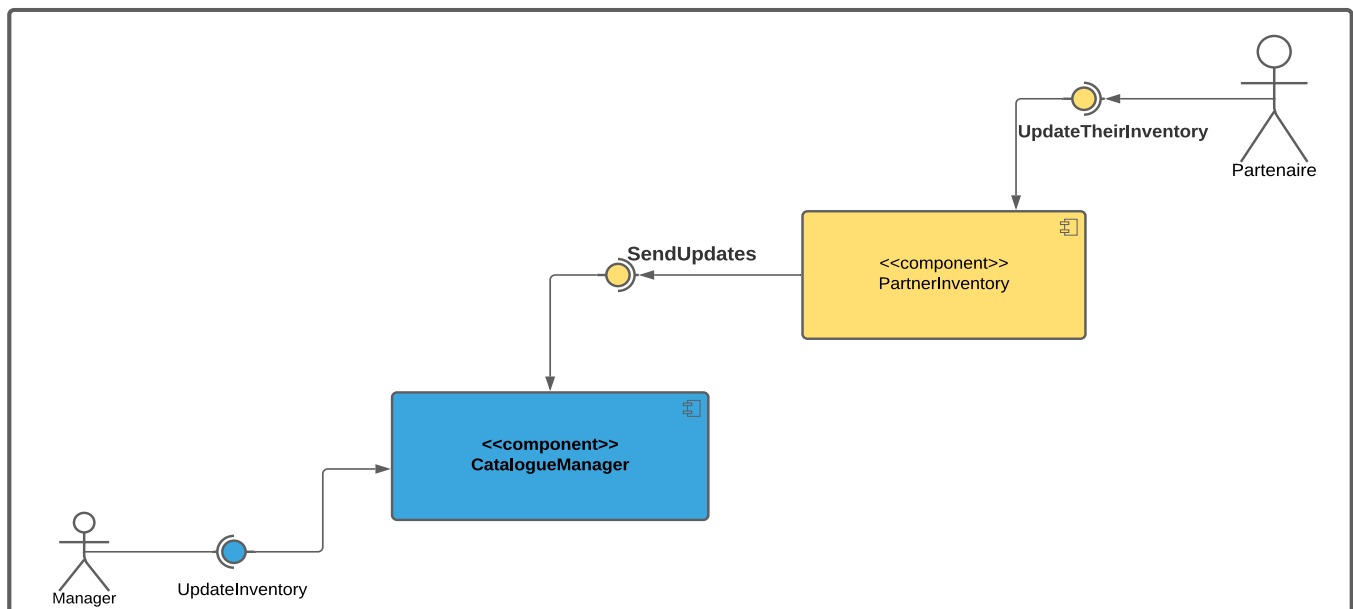
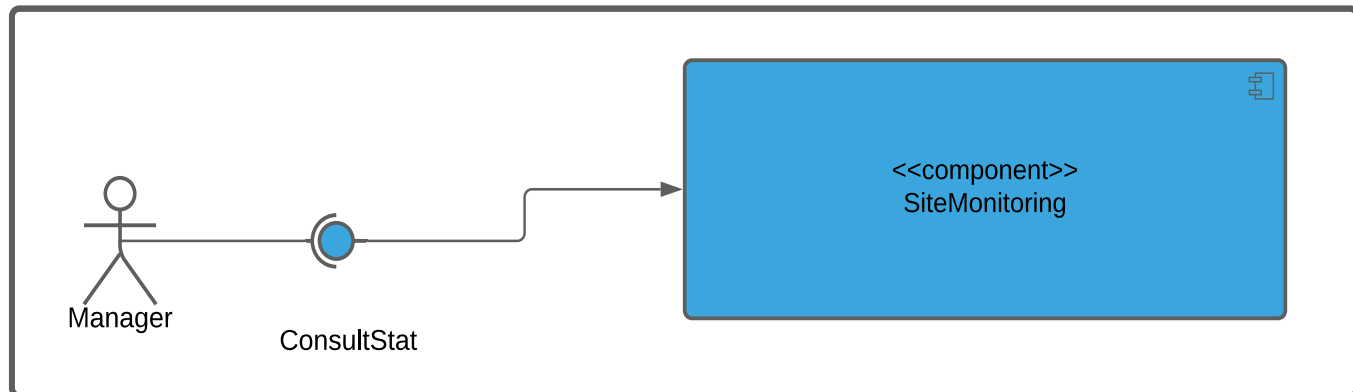


Figure 2 Mise à jour du catalogue

Pour ce scenario nous avons principalement repris la phase d'initialisation du scenario d'achat. Nous créons des produits, les stockons puis changeons leur quantité ainsi que leur prix. Un manager de **Polypet** et un manager d'un partenaire **interne** passent par la même interface. Lorsqu'un le prix du produit est modifié chez le partenaire, ce dernier fait un appel post contenant le produit modifié vers le catalogue via une autre interface.

- Consulter les statistiques afin d'adapter une stratégie



*Figure 3 Consultation des statistiques*

Pour ce scenario nous reprenons le scenario 1 pour la génération de données de recherches et d'achat. Ensuite nous contactons le service SiteMonitoring pour avoir accès aux statistiques des données générées.

## V. Supervision du système

Afin de superviser notre système sur GCP nous avons créé certains Policy. Une Policy correspond à une notification. Ici nous avons 2 Policy, le 1<sup>er</sup> correspond à monitorer et surveiller l'usage de la mémoire de notre cluster sur Kubernetes. Et la 2<sup>ème</sup> correspond surveille le nombre de connexions sur 10 minutes à notre base de données PostgreSQL.

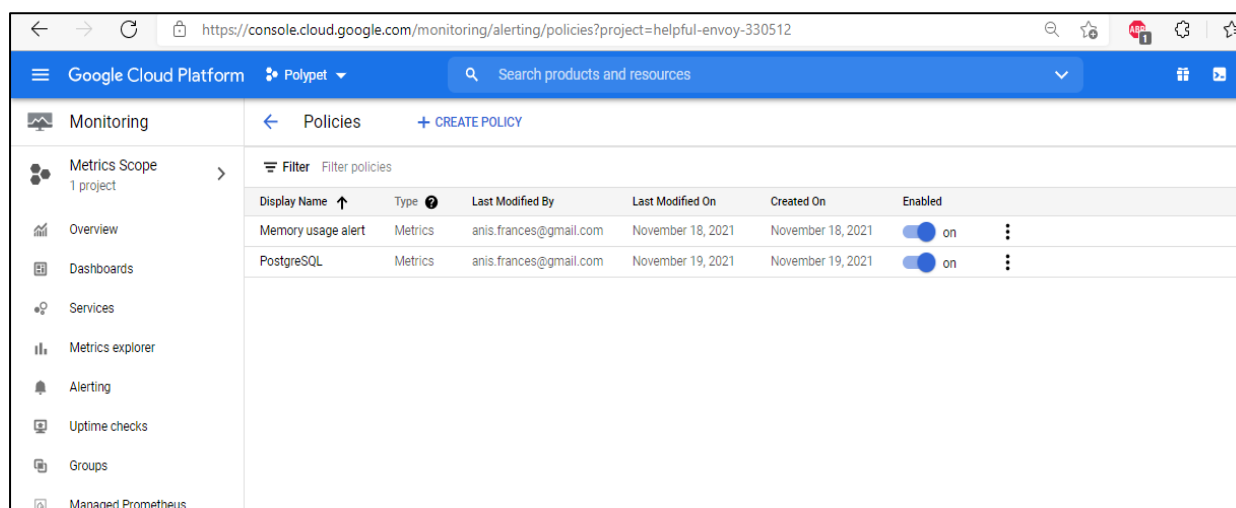
Lorsque la condition est violée un mail sera envoyé à un développeur de l'équipe. Il s'agit de notre canal de notification.

Le nombre de connexions sur notre serveur de base de données est un bon indicateur de combien de clients / trafic nous avons réellement étant donné que tous les services l'utilisent. Ainsi l'alerte sur la base de données permet aux développeurs de détecter s'il est nécessaire de faire des optimisations sur la base de données ou augmenter les ressources qui lui sont allouées.

De même l'alerte de consommation mémoire permet de détecter s'il est nécessaire d'augmenter manuellement les ressources de nos Pods, le nombre d'instances de nos services ou le nombre de nœuds. Ainsi ces alertes viennent en complément au système automatisé de passage à l'échelle de Kubernetes Engine.

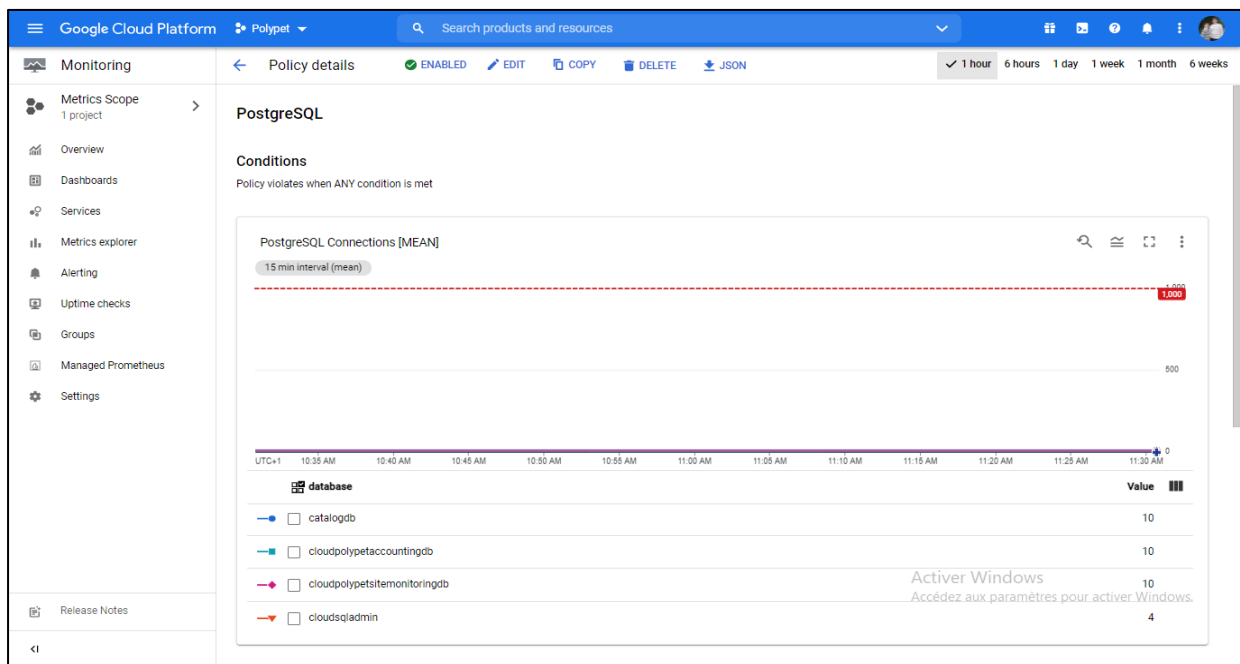
Une autre alerte intéressante qui serait la surveillance des logs pour détecter si un bug a lieu en production afin de pouvoir les réparer au plus vite pour limiter l'insatisfaction des clients. Ou encore une alert lors du retour d'une réponse 5xx, qui signale une erreur interne, d'une service donnée.

Vous retrouverez ci-dessous les captures d'écrans de nos policies.



The screenshot shows the Google Cloud Platform Monitoring Policies page. The left sidebar contains navigation links: Monitoring, Metrics Scope (1 project), Overview, Dashboards, Services, Metrics explorer, Alerting, Uptime checks, Groups, and Managed Prometheus. The main content area is titled 'Policies' and includes a '+ CREATE POLICY' button. Below this is a table listing the policies.

Display Name	Type	Last Modified By	Last Modified On	Created On	Enabled
Memory usage alert	Metrics	anis.frances@gmail.com	November 18, 2021	November 18, 2021	on
PostgreSQL	Metrics	anis.frances@gmail.com	November 19, 2021	November 19, 2021	on

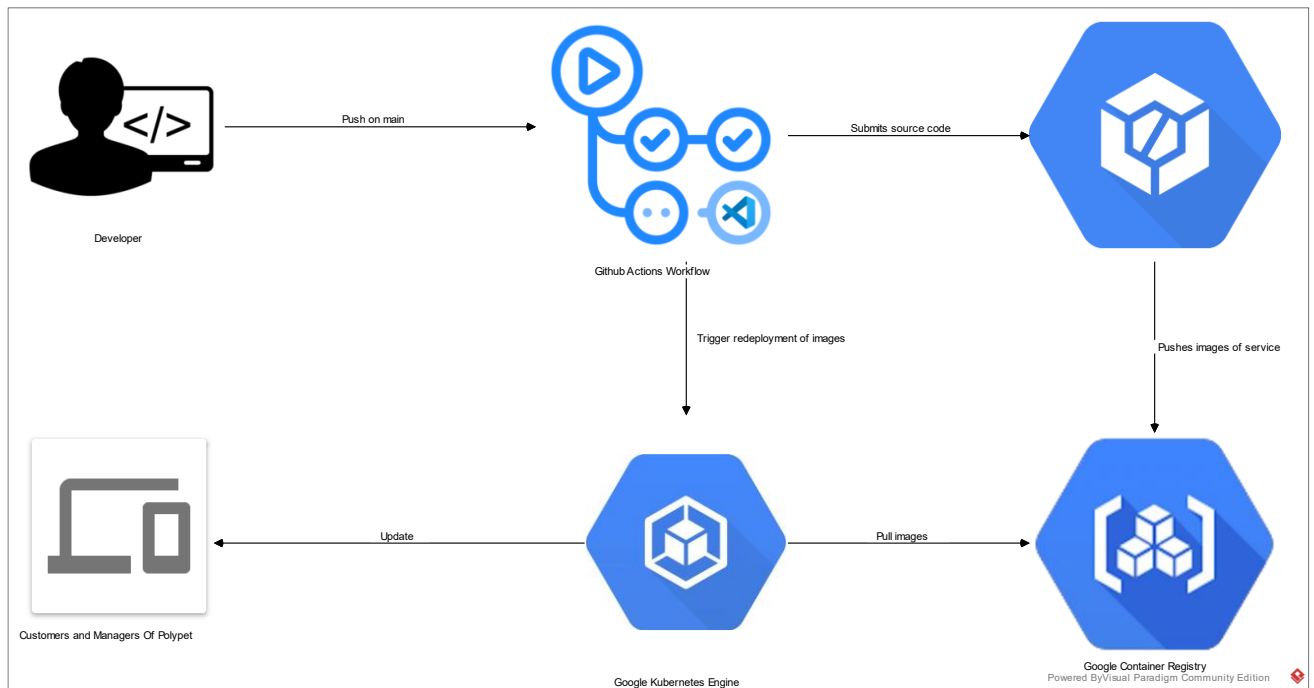


## VI. Système de déploiement

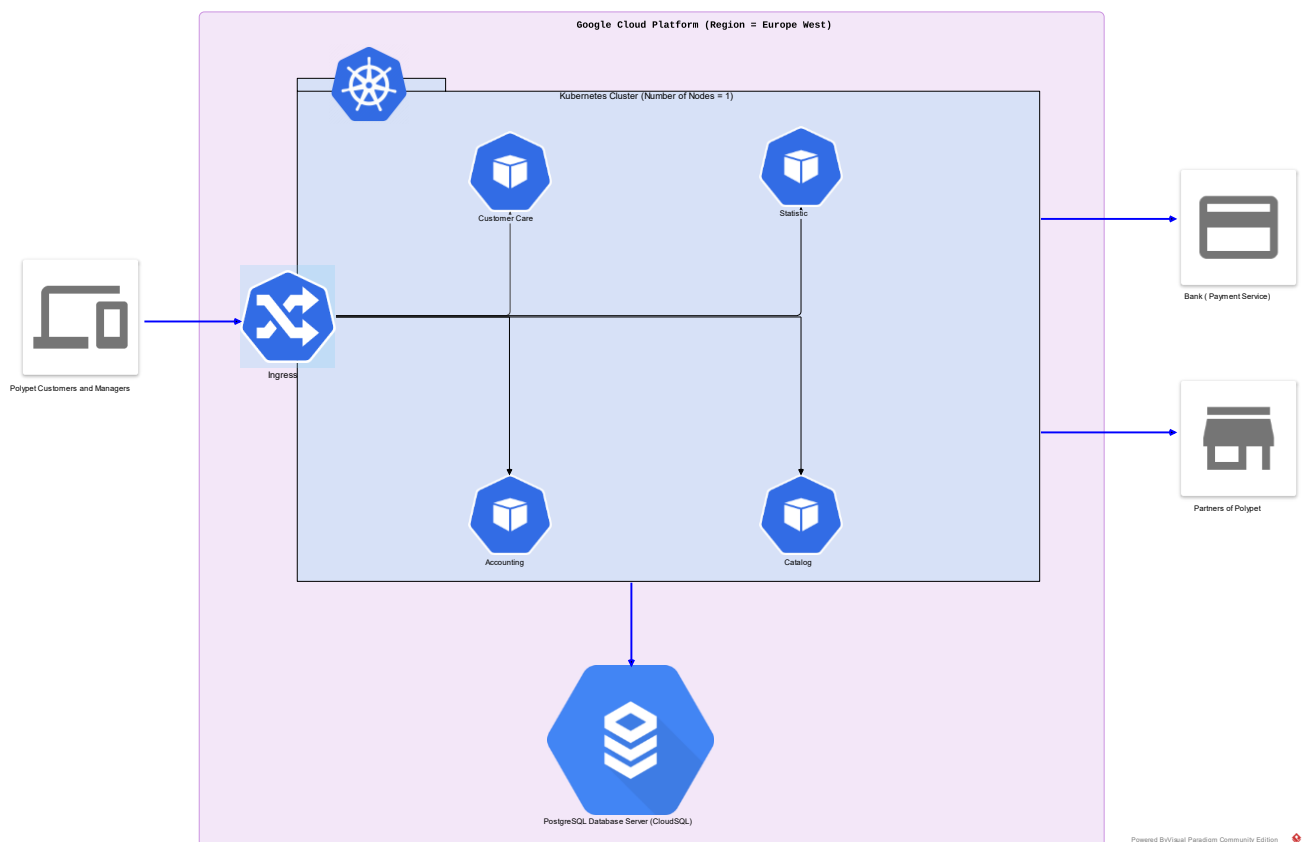
### A. Intégration continue

Nous avons mis en place un déploiement automatisé en utilisant Github Actions. Lorsqu'un développeur publie du code sur la branche *main*, des tests d'intégration sont déclenchés et exécutés sur ce code. Si les tests passent le code est soumis à Google Cloud Build qui crée des images Docker pour chacun des services de Polypet et les envoie sur Google Container Registry.

Notre workflow déclenche ensuite le redéploiement des Pods, Services et autres ressources Kubernetes en utilisant la configuration YAML et des modifications des variables d'environnement obligeant Google Kubernetes Engine à utiliser les dernières images de nos services. Nous suivons le principe 3 des "12 facteurs" en utilisant les secrets de GitHub pour éviter de mettre nos clés de sécurité et nos mots de passe dans le code source sur GitHub. Nous utilisons les variables d'environnement afin de fournir à nos services les informations de configuration comme les liens de la base de données et des autres services et les mots de passe.



## B. Diagramme de déploiement de Polypet sur Google cloud Platform



Nos services sont déployés dans un Kubernetes, chaque service est dans un pod et ils communiquent entre eux en utilisant les noms des services de façon similaire à docker-compose dans notre environnement de test. Nous avons un serveur de bases de données PostgreSQL sur Cloud SQL. Ce serveur est utilisé par l'ensemble des services, chaque service à sa propre base de données dans ce serveur. Des backups du serveur sont créés régulièrement. Il peut aussi être configuré pour utiliser des réplicas.

Pour appeler nos services nous exposons un unique Endpoint en utilisant un Ingress qui va rediriger les requêtes sur Polypet en fonction de l'url.

Ainsi par défaut les requêtes sont redirigées sur Customer Care sauf si l'URL commence par catalog, sitemonitoring ou accounting, auquel cas elles sont redirigées sur le service correspondant.

Notre cluster est localisé en Europe qui correspond à la région que Polypet sert principalement.

Pour le déploiement de nos services nous avons utilisé un seul nœud mais pour un déploiement réel nous utiliserions plus de nœuds en fonction du nombre de clients attendus.

Pour notre déploiement nous avons fait du **Caas** parce que nos services utilisaient du JDK17 qui n'était pas disponible sur Google Cloud en Paas et les conteneurs nous permettent de contrôler quelle version et binaire du JDK on utilise et donc d'obtenir les mêmes comportements en test et en production.

Nous avons choisi Kubernetes parce qu'il s'agissait d'une solution qui utilisait les conteneurs, mais qui n'est pas spécifique à Google Cloud. Ainsi Polypet ne serait pas dépendant de Google et pourrait déplacer le déploiement vers d'autres fournisseurs.

Aussi le choix du Caas nous a permis d'éviter un vendor lock-in, notamment en conteneurisant nos images docker et en spécifiant les dépendances dont notre application a besoin dans ces images pour s'exécuter sur le serveur du google cloud plateforme. Ainsi le GCP s'occupera du reste (OS, Stockage, middleware, etc.), pendant que notre équipe se concentre sur la partie du développement de l'application PolyPet.

## VII. Scénario de démonstration MVP

Gilbert souhaite acheter une laisse pour son chien. Il choisit de faire cet achat chez 'PolyPet' via leur application qui donne accès à un catalogue regroupant plusieurs produits de plusieurs vendeurs. Pendant sa recherche d'article, il décide de profiter de cette occasion pour acheter des produits supplémentaires : un sac de croquettes pour chien 'Medium Maxi - Adult' vendu par 'Ultima' (un gros partenaire de PolyPet) ainsi qu'une balle en caoutchouc vendu par 'ZooPlus' un commerçant (un petit partenaire).

Une fois le panier de Gilbert rempli, il souhaite passer à la validation de son achat en visualisant une dernière fois le contenu de son panier. L'application PolyPet fera alors une vérification de stock afin de s'assurer que tous les produits du panier sont disponibles chez les différents partenaires. Heureusement, tous les produits du panier sont disponibles en stock et Gilbert peut à présent passer au paiement de son panier.

Gilbert entre alors les informations de sa carte bancaire dans PolyPet, et l'application se charge de relayer ces informations au système de banque. Le système bancaire retourne un signal 'OK' à PolyPet et le paiement est validé. PolyPet génère alors un numéro de commande et la communique à Gilbert.

Finalement une demande est envoyée à l'entreprise Joly Drone, le partenaire de PolyPet chargé d'effectuer les livraisons à nos clients par drone.

## VIII. Estimation du prix de déploiement

Pour le déploiement de notre solution Polypet nous utilisons une solution IaaS pour notre serveur de base de données et nous utilisons des solutions CaaS pour le déploiement de nos services.

### A. Google Cloud

Sur Google Cloud nous utilisons une **Compute Engine** qui héberge notre serveur de base de données, et qui contient toutes les bases de données des services. Nous basons notre cluster en Europe. En se basant sur les indicateurs de prix disponibles [ici](#). Nous estimons le coût de notre base de données sur un an.

	Prix unitaire mensuel	Quantités	Durée (mois)	Prix
CPU	22,61	8	12	2170,56
Mémoire RAM	3,83	64	12	2941,44
Stockage	0,09	256	12	276,48
Adresse IPv4	8	1	12	96
				5484,48





Nous n'incluons pas le coût du trafic internet parce que ce trafic est effectué depuis nos services qui sont aussi hébergés sur des produits Google qui sont Europe (ce qui est gratuit).

Pour héberger les services qui composent le site de Polypet, nous utilisons Kubernetes.

Sur Google Cloud nous avons la possibilité de choisir entre GKE Autopilot et GKE Standard (GKE= Google Kubernetes Engine). Nous avons choisi GKE Autopilot parce que cette solution attribue les ressources aux pods dans nos nœuds puis les fait évoluer en fonction de l'utilisation. Cette solution peut donc permettre pour la première année de ne pas prendre trop de risques à surestimer ou sous-estimer l'utilisation du site Polypet.

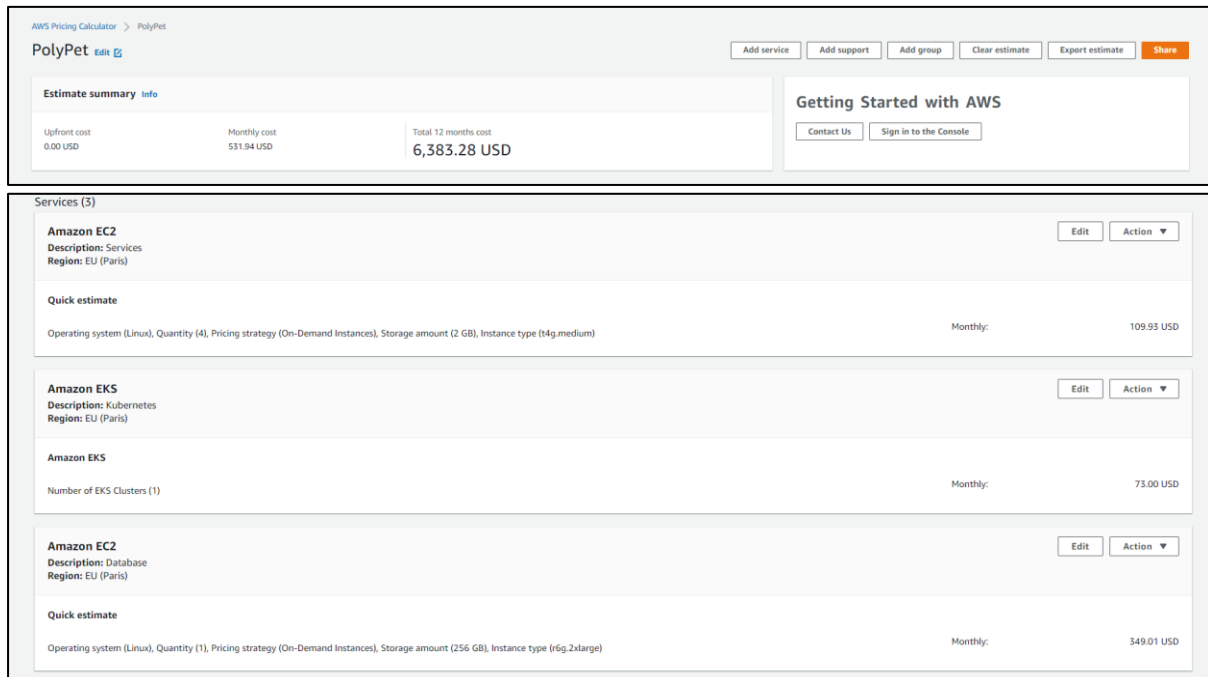
En se basant sur le calculateur de prix de Google Cloud et en attribuant des ressources de (2 CPU, 4 GB de mémoire, et 2 GB de stockage temporaires pour chacun des Pods de notre cluster Kubernetes) nous parvenons à un coût de 170 dollars par mois pour notre cluster Kubernetes. Sur une année nous avons donc **2040 dollars** pour la première année.



GKE Autopilot		
1 x		
Total vCPU Hours: 1,642.5		
Total Memory Hours: 2,920		
Total Storage Hours: 1,460		
Region: Belgium		
Estimated Component Cost: USD 96.40 per 1 month		
GKE Cluster Management Fee		
Regional Cluster: 730 hours		
USD 73.00		
<b>Total Estimated Cost: USD 169.40 per 1 month</b>		

## B. Amazon Web Services

Nous avons utilisé le 'AWS Pricing Calculator' afin d'analyser et de comparer les coûts prévisionnels au déploiement de notre application. Pour automatiser le déploiement de nos conteneurs en fonction de la charge, nous avons choisi de nous appuyer sur le service 'Elastic Kubernetes Service' (EKS) d'Amazon. En ce qui concerne les machines utilisées pour les services ainsi que la base de données, nous avons utilisé deux 'Elastic Compute Cloud' (EC2) avec les mêmes configurations que nous avons utilisées sur GCP.



**Estimate summary**

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	531.94 USD	6,383.28 USD

**Getting Started with AWS**

[Contact Us](#) [Sign in to the Console](#)

**Services (3)**

Service	Description	Region	Quick estimate	Monthly
Amazon EC2	Description: Services	EU (Paris)	Operating system (Linux), Quantity (4), Pricing strategy (On-Demand Instances), Storage amount (2 GB), Instance type (t4g.medium)	109.93 USD
Amazon EKS	Description: Kubernetes	EU (Paris)	Number of EKS Clusters (1)	73.00 USD
Amazon EC2	Description: Database	EU (Paris)	Operating system (Linux), Quantity (1), Pricing strategy (On-Demand Instances), Storage amount (256 GB), Instance type (r6g.2xlarge)	349.01 USD

Nous avons également quantifié le trafic au sein de notre application afin d'avoir une estimation des coûts que cela peut générer. Pour cela, nous avons analysé les différents appels de notre application et avons enregistré les informations suivantes :

			Calls						
			Internal				External		
		Calls per day	Customer Care	Accounting	Inventory	SiteMonitoring	Bank	Partner Inv.	Delivery
Scenario 1	Estimated local purchases	105 168	1	1			1		1
	Estimated big partner purchases	21 034	1	1			1	1	1
Scenario 2	Estimated update inventory (Local + Big partner)	315 504			1				
Scenario 3	Estimated consult stats	10 517				1			
Scenario 4	Estimated local product searches	631 008	1		1				
	Estimated big partner product searches	628 461	1		1			1	
Other 1	Estimated Customer authentication	42 067	1						
Other 2	Estimated Customer Registration	42 218	1						
Total (per day)			1 469 956	126 202	1 574 973	10 517	126 202	649 495	126 202
Total (per month)			44 098 680	3 786 060	47 249 190	315 510	3 786 060	19 484 850	3 786 060
Total (per year)			529 184 160	45 432 720	566 990 280	3 786 120	45 432 720	233 818 200	45 432 720

Nombre d'appels (internes/externes) pour chaque scénario

Payload	PaymentRequest (CreditCard, BankAccount, Amount)		ProductInventory Product Reference (String) + Quantity (Integer)		DeliveryDTO (PackagesOrder, Address)	
Size(b/Mo) per call	109	0	15		279	0
perDay	13 756 018	13	9 742 425	9	35 210 358	34
perMonth	412 680 540	394	56 790 900	54	1 056 310 740	1 007
perYear	4 952 166 480	4 723	681 490 800	650	12 675 728 880	12 089

*Taille des appels sortants*

AWS facture seulement les appels sortants de l'application et non les appels entrants. Cela nous permettra d'économiser des coûts supplémentaires, notamment dû aux appels interservices. De plus, AWS facture les appels en fonction de la taille des données utilisé par mois et non par le nombre d'appels. Il était donc nécessaire d'analyser les appels sortants de notre application ainsi que leurs payload. Cela nous a donc permis d'avoir une estimation sur la taille des données envoyé par chaque scénario vers chaque service externe. Finalement, nous avons effectué une estimation supplémentaire afin d'évaluer la charge de l'application en fonction du nombre de clients/visiteurs.

En conclusion, cette étude nous a permis de définir les services qui devront pouvoir supporter le plus d'appels externes, et d'avoir une estimation sur le prix que cela peut nous coûter. Dans notre cas, le CustomerCare est le service qui déclenche le plus d'appels mensuel, mais la taille des appels reste raisonnable car cela reste en dessous des 2Go même pour un trafic surestimé (ce qui revient à moins de 5 USD). Toutefois, il est important de porter une attention à ce facteur si l'application est amenée à évoluer par l'implémentation d'un nouveau service exploitant un nouveau service, par exemple on pourrait imaginer un service externe chargée de produire des annonces publicitaires, vers lequel nous enverrons régulièrement des images. Dans ce cas la taille des envois seraient bien plus grandes et nous devrions apporter une attention particulière à ces coûts.

À la suite de ces comparaisons, nous pouvons également conclure que notre solution est moins chère sur AWS que sur GCP.

## IX. Conclusion & répartition des points

Afin d'assurer le bon déroulement de ce projet, il était tout d'abord nécessaire d'appliquer les notions que nous avons étudié en cours 'Architecture Logicielle' et 'Service Oriented Application' pour optimiser le découpage de notre application en micro-services. Cela nous avait également permis de définir les différents échanges que chaque service sera amené à effectuer pour remplir les besoins d'un scénario.

Suivant notre premier rendu, nous avons revu une partie de l'architecture, notamment les services qui permettaient de retrouver les produits, ces dernières ayant un découpage beaucoup trop fin, nous avons choisi de fusionner le 'Search Engine' ainsi que 'Inventory' dans un service 'Catalogue'. Nous avons également suivi cette même logique pour la création du Customer Service.

D'un point de vue technique, le projet nous a permis d'explorer les différentes solutions cloud, et nous a permis de prendre connaissance des avantages ainsi que des inconvénients, pour un besoin donné. Cela nous a alors sensibilisé à l'exploitation des ressources afin d'estimer les coûts du déploiement d'une application.

### Répartition des points

ANAGONOU Sourou Patrick 100pt : Déploiement des services sur GCP + Intégration continue + Coûts prévisionnels du déploiement sur GCP

ANIGLO Vihoalé Jonas 100pt : Scénario Achat/recherche d'un produit + gestion de stock + mise à jour de l'inventaire

FRANCIS Anas 100pt : scénario Achat/recherche d'un produit + Consultation des statistiques + Monitoring in GCP + Intégration continue

ZABOURDINE Soulaïman 100pt : Coûts prévisionnels du déploiement sur AWS + scénario de livraison + services externes (Banque & partenaire)